Cobbler Documentation

Release 3.4.0

Enno Gotthold

Apr 17, 2024

CONTENTS

Cobbler is a provisioning (installation) and update server. It supports deployments via PXE (network booting), virtualization (Xen, QEMU/KVM, or VMware), and re-installs of existing Linux systems. The latter two features are enabled by usage of 'Koan' on the remote system. Update server features include yum mirroring and integration of those mirrors with automated installation files. Cobbler has a command line interface, WebUI, and extensive Python and XML-RPC APIs for integration with external scripts and applications.

If you want to explore tools or scripts which are using Cobbler please use the GitHub Topic: https://github.com/topics/cobbler

Here you should find a comprehensive overview about the usage of Cobbler.

CHAPTER

ONE

QUICKSTART

Cobbler can be a somewhat complex system to get started with, due to the wide variety of technologies it is designed to manage, but it does support a great deal of functionality immediately after installation with little to no customization needed. Before getting started with Cobbler, you should have a good working knowledge of PXE as well as the automated installation methodology of your chosen distribution(s).

We will assume you have successfully installed Cobbler, please refer to the *Installation Guide* for instructions for your specific operating system. Finally, this part guide will focus only on the CLI application.

1.1 Preparing your OS

1.1.1 SELinux

Before getting started with Cobbler, it may be convenient to either disable SELinux or set it to "permissive" mode, especially if you are unfamiliar with SELinux troubleshooting or modifying SELinux policy. Cobbler constantly evolves to assist in managing new system technologies, and the policy that ships with your OS can sometimes lag behind the feature-set we provide, resulting in AVC denials that break Cobbler's functionality.

1.1.2 Firewall

TBD

1.2 Changing settings

Before starting the *cobblerd* service, there are a few things you should modify.

Settings are stored in `/etc/cobbler/settings.yaml`. This file is a YAML formatted data file, so be sure to take care when editing this file as an incorrectly formatted file will prevent *cobblerd* from running.

1.2.1 Default encrypted password

This setting controls the root password that is set for new systems during the handsoff installation.

```
default_password_crypted: "$1$bfI7WLZz$PxXetL97LkScqJFxnW7KS1"
```

You should modify this by running the following command and inserting the output into the above string (be sure to save the quote marks):

```
openssl passwd -1
```

1.2.2 Server and next_server

The `server` option sets the IP that will be used for the address of the Cobbler server. **DO NOT** use 0.0.0.0, as it is not the listening address. This should be set to the IP you want hosts that are being built to contact the Cobbler server on for such protocols as HTTP and TFTP.

```
server: 127.0.0.1
```

The `next_server` option is used for DHCP/PXE as the IP of the TFTP server from which network boot files are downloaded. Usually, this will be the same IP as the server setting.

```
next_server: 127.0.0.1
```

1.3 DHCP management and DHCP server template

In order to PXE boot, you need a DHCP server to hand out addresses and direct the booting system to the TFTP server where it can download the network boot files. Cobbler can manage this for you, via the `manage_dhcp` setting:

```
manage_dhcp: 0
```

Change that setting to 1 so Cobbler will generate the `dhcpd.conf` file based on the `dhcp.template` that is included with Cobbler. This template will most likely need to be modified as well, based on your network settings:

```
$ vi /etc/cobbler/dhcp.template
```

For most uses, you'll only need to modify this block:

```
subnet 192.168.1.0 netmask 255.255.255.0 {
    option routers             192.168.1.1;
    option domain-name-servers 192.168.1.210,192.168.1.211;
    option subnet-mask         255.255.255.0;
    filename                   "/pxelinux.0";
    default-lease-time         21600;
    max-lease-time             43200;
    next-server                $next_server_v4;
}
```

No matter what, make sure you do not modify the `next-server $next_server_v4;` line, as that is how the next server setting is pulled into the configuration. This file is a cheetah template, so be sure not to modify anything starting after this line:

```
#for dhcp_tag in $dhcp_tags.keys():
```

Completely going through the `dhcpd.conf` configuration syntax is beyond the scope of this document, but for more information see the man page for more details:

```
$ man dhcpd.conf
```

1.4 Notes on files and directories

Cobbler makes heavy use of the `/var` directory. The `/var/www/cobbler/distro_mirror` directory is where all of the distribution and repository files are copied, so you will need 5-10GB of free space per distribution you wish to import.

If you have installed Cobbler onto a system that has very little free space in the partition containing `/var`, please read the *Relocating your installation* section of the Installation Guide to learn how you can relocate your installation properly.

1.5 Starting and enabling the Cobbler service

Once you have updated your settings, you're ready to start the service:

```
$ systemctl start cobblerd.service
$ systemctl enable cobblerd.service
$ systemctl status cobblerd.service
```

If everything has gone well, you should see output from the status command like this:

```
cobblerd.service - Cobbler Helper Daemon
    Loaded: loaded (/lib/systemd/system/cobblerd.service; enabled)
      Active: active (running) since Sun, 17 Jun 2012 13:01:28 -0500; 1min 44s ago
    Main PID: 1234 (cobblerd)
      CGroup: name=systemd:/system/cobblerd.service
              └ 1234 /usr/bin/python /usr/bin/cobblerd -F
```

1.6 Checking for problems and your first sync

Now that the *cobblerd* service is up and running, it's time to check for problems. Cobbler's check command will make some suggestions, but it is important to remember that these are mainly only suggestions and probably aren't critical for basic functionality. If you are running iptables or SELinux, it is important to review any messages concerning those that check may report.

```
$ cobbler check
The following are potential configuration items that you may want to fix:

1. ....
2. ....
```

Restart *cobblerd* and then run `cobbler sync` to apply changes.

If you decide to follow any of the suggestions, such as installing extra packages, making configuration changes, etc., be sure to restart the *cobblerd* service as it suggests so the changes are applied.

Once you are done reviewing the output of `cobbler check`, it is time to synchronize things for the first time. This is not critical, but a failure to properly sync at this point can reveal a configuration problem.

```
$ cobbler sync
task started: 2012-06-24_224243_sync
task started (id=Sync, time=Sun Jun 24 22:42:43 2012)
running pre-sync triggers
...
rendering DHCP files
generating /etc/dhcp/dhcpd.conf
```

(continues on next page)

(continued from previous page)

```
cleaning link caches
running: find /var/lib/tftpboot/images/.link_cache -maxdepth 1 -type f -links 1 -exec␣
↪rm -f '{}' ';'
received on stdout:
received on stderr:
running post-sync triggers
running python triggers from /var/lib/cobbler/triggers/sync/post/*
running python trigger cobbler.modules.sync_post_restart_services
running: dhcpd -t -q
received on stdout:
received on stderr:
running: service dhcpd restart
received on stdout:
received on stderr:
running shell triggers from /var/lib/cobbler/triggers/sync/post/*
running python triggers from /var/lib/cobbler/triggers/change/*
running python trigger cobbler.modules.scm_track
running shell triggers from /var/lib/cobbler/triggers/change/*
*** TASK COMPLETE ***
```

Assuming all went well and no errors were reported, you are ready to move on to the next step.

1.7 Importing your first distribution

Cobbler automates adding distributions and profiles via the `cobbler import` command. This command can (usually) automatically detect the type and version of the distribution your importing and create (one or more) profiles with the correct settings for you.

1.7.1 Download an ISO image

In order to import a distribution, you will need a DVD ISO for your distribution.

Note: You must use a full DVD, and not a "Live CD" ISO. For this example, we'll be using the Fedora 17 x86_64 ISO.

Warning: When running Cobbler via systemd, you cannot mount the ISO to `/tmp` or a sub-folder of it because we are using the option *Private Temporary Directory*, to enhance the security of our application.

Once this file is downloaded, mount it somewhere:

```
$ mount -t iso9660 -o loop,ro /path/to/isos/Fedora-17-x86_64-DVD.iso /mnt
```

1.7.2 Run the import

You are now ready to import the distribution. The name and path arguments are the only required options for import:

```
$ cobbler import --name=fedora17 --arch=x86_64 --path=/mnt
```

The `--arch` option need not be specified, as it will normally be auto-detected. We're doing so in this example in order to prevent multiple architectures from being found.

Listing objects

If no errors were reported during the import, you can view details about the distros and profiles that were created during the import.

```
$ cobbler distro list
$ cobbler profile list
```

The import command will typically create at least one distro/profile pair, which will have the same name as shown above. In some cases (for instance when a Xen-based kernel is found), more than one distro/profile pair will be created.

Object details

The report command shows the details of objects in Cobbler:

```
$ cobbler distro report --name=fedora17-x86_64
Name                           : fedora17-x86_64
Architecture                   : x86_64
TFTP Boot Files                : {}
Breed                          : redhat
Comment                        :
Fetchable Files                : {}
Initrd                         : /var/www/cobbler/distro_mirror/fedora17-x86_64/
↪images/pxeboot/initrd.img
Kernel                         : /var/www/cobbler/distro_mirror/fedora17-x86_64/
↪images/pxeboot/vmlinuz
Kernel Options                 : {}
Kernel Options (Post Install)  : {}
Automatic Installation Template Metadata : {'tree': 'http://@@http_server@@/cblr/
↪links/fedora17-x86_64'}
Management Classes             : []
OS Version                     : fedora17
Owners                         : ['admin']
Red Hat Management Key         : <<inherit>>
Red Hat Management Server      : <<inherit>>
Template Files                 : {}
```

As you can see above, the import command filled out quite a few fields automatically, such as the breed, OS version, and initrd/kernel file locations. The "Automatic Installation Template Metadata" field (`--autoinstall_meta` internally) is used for miscellaneous variables, and contains the critical "tree" variable. This is used in the automated installation templates to specify the URL where the installation files can be found.

Something else to note: some fields are set to `<<inherit>>`. This means they will use either the default setting (found in the settings file), or (in the case of profiles, sub-profiles, and systems) will use whatever is set in the parent object.

Creating a system

Now that you have a distro and profile, you can create a system. Profiles can be used to PXE boot, but most of the features in Cobbler revolve around system objects. The more information you give about a system, the more Cobbler will do automatically for you.

First, we'll create a system object based on the profile that was created during the import. When creating a system, the name and profile are the only two required fields:

```
$ cobbler system add --name=test --profile=fedora17-x86_64
$ cobbler system list
test
$ cobbler system report --name=test
Name                           : test
TFTP Boot Files                : {}
Comment                        :
Enable gPXE?                   : 0
Fetchable Files                : {}
Gateway                        :
Hostname                       :
Image                          :
IPv6 Autoconfiguration         : False
IPv6 Default Device            :
Kernel Options                 : {}
Kernel Options (Post Install)  : {}
Automatic Installation Template: <<inherit>>
Automatic Installation Template Metadata: {}
Management Classes             : []
Management Parameters          : <<inherit>>
Name Servers                   : []
Name Servers Search Path       : []
Netboot Enabled                : True
Owners                         : ['admin']
Power Management Address       :
Power Management ID            :
Power Management Password      :
Power Management Type          : ipmilanplus
Power Management Username      :
Profile                        : fedora17-x86_64
Proxy                          : <<inherit>>
Red Hat Management Key         : <<inherit>>
Red Hat Management Server      : <<inherit>>
Repos Enabled                  : False
Server Override                : <<inherit>>
Status                         : production
Template Files                 : {}
Virt Auto Boot                 : <<inherit>>
Virt CPUs                      : <<inherit>>
Virt Disk Driver Type          : <<inherit>>
Virt File Size(GB)             : <<inherit>>
Virt Path                      : <<inherit>>
Virt RAM (MB)                  : <<inherit>>
Virt Type                      : <<inherit>>
```

The primary reason for creating a system object is network configuration. When using profiles, you're limited to DHCP interfaces, but with systems you can specify many more network configuration options.

So now we'll setup a single, simple interface in the `192.168.1/24` network:

```
$ cobbler system edit --name=test --interface=eth0 --mac=00:11:22:AA:BB:CC --ip-
↪address=192.168.1.100 --netmask=255.255.255.0 --static=1 --dns-name=test.mydomain.
↪com
```

The default gateway isn't specified per-NIC, so just add that separately (along with the hostname):

```
$ cobbler system edit --name=test --gateway=192.168.1.1 --hostname=test.mydomain.com
```

The `--hostname` field corresponds to the local system name and is returned by the `hostname` command. The `--dns-name` (which can be set per-NIC) should correspond to a DNS A-record tied to the IP of that interface. Neither are required, but it is a good practice to specify both. Some advanced features (like configuration management) rely on the `--dns-name` field for system record look-ups.

Whenever a system is edited, Cobbler executes what is known as a "lite sync", which regenerates critical files like the PXE boot file in the TFTP root directory. One thing it will **NOT** do is execute service management actions, like regenerating the `dhcpd.conf` and restarting the DHCP service. After adding a system with a static interface it is a good idea to execute a full `cobbler sync` to ensure the dhcpd.conf file is rewritten with the correct static lease and the service is bounced.

CHAPTER

TWO

INSTALL GUIDE

Setting up and running *cobblerd* is not a easy task. Knowledge in Apache2 configuration (setting up SSL, virtual hosts, and apache proxy module) is needed. Certificates and some server administration knowledge is required too.

Cobbler is available for installation in several different ways, through packaging systems for each distribution or directly from source.

Cobbler has both definite and optional prerequisites, based on the features you'd like to use. This section documents the definite prerequisites for both a basic installation and when building/installing from source.

2.1 Known packages by distros

This is the most convenient way and should be the default for most people. Production usage is advised only from these four sources or from source with Git Tags.

- Fedora 37 - `dnf install cobbler`
- **CentOS 8**:
  - `dnf install epel-release`
  - `dnf module enable cobbler`
  - `dnf install cobbler`
- openSUSE Tumbleweed - `zypper in cobbler`
- openSUSE Leap 15.x - `zypper in cobbler`

2.2 Prerequisites

2.2.1 Packages

Please note that installing any of the packages here via a package manager (such as dnf/yum or apt) can and will require a large number of ancillary packages, which we do not document here. The package definition should automatically pull these packages in and install them along with Cobbler, however it is always best to verify these requirements have been met prior to installing Cobbler or any of its components.

First and foremost, Cobbler requires Python. Since 3.0.0 you will need Python 3. Cobbler also requires the installation of the following packages:

- A webserver that can act as a proxy (like Apache, Nginx, ...)
- wget and/or curl
- createrepo_c
- xorriso
- Gunicorn

- python-cheetah
- python-dns
- python-requests
- python-distro
- python-netaddr
- python-librepo
- python-schema
- python-gunicorn
- PyYAML / python-yaml
- fence-agents
- rsync
- syslinux
- tftp-server / atftpd

On dnf based systems please also install: `dnf-plugins-core`

If you decide to use the LDAP authentication, please also install manually in any case:

- python3-ldap (or via PyPi: ldap)

If you decide to require Windows auto-installation support, please also install manually:

- python-hivex
- python-pefile

If you are on an apt-based system our operation may be better for mirror detection if the `aptsources` Python module is available.

Koan can be installed apart from Cobbler. Please visit the Koan documentation for details.

Note: Not installing all required dependencies will lead to stacktraces in your Cobbler installation.

2.2.2 Source

Note: Please be aware that on some distributions the python packages are named differently. On Debian based systems everything which is named `something-devel` is named `something-dev` there. Also please remember that the case of some packages is slightly different.

Warning: Some distributions still have Python 2 available. It is your responsibility to adjust the package names to Python3.

Installation from source requires the following additional software:

- git
- make
- python3-devel (on Debian based distributions `python3-dev`)
- python3-Sphinx
- python3-coverage

- openssl

2.3 Installation

Cobbler is available for installation for many Linux variants through their native packaging systems. However, the Cobbler project also provides packages for all supported distributions which is the preferred method of installation.

2.3.1 Packages

We leave packaging to downstream; this means you have to check the repositories provided by your distribution vendor. However we provide docker files for

- Fedora 37
- openSUSE Leap 15.3
- openSUSE Tumbleweed
- Rocky Linux 8
- Debian 10 Buster
- Debian 11 Bullseye
- Debian 12 Bookworm

which will give you packages which will work better then building from source yourself.

Note: If you have a close look at our `docker` folder you may see more folders and files but they are meant for testing or other purposes. Please ignore them, this page is always aligned and up to date.

To build the packages you to need to execute the following in the root folder of the cloned repository:

- Fedora 37: `./docker/rpms/build-and-install-rpms.sh fc37 docker/rpms/Fedora_37/Fedora37.dockerfile`
- CentOS 8: `./docker/rpms/build-and-install-rpms.sh el8 docker/rpms/CentOS_8/CentOS8.dockerfile`
- Debian 10: `./docker/debs/build-and-install-debs.sh deb10 docker/debs/Debian_10/Debian10.dockerfile`
- Debian 11: `./docker/debs/build-and-install-debs.sh deb11 docker/debs/Debian_11/Debian11.dockerfile`
- Debian 12: `./docker/debs/build-and-install-debs.sh deb12 docker/debs/Debian_12/Debian12.dockerfile`

After executing the scripts you should have one folder owned by `root` which was created during the build. It is either called `rpm-build` or `deb-build`. In these directories you should find the built packages. They are obviously unsigned and thus will generate warnings in relation to that fact.

2.3.2 Packages from source

For some platforms it's also possible to build packages directly from the source tree.

2.4 RPM

```
$ make rpms
... (lots of output) ...
Wrote: /path/to/cobbler/rpm-build/cobbler-3.0.0-1.fc20.src.rpm
Wrote: /path/to/cobbler/rpm-build/cobbler-3.0.0-1.fc20.noarch.rpm
Wrote: /path/to/cobbler/rpm-build/koan-3.0.0-1.fc20.noarch.rpm
Wrote: /path/to/cobbler/rpm-build/cobbler-web-3.0.0-1.fc20.noarch.rpm
```

As you can see, an RPM is output for each component of Cobbler, as well as a source RPM. This command was run on a system running Fedora 20, hence the fc20 in the RPM name - this will be different based on the distribution you're running.

2.5 DEB

To install Cobbler from source on a Debian-Based system, the following steps need to be made (tested on Debian Buster):

```
$ a2enmod proxy
$ a2enmod proxy_http
$ a2enmod rewrite

$ ln -s /srv/tftp /var/lib/tftpboot

$ systemctl restart apache2
$ make debs
```

Change all `/var/www/cobbler` in `/etc/apache2/conf.d/cobbler.conf` to `/usr/share/cobbler/webroot/` Init script:

- add Required-Stop line
- path needs to be `/usr/local/...` or fix the install location

2.6 Multi-Build

In the repository root there is a file called `docker-compose.yml`. If you have `docker-compose` installed you may use that to build packages for multiple distros on a single run. Just execute:

```
$ docker-compose up -d
```

After some time all containers expect one should be exited and you should see two new folders owned by `root` called `rpm-build` and `deb-build`. The leftover docker container is meant to be used for testing and playing, if you don't require this playground you may just clean up with:

```
$ docker-compose down
```

2.7 Source

Warning: Cobbler is not suited to be run outside of custom paths or being installed into a virtual environment. We are working hard to get there but it is not possible yet. If you try this and it works, please report to our GitHub repository and tell us what is left to support this conveniently.

2.7.1 Installation

The latest source code is available through git:

```
$ git clone https://github.com/cobbler/cobbler.git
$ cd cobbler
```

The release30 branch corresponds to the official release version for the 3.0.x series. The main branch is the development series.

When building from source, make sure you have the correct prerequisites. The Makefile uses a script called *distro_build_configs.sh* which sets the correct environment variables. Be sure to source it if you do not use the Makefile.

If all prerequisites are met, you can install Cobbler with the following command:

```
$ make install
```

This command will rewrite all configuration files on your system if you have an existing installation of Cobbler (whether it was installed via packages or from an older source tree).

To preserve your existing configuration files, snippets and automatic installation files, run this command:

```
$ make devinstall
```

To install Cobbler, finish the installation in any of both cases, use these steps:

1. Copy the systemd service file for *cobblerd* from `/etc/cobbler/cobblerd.service` to your systemd unit directory (`/etc/systemd/system`).
2. Install `python3-gunicorn` or the package responsible for your distro.
3. Take the systemd service file `cobblerd-gunicorn-service` and copy it into your unit directory.
4. Enable the proxy module of Apache2 (`a2enmod proxy` or something similar) if not enabled.
5. Restart Apache, `cobblerd` and `cobblerd-gunicorn`.

Note: Depending on your distributions FHS implementation you might need to adjust `ExecStart` from `/usr/bin/cobblerd` to `/usr/local/bin/cobblerd` in the `cobblerd.service` file.

Be advised that we don't copy the service file into the correct directory and that the path to the binary may be wrong depending on the location of the binary on your system. Do this manually and then you should be good to go. The same is valid for the Apache webserver config.

2.7.2 Uninstallation

1. Stop the `cobblerd` and `apache2` daemon
2. Remove Cobbler related files from the following paths:
    1. `/usr/lib/python3.x/site-packages/cobbler/`
    2. `/etc/apache2/`
    3. `/etc/cobbler/`
    4. `/etc/systemd/system/`
    5. `/usr/local/bin/`
    6. `/var/lib/cobbler/`
    7. `/var/log/cobbler/`
3. Do a `systemctl daemon-reload`.

2.8 Relocating your installation

Often folks don't have a very large `/var` partition, which is what Cobbler uses by default for mirroring install trees and the like.

You'll notice you can reconfigure the webdir location just by going into `/etc/cobbler/settings.yaml`, but it's not the best way to do things – especially as the packaging process does include some files and directories in the stock path. This means that, for upgrades and the like, you'll be breaking things somewhat. Rather than attempting to reconfigure Cobbler, your Apache configuration, your file permissions, and your SELinux rules, the recommended course of action is very simple.

1. Copy everything you have already in `/var/www/cobbler` to another location – for instance, `/opt/cobbler_data`
2. Now just create a symlink or bind mount at `/var/www/cobbler` that points to `/opt/cobbler_data`.

Done. You're up and running.

If you decided to access Cobbler's data store over NFS (not recommended) you really want to mount NFS on `/var/www/cobbler` with SELinux context passed in as a parameter to mount versus the symlink. You may also have to deal with problems related to rootsquash. However if you are making a mirror of a Cobbler server for a multi-site setup, mounting read only is OK there.

Also Note: `/var/lib/cobbler` can not live on NFS, as this interferes with locking ("flock") Cobbler does around it's storage files.

CHAPTER

THREE

COBBLER CLI

This page contains a description for commands which can be used from the CLI.

Note: We are currently developing a new CLI which is independent from the server. This document redirects you to the new documentation once the new CLI is ready.

3.1 General Principles

This should just be a brief overview. For the detailed explanations please refer to Readthedocs.

3.1.1 Distros, Profiles and Systems

Cobbler has a system of inheritance when it comes to managing the information you want to apply to a certain system.

3.1.2 Images

3.1.3 Repositories

3.1.4 Management Classes

3.1.5 Deleting configuration entries

If you want to remove a specific object, use the remove command with the name that was used to add it.

```
cobbler distro|profile|system|repo|image|menu remove --name=string
```

3.1.6 Editing

If you want to change a particular setting without doing an `add` again, use the `edit` command, using the same name you gave when you added the item. Anything supplied in the parameter list will overwrite the settings in the existing object, preserving settings not mentioned.

```
cobbler distro|profile|system|repo|image|menu edit --name=string [parameterlist]
```

3.1.7 Copying

Objects can also be copied:

```
cobbler distro|profile|system|repo|image|menu copy --name=oldname --newname=newname
```

3.1.8 Renaming

Objects can also be renamed, as long as other objects don't reference them.

```
cobbler distro|profile|system|repo|image|menu rename --name=oldname --newname=newname
```

3.2 CLI-Commands

Short Usage: `cobbler command [subcommand] [--arg1=value1] [--arg2=value2]`

Long Usage:

```
cobbler <distro|profile|system|repo|image|menu> ... [add|edit|copy|get-
↪autoinstall*|list|remove|rename|report] [options|--help]
cobbler
↪<aclsetup|buildiso|import|list|mkloaders|replicate|report|reposync|sync|validate-
↪autoinstalls|version|signature|hardlink> [options|--help]
```

3.2.1 Cobbler distro

This first step towards configuring what you want to install is to add a distribution record to Cobbler's configuration.

If there is an rsync mirror, DVD, NFS, or filesystem tree available that you would rather `import` instead, skip down to the documentation about the `import` command. It's really a lot easier to follow the import workflow – it only requires waiting for the mirror content to be copied and/or scanned. Imported mirrors also save time during install since they don't have to hit external install sources.

If you want to be explicit with distribution definition, however, here's how it works:

```
$ cobbler distro add --name=string --kernel=path --initrd=path [--kernel-
↪options=string] [--kernel-options-post=string] [--autoinstall-meta=string] [--
↪arch=i386|x86_64|ppc|ppc64|ppc64le|arm64] [--breed=redhat|debian|suse] [--template-
↪files=string]
```

Name	Description
arch	Sets the architecture for the PXE bootloader and also controls how Koan's `--replace-self` option will operate. The default setting (`standard`) will use `pxelinux`. `x86` and `x86_64` effectively do the same thing as standard. If you perform a `cobbler import`, the arch field will be auto-assigned.
autoinstall-meta	This is an advanced feature that sets automatic installation template variables to substitute, thus enabling those files to be treated as templates. Templates are powered using Cheetah and are described further along in this manpage as well as on the Cobbler Wiki. Example: `--autoinstall-meta="foo=bar baz=3 asdf"` See the section on "Kickstart Templating" for further information.
boot-files	TFTP Boot Files (Files copied into tftpboot beyond the kernel/initrd).
boot-loaders	Boot loader space delimited list (Network installation boot loaders). Valid options for list items are <<inherit>>, *grub*, *pxe*, *ipxe*.
breed	Controls how various physical and virtual parameters, including kernel arguments for automatic installation, are to be treated. Defaults to `redhat`, which is a suitable value for Fedora and CentOS as well. It means anything Red Hat based. There is limited experimental support for specifying "debian", "ubuntu", or "suse", which treats the automatic installation template file as a preseed/autoyast file format and changes the kernel arguments appropriately. Support for other types of distributions is possible in the future. See the Wiki for the latest information about support for these distributions. The file used for the answer file, regardless of the breed setting, is the value used for `--autoinstall` when creating the profile.
comment	Simple attach a description (Free form text) to your distro.
fetchable-files	Fetchable Files (Templates for tftp or wget/curl)
initrd	An absolute filesystem path to a initrd image.
kernel	An absolute filesystem path to a kernel image.
kernel-options	Sets kernel command-line arguments that the distro, and profiles/systems depending on it, will use. To remove a kernel argument that may be added by a higher Cobbler object (or in the global settings), you can prefix it with a `!`. Example: `--kernel-options="foo=bar baz=3 asdf !gulp"` This example passes the arguments `foo=bar baz=3 asdf` but will make sure `gulp` is not passed even if it was requested at a level higher up in the Cobbler configuration.
kernel-options-post	This is just like `--kernel-options`, though it governs kernel options on the installed OS, as opposed to kernel options fed to the installer. The syntax is exactly the same. This requires some special snippets to be found in your automatic installation template in order for this to work. Automatic installation templating is described later on in this document. Example: `noapic`
mgmt-classes	Management Classes (Management classes for external config management).
name	A string identifying the distribution, this should be something like `rhel6`.
os-version	Generally this field can be ignored. It is intended to alter some hardware setup for virtualized instances when provisioning guests with Koan. The valid options for `--os-version` vary depending on what is specified for `--breed`. If you specify an invalid option, the error message will contain a list of valid OS versions that can be used. If you don't know the OS version or it does not appear in the list, omitting this argument or using `other` should be perfectly fine. If you don't encounter any problems with virtualized instances, this option can be safely ignored.
owners	Users with small sites and a limited number of admins can probably ignore this option. All Cobbler objects (distros, profiles, systems, and repos) can take a –owners parameter to specify what Cobbler users can edit particular objects.This only applies to the Cobbler WebUI and XML-RPC interface, not the "cobbler" command line tool run from the shell. Furthermore, this is only respected by the `authorization.ownership` module which must be enabled in the settings. The value for `--owners` is a space separated list of users and groups as specified in `/etc/cobbler/users.conf`. For more information see the users.conf file as well as the Cobbler Wiki. In the default Cobbler configuration, this value is completely ignored, as is `users.conf`.
redhat-management-key	Management Classes (Management classes for external config management).

3.2.2 Cobbler profile

A profile associates a distribution to additional specialized options, such as a installation automation file. Profiles are the core unit of provisioning and at least one profile must exist for every distribution to be provisioned. A profile might represent, for instance, a web server or desktop configuration. In this way, profiles define a role to be performed.

```
$ cobbler profile add --name=string --distro=string [--autoinstall=path] [--kernel-
↪options=string] [--autoinstall-meta=string] [--name-servers=string] [--name-servers-
↪search=string] [--virt-file-size=gigabytes] [--virt-ram=megabytes] [--virt-
↪type=string] [--virt-cpus=integer] [--virt-path=string] [--virt-bridge=string] [--
↪server] [--parent=profile] [--filename=string]
```

Arguments are the same as listed for distributions, save for the removal of "arch" and "breed", and with the additions listed below:

Name	Description
autoinstall	Local filesystem path to a automatic installation file, the file must reside under `/var/lib/cobbler/templates`
autoinstall-meta	Automatic Installation Metadata (Ex: *dog=fang agent=86*).
boot-files	TFTP Boot Files (Files copied into tftpboot beyond the kernel/initrd).
boot-loaders	Boot loader space delimited list (Network installation boot loaders). Valid options for list items are <<inherit>>, *grub*, *pxe*, *ipxe*.
comment	Simple attach a description (Free form text) to your distro.
dhcp-tag	DHCP Tag (see description in system).
distro	The name of a previously defined Cobbler distribution. This value is required.
enable-ipxe	Enable iPXE? (Use iPXE instead of PXELINUX for advanced booting options)
enable-menu	Enable PXE Menu? (Show this profile in the PXE menu?)
fetchable-files	Fetchable Files (Templates for tftp or wget/curl)
filename	This parameter can be used to select the bootloader for network boot. If specified, this must be a path relative to the TFTP servers root directory. (e.g. grub/grubx64.efi) For most use cases the default bootloader is correct and this can be omitted
menu	This is a way of organizing profiles and images in an automatically generated boot menu for *grub*, *pxe* and *ipxe* boot loaders. Menu created with `cobbler menu add` command.
name	A descriptive name. This could be something like `rhel5webservers` or `f9desktops`.
name-servers	If your nameservers are not provided by DHCP, you can specify a space separated list of addresses here to configure each of the installed nodes to use them (provided the automatic installation files used are installed on a per-system basis). Users with DHCP setups should not need to use this option. This is available to set in profiles to avoid having to set it repeatedly for each system record.

continues on next page

Table 1 – continued from previous page

Name	Description
name-servers-search	You can specify a space separated list of domain names to configure each of the installed nodes to use them as domain search path. This is available to set in profiles to avoid having to set it repeatedly for each system record.
next-server	To override the Next server.
owners	Users with small sites and a limited number of admins can probably ignore this option. All objects (distros, profiles, systems, and repos) can take a –owners parameter to specify what Cobbler users can edit particular objects.This only applies to the Cobbler WebUI and XML-RPC interface, not the "cobbler" command line tool run from the shell. Furthermore, this is only respected by the `authorization.ownership` module which must be enabled in the settings. The value for `--owners` is a space separated list of users and groups as specified in `/etc/cobbler/users.conf`. For more information see the users.conf file as well as the Cobbler Wiki. In the default Cobbler configuration, this value is completely ignored, as is `users.conf`.
parent	This is an advanced feature. Profiles may inherit from other profiles in lieu of specifying `--distro`. Inherited profiles will override any settings specified in their parent, with the exception of `--autoinstall-meta` (templating) and `--kernel-options` (kernel options), which will be blended together. Example: If profile A has `--kernel-options="x=7 y=2"`, B inherits from A, and B has `--kernel-options="x=9 z=2"`, the actual kernel options that will be used for B are `x=9 y=2 z=2`. Example: If profile B has `--virt-ram=256` and A has `--virt-ram=512`, profile B will use the value 256. Example: If profile A has a `--virt-file-size=5` and B does not specify a size, B will use the value from A.
proxy	Proxy URL.
redhat- management-key	Management Classes (Management classes for external config management).
repos	This is a space delimited list of all the repos (created with `cobbler repo add` and updated with `cobbler reposync`)that this profile can make use of during automated installation. For example, an example might be `--repos="fc6i386updates fc6i386extras"` if the profile wants to access these two mirrors that are already mirrored on the Cobbler server. Repo management is described in greater depth later in the manpage.

continues on next page

Table 1 – continued from previous page

Name	Description
server	This parameter should be useful only in select circumstances. If machines are on a subnet that cannot access the Cobbler server using the name/IP as configured in the Cobbler settings file, use this parameter to override that servername. See also `--dhcp-tag` for configuring the next server and DHCP information of the system if you are also using Cobbler to help manage your DHCP configuration.
template-files	This feature allows Cobbler to be used as a configuration management system. The argument is a space delimited string of `key=value` pairs. Each key is the path to a template file, each value is the path to install the file on the system. This is described in further detail on the Cobbler Wiki and is implemented using special code in the post install. Koan also can retrieve these files from a Cobbler server on demand, effectively allowing Cobbler to function as a lightweight templated configuration management system.
virt-auto-boot	(Virt-only) Virt Auto Boot (Auto boot this VM?).
virt-bridge	(Virt-only) This specifies the default bridge to use for all systems defined under this profile. If not specified, it will assume the default value in the Cobbler settings file, which as shipped in the RPM is `virbr0`. If not using NAT, this is most likely not correct. You may want to override this setting in the system object. Bridge settings are important as they define how outside networking will reach the guest. For more information on bridge setup, see the Cobbler Wiki, where there is a section describing Koan usage.
virt-cpus	(Virt-only) How many virtual CPUs should Koan give the virtual machine? The default is 1. This is an integer.
virt-disk-driver	(Virt-only) Virt Disk Driver Type (The on-disk format for the virtualization disk). Valid options are <<inherit>>, *raw*, *qcow2*, *qed*, *vdi*, *vmdk*
virt-file-size	(Virt-only) How large the disk image should be in Gigabytes. The default is 5. This can be a comma separated list (ex: `5,6,7`) to allow for multiple disks of different sizes depending on what is given to `--virt-path`. This should be input as a integer or decimal value without units.
virt-path	(Virt-only) Where to store the virtual image on the host system. Except for advanced cases, this parameter can usually be omitted. For disk images, the value is usually an absolute path to an existing directory with an optional filename component. There is support for specifying partitions `/dev/sda4` or volume groups `VolGroup00`, etc. For multiple disks, separate the values with commas such as `VolGroup00,VolGroup00` or `/dev/sda4, /dev/sda5`. Both those examples would create two disks for the VM.
virt-ram	(Virt-only) How many megabytes of RAM to consume. The default is 512 MB. This should be input as an integer without units.

continues on next page

Table 1 – continued from previous page

Name	Description
virt-type	(Virt-only) Koan can install images using either Xen paravirt (`xenpv`) or QEMU/KVM (`qemu/kvm`). Choose one or the other strings to specify, or values will default to attempting to find a compatible installation type on the client system("auto"). See the "Koan" manpage for more documentation. The default `--virt-type` can be configured in the Cobbler settings file such that this parameter does not have to be provided. Other virtualization types are supported, for information on those options (such as VMware), see the Cobbler Wiki.

3.2.3 Cobbler system

System records map a piece of hardware (or a virtual machine) with the Cobbler profile to be assigned to run on it. This may be thought of as choosing a role for a specific system.

Note that if provisioning via Koan and PXE menus alone, it is not required to create system records in Cobbler, though they are useful when system specific customizations are required. One such customization would be defining the MAC address. If there is a specific role intended for a given machine, system records should be created for it.

System commands have a wider variety of control offered over network details. In order to use these to the fullest possible extent, the automatic installation template used by Cobbler must contain certain automatic installation snippets (sections of code specifically written for Cobbler to make these values become reality). Compare your automatic installation templates with the stock ones in /var/lib/cobbler/templates if you have upgraded, to make sure you can take advantage of all options to their fullest potential. If you are a new Cobbler user, base your automatic installation templates off of these templates.

Read more about networking setup at: https://cobbler.readthedocs.io/en/release28/4_advanced/advanced%20networking.html

Example:

```
$ cobbler system add --name=string --profile=string [--mac=macaddress] [--ip-
↪address=ipaddress] [--hostname=hostname] [--kernel-options=string] [--autoinstall-
↪meta=string] [--autoinstall=path] [--netboot-enabled=Y/N] [--server=string] [--
↪gateway=string] [--dns-name=string] [--static-routes=string] [--power-
↪address=string] [--power-type=string] [--power-user=string] [--power-pass=string] [-
↪-power-id=string]
```

Adds a Cobbler System to the configuration. Arguments are specified as per "profile add" with the following changes:

Name	Description
autoinstall	While it is recommended that the `--autoinstall` parameter is only used within for the "profile add" command, there are limited scenarios when an install base switching to Cobbler may have legacy automatic installation files created on aper-system basis (one automatic installation file for each system, nothing shared) and may not want to immediately make use of the Cobbler templating system. This allows specifying a automatic installation file for use on a per-system basis. Creation of a parent profile is still required. If the automatic installation file is a filesystem location, it will still be treated as a Cobbler template.
autoinstall-meta	Automatic Installation Metadata (Ex: *dog=fang agent=86*).
boot-files	TFTP Boot Files (Files copied into tftpboot beyond the kernel/initrd).
boot-loaders	Boot loader space delimited list (Network installation boot loaders). Valid options for list items are <<inherit>>, *grub*, *pxe*, *ipxe*.
comment	Simple attach a description (Free form text) to your distro.
dhcp-tag	If you are setting up a PXE environment with multiple subnets/gateways, and are using Cobbler to manage a DHCP configuration, you will probably want to use this option. If not, it can be ignored. By default, the dhcp tag for all systems is "default" and means that in the DHCP template files the systems will expand out where $insert_cobbler_systems_definitions is found in the DHCP template. However, you may want certain systems to expand out in other places in the DHCP config file. Setting `--dhcp-tag=subnet2` for instance, will cause that system to expand out where $insert_cobbler_system_definitions_subnet2 is found, allowing you to insert directives to specify different subnets (or other parameters) before the DHCP configuration entries for those particular systems. This is described further on the Cobbler Wiki.
dns-name	If using the DNS management feature (see advanced section – Cobbler supports auto-setup of BIND and dnsmasq), use this to define a hostname for the system to receive from DNS. Example: `--dns-name=mycomputer.example.com` This is a per-interface parameter. If you have multiple interfaces, it may be different for each interface, for example, assume a DMZ / dual-homed setup.
enable-ipxe	Enable iPXE? (Use iPXE instead of PXELINUX for advanced booting options)
fetchable-files	Fetchable Files (Templates for tftp or wget/curl)
filename	This parameter can be used to select the bootloader for network boot. If specified, this must be a path relative to the TFTP servers root directory. (e.g. grub/grubx64.efi) For most use cases the default bootloader is correct and this can be omitted

continues on next page

Table 2 – continued from previous page

Name	Description
gateway and netmask	If you are using static IP configurations and the interface is flagged `--static=1`, these will be applied. Netmask is a per-interface parameter. Because of the way gateway is stored on the installed OS, gateway is a global parameter. You may use `--static-routes` for per-interface customizations if required.
hostname	This field corresponds to the hostname set in a systems `/etc/sysconfig/network` file. This has no bearing on DNS, even when manage_dns is enabled. Use `--dns-name` instead for that feature. This parameter is assigned once per system, it is not a per-interface setting.
interface	By default flags like `--ip`, `--mac`, `--dhcp-tag`, `--dns-name`, `--netmask`, `--virt-bridge`, and `--static-routes` operate on the first network interface defined for a system (eth0). However, Cobbler supports an arbitrary number of interfaces. Using `--interface=eth1` for instance, will allow creating and editing of a second interface. Interface naming notes: Additional interfaces can be specified (for example: eth1, or any name you like, as long as it does not conflict with any reserved names such as kernel module names) for use with the edit command. Defining VLANs this way is also supported, of you want to add VLAN 5 on interface eth0, simply name your interface eth0.5. Example: cobbler system edit –name=foo –ip-address=192.168.1.50 –mac=AA:BB:CC:DD:EE:A0 cobbler system edit –name=foo –interface=eth0 –ip-address=10.1.1.51 –mac=AA:BB:CC:DD:EE:A1 cobbler system report foo Interfaces can be deleted using the –delete-interface option. Example: cobbler system edit –name=foo –interface=eth2 –delete-interface

continues on next page

Table 2 – continued from previous page

Name	Description

interface-type, interface-master, bonding-opts, bridge-opts	One of the other advanced networking features supported by Cobbler is NIC bonding, bridging and BMC. You can use this to bond multiple physical network interfaces to one single logical interface to reduce single points of failure in your network, to create bridged interfaces for things like tunnels and virtual machine networks, or to manage BMC interface by DHCP. Supported values for the `--interface-type` parameter are "bond", "bond_slave", "bridge", "bridge_slave","bonded_bridge_slave" and "bmc". If one of the "_slave" options is specified, you also need to define the master-interface for this bond using `--interface-master=INTERFACE`. Bonding and bridge options for the master-interface may be specified using `--bonding-opts="foo=1 bar=2"` or `--bridge-opts="foo=1 bar=2"`.
	Example:
	`cobbler system edit --name=foo \ --interface=eth0 \ --mac=AA:BB:CC:DD:EE:00 \ --interface-type=bond_slave \ --interface-master=bond0`
	`cobbler system edit --name=foo \ --interface=eth1 \ --mac=AA:BB:CC:DD:EE:01 \ --interface-type=bond_slave \ --interface-master=bond0`
	`cobbler system edit --name=foo \ --interface=bond0 \ --interface-type=bond \ --bonding-opts="mode=active-backup miimon=100" \ --ip-address=192.168.0.63 \ --netmask=255.255.255.0 \ --gateway=192.168.0.1 \ --static=1`
	More information about networking setup is available at *Advanced Networking*
	To review what networking configuration you have for any object, run "cobbler system report" at any time:
	Example:
	`cobbler system report --name=foo`

continues on next page

Table 2 – continued from previous page

Name	Description
if-gateway	If you are using static IP configurations and have multiple interfaces, use this to define different gateway for each interface. This is a per-interface setting.
ip-address, ipv6-address	If Cobbler is configured to generate a DHCP configuration (see advanced section), use this setting to define a specific IP for this system in DHCP. Leaving off this parameter will result in no DHCP management for this particular system. Example: `--ip-address=192.168.1.50` If DHCP management is disabled and the interface is labelled `--static=1`, this setting will be used for static IP configuration. Special feature: To control the default PXE behavior for an entire subnet, this field can also be passed in using CIDR notation. If `--ip` is CIDR, do not specify any other arguments other than `--name` and `--profile`. When using the CIDR notation trick, don't specify any arguments other than `--name` and `--profile`, as they won't be used.
kernel-options	Sets kernel command-line arguments that the distro, and profiles/systems depending on it, will use. To remove a kernel argument that may be added by a higher Cobbler object (or in the global settings), you can prefix it with a `!`. Example: `--kernel-options="foo=bar baz=3 asdf !gulp"` This example passes the arguments `foo=bar baz=3 asdf` but will make sure `gulp` is not passed even if it was requested at a level higher up in the Cobbler configuration.
kernel-options-post	This is just like `--kernel-options`, though it governs kernel options on the installed OS, as opposed to kernel options fed to the installer. The syntax is exactly the same. This requires some special snippets to be found in your automatic installation template in order for this to work. Automatic installation templating is described later on in this document. Example: `noapic`

continues on next page

Table 2 – continued from previous page

Name	Description
mac, mac-address	Specifying a mac address via `--mac` allows the system object to boot directly to a specific profile via PXE, bypassing Cobbler's PXE menu. If the name of the Cobbler system already looks like a mac address, this is inferred from the system name and does not need to be specified. MAC addresses have the format AA:BB:CC:DD:EE:FF. It's highly recommended to register your MAC addresses in Cobbler if you're using static addressing with multiple interfaces, or if you are using any of the advanced networking features like bonding, bridges or VLANs. Cobbler does contain a feature (enabled in /etc/cobbler/settings.yaml) that can automatically add new system records when it finds profiles being provisioned on hardware it has seen before. This may help if you do not have a report of all the MAC addresses in your datacenter/lab configuration.
mgmt-classes	Management Classes (Management classes for external config management).
mgmt-parameters	Management Parameters which will be handed to your management application. (Must be valid YAML dictionary)
name	The system name works like the name option for other commands. If the name looks like a MAC address or an IP, the name will implicitly be used for either `--mac` or `--ip` of the first interface, respectively. However, it's usually better to give a descriptive name – don't rely on this behavior. A system created with name "default" has special semantics. If a default system object exists, it sets all undefined systems to PXE to a specific profile. Without a "default" system name created, PXE will fall through to local boot for unconfigured systems. When using "default" name, don't specify any other arguments than `--profile`, as they won't be used.
name-servers	If your nameservers are not provided by DHCP, you can specify a space separated list of addresses here to configure each of the installed nodes to use them (provided the automatic installation files used are installed on a per-system basis). Users with DHCP setups should not need to use this option. This is available to set in profiles to avoid having to set it repeatedly for each system record.
name-servers-search	You can specify a space separated list of domain names to configure each of the installed nodes to use them as domain search path. This is available to set in profiles to avoid having to set it repeatedly for each system record.

continues on next page

Table 2 – continued from previous page

Name	Description
netboot-enabled	If set false, the system will be provisionable through Koan but not through standard PXE. This will allow the system to fall back to default PXE boot behavior without deleting the Cobbler system object. The default value allows PXE. Cobbler contains a PXE boot loop prevention feature (pxe_just_once, can be enabled in /etc/cobbler/settings.yaml) that can automatically trip off this value after a system gets done installing. This can prevent installs from appearing in an endless loop when the system is set to PXE first in the BIOS order.
next-server	To override the Next server.
owners	Users with small sites and a limited number of admins can probably ignore this option. All objects (distros, profiles, systems, and repos) can take a –owners parameter to specify what Cobbler users can edit particular objects.This only applies to the Cobbler WebUI and XML-RPC interface, not the "cobbler" command line tool run from the shell. Furthermore, this is only respected by the `authorization.ownership` module which must be enabled in the settings. The value for `--owners` is a space separated list of users and groups as specified in `/etc/cobbler/users.conf`. For more information see the users.conf file as well as the Cobbler Wiki. In the default Cobbler configuration, this value is completely ignored, as is `users.conf`.
power-address, power-type, power-user, power-pass, power-id, power-options, power-identity-file	Cobbler contains features that enable integration with power management for easier installation, reinstallation, and management of machines in a datacenter environment. These parameters are described online at *power-management*. If you have a power-managed datacenter/lab setup, usage of these features may be something you are interested in.
profile	The name of Cobbler profile the system will inherite its properties.
proxy	Proxy URL.
redhat- management-key	Management Classes (Management classes for external config management).
repos-enabled	If set true, Koan can reconfigure repositories after installation. This is described further on the Cobbler Wiki,https://github.com/cobbler/cobbler/wiki/Manage-yum-repos.
static	Indicates that this interface is statically configured. Many fields (such as gateway/netmask) will not be used unless this field is enabled. This is a per-interface setting.
static-routes	This is a space delimited list of ip/mask:gateway routing information in that format. Most systems will not need this information. This is a per-interface setting.
virt-auto-boot	(Virt-only) Virt Auto Boot (Auto boot this VM?).

continues on next page

Table 2 – continued from previous page

Name	Description
virt-bridge	(Virt-only) This specifies the default bridge to use for all systems defined under this profile. If not specified, it will assume the default value in the Cobbler settings file, which as shipped in the RPM is `virbr0`. If no using NAT, this is most likely not correct. You may want to override this setting in the system object. Bridge settings are important as they define how outside networking will reach the guest. For more information on bridge setup, see the Cobbler Wiki, where there is a section describing Koan usage.
virt-cpus	(Virt-only) How many virtual CPUs should Koan give the virtual machine? The default is 1. This is an integer.
virt-disk-driver	(Virt-only) Virt Disk Driver Type (The on-disk format for the virtualization disk). Valid options are <<inherit>>, *raw*, *qcow2*, *qed*, *vdi*, *vmdk*
virt-file-size	(Virt-only) How large the disk image should be in Gigabytes. The default is 5. This can be a comma separated list (ex: `5,6,7`) to allow for multiple disks of different sizes depending on what is given to `--virt-path`. This should be input as a integer or decimal value without units.
virt-path	(Virt-only) Where to store the virtual image on the host system. Except for advanced cases, this parameter can usually be omitted. For disk images, the value is usually an absolute path to an existing directory with an optional filename component. There is support for specifying partitions `/dev/sda4` or volume groups `VolGroup00`, etc. For multiple disks, separate the values with commas such as `VolGroup00,VolGroup00` or `/dev/sda4, /dev/sda5`. Both those examples would create two disks for the VM.
virt-ram	(Virt-only) How many megabytes of RAM to consume. The default is 512 MB. This should be input as an integer without units.
virt-type	(Virt-only) Koan can install images using either Xen paravirt (`xenpv`) or QEMU/KVM (`qemu/kvm`). Choose one or the other strings to specify, or values will default to attempting to find a compatible installation type on the client system("auto"). See the "Koan" manpage for more documentation. The default `--virt-type` can be configured in the Cobbler settings file such that this parameter does not have to be provided. Other virtualization types are supported, for information on those options (such as VMware), see the Cobbler Wiki.

3.2.4 Cobbler repo

Repository mirroring allows Cobbler to mirror not only install trees ("cobbler import" does this for you) but also optional packages, 3rd party content, and even updates. Mirroring all of this content locally on your network will result in faster, more up-to-date installations and faster updates. If you are only provisioning a home setup, this will probably be overkill, though it can be very useful for larger setups (labs, datacenters, etc).

```
$ cobbler repo add --mirror=url --name=string [--rpmlist=list] [--creatrepo-
↪flags=string] [--keep-updated=Y/N] [--priority=number] [--arch=string] [--mirror-
↪locally=Y/N] [--breed=yum|rsync|rhn] [--mirror_type=baseurl|mirrorlist|metalink]
```

Name	Description
apt-components	Apt Components (apt only) (ex: main restricted universe)
apt-dists	Apt Dist Names (apt only) (ex: precise precise-updates)
arch	Specifies what architecture the repository should use. By default the current system arch (of the server) is used,which may not be desirable. Using this to override the default arch allows mirroring of source repositories(using `--arch=src`).
breed	Ordinarily Cobbler's repo system will understand what you mean without supplying this parameter, though you can set it explicitly if needed.
comment	Simple attach a description (Free form text) to your distro.
createrepo-flags	Specifies optional flags to feed into the createrepo tool, which is called when `cobbler reposync` is run for the given repository. The defaults are `-c cache`.
keep-updated	Specifies that the named repository should not be updated during a normal "cobbler reposync". The repo may still be updated by name. The repo should be synced at least once before disabling this feature. See "cobbler reposync" below.
mirror	The address of the yum mirror. This can be an `rsync://`-URL, an ssh location, or a `http://` or `ftp://` mirror location. Filesystem paths also work. The mirror address should specify an exact repository to mirror – just one architecture and just one distribution. If you have a separate repo to mirror for a different arch, add that repo separately. Here's an example of what looks like a good URL: • `rsync://yourmirror.example.com/fedora-linux-core/updates/6/i386` (for rsync protocol) • `http://mirrors.kernel.org/fedora/extras/6/i386/` (for http) • `user@yourmirror.example.com/fedora-linux-core/updates/6/i386` (for SSH) Experimental support is also provided for mirroring RHN content when you need a fast local mirror. The mirror syntax for this is `--mirror=rhn://channel-name` and you must have entitlements for this to work. This requires the Cobbler server to be installed on RHEL 5 or later. You will also need a version of `yum-utils` equal or greater to 1.0.4.
mirror-locally	When set to `N`, specifies that this yum repo is to be referenced directly via automatic installation files and not mirrored locally on the Cobbler server. Only `http://` and `ftp://` mirror urls are supported when using `--mirror-locally=N`, you cannot use filesystem URLs.
name	This name is used as the save location for the mirror. If the mirror represented, say, Fedora Core 6 i386 updates, a good name would be `fc6i386updates`. Again, be specific. This name corresponds with values given to the `--repos` parameter of `cobbler profile add`. If a profile has a `--repos`-value that matches the name given here, that repo can be automatically set up during provisioning (when supported) and installed systems will also use the boot server as a mirror (unless `yum_post_install_mirror` is disabled in the settings file). By default the provisioning server will act as a mirror to systems it installs, which may not be desirable for laptop configurations, etc. Distros that can make use of yum repositories during automatic installation include FC6 and later, RHEL 5 and later, and derivative distributions. See the documentation on `cobbler profile add` for more information.
owners | Users with small sites and a limited number of admins can probably ignore this option. All objects (distros, profiles, systems, and repos) can take a –owners parameter to specify what Cobbler users can edit particular objects.This only applies to the Cobbler WebUI and XML-RPC interface, not the "cobbler" command line tool run from the shell. Furthermore, this is only respected by the `authorization.ownership` module which must be enabled in the settings. The value for `--owners` is a space separated list of users and groups as specified in `/etc/cobbler/users.conf`. For more information see the users.conf file as well as the Cobbler	

```
$ cobbler repo autoadd
```

Add enabled yum repositories from `dnf repolist --enabled` list. The repository names are generated using the <repo id>-<releasever>-<arch> pattern (ex: fedora-32-x86_64). Existing repositories with such names are not overwritten.

3.2.5 Cobbler image

The primary and recommended use of Cobbler is to deploy systems by building them like from the OS manufacturer's distribution, e.g Redhat kickstart. This method is generally easier to work with and provides an infrastructure which is not only more sustainable but also much more flexible across varieties of hardware.

But Cobbler can also help with image-based booting, physically and virtually. Some manual use of other commands beyond what is typically required of Cobbler may be needed to prepare images for use with this feature and the usage of these commands varies substantially depending on the type of image.

For now we just have 1 example of using the "memdisk" image type:

Example:

```
$ cobbler image
```

memdisk - Oracle / Sun Maintenance CD

The 'memdisk' image type can be used to PXE boot Oracle / Sun maintenance CDs. Their manual gives details on how to copy the image from a CD to a PXE server. The procedure is even easier with Cobbler since the system takes care of most of it for you.

Take your ISO for the boot CD and mount it as a loopback mount somewhere on your Cobbler server then copy the `boot.img` file into your tftpboot directory. Then add an image of type `memdisk` which uses it. Right now the following shell command will fail due to a known bug but the web interface can be used instead to add the image.

```
> cobbler image add --name=MyName --image-type=memdisk --file=/tftpboot/oracle/SF2250/
↪boot.img
> usage: cobbler [options]
>
> cobbler: error: option --image-type: invalid choice: 'memdisk' (choose from 'iso',
↪'direct', 'virt-image')
```

Now just boot your machine from the network and select the image "MyName".

Memtest

If installed Cobbler will put an entry into all of your PXE menus allowing you to run memtest on physical systems without making changes in Cobbler. This can be handy for some simple diagnostics.

Steps to get memtest to show up in your PXE menus:

```
$ zypper/dnf install memtest86+
$ cobbler image add --name=memtest86+ --file=/path/to/memtest86+ --image-type=direct
$ cobbler sync
```

Targeted Memtesting

However if you already have a Cobbler system record for the system you won't get the menu. To solve this:

```
cobbler image add --name=foo --file=/path/to/memtest86 --image-type=direct
cobbler system edit --name=bar --mac=AA:BB:CC:DD:EE:FF --image=foo --netboot-enabled=1
```

The system will boot to memtest until you put it back to its original profile.

Warning: When restoring the system back from memtest, make sure you turn its netboot flag **off** if you have it set to PXE first in the BIOS order unless you want to reinstall the system!

```
$ cobbler system edit --name=bar --profile=old_profile_name --netboot-enabled=0
```

If you do want to reinstall it after running memtest, use `--netboot-enabled=true`.

3.2.6 Cobbler menu

By default, Cobbler builds a single-level boot menu for profiles and images. To simplify navigation through a large number of OS boot items, you can create *menu* objects and place any number of submenus, profiles, and images there. The menu is hierarchical, to indicate the nesting of one submenu in another, you can use the *parent* property. If the *parent* property for a submenu, or the *menu* property for a profile or images are not set or have an empty value, then the corresponding element will be displayed in the top-level menu. If a submenu does not have descendants in the form of profiles or images, then such a submenu will not be displayed in the boot menu.

```
$ cobbler menu add --name=string [--display-name=string] [--parent=string]
```

Name	Description
display-name	This is a human-readable name to display in the boot menu.
name	This name can be used as a *–parent* for a submenu, or as a *–menu* for a profile or image.
parent	This value can be set to indicate the nesting of this submenu in another.

3.2.7 Cobbler aclsetup

Example:

```
$ cobbler aclsetup
```

3.2.8 Cobbler buildiso

This command may not behave like you expect it without installing additional dependencies and configuration. The in depth explanation can be found at *Building ISOs*.

Note: Systems refers to systems that are profile based. Systems with a parent image based systems will be skipped.

Name	Description
iso	Output ISO to this file. If the file exists it will be truncated to zero before.
profiles	Use these profiles only for information collection.
systems	(net-only) Use these systems only for information collection.
tempdir	Working directory for building the ISO. The default value is set in the settings file.
distro	Used to detect the architecture of the ISO you are building. Specifies also the used Kernel and Initrd.
standalone	(offline-only) Creates a standalone ISO with all required distribution files but without any added repositories.
airgapped	(offline-only) Implies –standalone but additionally includes repo files for disconnected system installations.
source	(offline-only) Used with –standalone or –airgapped to specify a source for the distribution files.
exclude-dns	(net-only) Prevents addition of name server addresses to the kernel boot options.
xorriso-opts	Extra options for xorriso.

Example: The following command builds a single ISO file for all profiles and systems present under the distro *test*.

```
$ cobbler buildiso --distro=test
```

3.2.9 Cobbler import

Note: When running Cobbler via systemd, you cannot mount the ISO to `/tmp` or a sub-folder of it because we are using the option *Private Temporary Directory*, to enhance the security of our application.

Example:

```
$ cobbler import
```

3.2.10 Cobbler list

This list all the names grouped by type. Identically to `cobbler report` there are subcommands for most of the other Cobbler commands. (Currently: distro, profile, system, repo, image)

```
$ cobbler list
```

3.2.11 Cobbler replicate

Cobbler can replicate configurations from a master Cobbler server. Each Cobbler server is still expected to have a locally relevant `/etc/cobbler/settings.yaml`, as this file is not synced.

This feature is intended for load-balancing, disaster-recovery, backup, or multiple geography support.

Cobbler can replicate data from a central server.

Objects that need to be replicated should be specified with a pattern, such as `--profiles="webservers* dbservers*"` or `--systems="*.example.org"`. All objects matched by the pattern, and all dependencies of those objects matched by the pattern (recursively) will be transferred from the remote server to the central server. This is to say if you intend to transfer `*.example.org` and the definition of the systems have not changed, but a profile above them has changed, the changes to that profile will also be transferred.

In the case where objects are more recent on the local server, those changes will not be overridden locally.

Common data locations will be rsync'ed from the master server unless `--omit-data` is specified.

To delete objects that are no longer present on the master server, use `--prune`.

Warning: This will delete all object types not present on the remote server from the local server, and is recursive. If you use prune, it is best to manage Cobbler centrally and not expect changes made on the slave servers to be preserved. It is not currently possible to just prune objects of a specific type.

Example:

```
$ cobbler replicate --master=cobbler.example.org [--distros=pattern] [--
↪profiles=pattern] [--systems=pattern] [--repos-pattern] [--images=pattern] [--
↪prune] [--omit-data]
```

3.2.12 Cobbler report

This lists all configuration which Cobbler can obtain from the saved data. There are also `report` subcommands for most of the other Cobbler commands (currently: distro, profile, system, repo, image, menu).

```
$ cobbler report
```

3.2.13 Cobbler reposync

Example:

```
$ cobbler reposync [--only=ONLY] [--tries=TRIES] [--no-fail]
```

Cobbler reposync is the command to use to update repos as configured with `cobbler repo add`. Mirroring can take a long time, and usage of cobbler reposync prior to usage is needed to ensure provisioned systems have the files they need to actually use the mirrored repositories. If you just add repos and never run `cobbler reposync`, the repos will never be mirrored. This is probably a command you would want to put on a crontab, though the frequency of that crontab and where the output goes is left up to the systems administrator.

For those familiar with dnf's reposync, cobbler's reposync is (in most uses) a wrapper around the `dnf reposync` command. Please use `cobbler reposync` to update cobbler mirrors, as dnf's reposync does not perform all required steps. Also cobbler adds support for rsync and SSH locations, where as dnf's reposync only supports what dnf supports (http/ftp).

If you ever want to update a certain repository you can run: `cobbler reposync --only="reponame1" ...`

When updating repos by name, a repo will be updated even if it is set to be not updated during a regular reposync operation (ex: `cobbler repo edit -name=reponame1 -keep-updated=0`). Note that if a cobbler import provides enough information to use the boot server as a yum mirror for core packages, cobbler can set up automatic installation files to use the cobbler server as a mirror instead of the outside world. If this feature is desirable, it can be turned on by `setting yum_post_install_mirror` to `True` in `/etc/cobbler/settings.yaml` (and running `cobbler sync`). You should not use this feature if machines are provisioned on a different VLAN/network than production, or if you are provisioning laptops that will want to acquire updates on multiple networks.

The flags `--tries=N` (for example, `--tries=3`) and `--no-fail` should likely be used when putting re-posync on a crontab. They ensure network glitches in one repo can be retried and also that a failure to synchronize one repo does not stop other repositories from being synchronized.

3.2.14 Cobbler sync

The sync command is very important, though very often unnecessary for most situations. It's primary purpose is to force a rewrite of all configuration files, distribution files in the TFTP root, and to restart managed services. So why is it unnecessary? Because in most common situations (after an object is edited, for example), Cobbler executes what is known as a "lite sync" which rewrites most critical files.

When is a full sync required? When you are using `manage_dhcpd` (Managing DHCP) with systems that use static leases. In that case, a full sync is required to rewrite the `dhcpd.conf` file and to restart the dhcpd service.

Cobbler sync is used to repair or rebuild the contents `/tftpboot` or `/var/www/cobbler` when something has changed behind the scenes. It brings the filesystem up to date with the configuration as understood by Cobbler.

Sync should be run whenever files in `/var/lib/cobbler` are manually edited (which is not recommended except for the settings file) or when making changes to automatic installation files. In practice, this should not happen often, though running sync too many times does not cause any adverse effects.

If using Cobbler to manage a DHCP and/or DNS server (see the advanced section of this manpage), sync does need to be run after systems are added to regenerate and reload the DHCP/DNS configurations. If you want to trigger the DHCP/DNS regeneration only and do not want a complete sync, you can use `cobbler sync --dhcp` or `cobbler sync --dns` or the combination of both.

`cobbler sync --systems` is used to only write specific systems (must exists in backend storage) to the TFTP folder. The expected pattern is a comma separated list of systems e.g. `sys1.internal,sys2.internal,sys3.internal`.

Note: Please note that at least once a full sync has to be run beforehand.

The sync process can also be kicked off from the web interface.

Example:

```
$ cobbler sync
$ cobbler sync [--systems=sys1.internal,sys2.internal,sys3.internal]
$ cobbler sync [--dns]
$ cobbler sync [--dhcp]
$ cobbler sync [--dns --dhcp]
```

3.2.15 Cobbler validate-autoinstalls

Example:

```
$ cobbler validate-autoinstalls
```

3.2.16 Cobbler version

Example:

```
$ cobbler version
```

3.2.17 Cobbler signature

Example:

```
$ cobbler signature
```

3.2.18 Cobbler hardlink

Example:

```
$ cobbler hardlink
```

3.2.19 Cobbler mkloaders

This command is used for generating UEFI bootable GRUB 2 bootloaders. This command has no options and is configured via the settings file of Cobbler. If available on the operating system Cobbler is running on, then this also generates bootloaders for different architectures then the one of the system.

Note: This command should be executed every time the bootloader modules are being updated, running it more frequently does not help, running it less frequently will cause the bootloader to be possibly vulnerable.

Example:

```
$ cobbler mkloaders
```

3.3 EXIT_STATUS

Cobbler's command line returns a zero for success and non-zero for failure.

3.4 Additional Help

We have a Gitter Channel and you also can ask questions as GitHub issues. The IRC Channel on Freenode (#cobbler) is not that active but sometimes there are people who can help you.

The way we would prefer are GitHub issues as they are easily searchable.

CHAPTER

FOUR

COBBLERD

Cobbler - a provisioning and update server

4.1 Preamble

We will refer to *cobblerd* here as "cobbler" because *cobblerd* is short for cobbler-daemon which is basically the server. The CLI will be referred to as Cobbler-CLI and Koan as Koan.

4.2 Description

Cobbler manages provisioning using a tiered concept of Distributions, Profiles, Systems, and (optionally) Images and Repositories.

Distributions contain information about what kernel and initrd are used, plus metadata (required kernel parameters, etc).

Profiles associate a Distribution with an automated installation template file and optionally customize the metadata further.

Systems associate a MAC, IP, and other networking details with a profile and optionally customize the metadata further.

Repositories contain yum mirror information. Using cobbler to mirror repositories is an optional feature, though provisioning and package management share a lot in common.

Images are a catch-all concept for things that do not play nicely in the "distribution" category. Most users will not need these records initially and these are described later in the document.

The main advantage of cobbler is that it glues together many disjoint technologies and concepts and abstracts the user from the need to understand them. It allows the systems administrator to concentrate on what he needs to do, and not how it is done.

This manpage will focus on the cobbler command line tool for use in configuring cobbler. There is also mention of the Cobbler WebUI which is usable for day-to-day operation of Cobbler once installed/configured. Docs on the API and XML-RPC components are available online at https://cobbler.github.io or https://cobbler.readthedocs.io.

Most users will be interested in the Web UI and should set it up, though the command line is needed for initial configuration – in particular `cobbler check` and `cobbler import`, as well as the repo mirroring features. All of these are described later in the documentation.

4.3 Setup

After installing, run `cobbler check` to verify that cobbler's ecosystem is configured correctly. Cobbler check will direct you on how to modify it's config files using a text editor.

Any problems detected should be corrected, with the potential exception of DHCP related warnings where you will need to use your judgement as to whether they apply to your environment. Run `cobbler sync` after making any changes to the configuration files to ensure those changes are applied to the environment.

It is especially important that the server name field be accurate in `/etc/cobbler/settings.yaml`, without this field being correct, automatic installation trees will not be found, and automated installations will fail.

For PXE, if DHCP is to be run from the cobbler server, the DHCP configuration file should be changed as suggested by `cobbler check`. If DHCP is not run locally, the `next-server` field on the DHCP server should at minimum point to the cobbler server's IP and the filename should be set to `pxelinux.0`. Alternatively, cobbler can also generate your DHCP configuration file if you want to run DHCP locally – this is covered in a later section. If you don't already have a DHCP setup managed by some other tool, allowing cobbler to manage your DHCP environment will prove to be useful as it can manage DHCP reservations and other data. If you already have a DHCP setup, moving an existing setup to be managed from within cobbler is relatively painless – though usage of the DHCP management feature is entirely optional. If you are not interested in network booting via PXE and just want to use Koan to install virtual systems or replace existing ones, DHCP configuration can be totally ignored. Koan also has a live CD (see Koan's manpage) capability that can be used to simulate PXE environments.

4.4 Autoinstallation (AutoYaST/Kickstart)

For help in building kickstarts, try using the `system-config-kickstart` tool, or install a new system and look at the `/root/anaconda-ks.cfg` file left over from the installer. General kickstart questions can also be asked at kickstart-list@redhat.com. Cobbler ships some autoinstall templates in `/etc/cobbler` that may also be helpful.

For AutoYaST guides and help please refer to the opensuse project.

Also see the website or documentation for additional documentation, user contributed tips, and so on.

4.5 Options

-B –daemonize
If you pass no options this is the default one. The Cobbler-Server runs in the background.

-F –no-daemonize
The Cobbler-Server runs in the foreground.

-f –log-file
Choose a destination for the logfile (currently has no effect).

-l –log-level
Choose a loglevel for the application (currently has no effect).

–c –config
The location of the Cobbler configuration file.

--disable-automigration If given, do no execute automigration from older settings filles to the most recent.

CHAPTER

FIVE

COBBLER CONFIGURATION

5.1 Updates to the yaml-settings-file

5.1.1 Starting with 3.4.0

- TBD

5.1.2 Starting with 3.3.3

- `default_virt_file_size` is now a float as intended.
- We added the `proxies` key for first-level Uyuni & SUSE Manager support. It is optional, so you can ignore it if you don't run one of the two solutions or a derivative of it.

5.1.3 Starting with 3.3.2

- After community feedback we changed the default of the auto-migration to be disabled. It can be re-enabled via the already known methods `cobbler-settings`-Tool, the settings file key `auto_migrate_settings` and the Daemon flag. We have decided to not change the flag for existing installations.

5.1.4 Starting with 3.3.1

- There is a new setting `bootloaders_shim_location`. For details please refer to the appropriate section below.

5.1.5 Starting with 3.3.0

- The setting `enable_gpxe` was replaced with `enable_ipxe`.
- The `settings.d` directory (`/etc/cobbler/settings.d/`) was deprecated and will be removed in the future.
- There is a new CLI tool called `cobbler-settings` which can be used to validate and migrate settings files from differente versions and to modify keys in the current settings file. Have a look at the migration matrix in the next paragraph to see the supported migration paths. Furthermore the auto migration feature can be enabled or disabled.
- A new settings auto migration feature was implemented which automatically updates the settings when installing a new version. A backup of the old settings file will be created in the same folder beforehand.

5.1.6 Starting with 3.2.1

- We require the extension `.yaml` on our settings file to indicate the format of the file to editors and comply to standards of the YAML specification.
- We require the usage of booleans in the format of `True` and `False`. If you have old integer style booleans with `1` and `0` this is fine but you may should convert them as soon as possible. We may decide in a future version to enforce our new way in a stricter manner. Automatic conversion is only done on a best-effort/available-resources basis.
- We enforce the types of values to the keys. Additional unexpected keys will throw errors. If you have those used in Cobbler please report this in our issue tracker. We have decided to go this way to be able to rely on the existence of the values. This gives us the freedom to write fewer access checks to the settings without losing stability.

5.2 `settings.yaml`

5.2.1 auto_migrate_settings

If `True` Cobbler will auto migrate the settings file after upgrading from older versions. The current settings are backed up in the same folder before the upgrade.

default: `True`

5.2.2 allow_duplicate_hostnames

If `True`, Cobbler will allow insertions of system records that duplicate the `--dns-name` information of other system records. In general, this is undesirable and should be left False.

default: `False`

5.2.3 allow_duplicate_ips

If `True`, Cobbler will allow insertions of system records that duplicate the IP address information of other system records. In general, this is undesirable and should be left False.

default: `False`

5.2.4 allow_duplicate_macs

If `True`, Cobbler will allow insertions of system records that duplicate the mac address information of other system records. In general, this is undesirable.

default: `False`

5.2.5 allow_dynamic_settings

If `True`, Cobbler will allow settings to be changed dynamically without a restart of the `cobblerd` daemon. You can only change this variable by manually editing the settings file, and you MUST restart `cobblerd` after changing it.

default: `False`

5.2.6 always_write_dhcp_entries

Always write DHCP entries, regardless if netboot is enabled.

default: `False`

5.2.7 anamon_enabled

By default, installs are *not* set to send installation logs to the Cobbler server. With `anamon_enabled`, automatic installation templates may use the `pre_anamon` snippet to allow remote live monitoring of their installations from the Cobbler server. Installation logs will be stored under `/var/log/cobbler/anamon/`.

Note: This does allow an XML-RPC call to send logs to this directory, without authentication, so enable only if you are ok with this limitation.

---

default: `False`

5.2.8 auth_token_expiration

How long the authentication token is valid for, in seconds.

default: `3600`

5.2.9 authn_pam_service

If using `authentication.pam` under `modules.authentication.module`, this can be configured to change the PAM service authentication will be tested against.

default: `"login"`

5.2.10 autoinstall

If no autoinstall template is specified to profile add, use this template.

default: `default.ks`

5.2.11 autoinstall_scheme

This should contain the scheme over which the autoinstall-file is available.

This is setting does not setup your api for HTTPS, it just changes the way the url for your profiles and systems are generated.

Choices:

- `http`
- `https`

default: `http`

5.2.12 autoinstall_snippets_dir

This is a directory of files that Cobbler uses to make templating easier. See the Wiki for more information. Changing this directory should not be required.

default: `/var/lib/cobbler/snippets`

5.2.13 autoinstall_templates_dir

This is a directory of files that Cobbler uses to make templating easier. See the Wiki for more information. Changing this directory should not be required.

default: `/var/lib/cobbler/templates`

5.2.14 bind_chroot_path

Set to path of bind chroot to create bind-chroot compatible bind configuration files.

default: `""`

5.2.15 bind_master

Set to the ip address of the master bind DNS server for creating secondary bind configuration files.

default: `127.0.0.1`

5.2.16 bind_zonefile_path

Set to path where zonefiles of bind/named server are located.

default: `"@@bind_zonefiles@@"`

5.2.17 boot_loader_conf_template_dir

Location of templates used for boot loader config generation.

default: `"/etc/cobbler/boot_loader_conf"`

5.2.18 bootloaders_dir

A directory that "cobbler mkloaders" copies the built bootloaders into. "cobbler sync" searches for bootloaders in this directory.

default: `/var/lib/cobbler/loaders`

5.2.19 bootloaders_shim_folder

This Python Glob will be responsible for finding the installed shim folder. If you don't have shim installed this bootloader link will be skipped. If the Glob is not precise enough a message will be logged and the link will also be skipped.

default: Depending on your distro. See values below.

- (open)SUSE: `"/usr/share/efi/*/"`
- Debian/Ubuntu: `"/usr/lib/shim/"`
- CentOS/Fedora: `"/boot/efi/EFI/*/"`

5.2.20 bootloaders_shim_file

This is a Python Regex responsible for finding a single match in all files found by the Python Glob in `bootloaders_shim_folder`. If more or fewer files are found a message will be logged.

default: Depending on your distro. See values below.

- (open)SUSE: `"shim\.efi"`
- Debian/Ubuntu: `"shim*.efi.signed"`
- CentOS/Fedora: `"shim*.efi"`

secure_boot_grub_folder

This Python Glob is responsible for finding the installed secure boot bootloader folders. If the Glob is not precise enough a message will be logged and the link will also be skipped.

This glob is only used for grub formats that use the `use_secure_boot_grub` property.

default: Depending on your distro. See values below.

- (open)SUSE: `"/usr/share/efi/*/"`
- Debian/Ubuntu: `"/usr/lib/shim/"`
- CentOS/Fedora: `"/boot/efi/EFI/*/"`

secure_boot_grub_file

This is a Python Regex responsible to finding a single match for the secure boot grub bootloader in all files found by the `secure_boot_grub_folder` glob.

This regex is only used for grub formats that use the `use_secure_boot_grub` property.

default: Depending on your distro. See values below.

- (open)SUSE: `"grub\.efi"`
- Debian/Ubuntu: `"grub[a-zA-Z0-9]*\.efi"`
- CentOS/Fedora: `"grub\.efi"`

5.2.21 grub2_mod_dir

The directory where Cobbler looks for GRUB modules that are required for "cobbler mkloaders".

default: Depends on your distribution. See values below.

- (open)SUSE: `"/usr/share/grub2"`
- Debian/Ubuntu: `"/usr/lib/grub"`
- CentOS/Fedora: `"/usr/lib/grub"`

5.2.22 syslinux_dir

The directory where Cobbler looks for syslinux modules that are required for "cobbler mkloaders".

default: Depends on your distribution. See values below.

- (open)SUSE: `"/usr/share/syslinux"`
- Debian/Ubuntu: `"/usr/lib/syslinux/modules/bios/"`
- CentOS/Fedora: `"/usr/share/syslinux"`

5.2.23 bootloaders_modules

A list of all modules "cobbler mkloaders" includes when building grub loaders. Typically, a grub loader uses the modules for PXE or HTTP Boot.

default: Omited for readablity, please refer to the *settings.yaml* file in our GitHub repository.

5.2.24 bootloaders_formats

This is a mapping that has the following structure:

```
<loader name>:
   binary_name: filename
   extra_modules:
     - extra-module
   mod_dir: <different folder name then loader name>
   use_secure_boot_grub: True
```

The keys `extra_modules`, `mod_dir` and `use_secure_boot_grub` are optional. Under normal circumstances this setting does not need adjustments.

default: Omited for readablity, please refer to the *settings.yaml* file in our GitHub repository.

5.2.25 grubconfig_dir

The location where Cobbler searches for GRUB configuration files.

default: `/var/lib/cobbler/grub_config`

5.2.26 build_reporting_*

Email out a report when Cobbler finishes installing a system.

- enabled: Set to `true` to turn this feature on
- email: Which addresses to email
- ignorelist: A list of prefixes that defines mail topics that should not be sent.
- sender: Optional
- smtp_server: Used to specify another server for an MTA.
- subject: Use the default subject unless overridden.

defaults:

```
build_reporting_enabled: false
build_reporting_sender: ""
build_reporting_email: [ 'root@localhost' ]
build_reporting_smtp_server: "localhost"
build_reporting_subject: ""
build_reporting_ignorelist: [ "" ]
```

5.2.27 buildisodir

Used for caching the intermediate files for ISO-Building. You may want to use a SSD, a tmpfs or something which does not persist across reboots and can be easily thrown away but is also fast.

default: `/var/cache/cobbler/buildiso`

5.2.28 cheetah_import_whitelist

Cheetah-language autoinstall templates can import Python modules. while this is a useful feature, it is not safe to allow them to import anything they want. This whitelists which modules can be imported through Cheetah. Users can expand this as needed but should never allow modules such as subprocess or those that allow access to the filesystem as Cheetah templates are evaluated by `cobblerd` as code.

default:

- `random`
- `re`
- `time`
- `netaddr`

5.2.29 client_use_https

If set to `True`, all commands to the API (not directly to the XML-RPC server) will go over HTTPS instead of plain text. Be sure to change the `http_port` setting to the correct value for the web server.

default: `False`

5.2.30 client_use_localhost

If set to `True`, all commands will be forced to use the localhost address instead of using the above value which can force commands like `cobbler sync` to open a connection to a remote address if one is in the configuration and would traceback.

default: `False`

5.2.31 cobbler_master

Used for replicating the Cobbler instance.

default: `""`

5.2.32 convert_server_to_ip

Convert hostnames to IP addresses (where possible) so DNS isn't a requirement for various tasks to work correctly.

default: `False`

5.2.33 createrepo_flags

Default `createrepo_flags` to use for new repositories.

default: `"--cachedir=cache --update"`

5.2.34 default_name_*

Configure all installed systems to use these name servers by default unless defined differently in the profile. For DHCP configurations you probably do **not** want to supply this.

defaults:

```
default_name_servers: []
default_name_servers_search: []
```

5.2.35 default_ownership

if using the `authz_ownership` module, objects created without specifying an owner are assigned to this owner and/or group.

default:

- `admin`

5.2.36 default_password_crypted

Cobbler has various sample automatic installation templates stored in `/var/lib/cobbler/templates/`. This controls what install (root) password is set up for those systems that reference this variable. The factory default is "cobbler" and Cobbler check will warn if this is not changed. The simplest way to change the password is to run `openssl passwd -1` and put the output between the `""`.

default: `"$1$mF86/UHC$WvcIcX2t6crBz2onWxyac."`

5.2.37 default_template_type

The default template type to use in the absence of any other detected template. If you do not specify the template with `#template=<template_type>` on the first line of your templates/snippets, Cobbler will assume try to use the following template engine to parse the templates.

Note: Over time we will try to deprecate and remove Cheetah3 as a template engine. It is hard to package and there are fewer guides then with Jinja2. Making the templating independent of the engine is a task which complicates the code. Thus, please try to use Jinja2. We will try to support a seamless transition on a best-effort basis.

Current valid values are: `cheetah`, `jinja2`

default: `"cheetah"`

5.2.38 default_virt_bridge

For libvirt based installs in Koan, if no virt-bridge is specified, which bridge do we try? For defauit libvirt NAT nework use "virbr0". For bridged networks, use bridge device name (e.g. "br0"). This can be overridden on a per-profile basis or at the Koan command line though this saves typing to just set it here to the most common option.

default: `virbr0`

5.2.39 default_virt_disk_driver

The on-disk format for the virtualization disk.

default: `raw`

5.2.40 default_virt_file_size

Use this as the default disk size for virt guests (GB).

default: `5.0`

5.2.41 default_virt_ram

Use this as the default memory size for virt guests (MB).

default: `512`

5.2.42 default_virt_type

If Koan is invoked without `--virt-type` and no virt-type is set on the profile/system, what virtualization type should be assumed?

Current valid values are:

- `qemu`
- `kvm`
- `xenpv`
- `xenfv`
- `qemu`
- `vmware`
- `vmwarew`
- `openvz`
- `auto`

NOTE: this does not change what `virt_type` is chosen by import.

default: `kvm`

5.2.43 enable_ipxe

Enable iPXE booting? Enabling this option will cause Cobbler to copy the `undionly.kpxe` file to the TFTP root directory, and if a profile/system is configured to boot via iPXE it will chain load off `pxelinux.0`.

default: `False`

5.2.44 enable_menu

Controls whether Cobbler will add each new profile entry to the default PXE boot menu. This can be over-ridden on a per-profile basis when adding/editing profiles with `--enable-menu=False/True`. Users should ordinarily leave this setting enabled unless they are concerned with accidental reinstall from users who select an entry at the PXE boot menu. Adding a password to the boot menus templates may also be a good solution to prevent unwanted reinstallations.

default: `True`

5.2.45 http_port

Change this port if Apache is not running plain text on port 80. Most people can leave this alone.

default: `80`

5.2.46 iso_template_dir

Folder to search for the ISO templates. These will build the boot-menu of the built ISO.

default: `/etc/cobbler/iso`

5.2.47 jinja2_includedir

This is a directory of files that Cobbler uses to include files into Jinja2 templates. Per default this settings is commented out.

default: `/var/lib/cobbler/jinja2`

5.2.48 kernel_options

Kernel options that should be present in every Cobbler installation. Kernel options can also be applied at the distro/profile/system level.

default: `{}`

5.2.49 ldap_*

Configuration options if using the authn_ldap module. See the Wiki for details. This can be ignored if you are not using LDAP for WebUI/XML-RPC authentication.

defaults:

```
ldap_server: "ldap.example.com"
ldap_base_dn: "DC=example,DC=com"
ldap_port: 389
ldap_tls: true
ldap_anonymous_bind: true
ldap_search_bind_dn: ''
```

(continues on next page)

(continued from previous page)

```
ldap_search_passwd: ''
ldap_search_prefix: 'uid='
ldap_tls_cacertdir: ''
ldap_tls_cacertfile: ''
ldap_tls_certfile: ''
ldap_tls_keyfile: ''
ldap_tls_reqcert: 'hard'
ldap_tls_cipher_suite: ''
```

5.2.50 bind_manage_ipmi

When using the Bind9 DNS server, you can enable or disable if the BMCs should receive own DNS entries.

default: `False`

5.2.51 manage_dhcp

Set to `True` to enable Cobbler's DHCP management features. The choice of DHCP management engine is under `modules.dhcp.module`.

default: `True`

5.2.52 manage_dhcp_v4

Set to `true` to enable DHCP IPv6 address configuration generation. This currently only works with manager.isc DHCP module (isc dhcpd6 daemon). See `modules.dhcp.module` whether this isc module is chosen for dhcp generation.

default: `False`

5.2.53 manage_dhcp_v6

Set to `true` to enable DHCP IPv6 address configuration generation. This currently only works with manager.isc DHCP module (isc dhcpd6 daemon). See `modules.dhcp.module` whether this isc module is chosen for dhcp generation.

default: `False`

5.2.54 manage_dns

Set to `True` to enable Cobbler's DNS management features. The choice of DNS management engine is under the key `modules.dns.module`.

default: `False`

5.2.55 manage_*_zones

If using BIND (named) for DNS management in `modules.dns.module` and `manage_dns` is enabled (above), this lists which zones are managed. See *DNS management* for more information.

defaults:

```
manage_forward_zones: []
manage_reverse_zones: []
```

5.2.56 manage_genders

Whether or not to manage the genders file. For more information on that visit: github.com/chaos/genders

default: `False`

5.2.57 manage_rsync

Set to `True` to enable Cobbler's RSYNC management features.

default: `False`

5.2.58 manage_tftpd

Set to `True` to enable Cobbler's TFTP management features. The choice of TFTP management engine is under `modules.tftpd.module`.

default: `True`

5.2.59 mgmt_*

Cobbler has a feature that allows for integration with config management systems such as Puppet. The following parameters work in conjunction with `--mgmt-classes` and are described in further detail at *Configuration Management Integrations*.

```
mgmt_classes: []
mgmt_parameters:
    from_cobbler: true
```

5.2.60 next_server_v4

If using Cobbler with `manage_dhcp_v4`, put the IP address of the Cobbler server here so that PXE booting guests can find it. If you do not set this correctly, this will be manifested in TFTP open timeouts.

default: `127.0.0.1`

5.2.61 next_server_v6

If using Cobbler with `manage_dhcp_v6`, put the IP address of the Cobbler server here so that PXE booting guests can find it. If you do not set this correctly, this will be manifested in TFTP open timeouts.

default: `::1`

5.2.62 nsupdate_enabled

This enables or disables the replacement (or removal) of records in the DNS zone for systems created (or removed) by Cobbler.

Note: There are additional settings needed when enabling this. Due to the limited number of resources, this won't be done until 3.3.0. Thus please expect to run into troubles when enabling this setting.

---

default: `False`

5.2.63 nsupdate_log

The logfile to document what records are added or removed in the DNS zone for systems.

Note: The functionality this settings is related to is currently not tested due to tech-debt. Please use it with caution. This note will be removed once we were able to look deeper into this functionality of Cobbler.

- Required: No
- Default: `/var/log/cobbler/nsupdate.log`

5.2.64 nsupdate_tsig_algorithm

Note: The functionality this settings is related to is currently not tested due to tech-debt. Please use it with caution. This note will be removed once we were able to look deeper into this functionality of Cobbler.

- Required: No
- Default: `hmac-sha512`

5.2.65 nsupdate_tsig_key

Note: The functionality this settings is related to is currently not tested due to tech-debt. Please use it with caution. This note will be removed once we were able to look deeper into this functionality of Cobbler.

- Required: No
- Default: `[]`

5.2.66 power_management_default_type

Settings for power management features. These settings are optional. See *Power Management* to learn more.

Choices (refer to the fence-agents project for a complete list):

- apc_snmp
- bladecenter
- bullpap
- drac
- ether_wake
- ilo
- integrity
- ipmilan
- ipmilanplus
- lpar
- rsa
- virsh
- wti

default: `ipmilanplus`

5.2.67 proxies

This key is used by Uyuni (or one of its derivatives) for the Proxy scenario. More information can be found here

Cobbler only evaluates this if the key has a list of strings as value. An empty list means you don't have any proxies configured in your Uyuni setup.

default: `[]`

5.2.68 proxy_url_ext

External proxy which is used by the following commands: `reposync`, `signature update`

defaults:

```
http: http://192.168.1.1:8080
https: https://192.168.1.1:8443
```

5.2.69 proxy_url_int

Internal proxy which is used by systems to reach Cobbler for kickstarts.

e.g.: `proxy_url_int:  http://10.0.0.1:8080`

default: `""`

5.2.70 puppet_auto_setup

If enabled, this setting ensures that puppet is installed during machine provision, a client certificate is generated and a certificate signing request is made with the puppet master server.

default: `False`

5.2.71 puppet_parameterized_classes

Choose whether to enable puppet parameterized classes or not. Puppet versions prior to 2.6.5 do not support parameters.

default: `True`

5.2.72 puppet_server

Choose a `--server` argument when running puppetd/puppet agent during autoinstall.

default: `'puppet'`

5.2.73 puppet_version

Let Cobbler know that you're using a newer version of puppet. Choose version 3 to use: 'puppet agent'; version 2 uses status quo: 'puppetd'.

default: `2`

5.2.74 puppetca_path

Location of the puppet executable, used for revoking certificates.

default: `"/usr/bin/puppet"`

5.2.75 pxe_just_once

If this setting is set to `True`, Cobbler systems that pxe boot will request at the end of their installation to toggle the `--netboot-enabled` record in the Cobbler system record. This eliminates the potential for a PXE boot loop if the system is set to PXE first in it's BIOS order. Enable this if PXE is first in your BIOS boot order, otherwise leave this disabled. See the manpage for `--netboot-enabled`.

default: `True`

5.2.76 nopxe_with_triggers

If this setting is set to `True`, triggers will be executed when systems will request to toggle the `--netboot-enabled` record at the end of their installation.

default: `True`

5.2.77 redhat_management_permissive

If using `modules.authentication.module: "authentication.spacewalk"` in the settings to let Cobbler authenticate against Satellite/Spacewalk's auth system, by default it will not allow per user access into Cobbler Web and Cobbler XML-RPC. In order to permit this, the following setting must be enabled HOWEVER doing so will permit all Spacewalk/Satellite users of certain types to edit all of Cobbler's configuration. these roles are: `config_admin` and `org_admin`. Users should turn this on only if they want this behavior and do not have a cross-multi-org separation concern. If you have a single org in your satellite, it's probably safe to turn this on and then you can use CobblerWeb alongside a Satellite install.

default: `False`

5.2.78 redhat_management_server

This setting is only used by the code that supports using Uyuni/SUSE Manager/Spacewalk/Satellite authentication within Cobbler Web and Cobbler XML-RPC.

default: `"xmlrpc.rhn.redhat.com"`

5.2.79 uyuni_authentication_endpoint

This setting is only used by the code that supports using uyuni/SUSE Manager authentication within Cobbler Web and Cobbler XMLRPC. This is the endpoint for uyuni/SUSE Manager authentication: if empty redhat_management_server will be used.

e.g.: `uyuni_authentication_endpoint: http://localhost`

default: `""`

5.2.80 redhat_management_key

Specify the default Red Hat authorization key to use to register system. If left blank, no registration will be attempted. Similarly you can set the `--redhat-management-key` to blank on any system to keep it from trying to register.

default: `""`

5.2.81 register_new_installs

If set to `True`, allows `/usr/bin/cobbler-register` (part of the Koan package) to be used to remotely add new Cobbler system records to Cobbler. This effectively allows for registration of new hardware from system records.

default: `False`

5.2.82 remove_old_puppet_certs_automatically

When a puppet managed machine is reinstalled it is necessary to remove the puppet certificate from the puppet master server before a new certificate is signed (see above). Enabling the following feature will ensure that the certificate for the machine to be installed is removed from the puppet master server if the puppet master server is running on the same machine as Cobbler. This requires `puppet_auto_setup` above to be enabled

default: `False`

5.2.83 replicate_repo_rsync_options

Replication rsync options for repos set to override default value of `-avzH`.

default: `"-avzH"`

5.2.84 replicate_rsync_options

replication rsync options for distros, autoinstalls, snippets set to override default value of `-avzH`.

default: `"-avzH"`

5.2.85 reposync_flags

Flags to use for yum's reposync. If your version of yum reposync does not support some options, you may need to remove that options.

default: `"--newest-only --delete --refresh --remote-time"`

5.2.86 reposync_rsync_flags

Flags to use for rysync's reposync. If archive mode (-a,–archive) is used then createrepo is not ran after the rsync as it pulls down the repodata as well. This allows older OS's to mirror modular repos using rsync.

default: `"-rltDv --copy-unsafe-links"`

5.2.87 restart_*

When DHCP and DNS management are enabled, `cobbler sync` can automatically restart those services to apply changes. The exception for this is if using ISC for DHCP, then OMAPI eliminates the need for a restart. `omapi`, however, is experimental and not recommended for most configurations. If DHCP and DNS are going to be managed, but hosted on a box that is not on this server, disable restarts here and write some other script to ensure that the config files get copied/rsynced to the destination box. This can be done by modifying the restart services trigger. Note that if `manage_dhcp` and `manage_dns` are disabled, the respective parameter will have no effect. Most users should not need to change this.

defaults:

```
restart_dns: true
restart_dhcp: true
```

5.2.88 run_install_triggers

Install triggers are scripts in `/var/lib/cobbler/triggers/install` that are triggered in autoinstall pre and post sections. Any executable script in those directories is run. They can be used to send email or perform other actions. They are currently run as root so if you do not need this functionality you can disable it, though this will also disable `cobbler status` which uses a logging trigger to audit install progress.

default: `true`

5.2.89 scm_track_*

enables a trigger which version controls all changes to `/var/lib/cobbler` when add, edit, or sync events are performed. This can be used to revert to previous database versions, generate RSS feeds, or for other auditing or backup purposes. Git and Mercurial are currently supported, but Git is the recommend SCM for use with this feature.

default:

```
scm_track_enabled: false
scm_track_mode: "git"
scm_track_author: "cobbler <cobbler@localhost>"
scm_push_script: "/bin/true"
```

5.2.90 serializer_pretty_json

Sort and indent JSON output to make it more human-readable.

default: `False`

5.2.91 server

This is the address of the Cobbler server – as it is used by systems during the install process, it must be the address or hostname of the system as those systems can see the server. if you have a server that appears differently to different subnets (dual homed, etc), you need to read the `--server-override` section of the manpage for how that works.

default: `127.0.0.1`

5.2.92 sign_puppet_certs_automatically

When puppet starts on a system after installation it needs to have its certificate signed by the puppet master server. Enabling the following feature will ensure that the puppet server signs the certificate after installation if the puppet master server is running on the same machine as Cobbler. This requires `puppet_auto_setup` above to be enabled.

default: `false`

5.2.93 signature_path

The `cobbler import` workflow is powered by this file. Its location can be set with this config option.

default: `/var/lib/cobbler/distro_signatures.json`

5.2.94 signature_url

Updates to the signatures may happen more often then we have releases. To enable you to import new version we provide the most up to date signatures we offer on this like. You may host this file for yourself and adjust it for your needs.

default: `https://cobbler.github.io/signatures/3.0.x/latest.json`

5.2.95 tftpboot_location

This variable contains the location of the tftpboot directory. If this directory is not present Cobbler does not start.

Default: `/srv/tftpboot`

5.2.96 virt_auto_boot

Should new profiles for virtual machines default to auto booting with the physical host when the physical host reboots? This can be overridden on each profile or system object.

default: `true`

5.2.97 webdir

Cobbler's web directory. Don't change this setting – see the Wiki on "relocating your Cobbler install" if your /var partition is not large enough.

default: `@@webroot@@/cobbler`

5.2.98 webdir_whitelist

Directories that will not get wiped and recreated on a `cobbler sync`.

default:

```
webdir_whitelist:
  - misc
  - web
  - webui
  - localmirror
  - repo_mirror
  - distro_mirror
  - images
  - links
  - pub
  - repo_profile
  - repo_system
  - svc
  - rendered
  - .link_cache
```

5.2.99 windows_enabled

Set to true to enable the generation of Windows boot files in Cobbler.

default: `False`

For more information see *Windows installation with Cobbler*.

5.2.100 windows_template_dir

Location of templates used for Windows.

default: `/etc/cobbler/windows`

For more information see *Windows installation with Cobbler*.

5.2.101 samba_distro_share

Samba share name for distros

default: `DISTRO`

For more information see *Windows installation with Cobbler*.

5.2.102 xmlrpc_port

Cobbler's public XML-RPC listens on this port. Change this only if absolutely needed, as you'll have to start supplying a new port option to Koan if it is not the default.

default: `25151`

5.2.103 yum_distro_priority

The default yum priority for all the distros. This is only used if yum-priorities plugin is used. 1 is the maximum value. Tweak with caution.

default: `true`

5.2.104 yum_post_install_mirror

`cobbler repo add` commands set Cobbler up with repository information that can be used during autoinstall and is automatically set up in the Cobbler autoinstall templates. By default, these are only available at install time. To make these repositories usable on installed systems (since Cobbler makes a very convenient mirror) set this to `True`. Most users can safely set this to `True`. Users who have a dual homed Cobbler server, or are installing laptops that will not always have access to the Cobbler server may wish to leave this as `False`. In that case, the Cobbler mirrored yum repos are still accessible at `http://cobbler.example.org/cblr/repo_mirror` and YUM configuration can still be done manually. This is just a shortcut.

default: `True`

5.2.105 yumdownloader_flags

Flags to use for yumdownloader. Not all versions may support `--resolve`.

default: `"--resolve"`

5.2.106 modules

If you have own custom modules which are not shipped with Cobbler directly you may have additional sections here.

authentication

module

This settings decides the login mechanism is being used to log users

Choices:

- authentication.denyall – No one
- authentication.configfile – Use /etc/cobbler/users.digest (default)
- authentication.passthru – Ask Apache to handle it (used for kerberos)
- authentication.ldap – Authenticate against LDAP
- authentication.spacewalk – Ask Spacewalk/Satellite (experimental)
- authentication.pam – Use PAM facilities
- (user supplied) – You may write your own module

Note: A new web interface is in the making. At the moment we do not have any documentation, yet.

---

default: `authentication.configfile`

hash_algorithm

This parameter has currently only a meaning when the option `authentication.configfile` is used. The parameter decides what hashfunction algorithm is used for checking the passwords.

Choices:

- blake2b
- blake2s
- sha3_512
- sha3_384
- sha3_256
- sha3_224
- shake_128
- shake_256

default: `sha3_512`

authorization

module

Once a user has been cleared by the WebUI/XML-RPC, what can they do?

Choices:

- authorization.allowall – full access for all authenticated users (default)
- authorization.ownership – use users.conf, but add object ownership semantics
- (user supplied) – you may write your own module

Warning: If you want to further restrict Cobbler with ACLs for various groups, pick `authorization.ownership`. `authorization.allowall` does not support ACLs. Configuration file does but does not support object ownership which is useful as an additional layer of control.

Note: A new web interface is in the making. At the moment we do not have any documentation, yet.

default: `authorization.allowall`

dns

module

Chooses the DNS management engine if `manage_dns` is enabled in the settings, which is off by default.

Choices:

- managers.bind – default, uses BIND/named
- managers.dnsmasq – uses dnsmasq, also must select dnsmasq for DHCP below
- managers.ndjbdns – uses ndjbdns

Note: More configuration is still required in `/etc/cobbler`

For more information see *DNS management*.

default: `managers.bind`

dhcp

module

Chooses the DHCP management engine if `manage_dhcp` is enabled in the settings, which is off by default.

Choices:

- managers.isc – default, uses ISC dhcpd
- managers.dnsmasq – uses dnsmasq, also must select dnsmasq for DNS above

Note: More configuration is still required in `/etc/cobbler`

For more information see *DHCP Management*.

default: `managers.isc`

tftpd

module

Chooses the TFTP management engine if `manage_tftpd` is enabled in `/etc/cobbler/settings.yaml`, which is **on** by default.

Choices:

- managers.in_tftpd – default, uses the system's TFTP server

default: `managers.in_tftpd`

serializers

module

This decided where Cobbler stores the item data that is being entered into the application.

Choices:

- serializers.file
- serializers.mongodb
- serializers.sqlite

default: `serializers.file`

mongodb

host

The host where MongoDB is running.

default: `localhost`

port

The port where MongoDB is running.

default: `27017`

5.2.107 cache_enabled

If set to `True`, allows the results of some internal operations to be cached, but may slow down editing of objects.

default: `False`

5.2.108 lazy_start

Set to `True` to speed up the start of the Cobbler. When storing collections as files, the directory with the names of the collection elements will be scanned without reading and parsing the files themselves. In the case of storing collections in the database, a projection query is made that includes only the names of the collection elements. The first time an attribute of an element other than a name is accessed, a full read of all other attributes will be performed, and a recursive full read of all elements on which this element depends. At startup, a background task is also launched, which, when idle, fills in all the properties of the elements of the collections. Suitable for configurations with a large number of elements placed on a slow device (HDD, network).

default: `False`

The main configuration file is `settings.yaml`. It is located per default at `/etc/cobbler/`. The file is following the YAML specification.

Warning: If you are using `allow_dynamic_settings` or `auto_migrate_settings`, then the comments in the YAML file will vanish after the first change due to the fact that PyYAML doesn't support comments (Source)

5.3 Migration matrix

To/From	<2.8.5	2.8.5	3.0.0	3.0.1	3.1.0	3.1.1	3.1.2	3.2.0	3.2.1	3.3.0	3.3.1	3.3.2	3.3.3	3.4.0
2.8.5	x	o	–	–	–	–	–	–	–	–	–	–	–	–
3.0.0	x	x	o	–	–	–	–	–	–	–	–	–	–	–
3.0.1	x	x	x	o	–	–	–	–	–	–	–	–	–	–
3.1.0	x	x	x	x	o	–	–	–	–	–	–	–	–	–
3.1.1	x	x	x	x	x	o	–	–	–	–	–	–	–	–
3.1.2	x	x	x	x	x	x	o	–	–	–	–	–	–	–
3.2.0	x	x	x	x	x	x	x	o	–	–	–	–	–	–
3.2.1	x	x	x	x	x	x	x	x	o	–	–	–	–	–
3.3.0	x	x	x	x	x	x	x	x	x	o	–	–	–	–
3.3.1	x	x	x	x	x	x	x	x	x	x	o	–	–	–
3.3.2	x	x	x	x	x	x	x	x	x	x	x	o	–	–
3.3.3	x	x	x	x	x	x	x	x	x	x	x	x	o	–
3.4.0	x	x	x	x	x	x	x	x	x	x	x	x	x	o
main	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Legend: x: supported, o: same version, -: not supported

Note: Downgrades are not supported!

CHAPTER

SIX

USER GUIDE

6.1 DHCP Management

Cobbler can optionally help you manage a DHCP server. This feature is off by default.

The following options are available for `modules.dhcp.module`:

- `"managers.isc"`
- `"managers.dnsmasq"`

Set `manage_dhcp: true` and `manage_dhcp_v4` or `manage_dhcp_v6` to `true` for this setting to take effect.

This allows DHCP to be managed via "cobbler system add" commands, when you specify the MAC address and IP address for systems you add into Cobbler.

You must configure the templates for your networking environment. Read the file and understand how the particular app work before proceeding.

If you already have DHCP configuration data that you would like to preserve (such as DHCP that was manually configured earlier), insert the relevant portions of it into the template file, as running `cobbler sync` will overwrite your previous configuration.

By default, Cobbler updates the DHCP configuration file each time you run `cobbler sync`. Remember to use `cobbler sync` when you use this feature.

6.1.1 isc DHCP

Helpful links:

- Website: https://www.isc.org/dhcp/
- Documentation: https://kb.isc.org/docs/aa-00333

Templates used during generation:

- `/etc/cobbler/dhcp.template`
- `/etc/cobbler/dhcp6.template`

6.1.2 dnsmasq DHCP

Helpful links:

- Website: https://thekelleys.org.uk/dnsmasq/doc.html
- Documentation: https://thekelleys.org.uk/dnsmasq/docs/dnsmasq-man.html

Templates used during generation:

- `/etc/cobbler/dnsmasq.template`

6.1.3 Kea DHCP

Support for Kea is a not yet implemented feature request: https://github.com/cobbler/cobbler/issues/3609

Helpful links:

- Website https://www.isc.org/kea/
- Migration tool from isc: https://www.isc.org/dhcp_migration/
- Documentation: https://kea.readthedocs.io/en/latest/index.html

6.2 DNS management

Cobbler can optionally manage DNS configuration. This feature is off by default.

The following options are available for `modules.dns.module`:

- `"managers.bind"`
- `"managers.dnsmasq"`

For this setting to take effect `manage_dns` must be set to `True`.

All managed files will be updated each time `cobbler sync` is run, and not until then, so it is important to remember to use `cobbler sync` when using this feature.

6.2.1 bind DNS

If using BIND, you must define the zones to be managed with. This is done with two options

- `manage_forward_zones`: This option is a list of domain names.
- `manage_reverse_zones`: This option is a list of IP addresses.

If using BIND, Cobbler will use `/etc/cobbler/named.template` and `/etc/cobbler/zone.template` as a starting point for the `named.conf` and individual zone files, respectively. You may drop zone-specific template files in `/etc/cobbler/zone_templates/<name-of-zone>` which will override the default. These files must be user edited for the user's particular networking environment. Read the file and understand how BIND works before proceeding.

Helpful links:

- Website: https://www.isc.org/bind/
- Documentation: https://bind9.readthedocs.io/en/latest/#

Templates used during generation:

- `/etc/cobbler/named.template`
- `/etc/cobbler/zone.template`
- `/etc/cobbler/zone_templates/<name-of-zone>`

6.2.2 dnsmasq DNS

If using dnsmasq, the template is `/etc/cobbler/dnsmasq.template`. Read this file and understand how dnsmasq works before proceeding.

Helpful links:

- Website: https://thekelleys.org.uk/dnsmasq/doc.html
- Docs: https://thekelleys.org.uk/dnsmasq/docs/dnsmasq-man.html

Templates used during generation:

- `/etc/cobbler/dnsmasq.template`

6.2.3 ndjbdns DNS

If using ndjbdns, the template is `/etc/cobbler/ndjbdns.template`. Read the file and understand how ndjbdns works before proceeding.

For this the DNS server tools of D.J. Bernstein need to be installed.

Helpful links:

- Website: https://cr.yp.to/djbdns.html

Templates used during generation:

- `/etc/cobbler/ndjbdns.template`

6.3 Configuration Management Integrations

Cobbler contains features for integrating an installation environment with a configuration management system, which handles the configuration of the system after it is installed by allowing changes to configuration files and settings.

Resources are the lego blocks of configuration management. Resources are grouped together via Management Classes, which are then linked to a system. Cobbler supports two (2) resource types. Resources are configured in the order listed below.

The initial provisioning of client systems with cobbler is just one component of their management. We also need to consider how to continue to manage them using a configuration management system (CMS). Cobbler can help you provision and introduce a CMS onto your client systems.

One option is cobbler's own lightweight CMS. For that, see the document *Built-In Configuration Management*.

Here we discuss the other option: deploying a CMS such as

- cfengine3,
- puppet,
- bcfg2,
- Chef,
- etc.

Cobbler doesn't force you to chose a particular CMS (or to use one at all), though it helps if you do some things to link cobbler's profiles with the "profiles" of the CMS. This, in general, makes management of both a lot easier.

Note that there are two independent "variables" here: the possible client operating systems and the possible CMSes. We don't attempt to cover all details of all combinations; rather we illustrate the principles and give a small number of illustrative examples of particular OS/CMS combinations. Currently cobbler has better support for Red Hat based OSes and for Puppet so the current examples tend to deal with this combination.

6.3.1 Background considerations

Machine lifecycle

A typical computer has a lifecycle something like:

- installation
- initial configuration
- ongoing configuration and maintenance
- decommissioning

Typically installation happens once. Likewise, the initial configuration happens once, usually shortly after installation. By contrast ongoing configuration evolves over an extended period, perhaps of several years. Sometimes part of that ongoing configuration may involve re-installing an OS from scratch. We can regard this as repeating the earlier phase.

We need not consider decommissioning here.

Installation clearly belongs (in our context) to Cobbler. In a complementary manner, ongoing configuration clearly belongs to the CMS. But what about initial configuration?

Some sites consider their initial configuration as the final phase of installation: in our context, that would put it at the back end of Cobbler, and potentially add significant configuration-based complication to the installation-based Cobbler set-up.

But it is worth considering initial configuration as the first step of ongoing configuration: in our context that would put it as part of the CMS, and keep the Cobbler set-up simple and uncluttered.

Local package repositories

Give consideration to:

- local mirrors of OS repositories
- local repository of local packages
- local repository of pick-and-choose external packages

In particular consider having the packages for your chosen CMS in one of the latter.

Package management

Some sites set up Cobbler always to deploy just a minimal subset of packages, then use the CMS to install many others in a large-scale fashion. Other sites may set up Cobbler to deploy tailored sets of packages to different types of machines, then use the CMS to do relatively small-scale fine-tuning of that.

6.3.2 General scheme

We need to consider getting Cobbler to install and automatically invoke the CMS software.

Set up Cobbler to include a package repository that contains your chosen CMS:

```
cobbler repo add ...
```

Then (illustrating a Red Hat/Puppet combination) set up the kickstart file to say something like:

```
%packages
puppet

%post
/sbin/chkconfig --add puppet
```

The detail may need to be more substantial, requiring some other associated local packages, files and configuration. You may wish to manage this through kickstart snippets.

David Lutterkort has a walkthrough for kickstart. While his example is written for Red Hat (Fedora) and Puppet, the principles are useful for other OS/CMS combinations.

6.3.3 Built-In Configuration Management

Cobbler is not just an installation server, it can also enable two different types of ongoing configuration management system (CMS):

- integration with an established external CMS such as cfengine3, bcfg2, Chef, or puppet.
- its own, much simpler, lighter-weight, internal CMS, discussed here.

Setting up

Cobbler's internal CMS is focused around packages and templated configuration files, and installing these on client systems.

This all works using the same Cheetah-powered templating engine used in kickstart templating, so once you learn about the power of treating your distribution answer files as templates, you can use the same templating to drive your CMS configuration files.

For example:

```
cobbler profile edit --name=webserver --template-files=/srv/cobbler/x.template=/etc/
↪foo.conf
```

A client system installed via the above profile will gain a file `/etc/foo.conf` which is the result of rendering the template given by `/srv/cobbler/x.template`. Multiple files may be specified; each `template=destination` pair should be placed in a space-separated list enclosed in quotes:

```
--template-files="srv/cobbler/x.template=/etc/xfile.conf srv/cobbler/y.template=/etc/
↪yfile.conf"
```

Template files

Because the template files will be parsed by the Cheetah parser, they must conform to the guidelines described in kickstart templating. This is particularly important when the file is generated outside a Cheetah environment. Look for, and act on, Cheetah 'ParseError' errors in the Cobbler logs.

Template files follows general Cheetah syntax, so can include Cheetah variables. Any variables you define anywhere in the cobbler object hierarchy (distros, profiles, and systems) are available to your templates. To see all the variables available, use the command:

```
cobbler profile dumpvars --name=webserver
```

Cobbler snippets and other advanced features can also be employed.

Ongoing maintenance

Koan can pull down files to keep a system updated with the latest templates and variables:

```
koan --server=cobbler.example.org --profile=foo --update-files
```

You could also use `--server=bar` to retrieve a more specific set of templating. Koan can also autodetect the server if the MAC address is registered.

Further uses

This Cobbler/Cheetah templating system can serve up templates via the magic URLs (see "Leveraging Mod Python" below). To do this ensure that the destination path given to any `--template-files` element is relative, not absolute; then Cobbler and Koan won't download those files.

For example, in:

```
cobbler profile edit --name=foo --template-files="/srv/templates/a.src=/etc/foo/a.
↪conf /srv/templates/b.src=1"
```

Cobbler and koan would automatically download the rendered `a.src` to replace the file `/etc/foo/a.conf`, but the `b.src` file would not be downloaded to anything because the destination pathname 1 is not absolute.

This technique enables using the Cobbler/Cheetah templating system to build things that other systems can fetch and use, for instance, BIOS config files for usage from a live environment.

Leveraging Mod Python

All template files are generated dynamically at run-time. If a change is made to a template, a `--ks-meta` variable or some other variable in Cobbler, the result of template rendering will be different on subsequent runs. This is covered in more depth in the *Developer documentation <https://github.com/cobbler/cobbler/wiki>_*.

Possible future developments

- Serving and running scripts via `--update-files` (probably staging them through `/var/spool/koan`).
- Auto-detection of the server name if `--ip` is registered.

6.3.4 Terraform Provider

This is developed and maintained by the Cobbler community. You will find more information in the docs under https://registry.terraform.io/providers/cobbler/cobbler/latest/docs.

The code for the Terraform-Provider can be found at: https://github.com/cobbler/terraform-provider-cobbler

6.3.5 Ansible

Official integration:

- https://docs.ansible.com/ansible/latest/collections/community/general/cobbler_inventory.html#ansible-collections-community-general-cobbler-inventory

Community provided integration:

- https://github.com/ac427/my_cm
- https://github.com/AnKosteck/ansible-cluster
- https://github.com/osism/ansible-cobbler

- https://github.com/hakoerber/ansible-roles

6.3.6 Saltstack

Although we currently can not provide something official we can indeed link some community work here:

- https://github.com/hakoerber/salt-states/tree/master/cobbler

6.3.7 Vagrant

Although we currently can not provide something official we can indeed link some community work here:

- https://github.com/davegermiquet/vmwarevagrantcobblercentos
- https://github.com/dratushnyy/tools
- https://github.com/mkusanagi/cobbler-kickstart-playground

6.3.8 Puppet

There is also an example of Puppet deploying Cobbler: https://github.com/gothicfann/puppet-cobbler

This example is relatively advanced, involving Cobbler "mgmt-classes" to control different types of initial configuration. But if instead you opt to put most of the initial configuration into the Puppet CMS rather than here, then things could be simpler.

Keeping Class Mappings In Cobbler

First, we assign management classes to distro, profile, or system objects.

```
cobbler distro edit --name=distro1 --mgmt-classes="distro1"
cobbler profile add --name=webserver --distro=distro1 --mgmt-classes="webserver likes_
↪llamas" --autoinstall=/etc/cobbler/my.ks
cobbler system edit --name=system --profile=webserver --mgmt-classes="orange" --dns-
↪name=system.example.org
```

For Puppet, the `--dns-name` (shown above) must be set because this is what puppet will be sending to cobbler and is how we find the system. Puppet doesn't know about the name of the system object in cobbler. To play it safe you probably want to use the FQDN here (which is also what you want if you were using Cobbler to manage your DNS, which you don't have to be doing).

External Nodes

For more documentation on Puppet's external nodes feature, see https://docs.puppetlabs.com.

Cobbler provides one, so configure puppet to use `/usr/bin/cobbler-ext-nodes`:

```
[main]
external_nodes = /usr/bin/cobbler-ext-nodes
```

Note: if you are using puppet 0.24 or later then you will want to also add the following to your configuration file.

```
ode_terminus = exec
```

You may wonder what this does. This is just a very simple script that grabs the data at the following URL, which is a URL that always returns a YAML document in the way that Puppet expects it to be returned. This file contains all the parameters and classes that are to be assigned to the node in question. The magic URL being visited is powered by Cobbler.

```
http://cobbler/cblr/svc/op/puppet/hostname/foo
```

(for developer information about this magic URL, visit https://fedorahosted.org/cobbler/wiki/ModPythonDetails)

And this will return data such as:

```
---
classes:
    - distro1
    - webserver
    - likes_llamas
    - orange
parameters:
    tree: 'http://.../x86_64/tree'
```

Where do the parameters come from? Everything that cobbler tracks in `--ks-meta` is also a parameter. This way you can easily add parameters as easily as you can add classes, and keep things all organized in one place.

What if you have global parameters or classes to add? No problem. You can also add more classes by editing the following fields in `/etc/cobbler/settings.yaml`:

```
# cobbler has a feature that allows for integration with config management
# systems such as Puppet.  The following parameters work in conjunction with

# --mgmt-classes  and are described in furhter detail at:
# https://fedorahosted.org/cobbler/wiki/UsingCobblerWithConfigManagementSystem
mgmt_classes: []
mgmt_parameters:
   from_cobbler: 1
```

Alternate External Nodes Script

Attached at `puppet_node.py` is an alternate external node script that fills in the nodes with items from a manifests repository (at `/etc/puppet/manifests/`) and networking information from cobbler. It is configured like the above from the puppet side, and then looks for `/etc/puppet/external_node.yaml` for cobbler side configuration. The configuration is as follows.

```
base: /etc/puppet/manifests/nodes
cobbler: <%= cobbler_host %>
no_yaml: puppet::noyaml
no_cobbler: network::nocobbler
bad_yaml: puppet::badyaml
unmanaged: network::unmanaged
```

The output for network information will be in the form of a pseudo data structure that allows puppet to split it apart and create the network interfaces on the node being managed.

6.3.9 cfengine support

Documentation to be added

6.3.10 bcfg2 support

Documentation to be added

6.3.11 Chef support

Documentation to be added.

There is some integration information on bootstrapping chef clients with cobbler in this blog article

6.3.12 Conclusion

Hopefully this should get you started in linking up your provisioning configuration with your CMS implementation. The examples provided are for Puppet, but we can (in the future) presumably extend `--mgmt-classes` to work with other tools... Just let us know what you are interested in, or perhaps take a shot at creating a patch for it.

6.4 Autoinstallation

6.4.1 Autoinstallation Support

AutoYaST

Kickstart

Cobbler has built-in support for Kickstart guided autoinstallations. We supply a script called "Anamon" that sends client side installation logs back to the Cobbler server.

To learn more about the installer used by Fedora, RedHat Enterprise Linux (RHEL) and other distributions please visit one of the following websites:

- https://fedoraproject.org/wiki/Anaconda
- https://github.com/rhinstaller/anaconda
- https://anaconda-installer.readthedocs.io/en/latest/intro.html

Preseed

Cloud-Init

For the current status of cloud-init support please visit https://github.com/cobbler/cobbler/issues/3218

Ignition (and Combustion)

For the current status of Ignition support please visit:

- https://github.com/cobbler/cobbler/issues/3281
- https://github.com/cobbler/cobbler/issues/3282

Yomi

For the current status of Yomi support please visit https://github.com/cobbler/cobbler/issues/2209

Other auto-installation systems

To request a new type of auto-installation please open a feature request on GitHub: https://github.com/cobbler/cobbler/issues/new?assignees=&labels=enhancement&template=02_feature_request.md&title=

6.4.2 Automatic installation templating

The `--autoinstall_meta` options require more explanation.

If and only if `--autoinstall` options reference filesystem URLs, `--autoinstall-meta` allows for templating of the automatic installation files to achieve advanced functions. If the `--autoinstall-meta` option for a profile read `--autoinstall-meta="foo=7 bar=llama"`, anywhere in the automatic installation file where the string `$bar` appeared would be replaced with the string "llama".

To apply these changes, `cobbler sync` must be run to generate custom automatic installation files for each profile/system.

For NFS and HTTP automatic installation file URLs, the `--autoinstall_meta` options will have no effect. This is a good reason to let Cobbler manage your automatic installation files, though the URL functionality is provided for integration with legacy infrastructure, possibly including web apps that already generate automatic installation files.

Templated automatic files are processed by the templating program/package Cheetah, so anything you can do in a Cheetah template can be done to an automatic installation template. Learn more at https://cheetahtemplate.org/users_guide/intro.html

When working with Cheetah, be sure to escape any shell macros that look like `$(this)` with something like `\$(this)` or errors may show up during the sync process.

The Cobbler Wiki also contains numerous Cheetah examples that should prove useful in using this feature.

Also useful is the following repository: https://github.com/FlossWare/cobbler

6.4.3 Automatic installation snippets

Anywhere a automatic installation template mentions `SNIPPET::snippet_name`, the file named `/var/lib/cobbler/snippets/snippet_name` (if present) will be included automatically in the automatic installation template. This serves as a way to recycle frequently used automatic installation snippets without duplication. Snippets can contain templating variables, and the variables will be evaluated according to the profile and/or system as one would expect.

Snippets can also be overridden for specific profile names or system names. This is described on the Cobbler Wiki.

6.4.4 Autoinstall validation

To check for potential errors in auto-installation files, prior to installation, use `cobbler validate-autoinstalls`. This function will check all profile and system auto-installation files for detectable errors. Since `pykickstart` and related tools are not future-version aware in most cases, there may be some false positives. It should be noted that `cobbler validate-autoinstalls` runs on the rendered autoinstall output, not autoinstall templates themselves.

6.5 Windows installation with Cobbler

Supported installation options:

- UEFI iPXE install (via ipxe-shimx64.efi, ipxe.efi and wimboot tftp/http)
- BIOS iPXE install (via ipxe undionly.kpxe and wimboot tftp/http)
- BIOS PXE install (via syslinux pxelinux.0, linux.c32 and wimboot tftp/http)
- BIOS PXE install (via grub2 grub.0 and wimboot tftp/http)
- BIOS PXE install (via windows pxeboot.n12)

6.5.1 Installation Quickstart guide

- `dnf install python3-pefile python3-hivex wimlib-utils`
- enable Windows support in settings `/etc/cobbler/settings.yaml`:

```
windows_enabled: true
```

- Share `/var/www/cobbler` via Samba:

```
vi /etc/samba/smb.conf
        [DISTRO]
        path = /var/www/cobbler
        guest ok = yes
        browseable = yes
        public = yes
        writeable = no
        printable = no
```

- import the Windows distro:

```
cobbler import --name=win11 --path=/mnt
```

This command will determine the version and architecture of the Windows distribution, extract the files `pxeboot.n12`, `bootmgr.exe`, `winpe.wim` from the distro into the `/var/www/cobbler/distro_mirror/win11/boot` and create a distro and profile named `win11-x86_64`.

Customization winpe.wim

For customization winpe.win you need ADK for Windows.

```
Start -> Apps -> Windows Kits -> Deployment and Imaging Tools Environment
```

You can use either `winpe.wim` obtained either as a result of cobbler import, or take it from ADK:

```
copype.cmd <amd64|x86|arm> c:\winpe
```

If necessary, add drivers to the image:

```
dism /mount-wim /wimfile:media\sources\boot.wim /index:1 /mountdir:mount
dism /image:mount /add-driver /driver:D:\NetKVM\w11\amd64
dism /image:mount /add-driver /driver:D:\viostor\w11\amd64
dism /unmount-wim /mountdir:mount /commit
```

Copy the resulting WinPE image from Windows to the `/var/www/cobbler/distro_mirror/win11/boot` directory of the distro.

6.5.2 UEFI Secure Boot (SB)

For SB you can use `ipxe-shimx64.efi` (unsigned), `ipxe.efi` (unsigned) and `wimboot` (signed with a Microsoft key). Therefore, in this case, we will need our own keys in order to sign `ipxe-shimx64.efi`, `ipxe.efi` and computer fimware with them.

Creating Secure Boot Keys

```
export NAME="DEMO"
openssl req -new -x509 -newkey rsa:2048 -subj "/CN=$NAME PK/" -keyout PK.key \
        -out PK.crt -days 3650 -nodes -sha256
openssl req -new -x509 -newkey rsa:2048 -subj "/CN=$NAME KEK/" -keyout KEK.key \
        -out KEK.crt -days 3650 -nodes -sha256
openssl req -new -x509 -newkey rsa:2048 -subj "/CN=$NAME DB/" -keyout DB.key \
        -out DB.crt -days 3650 -nodes -sha256

export GUID=`python3 -c 'import uuid; print(str(uuid.uuid1()))'`
echo $GUID > myGUID.txt
```

Provide cobbler with bootloaders

```
wget https://github.com/ipxe/shim/releases/download/ipxe-15.7/ipxe-shimx64.efi
wget https://boot.ipxe.org/ipxe.iso
wget https://github.com/ipxe/wimboot/releases/latest/download/wimboot -P /var/lib/
↪cobbler/loaders

mkdir -p /mnt/{cdrom,disk}
mount -o loop,ro ipxe.iso /mnt/cdrom
mount -o loop,ro /mnt/cdrom/esp.img /mnt/disk
```

Signing EFI Binaries and replacing keys in firmware

Signing the bootloaders:

```
sbsign --key DB.key --cert DB.crt --output /var/lib/cobbler/loaders/ipxe-shimx64.efi␣
↪ipxe-shimx64.efi
sbsign --key DB.key --cert DB.crt --output /var/lib/cobbler/loaders/ipxe.efi /mnt/
↪disk/EFI/BOOT/BOOTX64.EFI
cobbler sync
```

Sign the computer firmware with your keys. For VM it can be done like this:

```
rpm -ql python3-virt-firmware | grep '\.pem$'
    /usr/lib/python3.9/site-packages/virt/firmware/certs/CentOSSecureBootCA2.pem
    /usr/lib/python3.9/site-packages/virt/firmware/certs/CentOSSecureBootCAkey1.pem
    /usr/lib/python3.9/site-packages/virt/firmware/certs/
↪MicrosoftCorporationKEKCA2011.pem
    /usr/lib/python3.9/site-packages/virt/firmware/certs/
↪MicrosoftCorporationUEFICA2011.pem
    /usr/lib/python3.9/site-packages/virt/firmware/certs/
↪MicrosoftWindowsProductionPCA2011.pem
    /usr/lib/python3.9/site-packages/virt/firmware/certs/RedHatSecureBootCA3.pem
    /usr/lib/python3.9/site-packages/virt/firmware/certs/RedHatSecureBootCA5.pem
    /usr/lib/python3.9/site-packages/virt/firmware/certs/RedHatSecureBootCA6.pem
    /usr/lib/python3.9/site-packages/virt/firmware/certs/RedHatSecureBootPKKEKkey1.pem
    /usr/lib/python3.9/site-packages/virt/firmware/certs/fedoraca-20200709.pem

virt-fw-vars \
    --input /usr/share/edk2/ovmf/OVMF_VARS.fd \
    --output /var/lib/libvirt/qemu/nvram/win11_VARS.fd \
    --set-pk  ${GUID} PK.crt \
    --add-kek ${GUID} KEK.crt \
    --add-kek 77fa9abd-0359-4d32-bd60-28f4e78f784b /usr/lib/python3.9/site-packages/
↪virt/firmware/certs/MicrosoftCorporationKEKCA2011.pem \
    --add-db  ${GUID} DB.crt \
    --add-db  77fa9abd-0359-4d32-bd60-28f4e78f784b /usr/lib/python3.9/site-packages/
↪virt/firmware/certs/MicrosoftWindowsProductionPCA2011.pem \
    --add-db  77fa9abd-0359-4d32-bd60-28f4e78f784b /usr/lib/python3.9/site-packages/
↪virt/firmware/certs/MicrosoftCorporationUEFICA2011.pem
```

6.5.3 Booting from UEFI iPXE HTTP

Change `dhcpd.conf` to use `ipxe-shimx64.efi`:

```
class "pxeclients" {
     match if substring (option vendor-class-identifier, 0, 9) = "PXEClient";
     next-server 192.168.126.1;

     if exists user-class and option user-class = "iPXE" {
         filename "/ipxe/default.ipxe";
     }
     # UEFI-64-1
     else if option system-arch = 00:07 {
         filename "ipxe-shimx64.efi";
     }
```

The HTTP protocol is used by default in the profile created with the `cobbler import` command:

```
cobbler profile report --name=win11-x86_64 | grep Metadata
    Automatic Installation Metadata :
        {'kernel': 'http://@@http_server@@/images/win11-x86_64/wimboot',
         'bootmgr': 'bootmgr.exe',
         'bcd': 'bcd',
         'winpe': 'winpe.wim',
         'answerfile': 'autounattended.xml',
         'post_install_script': 'post_install.cmd'}
```

```
cat /var/lib/tftpboot/ipxe/default.ipxe
:win11-x86_64
kernel http://192.168.124.1/images/win11-x86_64/wimboot
initrd --name boot.sdi  http://192.168.124.1/cobbler/images/win11-x86_64/boot.sdi␣
↪boot.sdi
initrd --name bootmgr.exe  http://192.168.124.1/cobbler/images/win11-x86_64/bootmgr.
↪exe bootmgr.exe
initrd --name bcd  http://192.168.124.1/cobbler/images/win11-x86_64/bcd bcd
initrd --name winpe.wim  http://192.168.124.1/cobbler/images/win11-x86_64/winpe.wim␣
↪winpe.wim
```

6.5.4 Booting from BIOS firmware

Booting from BIOS iPXE (via ipxe undionly.kpxe and wimboot tftp/http)

Change `dhcpd.conf` to use `undionly.kpxe`:

```
class "pxeclients" {
     match if substring (option vendor-class-identifier, 0, 9) = "PXEClient";
     next-server 192.168.126.1;

     if exists user-class and option user-class = "iPXE" {
         filename "/ipxe/default.ipxe";
     }
     else if option system-arch = 00:00 {
         filename "undionly.pxe";
     }
```

Import distro

```
cobbler import --name=win10 --path=/mnt
```

By default, an EFI partition is created for the profile `win10-x86_64` in the answerfile, and for BIOS boot we can create a profile with `uefi=False` in the metadata:

```
cobbler profile copy \
    --name=win10-x86_64 \
    --newname=win10-bios-pxe-wimboot-http-x86_64 \
    --autoinstall-meta="kernel=http://@@http_server@@/images/win10-x86_64/wimboot␣
↪bootmgr=bootmg2.exe bcd=bc2 winpe=winp2.wim answerfile=autounattende2.xml uefi=False
↪"
cobbler sync
```

If you do not want to use the HTTP protocol, you can either change an existing profile or create a new one with `kernel=wimboot` in the metadata:

```
cobbler profile copy \
    --name=win10-x86_64
    --newname=win10-bios-ipxe-wimboot-tftp-x86_64 \
    --autoinstall-meta="kernel=wimboot bootmgr=bootmg3.exe bcd=bc3 winpe=winp3.wim␣
↪answerfile=autounattende3.xml uefi=False"
cobbler sync
```

```
cat /var/lib/tftpboot/ipxe/default.ipxe
:win10-bios-ipxe-wimboot-tftp-x86_64
kernel /images/win10-x86_64/wimboot
initrd --name boot.sdi  /images/win10-x86_64/boot.sdi boot.sdi
initrd --name bootmgr.exe  /images/win10-x86_64/bootmg3.exe bootmgr.exe
initrd --name bcd  /images/win10-x86_64/bc3 bcd
initrd --name winp3.wim  /images/win10-x86_64/winp3.wim winp3.wim
boot
```

Booting from BIOS PXE (via syslinux pxelinux.0, linux.c32 and wimboot tftp/http)

The `win10-bios-pxe-wimboot-http-x86_64` and `win10-bios-ipxe-wimboot-tftp-x86_64` profiles created earlier are suitable for this boot method. You just need to change `dhcpd.conf` to boot via `pxelinux.0`.

```
class "pxeclients" {
     match if substring (option vendor-class-identifier, 0, 9) = "PXEClient";
     next-server 192.168.126.1;

     if exists user-class and option user-class = "iPXE" {
         filename "/ipxe/default.ipxe";
     }
     else if option system-arch = 00:00 {
         filename "pxelinux.0";
     }
```

```
cat /var/lib/tftpboot/pxelinux.cfg/default
LABEL win10-bios-ipxe-wimboot-tftp-x86_64
    MENU LABEL win10-bios-ipxe-wimboot-tftp-x86_64
    kernel linux.c32
    append /images/win10-x86_64/wimboot initrdfile=/images/win10-x86_64/boot.sdi@boot.
↪sdi initrdfile=/images/win10-x86_64/bootmg3.exe@bootmgr.exe initrdfile=/images/
↪win10-x86_64/bc3@bcd initrdfile=/images/win10-x86_64/winp3.wim@winp3.wim
LABEL win10-bios-pxe-wimboot-http-x86_64
    MENU LABEL win10-bios-pxe-wimboot-http-x86_64
    kernel linux.c32
    append http://192.168.124.1/images/win10-x86_64/wimboot initrdfile=http://192.168.
↪124.1/cobbler/images/win10-x86_64/boot.sdi@boot.sdi initrdfile=http://192.168.124.1/
↪cobbler/images/win10-x86_64/bootmg2.exe@bootmgr.exe initrdfile=http://192.168.124.1/
↪cobbler/images/win10-x86_64/bc2@bcd initrdfile=http://192.168.124.1/cobbler/images/
↪win10-x86_64/winp2.wim@winp2.wim
```

Booting from BIOS PXE (via grub2 grub.0 and wimboot tftp/http)

The `win10-bios-pxe-wimboot-http-x86_64` and `win10-bios-ipxe-wimboot-tftp-x86_64` profiles created earlier also suitable for this boot method. You just need to change `dhcpd.conf` to boot via `grub/grub.0`.

```
class "pxeclients" {
     match if substring (option vendor-class-identifier, 0, 9) = "PXEClient";
     next-server 192.168.126.1;

     if exists user-class and option user-class = "iPXE" {
         filename "/ipxe/default.ipxe";
     }
     else if option system-arch = 00:00 {
         filename "grub/grub.0";
     }
```

```
cat /var/lib/tftpboot/grub/x86_64_menu_items.cfg
menuentry 'win10-bios-ipxe-wimboot-tftp-x86_64' --class gnu-linux --class gnu --class␣
↪os {
  echo 'Loading kernel ...'
  clinux /images/win10-x86_64/wimboot
  echo 'Loading initial ramdisk ...'
  cinitrd  newc:boot.sdi:/images/win10-x86_64/boot.sdi newc:bootmgr.exe:/images/win10-
↪x86_64/bootmg3.exe newc:bcd:/images/win10-x86_64/bc3 newc:winp3.wim:/images/win10-
↪x86_64/winp3.wim
  echo '...done'
}
menuentry 'win10-bios-pxe-wimboot-http-x86_64' --class gnu-linux --class gnu --class␣
↪os {
  echo 'Loading kernel ...'
  clinux (http,192.168.124.1)/images/win10-x86_64/wimboot
  echo 'Loading initial ramdisk ...'
  cinitrd  newc:boot.sdi:(http,192.168.124.1)/cobbler/images/win10-x86_64/boot.sdi␣
↪newc:bootmgr.exe:(http,192.168.124.1)/cobbler/images/win10-x86_64/bootmg2.exe␣
↪newc:bcd:(http,192.168.124.1)/cobbler/images/win10-x86_64/bc2 newc:winp2.wim:(http,
↪192.168.124.1)/cobbler/images/win10-x86_64/winp2.wim
  echo '...done'
}
```

Booting from BIOS PXE install (via windows pxeboot.n12)

This is the only boot method that does not require `wimboot`. Booting can be done via syslinux (pxelinux.0) or ipxe (undionly.kpxe).

Create a file `/etc/tftpd.rules`:

```
rg  \\                                  / # Convert backslashes to slashes
r   (boot1e.\.exe)                      /images/win10-x86_64/\1
r   (/Boot/)(1E.)                       /images/win10-x86_64/\2
```

Change the tftp service

```
cp /usr/lib/systemd/system/tftp.service /etc/systemd/system
```

Replace the line in the `/etc/systemd/system/tftp.service`

```
ExecStart=/usr/sbin/in.tftpd -s /var/lib/tftpboot
    to:
ExecStart=/usr/sbin/in.tftpd -m /etc/tftpd.rules -s /var/lib/tftpboot
```

Restart the tftp service:

```
systemctl daemon-reload
systemctl restart tftp
```

Create a new profile

```
cobbler profile copy \
    --name=win10-x86_64 \
    --newname=win10-bios-syslinux-tftp-x86_64 \
    --autoinstall-meta="kernel=win10a.0 bootmgr=boot1ea.exe bcd=1Ea winpe=winp5.wim␣
↪answerfile=autounattende5.xml uefi=False"
cobbler sync
```

Boot entries were created for this profile:

```
cat /var/lib/tftpboot/pxelinux.cfg/default
LABEL win10-bios-syslinux-tftp-x86_64
    MENU LABEL win10-bios-syslinux-tftp-x86_64
    kernel /images/win10-x86_64/win10a.0

cat /var/lib/tftpboot/ipxe/default.ipxe
:win10-bios-syslinux-tftp-x86_64
kernel /images/win10-x86_64/win10a.0
initrd /images/win10-x86_64/boot.sdi
boot
```

6.5.5 Additional Windows metadata

Additional metadata for preparing Windows boot files can be passed through the `--autoinstall-meta` option for distro, profile or system. The source files for Windows boot files should be located in the `/var/www/cobbler/distro_mirror/<distro_name>/Boot` directory. The trigger copies them to `/var/lib/tftpboot/images/<distro_name>` with the new names specified in the metadata and and changes their contents. The resulting files will be available via tftp and http.

The `sync_post_wingen` trigger uses the following set of metadata:

- kernel

  `kernel` in autoinstall-meta is only used if the boot kernel is `pxeboot.n12` (`--kernel=/path_to_kernel/pxeboot.n12` in distro). In this case, the trigger copies the `pxeboot.n12` file into a file with a new name and replaces:

  - `bootmgr.exe` substring in it with the value passed through the `bootmgr` metadata key in case of using Micrisoft ADK.
  - `NTLDR` substring in it with the value passed through the `bootmgr` metadata key in case of using Legacy RIS.

  Value of the `kernel` key in `autoinstall-meta` will be the actual first boot file. If `--kernel=/path_to_kernel/wimboot` is in distro, then `kernel` key is not used in `autoinstall-meta`.

- bootmgr

  The bootmgr key value is passed the name of the second boot file in the Windows boot chain. The source file to create it can be:

  - `bootmgr.exe` in case of using Micrisoft ADK
  - `setupldr.exe` for Legacy RIS

  Trigger copies the corresponding source file to a file with the name given by this key and replaces in it:

  - substring `\Boot\BCD` to `\Boot\<bcd_value>`, where `<bcd_value>` is the metadata `bcd` key value for Micrisoft ADK.
  - substring `winnt.sif` with the value passed through the `answerfile` metadata key in case of using Legacy RIS.

- bcd

  This key is used to pass the value of the `BCD` file name in case of using Micrisoft ADK. Any `BCD` file from the Windows distribution can be used as a source for this file. The trigger copies it, then removes all boot information from the copy and adds new data from the `initrd` value of the distro and the value passed through the `winpe` metadata key.

- winpe

  This metadata key allows you to specify the name of the WinPE image. The image is copied by the cp utility trigger with the `--reflink=auto` option, which allows to reduce copying time and the size of the disk space on CoW file systems. In the copy of the file, the tribger changes the `/Windows/System32/startnet.cmd` script to the script generated from the `startnet.template` template.

- answerfile

  This is the name of the answer file for the Windows installation. This file is generated from the `answerfile.template` template and is used in:

  - `startnet.cmd` to start WinPE installation
  - the file name is written to the binary file `setupldr.exe` for RIS

- post_install_script

  This is the name of the script to run immediately after the Windows installation completes. The script is specified in the Windows answer file. All the necessary completing the installation actions can be performed directly in this script, or it can be used to get and start additional steps from `http://<server>/cblr/svc/op/autoinstall/<profile|system>/name`. To make this script available after the installation is complete, the trigger creates it in `/var/www/cobbler/distro_mirror/<distro_name>/$OEM$/$1` from the `post_inst_cmd.template` template.

6.5.6 Legacy Windows XP and Windows 2003 Server

- WinPE 3.0 and wimboot can be used to install legacy versions of Windows. `startnet.template` contains the code for starting such an installation via `winnt32.exe`.
  - copy `bootmgr.exe`, `bcd`, `boot.sdi` from Windows 7 and `winpe.wim` from WAIK to the `/var/www/cobbler/distro_mirror/WinXp_EN-i386/boot`

```
cobbler distro add --name=WinXp_EN-i386 \
--kernel=/var/lib/tftpboot/wimboot \
--initrd=/var/www/cobbler/distro_mirror/WinXp_EN-i386/boot/boot.sdi \
--remote-boot-kernel=http://@@http_server@@/cobbler/images/@@distro_name@@/wimboot \
--remote-boot-initrd=http://@@http_server@@/cobbler/images/@@distro_name@@/boot.sdi \
--arch=i386 --breed=windows --os-version=xp \
--boot-loaders=ipxe --autoinstall-meta='clean_disk'

cobbler distro add --name=Win2k3-Server_EN-x64 \
```

(continues on next page)

(continued from previous page)

```
--kernel=/var/lib/tftpboot/wimboot \
--initrd=/var/www/cobbler/distro_mirror/Win2k3-Server_EN-x64/boot/boot.sdi \
--remote-boot-kernel=http://@@http_server@@/cobbler/images/@@distro_name@@/wimboot \
--remote-boot-initrd=http://@@http_server@@/cobbler/images/@@distro_name@@/boot.sdi \
--arch=x86_64 --breed=windows --os-version=2003 \
--boot-loaders=ipxe --autoinstall-meta='clean_disk'

cobbler profile add --name=WinXp_EN-i386 --distro=WinXp_EN-i386 --autoinstall=win.ks \
--autoinstall-meta='bootmgr=bootxea.exe bcd=XEa winpe=winpe.wim answerfile=wine0.sif␣
↪post_install_script=post_install.cmd'

cobbler profile add --name=Win2k3-Server_EN-x64 --distro=Win2k3-Server_EN-x64 --
↪autoinstall=win.ks \
--autoinstall-meta='bootmgr=boot3ea.exe bcd=3Ea winpe=winpe.wim answerfile=wi2k3.sif␣
↪post_install_script=post_install.cmd'
```

- WinPE 3.0 without `wimboot` also can be used to install legacy versions of Windows.
  - copy `pxeboot.n12`, `bootmgr.exe`, `bcd`, `boot.sdi` from Windows 7 and `winpe.wim` from WAIK to the `/var/www/cobbler/distro_mirror/WinXp_EN-i386/boot`

```
cobbler distro add --name=WinXp_EN-i386 \
--kernel=/var/www/cobbler/distro_mirror/WinXp_EN-i386/boot/pxeboot.n12 \
--initrd=/var/www/cobbler/distro_mirror/WinXp_EN-i386/boot/boot.sdi \
--arch=i386 --breed=windows --os-version=xp \
--autoinstall-meta='clean_disk'

cobbler distro add --name=Win2k3-Server_EN-x64 \
--kernel=/var/www/cobbler/distro_mirror/Win2k3-Server_EN-x64/boot/pxeboot.n12 \
--initrd=/var/www/cobbler/distro_mirror/Win2k3-Server_EN-x64/boot/boot.sdi \
--arch=x86_64 --breed=windows --os-version=2003 \
--autoinstall-meta='clean_disk'

cobbler profile add --name=WinXp_EN-i386 --distro=WinXp_EN-i386 --autoinstall=win.ks \
--autoinstall-meta='kernel=wine0.0 bootmgr=bootxea.exe bcd=XEa winpe=winpe.wim␣
↪answerfile=wine0.sif post_install_script=post_install.cmd'

cobbler profile add --name=Win2k3-Server_EN-x64 --distro=Win2k3-Server_EN-x64 --
↪autoinstall=win.ks \
--autoinstall-meta='kernel=w2k0.0 bootmgr=boot3ea.exe bcd=3Ea winpe=winpe.wim␣
↪answerfile=wi2k3.sif post_install_script=post_install.cmd'
```

- Although the ris-linux package is no longer supported, it also can still be used to install older Windows versions.

For example on Fedora 33:

```
dnf install chkconfig python27
dnf install ris-linux --releasever=24 --repo=updates,fedora
dnf install python3-dnf-plugin-versionlock
dnf versionlock add ris-linux
sed -i -r 's/(python)/\12/g' /sbin/ris-linuxd
sed -i -r 's/(\/winos\/inf)\//\1/g' /etc/sysconfig/ris-linuxd
sed -i -r 's/(\/usr\/share\/ris-linux\/infparser.py)/python2 \1/g' /etc/rc.d/init.d/
↪ris-linuxd
sed -i 's/p = p + chr(252)/#&/g' /usr/share/ris-linux/binlsrv.py
mkdir -p /var/lib/tftpboot/winos/inf
```

To support 64 bit distributions:

```
cd /sbin
ln -s ris-linux ris-linux64
cd /etc/sysconfig
cp ris-linuxd ris-linuxd64
sed -i -r 's/(linuxd)/\164/g' ris-linuxd64
sed -i -r 's/(inf)/\164/g' ris-linuxd64
sed -i -r 's/(BINLSRV_OPTS=)/\1--port=4012/g' ris-linuxd64
cd /etc/rc.d/init.d
cp ris-linuxd ris-linuxd64
sed -i -r 's/(linuxd)/\164/g' ris-linuxd64
sed -i -e 's/RIS/RIS64/g' ris-linuxd64
systemctl daemon-reload
mkdir -p /var/lib/tftpboot/winos/inf64
```

copy the Windows network drivers to `/var/lib/tftpboot/winos/inf[64]` and start `ris-linuxd[64]`:

```
systemctl start ris-linuxd
systemctl start ris-linuxd64
```

Preparing boot files for RIS and legacy Windows XP and Windows 2003 Server

```
dnf install cabextract
cd /var/www/cobbler/distro_mirror/<distro_name>
mkdir boot
cp i386/ntdetect.com /var/lib/tftpboot
cabextract -dboot i386/setupldr.ex_
```

If you need to install Windows 2003 Server in addition to Windows XP, then to avoid a conflict, you can rename the `ntdetect.com` file:

```
mv /var/lib/tftpboot/ntdetect.com /var/lib/tftpboot/ntdetect.wxp
sed -i -e 's/ntdetect\.com/ntdetect\.wxp/g' boot/setupldr.exe

cp /var/www/cobbler/distro_mirror/Win2k3-Server_EN-x64/i386/ntdetect.com /var/lib/
↪tftpboot/ntdetect.2k3
sed -i -e 's/ntdetect\.com/ntdetect\.2k3/g' /var/www/cobbler/distro_mirror/Win2k3-
↪Server_EN-x64/boot/setupldr.exe
sed -bi "s/\x0F\xAB\x00\x00/\x0F\xAC\x00\x00/" /var/www/cobbler/distro_mirror/Win2k3-
↪Server_EN-x64/boot/setupldr.exe
```

```
cabextract -dboot i386/startrom.n1_
mv Boot/startrom.n12 boot/pxeboot.n12
touch boot/boot.sdi
```

Copy the required drivers to the `i386`

```
cobbler distro add --name=WinXp_EN-i386 \
--kernel=/var/www/cobbler/distro_mirror/WinXp_EN-i386/boot/pxeboot.n12 \
--initrd=/var/www/cobbler/distro_mirror/WinXp_EN-i386/boot/boot.sdi \
--boot-files='@@local_img_path@@/i386/=@@web_img_path@@/i386/*.*' \
--arch=i386 --breed=windows -os-version=xp

cobbler distro add --name=Win2k3-Server_EN-x64 \
--kernel=/var/www/cobbler/distro_mirror/Win2k3-Server_EN-x64/boot/pxeboot.n12 \
```

(continues on next page)

(continued from previous page)

```
--initrd=/var/www/cobbler/distro_mirror/Win2k3-Server_EN-x64/boot/boot.sdi \
--boot-files='@@local_img_path@@/i386/=@@web_img_path@@/[ia][3m][8d]6*/*.*' \
--arch=x86_64 --breed=windows --os-version=2003

cobbler profile add --name=WinXp_EN-i386 --distro=WinXp_EN-i386 --autoinstall=win.ks \
--autoinstall-meta='kernel=wine0.0 bootmgr=xple0 answerfile=wine0.sif'

cobbler profile add --name=Win2k3-Server_EN-x64 --distro=Win2k3-Server_EN-x64 --
↪autoinstall=win.ks \
--autoinstall-meta='kernel=w2k0.0 bootmgr=w2k3l answerfile=wi2k3.sif'
```

6.5.7 Useful links

Managing EFI Boot Loaders for Linux: Controlling Secure Boot

6.6 VMware ESXi installation with cobbler

What works (DHCPv4):

- BIOS PXE install (via syslinux-3.86 `pxelinux.0` and `mboot.c32`)
- BIOS iPXE install (via ipxe `undionly.kpxe` chainloading syslinux-3.86 `pxelinux.0`)
- UEFI PXE install (via ESXi UEFI bootloader `mboot.efi`)
- UEFI iPXE install (via ipxe `snponly.efi` chainloading ESXi UEFI bootloader `mboot.efi`)

What does not work:

- using DHCPv6 to install ESXi.
- UEFI firmware HTTP install
- Profile boot menus

6.6.1 Installation Quickstart guide

This quickstart guide will assume default settings.

Provide cobbler with ESXi bootloaders

- For a BIOS firmware PXE install, you will need `pxelinux.0` from syslinux version 3.86
- For a UEFI firmware PXE install, you will need the `efi/boot/bootx64.efi` file from the ESXi installer ISO image copied as `mboot.efi`

iPXE booting is documented later. Note that this step will only need to be run once.

```
# STEP 1: Create esxi dir in cobbler bootloaders_dir
mkdir /var/lib/cobbler/loaders/esxi

# STEP 2: If installing from BIOS firmware, pxelinux.0 from syslinux version 3.86 is␣
↪needed
curl https://mirrors.edge.kernel.org/pub/linux/utils/boot/syslinux/3.xx/syslinux-3.86.
↪tar.gz | tar -zx -C /tmp
cp /tmp/syslinux-3.86/core/pxelinux.0 /var/lib/cobbler/loaders/esxi/
```

(continues on next page)

(continued from previous page)

```
# STEP 3: If installing from UEFI firmware, copy efi/boot/bootx64.efi as mboot.efi
# try using your latest ESXi ISO for compatibility
mount -t iso9660 VMware-VMvisor-Installer-7.0U3d-19482537.x86_64.iso /mnt
cp /mnt/efi/boot/bootx64.efi /var/lib/cobbler/loaders/esxi/mboot.efi
umount /mnt

# STEP 4: sync cobbler so bootloaders are copied to tftpboot location
cobbler sync
```

Import an ESXi distro

```
mount -t iso9660 /srv/VMware-VMvisor-Installer-7.0U3d-19482537.x86_64.iso /mnt
cobbler import --name=esxiv70U3d --path=/mnt --arch=x86_64
```

Import will detect the breed as `vmware` and os-version as `esxi70`; it will create a distro named `esxiv70U3d-x86_64` and a profile with the same `esxiv70U3d-x86_64` name.

Add a system

Now add a system with the previously created profile

```
cobbler system add --name some-esxi-host --profile esxiv70U3d-x86_64 --netboot-
↪enabled=true \
    --interface="vmnic0" --mac-address="01:23:45:67:89:ab" --dns-name=some-esxi-host.
↪localdomain
```

Warning: Note that you **must** provide a MAC address for the ESXi system in order to be provisioned via cobbler

Entries in the `/etc/dhcp/dhcpd.conf` file should have been generated for system `some-esxi-host`.

```
# group for Cobbler DHCP tag: default
group {
    ...
    host some-esxi-host.localdomain-vmnic0 {
        hardware ethernet 01:23:45:67:89:ab;
        option host-name "some-esxi-host.localdomain";
        if option system-arch = 00:07 or option system-arch = 00:09 {
            filename = "esxi/system/01-01-23-45-67-89-ab/mboot.efi";
        } else {
            filename = "esxi/pxelinux.0";
        }
        next-server 192.168.1.1;
    }
    ...
}
```

You should now be able to pxe boot your system (BIOS or UEFI firmware) and install ESXi.

6.6.2 Providing Cobbler the ESXi bootloaders

ESXi own bootloader is available on github; this guides uses the ESXi install ISO as an easier way to provide cobbler with the ESXi bootloaders, instead of compiling from source.

Note: ESXi **does not support GRUB**; you can find the details on this VMware community thread; (useful comments from the esx-boot author TimMann).

ESXi provides network bootloaders for:

- BIOS firmware (`mboot.c32`).
- UEFI firmware (`mboot.efi`).
- It is also possible to use iPXE (BIOS and UEFI), and then chainload the ESXi bootloaders.

A cobbler setup with all the ESXi bootloaders would look like:

```
cobbler:~ # ls -alh /var/lib/cobbler/loaders/esxi/
total 488K
drwxr-xr-x 2 root root 4.0K Jul 18 10:47 .
drwxr-xr-x 4 root root 4.0K Jul 18 07:25 ..
-r-xr-xr-x 1 root root 197K Jul 13 11:18 mboot.efi
-rwxr-xr-x 1 root root  17K Jul 13 18:04 pxelinux.0
-rw-r--r-- 1 root root 185K Jul 14 13:54 snponly.efi
-rw-r--r-- 1 root root  72K Jul 18 07:26 undionly.pxe
```

Note that `mboot.c32`, the esxi network bootloader for BIOS firmware, is not listed as it will be downloaded from the images/distro directory in the tftp boot location.

Booting from BIOS firmware

Note: As stated on VMware docs, *The ESXi boot loader for BIOS systems, mboot.c32, runs as a SYSLINUX plugin. VMware builds the mboot.c32 plugin to work with SYSLINUX version 3.86 and tests PXE booting only with that version. Other versions might be incompatible.*

SYSLINUX packages (all versions) can be found at https://mirrors.edge.kernel.org/pub/linux/utils/boot/syslinux/. While syslinux 4.x still worked for ESXi (as for example syslinux 4.05 on rhel7), latest syslinux 6.x is not compatible with the `mboot.c32` plugin (as for example syslinux 6.04 on rhel8).

Providing cobbler with `pxelinux.0` from syslinux 3.86 is therefore needed to pxe boot the ESXi installer. To avoid overwriting other `pxelinux.0` such as the provided via `cobbler mkloaders` command, version 3.86 should be placed on the esxi directory of the *bootloaders_dir*.

The following code snippet shows how to provide cobbler with `pxelinux.0` from syslinux version 3.86:

```
# Create esxi dir in cobbler bootloaders_dir
mkdir /var/lib/cobbler/loaders/esxi
# Obtain syslinux version 3.86
curl https://mirrors.edge.kernel.org/pub/linux/utils/boot/syslinux/3.xx/syslinux-3.86.
↪tar.gz | tar -zx -C /tmp
# Copy pxelinux.0
cp /tmp/syslinux-3.86/core/pxelinux.0 /var/lib/cobbler/loaders/esxi/
# sync cobbler to copy bootloaders to tftp root
cobbler sync
```

During the network boot process:

- the DHCP server will provide the booting host with the IP address of the TFTP server and the location of filename `esxi/pxelinux.0`.
- On the booting host (with MAC address `01:23:45:67:89:ab`) , PXELINUX will request the file `esxi/pxelinux.cfg/01-01-23-45-67-89-ab`
- that file will provide the kernel tftp path to `mboot.c32` (from the distro images link), and append the `boot.cfg` file for the host:

```
cobbler:~ # cat /var/lib/tftpboot/esxi/pxelinux.cfg/01-01-23-45-67-89-ab
timeout 1
prompt 0
default some-esxi-host
ontimeout some-esxi-host
LABEL some-esxi-host
    MENU LABEL some-esxi-host
    kernel /images/esxiv70U3d-x86_64/mboot.c32
    append -c system/01-01-23-45-67-89-ab/boot.cfg
    ipappend 2
```

Booting from UEFI firmware

The ESXi UEFI bootloader can be found in the ESXi installation iso at `efi/boot/bootx64.efi`. You will need to provide the `bootx64.efi` bootloader to cobbler, renamed as `mboot.efi`, on the esxi directory of the *bootloaders_dir*.

Note: As stated on VMware docs, *try to provide cobbler with the latest ESXi UEFI bootloader: Newer versions of mboot.efi can generally boot older versions of ESXi, but older versions of mboot.efi might be unable to boot newer versions of ESXi. If you plan to configure different hosts to boot different versions of the ESXi installer, use the mboot.efi from the newest version.*

The following code snippet shows how to provide cobbler with the `mboot.efi` bootloader:

```
# Create esxi dir in cobbler bootloaders_dir
mkdir /var/lib/cobbler/loaders/esxi
# mount your latest ESXi ISO for compatibility
# example here is VMware-VMvisor-Installer-7.0U3d-19482537.x86_64.iso
mount -t iso9660 VMware-VMvisor-Installer-7.0U3d-19482537.x86_64.iso /mnt
# copy to bootloaders_dir/esxi and rename file to mboot.efi
cp /mnt/efi/boot/bootx64.efi /var/lib/cobbler/loaders/esxi/mboot.efi
# umount and sync cobbler
umount /mnt
cobbler sync
```

- During the network process, for a system with MAC address `01:23:45:67:89:ab`, the DHCP server will provide the booting host with the IP address of the TFTP server and the location of filename `esxi/system/01-01-23-45-67-89-ab/mboot.efi`.
- Then `mboot.efi` will try to download the `boot.cfg` file from the same location: `esxi/system/01-01-23-45-67-89-ab/boot.cfg`

Booting from iPXE

iPXE can be used to boot the ESXi installer:

- For BIOS firmware, iPXE works chainloading the syslinux `pxelinux.0` (from version 3.86). We need to provide cobbbler the iPXE `undionly.kpxe` driver renamed as `undionly.pxe` for consistency with the naming in cobbler.
- For UEFI firmware, iPXE works chainloading the ESXi UEFI bootloader (`mboot.efi`). We need to provide cobbler the iPXE `snponly.efi`. driver.

Note: As iPXE will chainload `pxelinux.0` (syslinux version 3.86) for BIOS and `mboot.efi` for UEFI, you already need to have provided cobbler previously with both.

Some distros already provide a compiled binary of undionly.kpxe and snponly.efi files. This snippet is valid for rhel8 and derivates:

```
# This is an example valid for rhel8 and derivates.
# install ipxe-bootimgs-x86
dnf -y install ipxe-bootimgs-x86
# copy undionly.kpxe to bootloaders_dir/esxi and rename file to undionly.pxe
cp /usr/share/ipxe/undionly.kpxe /var/lib/cobbler/loaders/esxi/undionly.pxe
# copy ipxe-snponly-x86_64.efi to bootloaders_dir/esxi and rename file to snponly.pxe
cp /usr/share/ipxe/ipxe-snponly-x86_64.efi /var/lib/cobbler/loaders/esxi/snponly.efi
# sync cobbler to copy bootloaders to tftp root
cobbler sync
```

Another option is obtaining the binaries from source ipxe:

```
# obtain source ipxe
git clone https://github.com/ipxe/ipxe.git
cd ipxe/src
# make undionly.kpxe
make bin/undionly.kpxe
# copy undionly.kpxe to bootloaders_dir/esxi and rename file to undionly.pxe
cp bin/undionly.kpxe /var/lib/cobbler/loaders/esxi/undionly.pxe
# make snponly.efi
make bin-x86_64-efi/snponly.efi
# copy snponly.efi to bootloaders_dir/esxi
cp bin-x86_64-efi/snponly.efi /var/lib/cobbler/loaders/esxi/
# sync cobbler so bottloaders are copied to tftpboot location
cobbler sync
```

iPXE boot can be enabled on a profile or system basis.

```
cobbler system edit --name some-esxi-host --enable-ipxe=true
```

After enabling iPXE, you shoud see a different DHCP configuration for the host.

```
...
# group for Cobbler DHCP tag: default
group {
...
    host some-esxi-host.localdomain-vmnic0 {
        hardware ethernet 01:23:45:67:89:ab;
        option host-name "some-esxi-host.localdomain";
        if option system-arch = 00:07 or option system-arch = 00:09 {
            if exists user-class and option user-class = "iPXE" {
```

(continues on next page)

(continued from previous page)

```
                filename = "esxi/system/01-01-23-45-67-89-ab/mboot.efi";
            } else {
                filename = "esxi/snponly.efi";
            }
        } else {
            if exists user-class and option user-class = "iPXE" {
                filename = "esxi/pxelinux.0";
            } else {
                filename = "esxi/undionly.pxe";
           }
        }
        next-server 192.168.1.1;
    }
...
}
```

Booting from UEFI HTTP

This is not currently supported.

6.6.3 The boot.cfg file

Note: As stated on VMware docs, *the boot loader configuration file boot.cfg specifies the kernel, the kernel options, and the boot modules that the mboot.c32 or mboot.efi boot loader uses in an ESXi installation. The boot.cfg file is provided in the ESXi installer. You can modify the kernelopt line of the boot.cfg file to specify the location of an installation script or to pass other boot options.*

Cobbler will provide with boot.cfg configuration files from systems and profiles. They are generated via the `bootcfg.template`. You can obtain cobbler's boot.cfg file for a system and profile via HTTP API.

Example call for profile (modules shortened for readability)

```
cobbler:~ # curl http://localhost/cblr/svc/op/bootcfg/profile/esxiv70U3d-x86_64
bootstate=0
title=Loading ESXi installer
prefix=/images/esxiv70U3d-x86_64
kernel=b.b00
kernelopt=runweasel  ks=http://10.4.144.14/cblr/svc/op/autoinstall/profile/esxiv70U3d-
↪x86_64
modules=jumpstrt.gz --- useropts.gz --- features.gz --- k.b00 --- uc_intel.b00 --- uc_
↪amd.b00 --- uc_hygon.b00
build=
updated=0
```

Example call for system (modules shortened for readability). Note that as system is iPXE enabled, prefix is now an http location.

```
cobbler:~ # curl http://localhost/cblr/svc/op/bootcfg/system/some-esxi-host
bootstate=0
title=Loading ESXi installer
prefix=http://10.4.144.14:80/cobbler/links/esxiv70U3d-x86_64
kernel=b.b00
kernelopt=runweasel  ks=http://10.4.144.14/cblr/svc/op/autoinstall/system/some-esxi-
```

(continues on next page)

(continued from previous page)

```
↪host
modules=jumpstrt.gz --- useropts.gz --- features.gz --- k.b00 --- uc_intel.b00 --- uc_
↪amd.b00 --- uc_hygon.b00
build=
updated=0
```

Kernel Options

Kernel options can be added to profiles and to systems. Systems will inherit their profile kernel options.

Example adding a kernel option to profile and system, and the generated boot.cfg file:

```
cobbler:~ # cobbler profile edit --name esxiv70U3d-x86_64 --kernel-options="vlanid=203
↪"
cobbler:~ # cobbler system edit --name some-esxi-host --kernel-options=
↪"systemMediaSize=small"
cobbler:~ # curl http://localhost/cblr/svc/op/bootcfg/system/some-esxi-host
bootstate=0
title=Loading ESXi installer
prefix=http://10.4.144.14:80/cobbler/links/esxiv70U3d-x86_64
kernel=b.b00
kernelopt=runweasel vlanid=203 systemMediaSize=small  ks=http://10.4.144.14/cblr/svc/
↪op/autoinstall/system/some-esxi-host
modules=jumpstrt.gz --- useropts.gz --- features.gz --- k.b00 --- uc_intel.b00 --- uc_
↪amd.b00 --- uc_hygon.b00
build=
updated=0
```

6.6.4 TFTP esxi directory

On the tftp root directory, tree would look like:

```
cobbler:~ # tree /var/lib/tftpboot/esxi
/var/lib/tftpboot/esxi
├── images -> ../images
├── mboot.efi
├── pxelinux.0
├── pxelinux.cfg -> ../pxelinux.cfg
├── snponly.efi
├── system
│   ├── 01-01-23-45-67-89-ab
│   │   ├── boot.cfg
│   │   └── mboot.efi -> ../../mboot.efi
│   └── 01-98-40-bb-c8-36-00
│       ├── boot.cfg
│       └── mboot.efi -> ../../mboot.efi
└── undionly.pxe
```

The directory contains:

- Bootloaders and helper files (`pxelinux.0`, `mboot.efi`, `undionly.pxe`, `snponly.efi`)
- Symlink from `esxi/images` to `images`
- Symlink from `esxi/pxelinux.cfg` to `pxelinux.cfg`

- Directory `system`, with a subdirectory per system mac address. On each system/mac directory, the `boot.cfg` file and a symlink to `mboot.efi`.

6.6.5 Useful links

- VMware ESXi 7 Network Boot Install
- boot.cfg file description
- ESXi boot options

6.7 Extending Cobbler

This section covers methods to extend the functionality of Cobbler through the use of *Triggers* and *Modules*, as well as through extensions to the Cheetah templating system.

6.7.1 Triggers

About

Cobbler triggers provide a way to tie user-defined actions to certain Cobbler commands – for instance, to provide additional logging, integration with apps like Puppet or cfengine, set up SSH keys, tieing in with a DNS server configuration script, or for some other purpose.

Cobbler Triggers should be Python modules written using the low-level Python API for maximum speed, but could also be simple executable shell scripts.

As a general rule, if you need access to Cobbler's object data from a trigger, you need to write the trigger as a module. Also never invoke Cobbler from a trigger, or use Cobbler XMLRPC from a trigger. Essentially, Cobbler triggers can be thought of as plugins into Cobbler, though they are not essentially plugins per se.

Trigger Names (for Old-Style Triggers)

Cobbler script-based triggers are scripts installed in the following locations, and must be made `chmod +x`.

- `/var/lib/cobbler/triggers/add/system/pre/*`
- `/var/lib/cobbler/triggers/add/system/post/*`
- `/var/lib/cobbler/triggers/add/profile/pre/*`
- `/var/lib/cobbler/triggers/add/profile/post/*`
- `/var/lib/cobbler/triggers/add/distro/pre/*`
- `/var/lib/cobbler/triggers/add/distro/post/*`
- `/var/lib/cobbler/triggers/add/repo/pre/*`
- `/var/lib/cobbler/triggers/add/repo/post/*`
- `/var/lib/cobbler/triggers/sync/pre/*`
- `/var/lib/cobbler/triggers/sync/post/*`
- `/var/lib/cobbler/triggers/install/pre/*`
- `/var/lib/cobbler/triggers/install/post/*`

And the same as the above replacing "add" with "remove".

Pre-triggers are capable of failing an operation if they return anything other than 0. They are to be thought of as "validation" filters. Post-triggers cannot fail an operation and are to be thought of notifications.

We may add additional types as time goes on.

Pure Python Triggers

As mentioned earlier, triggers can be written in pure Python, and many of these kinds of triggers ship with Cobbler as stock.

Look in your `site-packages/cobbler/modules` directory and cat "`install_post_report.py`" for an example trigger that sends email when a system finished installation.

Notice how the trigger has a register method with a path that matches the shell patterns above. That's how we find out the type of trigger.

You will see the path used in the trigger corresponds with the path where it would exist if it was a script – this is how we know what type of trigger the module is providing.

The Simplest Trigger Possible

1. Create `/var/lib/cobbler/triggers/add/system/post/test.sh`.
2. `chmod +x` the file.

```
#!/bin/bash
echo "Hi, my name is $1 and I'm a newly added system"
```

However that's not very interesting as all you get are the names passed across. For triggers to be the most powerful, they should take advantage of the Cobbler API – which means writing them as a Python module.

Performance Note

If you have a very large number of systems, using the Cobbler API from scripts with old style (non-Python modules, just scripts in `/var/lib/cobbler/triggers`) is a very very bad idea. The reason for this is that the Cobbler API brings the Cobbler engine up with it, and since it's a seperate process, you have to wait for that to load. If you invoke 3000 triggers editing 3000 objects, you can see where this would get slow pretty quickly. However, if you write a modular trigger (see above) this suffers no performance penalties – it's crazy fast and you experience no problems.

Permissions

The `/var/lib/cobbler/triggers` directory is only writeable by root (and are executed by Cobbler on a regular basis). For security reasons do not loosen these permissions.

Example trigger for resetting Cfengine keys

Here is an example where Cobbler and cfengine are running on two different machines and XMLRPC is used to communicate between the hosts.

Note that this uses the Cobbler API so it's somewhat inefficient – it should be converted to a Python module-based trigger. If it would be a pure Python modular trigger, it would fly.

On the Cobbler box: `/var/lib/cobbler/triggers/install/post/clientkeys.py`

```
#!/usr/bin/python

import socket
import xmlrpclib
import sys
from cobbler import api
cobbler_api = api.BootAPI()
systems = cobbler_api.systems()
box = systems.find(sys.argv[2])
server = xmlrpclib.ServerProxy("http://cfengine:9000")
server.update(box.get_ip_address())
```

On the cfengine box, we run a daemon that does the following (along with a few steps to update our ssh_known_hosts- file):

```
#!/usr/bin/python

import SimpleXMLRPCServer
import os


class Keys(object):
    def update(self, ip):
        try:
            os.unlink('/var/cfengine/ppkeys/root-%s.pub' % ip)
        except OSError:
            pass


keys = Keys()
server = SimpleXMLRPCServer.SimpleXMLRPCServer(("cfengine", 9000))
server.register_instance(keys)
server.serve_forever()
```

See Also

- Post by Ithiriel: Writing triggers

6.7.2 Modules

Certain Cobbler features can be user extended (in Python) by Cobbler users.

These features include storage of data (serialization), authorization, and authentication. Over time, this list of module types will grow to support more options. *Triggers* are basically modules.

See Also

- The Cobbler command line itself (it's implemented in Cobbler modules so it's easy to add new commands)

Python Files and the configuration

To create a module, add a Python file in `/usr/lib/python$version/site-packages/cobbler/modules`. Then, in the appropriate part of the configuration, reference the name of your module so Cobbler knows that you want to activate the module.

(*Triggers* that are Python modules, as well as CLI Python modules don't need to be listed in this file, they are auto-loaded)

An example from the serializers is:

```
modules:
  serializers:
    module: "serializer.file"
```

Each module, regardless of it's nature, must have the following function that returns the type of module (as a string) on an acceptable load (when the module can be loaded) or raises an exception otherwise.

The trivial case for a cli module is:

```
def register():
    return "cli"
```

Other than that, modules do not have a particular API signature – they are "Duck Typed" based on how they are employed. When starting a new module, look at other modules of the same type to see what functions they possess.

6.7.3 Cheetah Macros

Cobbler uses Cheetah for its templating system, it also wants to support other choices and may in the future support others.

It is possible to add new functions to the templating engine, much like snippets that provide the ability to do macro-based things in the template. If you are new to Cheetah, see the documentation at Cheetah User Guide and pay special attention to the `#def` directive.

To create new functions, add your Cheetah code to `/etc/cobbler/cheetah_macros`. This file will be sourced in all Cheetah templates automatically, making it possible to write custom functions and use them from this file.

You will need to restart `cobblerd` after changing the macros file.

6.8 Terraform Provider for Cobbler

First have a brief look at Introduction to Terraform.

Next check out the Cobbler Provider official documentation.

- On GitHub: https://github.com/cobbler/terraform-provider-cobbler
- Releases: https://github.com/cobbler/terraform-provider-cobbler/releases

6.8.1 Why Terraform for Cobbler

Note: This document is written with Cobbler 3.2 and higher in mind, so the examples used here can not be used for Cobbler 2.x and `terraform-provider-cobbler` version 1.1.0 (and older).

There are multiple ways to add new systems, profiles, distro's into Cobbler, eg. through the web-interface or using shell-scripts on the Cobbler-host itself.

One of the main advantages of using the Terraform Provider for Cobbler is speed: you do not have to login into the web-interface or SSH to the host itself and adapt shell-scripts. When Terraform is installed on a VM or your local computer, it adds new assets through the Cobbler API.

6.8.2 Configure Cobbler

Configure Cobbler to have **caching disabled**.

In file `/etc/cobbler/settings`, set `cache_enabled: 0`.

6.8.3 Install Terraform

Terraform comes as a single binary, written in Go. Download an OS-specific package to install on your local system via the Terraform downloads. Unpack the ZIP-file and move the binary-file into `/usr/local/bin`.

Make sure you're using at least **Terraform v0.14 or higher**. Check with `terraform version`:

```
$ terraform version
Terraform v0.14.5
```

Install terraform-provider-cobbler

Since Terraform version 0.13, you can use the Cobbler provider via the Terraform provider registry.

After setting up a Cobbler Terraform repository for the first time, run `terraform init` in the **basedir**, so the Cobbler provider gets installed automatically in `tf_cobbler/.terraform/providers`.

```
$ terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of cobbler/cobbler from the dependency lock file
- Installing cobbler/cobbler v2.0.2...
- Installed cobbler/cobbler v2.0.2 (self-signed, key ID B2677721AC1E7A84)

Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://www.terraform.io/docs/plugins/signing.html

Terraform has made some changes to the provider dependency selections recorded
in the .terraform.lock.hcl file. Review those changes and commit them to your
version control system if they represent changes you intended to make.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
```

(continues on next page)

(continued from previous page)

```
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

If you ever run into this error: `Error: Could not load plugin`, re-run `terraform init` in the **basedir** to reinstall / upgrade the Cobbler provider.

When you initialize a Terraform configuration for the first time with Terraform 0.14 or later, Terraform will generate a new `.terraform.lock.hcl` file in the current working directory. You should include the lock file in your version control repository to ensure that Terraform uses the same provider versions across your team and in ephemeral remote execution environments.

6.8.4 Repository setup & configurations

Create a git repository (for example `tf_cobbler`) and use a phased approach of software testing and deployment in the DTAP-style:

- **development** - holds development systems
- **test** - holds test systems
- **staging** - holds staging / acceptance systems
- **production** - holds production systems
- **profiles** - holds system profiles
- **templates** - holds kickstarts and preseed templates
- **snippets** - holds Cobbler snippets (written in Python Cheetah or Jinja2)
- **distros** - holds OS distributions

The directory-tree would look something like this:

```
├── .gitignore
├── .terraform
│   └── prioviders
├── .terraform.lock.hcl
├── README.md
├── templates
│   ├── main.tf
│   ├── debian10.seed
│   ├── debian10_VMware.seed
│   ├── ...
├── staging
│   ├── db-staging
│   ├── lb-staging
│   ├── web-staging
│   └── ...
├── development
├── production
│   ├── database
│   ├── load_balancer
│   ├── webserver
│   ├── ...
├── set_links.sh
```

(continues on next page)

(continued from previous page)

```
├── snippets
│   ├── partitioning-VMware.file
│   ├── main.tf
│   ├── ...
├── test
│   └── web-test
│   ├── ...
├── distros
│   └── distro-debian10-x86_64.tf
├── profiles
│   └── profile-debian10-x86_64.tf
├── terraform.tfvars
├── variables.tf
└── versions.tf
```

Each host-subdirectory consists of a Terraform-file named `main.tf`, one **symlinked** directory `.terraform` and files **symlinked** from the root: `versions.tf`, `variables.tf`. `.terraform.lock.hcl` and `terraform.tfvars`:

```
tf_cobbler/production/webserver
.
├── .terraform -> ../../.terraform
├── .terraform.lock.hcl -> ../../.terraform.lock.hcl
├── main.tf
├── terraform.tfstate
├── terraform.tfstate.backup
├── terraform.tfvars -> ../../terraform.tfvars
├── variables.tf -> ../../variables.tf
└── versions.tf -> ../../versions.tf
```

The files `terraform.tfstate` and `terraform.tfstate.backup` are the state files once Terraform has run succesfully.

File `versions.tf`

The block in this file specifies the required provider version and required Terraform version for the configuration.

```
terraform {
  required_version = ">= 0.14"
  required_providers {
    cobbler = {
      source = "cobbler/cobbler"
      version = "~> 2.0.1"
    }
  }
}
```

Credentials

You must add the `cobbler_username`, `cobbler_password` and the `cobbler_url` to the Cobbler API into a new file named `terraform.tfvars` in the basedir of your repo.

File `terraform.tfvars`

```
cobbler_username = "cobbler"
cobbler_password = "<the Cobbler-password>"
cobbler_url      = "https://cobbler.example.com/cobbler_api"
```

Terraform automatically loads `.tfvars`-files to populate variables defined in `variables.tf`.

Warning: When using a git repo, do not (force) push the file `terraform.tfvars`, since it contains login credentials!

File `variables.tf`

Tip: We recommend you always add variable descriptions. You never know who'll be using your code, and it'll make their (and your) life a lot easier if every variable has a clear description. Comments are fun too.

Excerpt from: James Turnbull, "The Terraform Book."

```
variable "cobbler_username" {
  description = "Cobbler admin user"
  default     = "some_user"
}

variable "cobbler_password" {
  description = "Password for the Cobbler admin"
  default     = "some_password"
}

variable "cobbler_url" {
  description = "Where to reach the Cobbler API"
  default     = "http://some_server/cobbler_api"
}

provider "cobbler" {
  username = var.cobbler_username
  password = var.cobbler_password
  url      = var.cobbler_url
}
```

Example configuration - system

This is the `main.tf` for system `webserver`, written in so called HCL (HashiCorp Configuration Language). It has been cleaned up with the terraform fmt command, to rewrite Terraform configuration files to a canonical format and style:

Important: Make sure there is only **ONE** gateway defined on **ONE** interface!

```
resource "cobbler_system" "webserver" {
  count            = "1"
  name             = "webserver"
  profile          = "debian10-x86_64"
  hostname         = "webserver.example.com"       # Use FQDN
  autoinstall      = "debian10_VMware.seed"
  # NOTE: Extra spaces at the end are there for a reason!
  # When reading these resource states, the terraform-provider-cobbler
  # parses these fields with an extra space. Adding an extra space in the
  # next 2 lines prevents Terraform from constantly changing the resource.
  kernel_options   = "netcfg/choose_interface=eth0 "
  autoinstall_meta = "fs=ext4 swap=4096 "
  status           = "production"
  netboot_enabled  = "1"

  # Backend interface ##############################
  interface {
    name          = "ens18"
    mac_address   = "0C:C4:7A:E3:C3:12"
    ip_address    = "10.11.15.106"
    netmask       = "255.255.255.0"
    dhcp_tag      = "grqproduction"
    dns_name      = "webserver.example.org"
    static_routes = ["10.11.14.0/24:10.11.15.1"]
    static        = true
    management    = true
  }

  # Public interface ###############################
  interface {
    name        = "ens18.15"
    mac_address = "0C:C4:7A:E3:C3:12"
    ip_address  = "127.28.15.106"
    netmask     = "255.255.255.128"
    gateway     = "127.28.15.1"
    dns_name    = "webserver.example.com"
    static      = true
  }
}
```

Example configuration - snippet

This is the `main.tf` for a snippet:

```
resource "cobbler_snippet" "partitioning-VMware" {
  name = "partitioning-VMware"
  body = file("partitioning-VMware.file")
}
```

In the same folder a file named `partitioning-VMware.file` holds the actual snippet.

Example configuration - repo

```
resource "cobbler_repo" "debian10-x86_64" {
  name           = "debian10-x86_64"
  breed          = "apt"
  arch           = "x86_64"
  apt_components = ["main universe"]
  apt_dists      = ["buster buster-updates buster-security"]
  mirror         = "http://ftp.nl.debian.org/debian/"
}
```

Example configuration - distro

```
resource "cobbler_distro" "debian10-x86_64" {
  name            = "debian10-x86_64"
  breed           = "debian"
  os_version      = "buster"
  arch            = "x86_64"
  kernel          = "/var/www/cobbler/distro_mirror/debian10-x86_64/install.amd/linux"
  initrd          = "/var/www/cobbler/distro_mirror/debian10-x86_64/install.amd/
↪initrd.gz"
}
```

Example configuration - profile

```
resource "cobbler_profile" "debian10-x86_64" {
  name                = "debian10-x86_64"
  distro              = "debian10-x86_64"
  autoinstall         = "debian10.seed"
  autoinstall_meta    = "release=10 swap=2048"
  kernel_options      = "fb=false ipv6.disable=1"
  name_servers        = ["1.1.1.1", "8.8.8.8"]   # Should be a list
  name_servers_search = ["example.com"]
  repos               = ["debian10-x86_64"]
}
```

Example configuration - combined

It is also possible to combine multiple resources into one file. For example, this will combine an Ubuntu Bionic distro, a profile and a system:

```
resource "cobbler_distro" "foo" {
    name = "foo"
    breed = "ubuntu"
    os_version = "bionic"
    arch = "x86_64"
    boot_loaders = ["grub"]
    kernel = "/var/www/cobbler/distro_mirror/Ubuntu-18.04/install/netboot/ubuntu-
↪installer/amd64/linux"
    initrd = "/var/www/cobbler/distro_mirror/Ubuntu-18.04/install/netboot/ubuntu-
↪installer/amd64/initrd.gz"
  }

  resource "cobbler_profile" "foo" {
    name = "foo"
    distro = "foo"
  }

  resource "cobbler_system" "foo" {
    name = "foo"
    profile = "foo"
    name_servers = ["8.8.8.8", "8.8.4.4"]
    comment = "I'm a system"
    interface {
      name = "ens18"
      mac_address = "aa:bb:cc:dd:ee:ff"
      static = true
      ip_address = "1.2.3.4"
      netmask = "255.255.255.0"
    }
    interface {
      name = "ens19"
      mac_address = "aa:bb:cc:dd:ee:fa"
      static = true
      ip_address = "1.2.3.5"
      netmask = "255.255.255.0"
    }
  }
```

File `set_links.sh`

The file `set_links.sh` is used to symlink to the default variables. We need these in every subdirectory.

```
#!/bin/sh

ln -s ../../variables.tf
ln -s ../../versions.tf
ln -s ../../.terraform
ln -s ../../terraform.tfvars
ln -s ../../.terraform.lock.hcl
```

Adding a new system

```
git pull --rebase <-- Refresh the repository

mkdir production/hostname
cd production/hostname

vi main.tf          <-- Add a-based configuration as described above.

../../set_links.sh  # This will create symlinks to .terraform, variables.tf and␣
↪terraform.tfvars

terraform fmt       <-- Rewrites the file "main.tf" to canonical format.

terraform validate  <-- Validates the .tf file (optional).

terraform plan      <-- Create the execution plan.

terraform apply     <-- Apply changes, eg. add this system to the (remote) Cobbler.
```

When `terraform apply` gives errors it is safe to run `rm terraform.tfstate*` in the "hostname" directory and run `terraform apply` again.

6.9 Building ISOs

Since Cobbler uses the systemd hardening option "PrivateTmp" you can't write or read files from your `/tmp` when you run Cobbler via systemd as a service.

Per default this builds an ISO for all available systems and profiles.

Note: All systems refers to systems that are profile based. Systems with a parent image based systems will be skipped.

If you want to generate multiple ISOs you need to execute this command multiple times (with different `--iso` names).

NOTE: This feature is currently only supported for the following architectures: x86_64, ppc, ppc64, ppc64le and ppc64el.

6.9.1 Under the hood

Under the hood the tool "xorriso" is used. It is being executed in the "mkisofs" (the predecessor) compatibility mode. Thus we don't execute "mkisofs" anymore. Please be aware of this when adding CLI options.

On the Python side we are executing the following command:

```
xorriso -as mkisofs $XORRISOFS_OPTS -isohybrid-mbr $ISOHDPFX_location -c isolinux/
↪boot.cat \
  -b isolinux/isolinux.bin -no-emul-boot -boot-load-size 4 -boot-info-table -eltorito-
↪alt-boot \
  -e $EFI_IMG_LOCATION -no-emul-boot -isohybrid-gpt-basdat -V \"Cobbler Install\" \
  -o $ISO $BUILDISODIR
```

Explanation what this command is doing:

```
xorriso -as mkisofs \
  -isohybrid-mbr /usr/share/syslinux/isohdpfx.bin \  # --> Makes the image MBR␣
↪bootable and specifies the MBR File
  -c isolinux/boot.cat \                             # --> Boot Catalog ->␣
↪Automatically created according to Syslinux wiki
  -b isolinux/isolinux.bin \                         # --> Boot file which is␣
↪manipulated by mkisofs/xorriso
  -no-emul-boot \                                    # --> Does not run in emulated␣
↪disk mode when being booted
  -boot-load-size 4 \                                # --> Size of 512 sectors to␣
↪boot in no-emulation mode
  -boot-info-table \                                 # --> Store CD layout in the␣
↪image
  -eltorito-alt-boot \                               # --> Allows to have more then␣
↪one El Torito boot on a CD
  -e /var/lib/cobbler/loaders/grub/x64.efi \         # --> Boot image file which is␣
↪EFI bootable, relative to root directory
  -no-emul-boot \                                    # --> See above
  -isohybrid-gpt-basdat \                            # --> Add GPT additionally to MBR
  -V "Cobbler Install" \                             # --> Name when the image is␣
↪recognized by the OS
  -o /root/generated.iso \                           # --> Produced ISO file name and␣
↪path
  /var/cache/cobbler/buildiso                        # --> Root directory for the␣
↪build
```

6.9.2 Common options for building ISOs

- `--iso`: This defines the name of the built ISO. It defaults to `autoinst.iso`.
- `--distro`: Used to detect the architecture of the ISO you are building. Specifies also the used Kernel and Initrd.
- `--buildisodir`: The temporary directory where Cobbler will build the ISO. If you have enough RAM to build the ISO you should really consider using a tmpfs for performance.
- `--profiles`: Modify the profiles Cobbler builds ISOs for. If this is omitted, ISOs for all profiles will be built.
- `--xorrisofs-opts`: The options which are passed to xorriso additionally to the above shown.

6.9.3 Building standalone ISOs

have to provide the following parameters:

- `--standalone`: If this flag is present, Cobbler will build an ISO which can be installed without network access.
- `--airgapped`: If this flag is present, Cobbler will build an ISO which contains all mirrored repositories for extended installations.
- `--source`: The directory with the sources for the image.

6.9.4 Building net-installer ISOs

You have to provide the following parameters:

- `--systems`: Filter the systems you want to build the ISO for.
- `--exclude-dns`: Flag to add the nameservers (and other DNS information) to the append line or not. This only has an effect in case you supply `--systems`.

6.9.5 Examples

Building exactly one network installer ISO for a specific profile (suitable for all underlying systems):

Building exactly one network installer ISO for a specific system:

Building exactly one airgapped installable ISO for a specific system:

6.9.6 Links with further information

- xorriso homepage
- xorriso manpage
- mkisofs manpage

6.10 GRUB and everything related

The directory `/var/lib/cobbler/grub_config` contains GRUB boot loader (version 2.02) configuration files.

The directory structure is exactly synced (e.g. via `cobbler sync`) to the TFTP (or http/www for http network boot) directory and must be kept as is.

6.10.1 Additional dependencies

If you wish to generate GRUB2 bootloaders in the EFI format please install the dependencies according to the arches you wish to boot with your Cobbler installation: `grub2-ARCH-efi-modules`.

6.10.2 The command "cobbler mkloaders"

This command can create a bootable GRUB2 bootloader in the EFI format. Thus it collects all modules and creates a bootable GRUB2 bootloader. The folder where this is executed is not relevant.

To build GRUB bootloaders for other architectures install the packages and then execute the command against the newly installed directories. openSUSE has enabled you to do this but other distros may not decide to do this. If your distro does not enable you to do this you need to enable yourself for this. For this you need advanced GRUB knowledge, thus this is not part of the tutorial.

This command must be ran after every GRUB2 package update.

The command can be manipulated by changing the settings of Cobbler. The following are being used:

- bootloaders_dir
- grub2_mod_dir
- bootloaders_formats
- bootloaders_modules
- syslinux_dir

- syslinux_memdisk_folder
- syslinux_pxelinux_folder
- bootloaders_shim_folder
- bootloaders_shim_file
- secure_boot_grub_folder
- secure_boot_grub_file
- bootloaders_ipxe_folder

6.10.3 Current workflow

1. Check the settings for above mentioned keys.
2. Create a bootable grubx64.efi loader via `cobbler mkloaders`
3. In `/etc/cobbler/settings.yaml` `grubconfig_dir` has to be set to `/var/lib/cobbler/grub_config`
4. `cobbler sync` automatically populates the GRUB configuration directory now in the TFTP root folder
5. On your DHCP server, point option 67 (`filename`) to `grubx64.efi` (assuming you have configured the other options already)

When you want to use cloud init with the new subiquity installer in Ubuntu 20.04, please keep in mind that the nocloud source has to be quoted in GRUB, otherwise it won't work. For syslinux however, the nocloud source mustn't be quoted! That said, currently you can't use cloud init profiles for Ubuntu 20.04 simultaneously in both Syslinux and GRUB.

6.10.4 IMPORTANT FILES

config/grub

grub.cfg

This file in the main TFTP directory is a fallback for broken firmware. Normally GRUB should already set the prefix to the directory where it has been loaded from (GRUB subdirectory in our case). It is known for (specific versions?) KVM and ppc64le that GRUB may end up loading this as first `grub.cfg`. We simply set `prefix="grub"` and manually load the main config file `grub/grub.cfg`.

grub/grub.cfg

This is the main entry point for all architectures. We always load this config file.

grub/grub/local_*.cfg

This are the architecture specific config files providing local (hard disk) boot entries. These may need adjusting over the time, depending how distributions name their `*.efi` executable for local boot.

grub/grub/system/*

Empty directory where Cobbler will sync machine specific configuration (typically setting local boot or an (auto-)install menu entry). These are named after the mac address of a machine, e.g.: `grub/system/52:54:00:42:07:04` This config file is tried to be loaded from the main `grub.cfg`.

grub/grub/system_link.*

Empty directory where Cobbler will create symlinks, named after the Cobbler name of the machine and it links to above described mac address file in `../system/${mac}` This is only for easier reading and debugging of machine specific GRUB settings.

/var/lib/cobbler/loaders

This directory holds network bootloaders (or links to them) and is also synced to `/srv/tftp` root directory 1 to 1.

It creates GRUB executables for each installed grub2-$arch via: `grub2-mkimage` and links in the corresponding GRUB2 modules and other supported bootloaders (`pxelinux.0`, ...)

If you have installed e.g. a new GRUB or Syslinux version, you should re-run `cobbler mkloaders` to build new GRUB executables. For other, static or already compiled/linked bootloaders like, shim, `pxelinux.0` or a precompiled, signed `grub.efi` executable, it is enough to call `cobbler sync` now (we store links to these now).

The GRUB specific files generated/linked via `cobbler mkloaders` are also described here:

.cobbler_postun_cleanup

Filled up with generated `grub2-mkimage` binaries and created links.

This is needed in `postun` `cobbler.spec` section to remove things again. This is the only, not synced file.

grub/grub.0

- 32 bit PXE (x86 legacy) GRUB executable.
- `grub2-mkimage` generated.
- This can/should be used instead of `pxelinux.0`. You then get the full grub boot process.
- The bootloader is named `grub.0`, because `pxelinux.0` can chain boot this grub executable via network. But it (or specific versions?) wants bootloaders with a filename ending on `.0`.

grub/{grubaa64.efi,grub.ppc64le,grubx64.efi}

Also `grub2-mkimage` generated, architecture specific GRUB executables. These, can directly be network booted on the corresponding/matching architecture. Please have a look at the `dhcpd.conf` template for getting an idea how architecture differing (via DHCP request network packets) works.

On `grub-${arch}` package updates, please call `cobbler mkloaders` to get up-to-date executables. The names of these executables are derived from GRUB2 sources. These are the default names as they should get generated on all distributions by default. These map to `${grub-cpu}-${grub-platform}` as seen below the modules directory structure. Unfortunately this does not map 1 to 1.

grub/{arm64-efi,i386-pc,powerpc-ieee1275,x86_64-efi}

Links to architecture specific GRUB modules. From these `grub2-mkimage` generates above executables.

These directories (where the links point to) have to be named exactly like this. GRUB may download missing/needed modules from `/srv/tftp/${prefix}/${grub-cpu}-${grub-platform}` on the fly as needed.

E.g. using the `grub.cfg` command: hello, will end up in downloading `hello.mod` then doing automatically an `insmod hello`...

grub/{grub.efi,shim.efi}

- Links to precompiled from distribution provided and signed shim and GRUB EFI executables.
- By default `shim.efi` is used in UEFI (x86 at least) case.
- `shim.efi` automatically tries to load `grub.efi`.
- Module loading via network using a signed `grub.efi` loader, does not work.
- All GRUB modules need `grub.cfg` and later sourced config files must be present in the signed `grub.efi` executable.
- For example the "tr" GRUB module was not part of SLES 12 and therefore the reforming of the `${mac}` address to the previous `pxelinux.0` style, e.g.: `52:54:00:42:56:58` -> `01-52-54-00-42-56-58` does not work. But this is overhead anyway, so we now use the plain mac address as filenames for system specific grub configuration.

Use the `use_secure_boot_grub` property to use a pre-built secure boot grub bootloader for a bootloader format, for example:

```
bootloaders_formats:
 x86_64-efi:
   use_secure_boot_grub: true
   binary_name: grubx64.efi
   extra_modules:
     - chain
     - efinet
```

6.11 Repository Management

6.11.1 General

This has already been covered a good bit in the command reference section, for details see: *Cobbler reposync*

Yum repository management is an optional feature and is not required to provision through Cobbler. However, if Cobbler is configured to mirror certain repositories, this feature can be used to associate profiles with those repositories. Systems installed under those profiles will be autoconfigured to use these repository mirrors in `/etc/yum.repos.d`, and if supported (Fedora Core 6 and later), these repositories can be leveraged within Anaconda.

This can be useful if

1. you have a large install base, or
2. you want fast installation and upgrades for your systems, or
3. have some extra software not in a standard repository but want provisioned systems to know about that repository.

Make sure there is plenty of space in Cobbler's webdir, which defaults to `/var/www/cobbler`.

```
cobbler reposync [--only=ONLY] [--tries=N] [--no-fail]
```

`cobbler reposync` is used to update repos known to Cobbler. The command is required to be executed prior to the first provisioning of a system if Cobbler is configured as a mirror. If you just add repos and never run `cobbler reposync`, the content of the repos will be missing. This is probably a command you should include in a crontab. The configuration is left up to the systems administrator.

Note: Mirroring can take a long time because of the amount of data being downloaded.

For those familiar with dnf's reposync, Cobbler's reposync is mostly a wrapper around the `dnf reposync` command. use "cobbler reposync" to update Cobbler mirrors, as dnf's reposync does not perform all required steps. Also Cobbler adds support for rsync and SSH locations, where as dnf's reposync only supports what yum supports (http/ftp).

If you want to update a certain repository, run:

```
cobbler reposync --only="reponame1" ...
```

When updating repos by name, a repo will be updated even if it is set to be not updated during a regular reposync operation (ex: `cobbler repo edit --name=reponame1 --keep-updated=False`).

For distributions using dnf/yum Cobbler can act as a mirror and generate the `.repo` files for the core system packages. This is only possible if the `cobbler import` command provided enough information. If this feature is desirable, it can be turned on by setting `yum_post_install_mirror` to `True` in `/etc/cobbler/settings.yaml` (and running `cobbler sync`). You should not use this feature if machines are provisioned on a different VLAN/network than production, or if you are provisioning laptops that will want to acquire updates on multiple networks.

The flags `--tries=N` (for example, `--tries=3`) and `--no-fail` should likely be used when putting reposync on a crontab. They ensure network glitches in one repo can be retried and also that a failure to synchronize one repo does not stop other repositories from being synchronized.

6.11.2 Importing trees workflow

Cobbler can auto-add distributions and profiles from remote sources, whether this is a filesystem path or an rsync mirror. This can save a lot of time when setting up a new provisioning environment. Import is a feature that many users will want to take advantage of, and is very simple to use.

After an import is run, Cobbler will try to detect the distribution type and automatically assign automatic installation files. By default, it will provision the system by erasing the hard drive, setting up eth0 for DHCP, and using a default password of "cobbler". If this is undesirable, edit the automatic installation files in `/etc/cobbler` to do something else or change the automatic installation setting after Cobbler creates the profile.

Mirrored content is saved automatically in `/var/www/cobbler/distro_mirror`.

Examples:

- `cobbler import --path=rsync://mirrorserver.example.com/path/ --name=fedora --arch=x86`
- `cobbler import --path=root@192.168.1.10:/stuff --name=bar`
- `cobbler import --path=/mnt/dvd --name=baz --arch=x86_64`
- `cobbler import --path=/path/to/stuff --name=glorp`
- `cobbler import --path=/path/where/filer/is/mounted --name=anyname --available-as=nfs://nfs.example.org:/where/mounted/`

Once imported, run a `cobbler list` or `cobbler report` to see what you've added.

By default, the rsync operations will exclude content of certain architectures, debug RPMs, and ISO images – to change what is excluded during an import, see `/etc/cobbler/rsync.exclude`.

Note that all of the import commands will mirror install tree content into `/var/www/cobbler` unless a network accessible location is given with `--available-as`. The option `--available-as` will be primarily used when importing distros stored on an external NAS box, or potentially on another partition on the same machine that is already accessible via HTTP or FTP.

For import methods using rsync, additional flags can be passed to rsync with the option `--rsync-flags`.

Should you want to force the usage of a specific Cobbler automatic installation template for all profiles created by an import, feed the option `--autoinstall` to import, to bypass the built-in automatic installation file auto-detection.

6.11.3 Repository mirroring workflow

The following example shows:

- How to set up a repo mirror for all enabled Cobbler host repositories and two additional repositories.
- Create a profile that will auto install those repository configurations on provisioned systems using that profile.

```
cobbler check
# set up your cobbler distros here.
cobbler autoadd
cobbler repo add --mirror=http://mirrors.kernel.org/fedora/core/updates/6/i386/ --
↪name=fc6i386updates
cobbler repo add --mirror=http://mirrors.kernel.org/fedora/extras/6/i386/ --
↪name=fc6i386extras
cobbler reposync
cobbler profile add --name=p1 --distro=existing_distro_name --autoinstall=/etc/
↪cobbler/kickstart_fc6.ks --repos="fc6i386updates fc6i386extras"
```

6.11.4 Import Workflow

This example shows:

- How to create a provisioning infrastructure from a distribution mirror or from ISO media.
- Create a default PXE configuration, so that by default systems will PXE boot into a fully automated install process for that distribution.

You can use a network rsync mirror, a mounted DVD location, or a tree you have available via a network filesystem.

Import knows how to autodetect the architecture of what is being imported. To make sure things are named correctly, it's a good idea to specify `--arch`. For instance, if you import a distribution named "fedora8" from an x86_64 ISO, specify `--arch=x86_64` and the distro will be named "fedora8-x86_64" automatically, and the right architecture field will also be set on the distribution object. If you are batch importing an entire mirror (containing multiple distributions and arches), you don't have to do this. Cobbler will set the names for things based on the paths it finds for you.

```
cobbler check
cobbler import --path=rsync://yourfavoritemirror.com/rhel/5/os/x86_64 --name=rhel5 --
↪arch=x86_64
# OR
cobbler import --path=/mnt/dvd --name=rhel5 --arch=x86_64
# OR (using an external NAS box without mirroring)
cobbler import --path=/path/where/filer/is/mounted --name=anyname --available-as=nfs:/
↪/nfs.example.org:/where/mounted/
# wait for mirror to rsync...
cobbler report
```

(continues on next page)

(continued from previous page)

```
cobbler system add --name=default --profile=name_of_a_profile1
cobbler system add --name=AA:BB:CC:DD:EE:FF --profile=name_of_a_profile2
cobbler sync
```

6.12 The TFTP Directory

For booting machines in a PXE and/or HTTP-Boot environment the TFTP directory is the most important directory. This folder contains all static files required for booting a system.

The folder of this is dependant on your distro and can be changed in the Cobbler settings. The default should be correctly set during the package build of your Linux distro or during the installation process (if you are use the source installation).

6.12.1 Behaviour

A good explanation of `cobbler sync` can be found here: *Cobbler sync*

In the following we will examine the behaviour for the TFTP directory more in details.

1. `cobbler sync` is executed (we assume a full one for now).
2. The pre-sync triggers are executed.
3. **If the following directories do not exist they are created:**
    1. `pxelinux.cfg`
    2. `grub`
    3. `images`
    4. `ipxe`
    5. `esxi`. Symlinks from `esxi/images` to `images` and from `esxi/pxelinux.cfg` to `pxelinux.cfg`
    6. A symlink from `grub/images` to `images`
4. The content of in above mentioned directories is being fully deleted.
5. All bootloaders are being copied
6. All kernel and initrds are being copied
7. All images (if created) are being copied
8. The PXE menu is being generated and written to disk
9. The post-sync triggers are being executed

Note: If you only sync DHCP, DNS or specific systems the order and actions might be slightly different.

Warning: A `cobbler sync` is not required. Due to the file copying of a lot of small files this is a very expensive operation. Under normal operation Cobbler should move the files automatically to the right places. Only use this command when you encounter problems.

6.12.2 Layout

This is how an example TFTP-Boot Directory could look like. In the following sections we will cover the details of the files and folders.

```
cobbler:~ # ls -alh /srv/tftpboot/
total 105M
drwxr-xr-x 17 root   root  327 Dez 17 14:29 .
drwxr-xr-x  4 root   root   44 Mär  3  2021 ..
drwxr-xr-x  8 root   root 4,0K Nov 18 14:30 grub
-rw-r--r--  1 root   root  429 Okt 21 16:13 grub.cfg
drwxr-xr-x 36 root   root 4,0K Jan 10 14:20 images
-rw-r--r--  1 root   root  96M Jan 28  2021 initrd
drwxr-xr-x  2 root   root   26 Dez  1 15:12 ipxe
-rw-r--r--  1 root   root 8,6M Jan 28  2021 linux
-rw-r--r--  1 root   root  26K Mär 17  2021 memdisk
-rw-r--r--  1 root   root  54K Mär 17  2021 menu.c32
drwxr-xr-x  2 root   root   24 Dez 11  2020 others
-rw-r--r--  1 root   root  26K Mär 17  2021 pxelinux.0
drwxr-xr-x  2 root   root  20K Jan 17 13:02 pxelinux.cfg
drwxr-xr-x  4 root   root 4096 Jul 18 11:02 esxi
```

All files or folders not covered by below explanations are specific to the environment the directory listing was taken from. Those files should not be touched by Cobbler and should survive even a `cobbler sync`.

- `tftpboot/grub/`: Contains the GRUB bootloaders and additional configuration not covered by `tftpboot/grub.cfg`. If available this directory will also contain the `shim.efi` file.
- `tftpboot/grub/system`: Normally contains the GRUB config for the MAC in the filename.

Note: In case Cobbler is not able to find a MAC for the interface it tries to generate an entry for, it applies a fallback strategy. First it tries the IP address. If that was not successful, it finally uses the name if no IP address is known to Cobbler.

- `tftpboot/grub.cfg`: Rescue config file which serves as a pointer on the client side because the error message shows that this is the wrong location for the `grub.cfg` file. GRUB should always try to load `tftpboot/grub/grub.cfg`.
- `tftpboot/images/<distro>/`: Contains always the kernel and initrd of the distro you add to Cobbler. During a `cobbler sync` all folder with distros will be deleted and the structure will be recreated by the paths saved in the `kernel` and `initrd` attributes in a Cobbler distro item.
- `tftpboot/ipxe/default.ipxe`: Cobbler will generate the iPXE menu for you. This is the file where all menu entries will be stored. It will be overwritten regularly by either a change in a distro or by the command `cobbler sync`.
- `tftpboot/pxelinux.0`: The binary for executing the pxelinux bootloader. This is taken from your system at `cobbler sync` time.
- **`tftpboot/pxelinux.cfg`: Normally this directory contains two types of files**
    1. The configuration for each system where the file name is the MAC of the system.
    2. The file named `default` which is used for all PXE Clients not known by MAC address.

Note: In case Cobbler is not able to find a MAC for the interface it tries to generate an entry for, it falls back first to the IP and finally uses the name if no IP is known to Cobbler.

6.13 Internal Database

Note: This document describes advanced topics for system administrators.

The internal database of Cobbler is held at `/var/lib/cobbler/collections`.

6.13.1 Items

An item in Cobbler is a set of attributes grouped together and given a name. An example for this would be a `distro`. On disk those items are represented using JSON. By default, the JSON is minified, however you can make the serializer produce "pretty" JSON files by changing `serializer_pretty_json` to `true` in the Cobbler Settings.

The name of the saved file is the name of the item.

6.13.2 Collections

A collection in Cobbler is a number of `n` Cobbler items that are living inside the same folder.

6.13.3 Notes

If you want to have a backup use the `scm_track` module of Cobbler. It will use Git for version control of the complete `/var/lib/cobbler/` folder.

A rename operation does the following: Delete the item with the old name and create a new item with the new name. This is reflected on disk and thus if Cobbler is being terminated at the wrong point in time, this specific item can get lost. It's unlikely, but if you have items dependent onto that item you will receive errors on the next Cobbler startup.

If you deem yourself a Cobbler expert you may edit the JSON files directly once Cobbler is not running. If Cobbler is running you risk a corruption of the complete application. Please take all actions here with huge precautions and only if you have backups!

6.14 HTTP API

6.14.1 Error codes

status code	status message	Description
200	ok	
404	not found	
500	server error	

6.14.2 Http endpoints

All Http endpoints are found at `http(s)://<fqdn>/cblr/svc/op/<endpoint>`

settings

Returns the currently loaded settings. For specific settings please see *the settings.yaml documentation*.

Example Call:

```
curl http://localhost/cblr/svc/op/setting
```

Example Output:

```
#{
    "allow_duplicate_hostnames": false,
    "allow_duplicate_ips": false,
    "allow_duplicate_macs": false,
    "allow_dynamic_settings": false
...
        "gcry_sha1",
        "gcry_sha256"
    ],
    "grub2_mod_dir": "/usr/share/grub2"
}
```

autoinstall

Autoinstallation files for either a profile or a system.

Profile

Example Call:

```
curl http://localhost/cblr/svc/op/autoinstall/profile/example_profile
```

Example Output:

```
# this file intentionally left blank
# admins:  edit it as you like, or leave it blank for non-interactive install
```

System

Example Call:

```
curl http://localhost/cblr/svc/op/autoinstall/system/example_system
```

Example Output:

```
# this file intentionally left blank
# admins:  edit it as you like, or leave it blank for non-interactive install
```

ks

Autoinstallation files for either a profile or a system. This is used only for backward compatibility with Cobbler 2.6.6 and lower, please use autoinstall if possible.

Profile

Example Call:

```
curl http://localhost/cblr/svc/op/ks/profile/example_profile
```

Example Output:

```
# this file intentionally left blank
# admins:  edit it as you like, or leave it blank for non-interactive install
```

System

Example Call:

```
curl http://localhost/cblr/svc/op/ks/system/example_system
```

Example Output:

```
# this file intentionally left blank
# admins:  edit it as you like, or leave it blank for non-interactive install
```

iPXE

The iPXE configuration for a profile, an image or a system.

Profile

Example Call:

```
curl http://localhost/cblr/svc/op/ipxe/profile/example_profile
```

Example Output:

```
:example_profile
kernel /images/example_distro/vmlinuz
initrd /images/example_distro/initramfs
boot
```

> **Warning:** If the specified profile doesn't exist there is currently no output.

Image

Example Call:

```
curl http://localhost/cblr/svc/op/ipxe/image/example_image
```

Example Output:

Warning: This endpoint is currently broken and will probably have no output.

System

Example Call:

```
curl http://localhost/cblr/svc/op/ipxe/system/example_system
```

Example Output:

```
#!ipxe
iseq ${smbios/manufacturer} HP && exit ||
sanboot --no-describe --drive 0x80
```

Warning: If the specified system doesn't exist there is currently no output.

bootcfg

boot.cfg configuration file for either a profile or a system.

Profile

Example Call:

```
curl http://localhost/cblr/svc/op/bootcfg/profile/example_profile
```

Example Output:

```
bootstate=0
title=Loading ESXi installer
prefix=/images/example_distro
kernel=b.b00
kernelopt=runweasel ks=http://192.168.1.1:80/cblr/svc/op/ks/profile/example_profile
modules=$esx_modules
build=
updated=0
```

System

Example Call:

```
curl http://localhost/cblr/svc/op/bootcfg/system/example_system
```

Example Output:

```
bootstate=0
title=Loading ESXi installer
prefix=/images/example_distro
kernel=b.b00
kernelopt=runweasel ks=http://192.168.1.1:80/cblr/svc/op/ks/system/example_system
modules=$esx_modules
build=
updated=0
```

script

A generated script based on snippets.

Profile

Example Call:

```
curl http://localhost/cblr/svc/op/script/profile/example_profile
```

Example Output:

Warning: This endpoint is currently broken and returns an Error 500.

System

Example Call:

```
curl http://localhost/cblr/svc/op/script/system/example_system
```

Example Output:

Warning: This endpoint is currently broken and returns an Error 500.

events

Returns events associated with the specified user, if no user is given returns all events.

Example Call:

```
curl http://localhost/cblr/svc/op/events/user/example_user
```

Example Output:

```
[]
```

Warning: If the specified user doesn't exist there is currently no output.

template

A rendered template for a system, or for a system linked to a profile.

Profile

Example Call:

```
curl http://localhost/cblr/svc/op/template/profile/example_profile
```

Example Output:

Warning: This endpoint is currently broken.

System

Example Call:

```
curl http://localhost/cblr/svc/op/template/system/example_system
```

Example Output:

Warning: This endpoint is currently broken.

yum

Repository configuration for a profile or a system.

Profile

Example Call:

```
curl http://localhost/cblr/svc/op/yum/profile/example_profile
```

Example Output:

Warning: This endpoint is currently broken and will probably have no output.

System

Example Call:

```
curl http://localhost/cblr/svc/op/yum/system/example_system
```

Example Output:

Warning: This endpoint is currently broken and will probably have no output.

trig

Hook to install triggers.

Example Call:

```
curl http://localhost/cblr/svc/op/trig
```

Example Output:

```
False
```

Profile

Example Call:

```
curl http://localhost/cblr/svc/op/trig/profile/example_profile
```

Example Output:

```
False
```

System

Example Call:

```
curl http://localhost/cblr/svc/op/trig/system/example_system
```

Example Output:

```
False
```

noPXE

If network boot is enabled for specified system.

Example Call:

```
curl http://localhost/cblr/svc/op/nopxe/system/example_system
```

Example Output:

```
True
```

list

Lists all instances of a specified type. Currently the valid options are: `systems`, `profiles`, `distros`, `images`, `repos`, `menus` If no option is selected the endpoint will default to `systems`. If the selected option is not valid the endpoint will return `?`.

Example Call:

```
curl http://localhost/cblr/svc/op/list/what/profiles
```

Example Output:

```
example_profile
example_profile2
```

Warning: currently no output if parameter has no instances.

autodetect

Autodetects the system, returns an error if more than one system is found.

Example Call:

```
curl http://localhost/cblr/svc/op/autodetect
```

Example Output:

Warning: This endpoint is currently broken.

find autoinstall

Find the autoinstallation file for a profile or system.

Profile

Example Call:

```
curl http://localhost/cblr/svc/op/find_autoinstall/profile/example_profile
```

Example Output:

Warning: This endpoint is currently broken.

System

Example Call:

```
curl http://localhost/cblr/svc/op/find_autoinstall/system/example_system
```

Example Output:

Warning: This endpoint is currently broken.

find ks

Find the autoinstallation files for either a profile or a system. This is used only for backward compatibility with Cobbler 2.6.6 and lower, please use `find autoinstall` if possible.

Profile

Example Call:

```
curl http://localhost/cblr/svc/op/findks/profile/example_profile
```

Example Output:

Warning: This endpoint is currently broken.

System

Example Call:

```
curl http://localhost/cblr/svc/op/findks/system/example_system
```

Example Output:

Warning: This endpoint is currently broken.

puppet

Dump puppet data for specified hostname, returns yaml file for host.

Example Call:

```
curl http://localhost/cblr/svc/op/puppet/hostname/example_hostname
```

Example Output:

Warning: This endpoint is currently broken.

Author

Nico Krapp

6.15 HTTP boot

6.15.1 Create Configuration

HTTP configuration

On the Cobbler server create the following files in `/etc/apache2/conf.d/http-tftpboot.conf` with the following content:

```
# allow http access to /srv/tftpboot/grub
Alias "/httpboot" "/srv/tftpboot"

<Directory "/srv/tftpboot">
    Options Indexes FollowSymLinks
    AddType application/efi efi
    <IfVersion <= 2.2>
        Order allow,deny
        Allow from all
    </IfVersion>
    <IfVersion >= 2.4>
        Require all granted
    </IfVersion>
</Directory>
```

After the changes have been made issue the following command:

```
systemctl restart apache2.service
```

DHCP configuration

To use HTTP-boot the following 2 entries need to be added:

```
option vendor-class-identifier "HTTPClient";
filename "http://<ip address SUSE Manager Server>/httpboot/grub/shim.efi";
```

The following example can be used if both traditional and HTTP boot are needed. It is recommended to use `class`-ses for this.

Example Configuration:

```
class "pxeclients" {
  match if substring (option vendor-class-identifier, 0, 9) = "PXEClient";
  next-server <ip address SUSE Manager Server>;
  filename "pxelinux.0";
}

class "httpclients" {
  match if substring (option vendor-class-identifier, 0, 10) = "HTTPClient";
  option vendor-class-identifier "HTTPClient";
  filename "http://<ip address SUSE Manager Server>/httpboot/grub/shim.efi";
}
```

Author

Michael Brookhuis

6.16 Power Management

Cobbler allows for linking your power management systems with cobbler, making it very easy to make changes to your systems when you want to reinstall them, or just use it to remember what the power management settings for all of your systems are. For instance, you can just change what profile they should run and flip their power states to begin the reinstall!

6.16.1 What's Supported

All of the following modes are supported. Most all of them use the fence scripts internally so you will want fence installed. This is part of the 'cman' package for some distributions, though it's fence-agents in Fedora 11 and later (which cobbler has as a dependency on that OS for newer versions).

- bullpap
- wti
- apc_snmp
- ether_wake
- ipmilan
- drac
- ipmitool
- ilo
- rsa
- lpar
- bladecenter
- and many more…

6.16.2 Example of Set Up

You have a WTI powerbar. Define that system foo is a part of that powerbar on plug 7

```
cobbler system edit --name foo --power-type=wti --power-address=foo-mgmt.example.org -
↪-power-user Administrator --power-pass PASSWORD --power-id 7
```

You have a DRAC based blade:

```
cobbler system edit --name blade7 --power-type=drac --power-address=blade-mgmt.
↪example.org --power-user Administrator --power-pass=PASSWORD --power-id blade7
```

You have an IPMI based system:

```
cobbler system edit --name foo --power-type=ipmi --power-address=foo-mgmt.example.org␣
↪--power-user Administrator --power-pass=PASSWORD
```

You have a IBM HMC managed system:

```
cobbler system edit --name 9115-505 --power-type=lpar --power-address=ibm-hmc.example.
↪org --power-user hscroot --power-pass=PASSWORD --power-id system:partition
```

Note: The *–power-id* option is used to indicate **both** the managed system name and a logical partition name. Since an IBM HMC is responsible for managing more than one system, you must supply the managed system name and logical partition name separated by a colon (':') in the –power-id command-line option.

You have an IBM Bladecenter:

```
cobbler system edit --name blade-06 --power-type=bladecenter --power-address=blademm.
↪example.org --power-user USERID --power-pass=PASSW0RD --power-id 6
```

Note: The *`--power-id`* option is used to specify what slot your blade is connected.

6.16.3 Data Entry

Tip: to make life easier, you can use cobbler find + xargs [CommandLineSearch](Command Line Search) to batch populate the settings for lots of systems.

6.16.4 Defaults

If `--power-user` and `--power-pass` are left blank, the values of `default_power_user` and `default_power_pass` will be loaded from cobblerd's environment at the time of usage.

`--power-type` also has a default value in our settings, initially set to "ipmilanplus".

6.16.5 Using the Power Management Features

Assigning A System To Be Installed To A New Profile

```
obbler system edit --name=foo --netboot-enabled=1 --profile=install-this-profile-name-
↪instead
```

Powering Off A System

```
cobbler system poweroff --name=foo
```

Powering On A System

```
cobbler system poweron --name=foo
```

Rebooting A System (if netboot-enabled is turned on, it will now reinstall to the new profile – assuming PXE is working)

```
cobbler system reboot --name=foo
```

Since not all power management systems support reboot, this is a "power off, sleep for 1 second, and power on" operation.

6.16.6 Implementation

The individual command syntaxes are generated from Cheetah templates in /etc/cobbler/power in case you need to modify the commands or add additional options. You can also add new power types if you like if you are using Cobbler 2.0 and later, just by making new files in that directory.

6.16.7 Important: Security Implications

Storing the power control usernames and passwords in Cobbler means that information is essentially public (this data is available via XMLRPC without access control), therefore you will want to control what machines have network access to contact the power management devices if you use this feature (such as /only/ the cobbler machine, and then control who has local access to the cobbler machine). Also do not reuse important passwords for your power management devices. If this concerns you, you can still use this feature, just don't store the username/password in Cobbler for your power management devices.

If you are not going to to store power control passwords in Cobbler, leave the username and password fields blank.

Cobbler will first try to source them from it's environment using the `COBBLER_POWER_USER` and `COBBLER_POWER_PASS` variables.

This may also be too insecure for some, so in this case, don't set these, and supply `--power-user` and `--power-pass` when running commands like `cobbler system poweron` and `cobbler system poweroff`. The values used on the command line are always used, regardless of the value stored in Cobbler or the environment, if so provided.

```
cobbler system poweron --name=foo --power-user=X --power-pass=Y
```

Be advised of current limitations in storing passwords, make your choices accordingly and in relation to the ease-of-use that you need, and secure your networks appropriately.

6.17 Boot CD

Cobbler can build all of it's profiles into a bootable CD image using the `cobbler buildiso` command. This allows for PXE-menu like bring up of bare metal in environments where PXE is not possible. Another more advanced method is described in the Koan manpage, though this method is easier and sufficient for most applications.

6.18 Advanced networking

First off, read the cobbler manpage for all the settings you can set on a system object.

This page details some of the networking tips and tricks in more detail, regarding what you can set on system records to set up networking, without having to know a lot about kickstart/Anaconda.

These features include:

- Arbitrary NIC naming (the interface is matched to a physical device using it's MAC address)
- Configuring DNS nameserver addresses
- Setting up NIC bonding
- Defining for static routes
- Support for VLANs

If you want to use any of these features, it's highly recommended to add the MAC addresses for the interfaces you're using to Cobbler for each system.

6.18.1 Arbitrary NIC naming

You can give your network interface (almost) any name you like.

```
cobbler system edit --name=foo1.bar.local --interface=mgmt --mac=AA:BB:CC:DD:EE:F0]
cobbler system edit --name=foo1.bar.local --interface=dmz --mac=AA:BB:CC:DD:EE:F1
```

The default interface is named `default`, but you don't have to call it that.

Note that you can't name your interface after a kernel module you're using. For example: if a NIC is called `drbd`, the module `drbd.ko` would stop working. This is due to an "alias" line in `/etc/modprobe.conf`.

6.18.2 Name Servers

For static systems, the `--name-servers` parameter can be used to specify a list of name servers to assign to the systems.

```
cobbler system edit --name=foo --interface=eth0 --mac=AA:BB:CC::DD:EE:FF --static=1 --
↪name-servers="<ip1> <ip2>"
```

DNS and DHCP Management

See [ManageDns](Dns Management) and [ManageDhcp](Dhcp Management) for how cobbler can help control your DHCP and DNS servers.

6.18.3 NIC bonding

Bonding is also known as trunking, or teaming. Different vendors use different names. It's used to join multiple physical interfaces to one logical interface, for redundancy and/or performance.

You can set up a bond, to join interfaces eth0 and `eth1` to a failover (active-backup) interface `bond0` as follows:

```
cobbler system edit --name=foo2.bar.local --interface=eth0 --mac=AA:BB:CC:DD:EE:F0 --
↪bonding=slave --bonding-master=bond0
cobbler system edit --name=foo2.bar.local --interface=eth1 --mac=AA:BB:CC:DD:EE:F1 --
↪bonding=slave --bonding-master=bond0
cobbler system edit --name=foo2.bar.local --interface=bond0 --bonding=master --
↪bonding-opts="miimon=100 mode=1"
```

6.18.4 Static routes

You can define static routes for a particular interface to use with `--static-routes`.

The format of a static route is: `network/CIDR:gateway`

So, for example to route the `192.168.1.0/24` network through `192.168.1.254`:

```
cobbler system edit --name=foo --interface=eth0 --static-routes="192.168.1.0/24:192.
↪168.1.254"
```

As with all lists in cobbler, the `--static-routes` list is space-separated so you can specify multiple static routes if needed.

6.18.5 VLANs

You can now add VLAN tags to interfaces from Cobbler. In this case we have two VLANs on `eth0`: 10 and 20. The default VLAN (untagged traffic) is not used:

```
cobbler system edit --name=foo3.bar.local --interface=eth0 --mac=AA:BB:CC:DD:EE:F0 --
↪static=1
cobbler system edit --name=foo3.bar.local --interface=eth0.10 --static=1 --ip=10.0.10.
↪5 --subnet=255.255.255.0
cobbler system edit --name=foo3.bar.local --interface=eth0.20 --static=1 --ip=10.0.20.
↪5 --subnet=255.255.255.0
```

You have to install the vconfig package for this to work.

6.18.6 Kickstart Notes

Three different networking Kickstart Snippets must be present in your kickstart files for this to work:

- `pre_install_network_config`
- `network_config`
- `post_install_network_config`

The default kickstart templates (`/var/lib/cobbler/kickstart/sample\*.ks`) have these installed by default so they work out of the box.

6.19 SELinux

Providing working policies for SELinux (and AppArmor) is the responsibility of downstream (e.g. your Linux or repo vendor). Unfortunately, every now and then issues tend to pop up on the mailing lists or in the issue tracker. Since we're really not in the position to resolve SELinux issues, all reported bugs will be closed. All we can do is try to document these issues here, hopefully the community is able to provide some feedback/workarounds/fixes.

6.19.1 General Tips - Fedora

Service Specific Manpages

Manpages are automatically generated for SELinux, and many application that are restricted by SELinux. This documentation is provided by the `selinux-policy-devel` package. For example, to see the SELinux restrictions on cobbler, try:

```
yum install selinux-policy-devel
man cobblerd_selinux
```

Booleans

Many SELinux restrictions can easily be remedied by switching a boolean specifically designed for the purpose. For example, many cobbler deployments require `cobbler_can_network_connect` to be true.

To find and set booleans that might affect the service you're working with, do:

```
getsebool -a|grep cobbler
setsebool -P cobbler_can_network_connect 1
```

Context

File context labelling is also addressed in `man cobblerd_selinux`. Remember, `mv` will retain a file's current context, and `cp` will make the file inherit the target directory's context. The first step and easiest step in troubleshooting context denials is to simply ensure the default labels are applied:

```
restorecon -R /var/lib/cobbler/
```

See the aforementioned manpage to learn of applying contexts to non-default paths.

Other policy issues

SELinux denials can be caused by policies or labelling not applied (requiring admin action) or by improper default policy (requiring developer action). You can create custom policy modules, if needed:

> yum install policycoreutils-python checkpolicy grep cobbler /var/log/audit/audit.log | audit2why # Read over the denials, check for booleans, labelling problems etc

Create a policy module for a specific denial:

```
grep "audit(1388259039.970:1931)" /var/log/audit/audit.log | audit2allow -M sensible_
↪module_name
semodule -i sensible_module_name.pp
```

Custom Policy Best Practices

Applying custom modules atomically ensures appropriate restrictions and helps to identify individual policy or labelling issues. Some denials are caused by booleans or labelling that are not yet applied (requiring admin action); some denials are caused by the default policy not matching the behaviour of the code (requiring developer action). By providing feedback to both SELinux policy maintainers and application developers in bug reports, you can help make secure use of cobbler (and other services) easier for everyone.

6.19.2 Fedora 16 / RHEL6 / CentOS6 - Python MemoryError

Obscure error message for which a solution is unknown. The workaround is to disable SELinux or build a custom SELinux module to run cobbler unconfined. See also https://bugzilla.redhat.com/show_bug.cgi?id=816309

> Starting cobbler daemon: Traceback (most recent call last): File "/usr/bin/cobblerd", line 76, in main api = cobbler_api.BootAPI(is_cobblerd=True) File "/usr/lib/python2.6/site-packages/cobbler/api.py", line 127, in init module_loader.load_modules() File "/usr/lib/python2.6/site-packages/cobbler/module_loader.py", line 62, in load_modules blip = import("modules.%s" % (modname), globals(), locals(), [modname]) File "/usr/lib/python2.6/site-packages/cobbler/modules/authn_pam.py", line 53, in from ctypes import CDLL, POINTER, Structure, CFUNCTYPE, cast, pointer, sizeof File "/usr/lib64/python2.6/ctypes/init.py", line 546, in CFUNCTYPE(c_int)(lambda: None) MemoryError

To run cobbler unconfined, build the following SELinux module using the instructions http://www.city-fan.org/tips/BuildSeLinuxPolicyModules

```
root@system # cat cobbler_unconfined.te
policy_module(cobbler_unconfined, 1.0)
gen_require('type cobblerd_t;')
unconfined_domain(cobblerd_t)
root@system # make -f /usr/share/selinux/devel/Makefile cobbler_unconfined.pp
root@system # semodule -i cobbler_unconfined.pp
root@system # semodule -l | grep cobbler
cobbler    1.1.0
cobbler_unconfined 1.0
root@system #
```

6.19.3 Fedora 14

While many users with SELinux distributions opt to turn SELinux off, you may wish to keep it on. For Fedora 14 you might want to amend the SELinux policy settings:

```
/usr/sbin/semanage fcontext -a -t public_content_rw_t "/var/lib/tftpboot/.*"
/usr/sbin/semanage fcontext -a -t public_content_rw_t "/var/www/cobbler/images/.*"
restorecon -R -v "/var/lib/tftpboot/"
restorecon -R -v "/var/www/cobbler/images.*"
# Enables cobbler to read/write public_content_rw_t
setsebool cobbler_anon_write on
# Enable httpd to connect to cobblerd (optional, depending on if web interface is␣
↪installed)
# Notice: If you enable httpd_can_network_connect_cobbler and you should switch httpd_
↪can_network_connect off
setsebool httpd_can_network_connect off
setsebool httpd_can_network_connect_cobbler on
#Enabled cobbler to use rsync etc.. (optional)
setsebool cobbler_can_network_connect on
#Enable cobbler to use CIFS based filesystems (optional)
setsebool cobbler_use_cifs on
# Enable cobbler to use NFS based filesystems (optional)
setsebool cobbler_use_nfs on
# Double check your choices
getsebool -a|grep cobbler
```

The information suggested by `cobbler check` should be sufficient for older distributions. These is just a few `fcontext` commands and setting `httpd_can_network_connect`.

6.19.4 ProtocolError: <ProtocolError for x.x.x.x:80/cobbler_api: 503 Service Temporarily Unavailable>

If you see this when you run `cobbler check` or any other Cobbler command, it means SELinux is blocking httpd from talking with cobblerd. The command to fix this is:

```
setsebool -P httpd_can_network_connect true
```

6.20 API

Cobbler also makes itself available as an XML-RPC API for use by higher level management software. Learn more at https://cobbler.github.io

6.21 Triggers

Triggers provide a way to integrate Cobbler with arbitrary 3rd party software without modifying Cobbler's code. When adding a distro, profile, system, or repo, all scripts in `/var/lib/cobbler/triggers/add` are executed for the particular object type. Each particular file must be executable and it is executed with the name of the item being added as a parameter. Deletions work similarly – delete triggers live in `/var/lib/cobbler/triggers/delete`. Order of execution is arbitrary, and Cobbler does not ship with any triggers by default. There are also other kinds of triggers – these are described on the Cobbler Wiki. For larger configurations, triggers should be written in Python – in which case they are installed differently. This is also documented on the Wiki.

6.22 Images

Cobbler can help with booting images physically and virtually, though the usage of these commands varies substantially by the type of image. Non-image based deployments are generally easier to work with and lead to more sustainable infrastructure. Some manual use of other commands beyond of what is typically required of Cobbler may be needed to prepare images for use with this feature.

6.23 Non-import (manual) workflow

The following example uses a local kernel and initrd file (already downloaded), and shows how profiles would be created using two different automatic installation files – one for a web server configuration and one for a database server. Then, a machine is assigned to each profile.

```
cobbler check
cobbler distro add --name=rhel4u3 --kernel=/dir1/vmlinuz --initrd=/dir1/initrd.img
cobbler distro add --name=fc5 --kernel=/dir2/vmlinuz --initrd=/dir2/initrd.img
cobbler profile add --name=fc5webservers --distro=fc5-i386 --autoinstall=/dir4/kick.
↪ks --kernel-options="something_to_make_my_gfx_card_work=42 some_other_parameter=foo"
cobbler profile add --name=rhel4u3dbservers --distro=rhel4u3 --autoinstall=/dir5/kick.
↪ks
cobbler system add --name=AA:BB:CC:DD:EE:FF --profile=fc5-webservers
cobbler system add --name=AA:BB:CC:DD:EE:FE --profile=rhel4u3-dbservers
cobbler report
```

6.24 Virtualization

For Virt, be sure the distro uses the correct kernel (if paravirt) and follow similar steps as above, adding additional parameters as desired:

```
cobbler distro add --name=fc7virt [options...]
```

Specify reasonable values for the Virt image size (in GB) and RAM requirements (in MB):

```
cobbler profile add --name=virtwebservers --distro=fc7virt --autoinstall=path --virt-
↪file-size=10 --virt-ram=512 [...]
```

Define systems if desired. Koan can also provision based on the profile name.

```
cobbler system add --name=AA:BB:CC:DD:EE:FE --profile=virtwebservers [...]
```

If you have just installed Cobbler, be sure that the *cobblerd* service is running and that port 25151 is unblocked.

See the manpage for Koan for the client side steps.

6.25 Network Topics

6.25.1 PXE Menus

Cobbler will automatically generate PXE menus for all profiles that have the `enable_menu` property set. You can enable this with:

```
cobbler profile edit --name=PROFILE --enable-menu=yes
```

Running `cobbler sync` is required to generate and update these menus.

To access the menus, type `menu` at the `boot:` prompt while a system is PXE booting. If nothing is typed, the network boot will default to a local boot. If "menu" is typed, the user can then choose and provision any Cobbler profile the system knows about.

If the association between a system (MAC address) and a profile is already known, it may be more useful to just use `system add` commands and declare that relationship in Cobbler; however many use cases will prefer having a PXE system, especially when provisioning is done at the same time as installing new physical machines.

If this behavior is not desired, run `cobbler system add --name=default --profile=plugh` to default all PXE booting machines to get a new copy of the profile `plugh`. To go back to the menu system, run `cobbler system remove --name=default` and then `cobbler sync` to regenerate the menus.

When using PXE menu deployment exclusively, it is not necessary to make Cobbler system records, although the two can easily be mixed.

Additionally, note that all files generated for the PXE menu configurations are templatable, so if you wish to change the color scheme or equivalent, see the files in `/etc/cobbler`.

6.25.2 Default PXE Boot behavior

What happens when PXE booting a system when Cobbler has no record of the system being booted?

By default, Cobbler will configure PXE to boot to the contents of `/etc/cobbler/default.pxe`, which (if unmodified) will just fall through to the local boot process. Administrators can modify this file if they like to change that behavior.

An easy way to specify a default Cobbler profile to PXE boot is to create a system named `default`. This will cause `/etc/cobbler/default.pxe` to be ignored. To restore the previous behavior do a `cobbler system remove` on the `default` system.

```
cobbler system add --name=default --profile=boot_this
cobbler system remove --name=default
```

As mentioned in earlier sections, it is also possible to control the default behavior for a specific network:

```
cobbler system add --name=network1 --ip-address=192.168.0.0/24 --profile=boot_this
```

6.25.3 PXE boot loop prevention

If you have your machines set to PXE first in the boot order (ahead of hard drives), change the `pxe_just_once` flag in `/etc/cobbler/settings.yaml` to 1. This will set the machines to not PXE on successive boots once they complete one install. To re-enable PXE for a specific system, run the following command:

```
cobbler system edit --name=name --netboot-enabled=1
```

6.25.4 Automatic installation tracking

Cobbler knows how to keep track of the status of automatic installation of machines.

```
cobbler status
```

Using the status command will show when Cobbler thinks a machine started automatic installation and when it finished, provided the proper snippets are found in the automatic installation template. This is a good way to track machines that may have gone interactive (or stalled/crashed) during automatic installation.

6.26 Containerization

We have a test-image which you can find in the Cobbler repository and an old image made by the community: https://github.com/osism/docker-cobbler

6.27 Web-Interface

Please be patient until we have time with the 4.0.0 release to create a new web UI. The old Django based was preventing needed change inside the internals in Cobbler.

CHAPTER

SEVEN

DEVELOPER GUIDE

Our project lives on GitHub! Please visit our wiki there to get familiar with developer specific instructions: GitHub Cobbler Wiki

CHAPTER

EIGHT

COBBLER PACKAGE

8.1 Subpackages

8.1.1 cobbler.actions package

Subpackages

cobbler.actions.buildiso package

Submodules

cobbler.actions.buildiso.netboot module

This module contains the specific code to generate a network bootable ISO.

class cobbler.actions.buildiso.netboot.**AppendLineBuilder**(*distro_name: str*, *data: Dict[str, Any]*)

Bases: object

This class is meant to be initiated for a single append line. Afterwards the object should be disposed.

generate_profile(*distro_breed: str*, *os_version: str*, *protocol: str = 'http'*) → str

Generate the append line for the kernel for a network installation. :param distro_breed: The name of the distribution breed. :param os_version: The OS version of the distribution. :param protocol: The scheme that is used to read the autoyast file from the server :return: The generated append line.

generate_system(*dist:* Distro, *system:* System, *exclude_dns: bool*, *scheme: str = 'http'*) → str

Generate the append-line for a net-booting system.

Parameters

- **dist** – The distribution associated with the system.
- **system** – The system itself
- **exclude_dns** – Whether to include the DNS config or not.
- **scheme** – The scheme that is used to read the autoyast file from the server

class cobbler.actions.buildiso.netboot.**NetbootBuildiso**(*api:* CobblerAPI)

Bases: *BuildIso*

This class contains all functionality related to building network installation images.

filter_systems(*selected_items: Optional[List[str]] = None*) → List[Any]

Return a list of valid system objects selected from all systems by name, or everything if `selected_items` is empty.

Parameters
selected_items – A list of names to include in the returned list.

Returns
A list of valid systems. If an error occurred this is logged and an empty list is returned.

make_shorter(*distname: str*) → str

Return a short distro identifier which is basically an internal counter which is mapped via the real distro name.

Parameters
distname – The distro name to return an identifier for.

Returns
A short distro identifier

run(*iso: str = 'autoinst.iso'*, *buildisodir: str = ''*, *profiles: Optional[List[str]] = None*, *xorrisofs_opts: str = ''*, *distro_name: str = ''*, *systems: Optional[List[str]] = None*, *exclude_dns: bool = False*, ***kwargs: Any*)

Generate a net-installer for a distribution.

By default, the ISO includes all available systems and profiles. Specify `profiles` and `systems` to only include the selected systems and profiles. Both parameters can be provided at the same time.

Parameters

- **iso** – The name of the iso. Defaults to "autoinst.iso".
- **buildisodir** – This overwrites the directory from the settings in which the iso is built in.
- **profiles** – The filter to generate the ISO only for selected profiles.
- **xorrisofs_opts** – `xorrisofs` options to include additionally.
- **distro_name** – For detecting the architecture of the ISO.
- **systems** – Don't use that when building standalone ISOs. The filter to generate the ISO only for selected systems.
- **exclude_dns** – Whether the repositories have to be locally available or the internet is reachable.

cobbler.actions.buildiso.standalone module

This module contains the specific code for generating standalone or airgapped ISOs.

class cobbler.actions.buildiso.standalone.**StandaloneBuildiso**(*api:* CobblerAPI)

Bases: *BuildIso*

This class contains all functionality related to building self-contained installation images.

run(*iso: str = 'autoinst.iso'*, *buildisodir: str = ''*, *profiles: Optional[List[str]] = None*, *xorrisofs_opts: str = ''*, *distro_name: str = ''*, *airgapped: bool = False*, *source: str = ''*, ***kwargs: Any*)

Run the whole iso generation from bottom to top. Per default this builds an ISO for all available systems and profiles. This is the only method which should be called from non-class members. The `profiles` and `system` parameters can be combined. :param iso: The name of the iso. Defaults to "autoinst.iso". :param buildisodir: This overwrites the directory from the settings in which the iso is built in. :param profiles: The filter to generate the ISO only for selected profiles. :param xorrisofs_opts: `xorrisofs` options to include additionally. :param distro_name: For detecting the architecture of the ISO. :param airgapped: This option implies `standalone=True`. :param source: If the iso should be offline available this is the path to the sources of the image.

validate_repos(*profile_name: str*, *repo_names: List[str]*, *repo_mirrordir: Path*)

Sanity checks for repos to sync.

This function checks that repos are known to cobbler and have a local mirror directory. Raises ValueError if any repo fails the validation.

Module contents

Builds bootable CD images that have PXE-equivalent behavior for all Cobbler distros/profiles/systems currently in memory.

class cobbler.actions.buildiso.**Autoinstall**(*config*, *repos*)

Bases: NamedTuple

config: str

Alias for field number 0

repos: List[str]

Alias for field number 1

class cobbler.actions.buildiso.**BootFilesCopyset**(*src_kernel*, *src_initrd*, *new_filename*)

Bases: NamedTuple

new_filename: str

Alias for field number 2

src_initrd: str

Alias for field number 1

src_kernel: str

Alias for field number 0

class cobbler.actions.buildiso.**BuildIso**(*api: CobblerAPI*)

Bases: object

Handles conversion of internal state to the isolinux tree layout

calculate_grub_name(*desired_arch: Archs*) → str

This function checks the bootloaders_formats in our settings and then checks if there is a match between the architectures and the distribution architecture. :param distro: The distribution to get the GRUB2 loader name for.

create_buildiso_dirs_ppc64le(*buildiso_root: str*) → *BuildisoDirsPPC64LE*

Create directories in the buildiso root.

Layout: . ├── autoinstall ├── boot ├── ppc └── repo_mirror

create_buildiso_dirs_x86_64(*buildiso_root: str*) → *BuildisoDirsX86_64*

Create directories in the buildiso root.

Layout: . ├── autoinstall ├── EFI │ └── BOOT ├── isolinux └── repo_mirror

filter_items(*all_objs: Collection[ITEM]*, *selected_items: List[str]*) → List[ITEM]

Return a list of valid profile or system objects selected from all profiles or systems by name, or everything if selected_items is empty.

Parameters

- **all_objs** – The collection of items to filter.
- **selected_items** – The list of names

Raises
ValueError – Second option that this error is raised when the list of filtered systems or profiles is empty.

Returns
A list of valid profiles OR systems. If an error occurred this is logged and an empty list is returned.

filter_profiles(*selected_items: Optional[List[str]] = None*) → List[*Profile*]

Return a list of valid profile objects selected from all profiles by name, or everything if `selected_items` is empty. :param selected_items: A list of names to include in the returned list. :return: A list of valid profiles. If an error occurred this is logged and an empty list is returned.

parse_distro(*distro_name: str*) → *Distro*

Find and return distro object.

Parameters
distro_name – Name of the distribution to parse.

Raises
ValueError – If the distro is not found.

parse_profiles(*profiles: Optional[List[str]]*, *distro_obj:* Distro) → List[*Profile*]

TODO

Parameters

- **profiles** – TODO
- **distro_obj** – TODO

class cobbler.actions.buildiso.**BuildisoDirsPPC64LE**(*root*, *grub*, *ppc*, *autoinstall*, *repo*)

Bases: NamedTuple

autoinstall: Path

Alias for field number 3

grub: Path

Alias for field number 1

ppc: Path

Alias for field number 2

repo: Path

Alias for field number 4

root: Path

Alias for field number 0

class cobbler.actions.buildiso.**BuildisoDirsX86_64**(*root*, *isolinux*, *grub*, *autoinstall*, *repo*)

Bases: NamedTuple

autoinstall: Path

Alias for field number 3

grub: Path

Alias for field number 2

isolinux: Path

Alias for field number 1

repo: Path

Alias for field number 4

root: Path

Alias for field number 0

class cobbler.actions.buildiso.**LoaderCfgsParts**(*isolinux*, *grub*, *bootfiles_copysets*)

Bases: NamedTuple

bootfiles_copysets: List[*BootFilesCopyset*]

Alias for field number 2

grub: List[str]

Alias for field number 1

isolinux: List[str]

Alias for field number 0

cobbler.actions.buildiso.**add_remaining_kopts**(*kopts: Dict[str, Union[str, List[str]]]*) → str

Add remaining kernel_options to append_line :param kopts: The kernel options which are not present in append_line. :return: A single line with all kernel options from the dictionary in the string. Starts with a space.

Submodules

cobbler.actions.acl module

Configures acls for various users/groups so they can access the Cobbler command line as non-root. Now that CLI is largely remoted (XMLRPC) this is largely just useful for not having to log in (access to shared-secret) file but also grants access to hand-edit various cobbler_collections files and other useful things.

class cobbler.actions.acl.**AclConfig**(*api:* CobblerAPI)

Bases: object

TODO

modacl(*isadd: bool*, *isuser: bool*, *who: str*) → None

Modify the acls for Cobbler on the filesystem.

Parameters

- **isadd** – If true then the who will be added. If false then who will be removed.
- **isuser** – If true then the who may be a user. If false then who may be a group.
- **who** – The user or group to be added or removed.

run(*adduser: Optional[str] = None*, *addgroup: Optional[str] = None*, *removeuser: Optional[str] = None*, *removegroup: Optional[str] = None*) → None

Automate setfacl commands. Only one of the four may be specified but one option also must be specified.

Parameters

- **adduser** – Add a user to be able to manage Cobbler.
- **addgroup** – Add a group to be able to manage Cobbler.
- **removeuser** – Remove a user to be able to manage Cobbler.
- **removegroup** – Remove a group to be able to manage Cobbler.

Raises

CX – Raised in case not enough arguments are specified.

cobbler.actions.check module

Cobbler Trigger Module that checks against a list of hardcoded potential common errors in a Cobbler installation.

class cobbler.actions.check.**CobblerCheck**(*api:* CobblerAPI)

Bases: object

Validates whether the system is reasonably well configured for serving up content. This is the code behind 'cobbler check'.

static check_bind_bin(*status: List[str]*) → None

Check if bind is installed.

Parameters
status – The status list with possible problems.

static check_bootloaders(*status: List[str]*) → None

Check if network bootloaders are installed

Parameters
status – The status list with possible problems.

check_ctftpd_dir(*status: List[str]*) → None

Check if `cobbler.conf`'s tftpboot directory exists.

Parameters
status – The status list with possible problems.

check_debmirror(*status: List[str]*) → None

Check if debmirror is available and the config file for it exists. If the distro family is suse then this will pass without checking.

Parameters
status – The status list with possible problems.

static check_dhcpd_bin(*status: List[str]*) → None

Check if dhcpd is installed.

Parameters
status – The status list with possible problems.

check_dhcpd_conf(*status: List[str]*) → None

NOTE: this code only applies if Cobbler is *NOT* set to generate a `dhcp.conf` file.

Check that dhcpd *appears* to be configured for pxe booting. We can't assure file correctness. Since a Cobbler user might have dhcp on another server, it's okay if it's not there and/or not configured correctly according to automated scans.

Parameters
status – The status list with possible problems.

static check_dnsmasq_bin(*status: List[str]*) → None

Check if dnsmasq is installed.

Parameters
status – The status list with possible problems.

static check_for_cman(*status: List[str]*) → None

Check if the fence agents are available. This is done through checking if the binary `fence_ilo` is present in `/sbin` or `/usr/sbin`.

Parameters
status – The status list with possible problems. The status list with possible problems.

check_for_default_password(*status: List[str]*) → None

Check if the default password of Cobbler was changed.

Parameters
status – The status list with possible problems.

check_for_ksvalidator(*status: List[str]*) → None

Check if the `ksvalidator` is present in `/usr/bin`.

Parameters
status – The status list with possible problems. The status list with possible problems.

check_for_unreferenced_repos(*status: List[str]*) → None

Check if there are repositories which are not used and thus could be removed.

Parameters
status – The status list with possible problems.

check_for_unsynced_repos(*status: List[str]*) → None

Check if there are unsynchronized repositories which need an update.

Parameters
status – The status list with possible problems.

static check_for_wget_curl(*status: List[str]*) → None

Check to make sure wget or curl is installed

Parameters
status – The status list with possible problems.

check_iptables(*status: List[str]*) → None

Check if iptables is running. If yes print the needed ports. This is unavailable on Debian, SUSE and CentOS7 as a service. However this only indicates that the way of persisting the iptable rules are persisted via other means.

Parameters
status – The status list with possible problems.

check_name(*status: List[str]*) → None

If the server name in the config file is still set to localhost automatic installations run from koan will not have proper kernel line parameters.

Parameters
status – The status list with possible problems.

check_rsync_conf(*status: List[str]*) → None

Check that rsync is enabled to autostart.

Parameters
status – The status list with possible problems.

check_selinux(*status: List[str]*) → None

Suggests various SELinux rules changes to run Cobbler happily with SELinux in enforcing mode.

Parameters
status – The status list with possible problems.

check_service(*status: List[str]*, *which: str*, *notes: str = ''*) → None

Check if the service command is available or the old init.d system has to be used.

Parameters

- **status** – The status list with possible problems.
- **which** – The service to check for.
- **notes** – A manual not to attach.

check_tftpd_dir(*status: List[str]*) → None

Check if cobbler.conf's tftpboot directory exists

Parameters
status – The status list with possible problems.

check_yum(*status: List[str]*) → None

Check if the yum-stack is available. On Debian based distros this will always return without checking.

Parameters
status – The status list with possible problems.

run() → List[str]

The CLI usage is "cobbler check" before "cobbler sync".

Returns
None if there are no errors, otherwise returns a list of things to correct prior to running application 'for real'.

cobbler.actions.hardlink module

Hard links Cobbler content together to save space.

class cobbler.actions.hardlink.**HardLinker**(*api: CobblerAPI*)

Bases: object

TODO

run() → int

Simply hardlinks directories that are Cobbler managed.

cobbler.actions.importer module

This module contains the logic that kicks of the `cobbler import` process. This is extracted logic from `api.py` that is essentially calling `modules/mangers/import_signatures.py` with some preparatory code.

class cobbler.actions.importer.**Importer**(*api: CobblerAPI*)

Bases: object

Wrapper class to adhere to the style of all other actions.

run(*mirror_url: str*, *mirror_name: str*, *network_root: Optional[str] = None*, *autoinstall_file: Optional[str] = None*, *rsync_flags: Optional[str] = None*, *arch: Optional[str] = None*, *breed: Optional[str] = None*, *os_version: Optional[str] = None*) → bool

Automatically import a directory tree full of distribution files.

Parameters

- **mirror_url** – Can be a string that represents a path, a user@host syntax for SSH, or an rsync:// address. If mirror_url is a filesystem path and mirroring is not desired, set network_root to something like "nfs://path/to/mirror_url/root"
- **mirror_name** – The name of the mirror.
- **network_root** – the remote path (nfs/http/ftp) for the distro files
- **autoinstall_file** – user-specified response file, which will override the default
- **rsync_flags** – Additional flags that will be passed to the rsync call that will sync everything to the Cobbler webroot.
- **arch** – user-specified architecture
- **breed** – user-specified breed

- **os_version** – user-specified OS version

cobbler.actions.log module

Cobbler Trigger Module that managed the logs associated with a Cobbler system.

class cobbler.actions.log.**LogTool**(*system:* System, *api:* CobblerAPI)

Bases: object

Helpers for dealing with System logs, anamon, etc..

clear() → None

Clears the system logs

cobbler.actions.mkloaders module

Cobbler action to create bootable Grub2 images.

This action calls grub2-mkimage for all bootloader formats configured in Cobbler's settings. See man(1) grub2-mkimage for available formats.

class cobbler.actions.mkloaders.**MkLoaders**(*api:* CobblerAPI)

Bases: object

Action to create bootloader images.

create_directories() → None

Create the required directories so that this succeeds. If existing, do nothing. This should create the tree for all supported bootloaders, regardless of the capabilities to symlink/install/build them.

make_grub() → None

Create symlink of the GRUB 2 bootloader in case it is available on the system. Additionally build the loaders for other architectures if the modules to do so are available.

make_ipxe() → None

Create symlink of the iPXE bootloader in case it is available on the system.

make_shim() → None

Create symlink of the shim bootloader in case it is available on the system.

make_syslinux() → None

Create symlink of the important syslinux bootloader files in case they are available on the system.

run() → None

Run GrubImages action. If the files or executables for the bootloader is not available we bail out and skip the creation after it is logged that this is not available.

cobbler.actions.mkloaders.**find_file**(*glob_path: Path*, *file_regex: Pattern[str]*) → Optional[Path]

Given a path glob and a file regex, return a full path of the file.

Param
glob_path: Glob of a path, e.g. Path('/var/*/rhn')

Param
file_regex: A regex for a filename in the path

Returns
The full file path or None if no file was found

cobbler.actions.mkloaders.**get_syslinux_version**() → int

This calls syslinux and asks for the version number.

Returns
The major syslinux release number.

Raises
subprocess.CalledProcessError – Error raised by `subprocess.run` in case syslinux does not return zero.

cobbler.actions.mkloaders.**mkimage**(*image_format: str*, *image_filename: Path*, *modules: List[str]*) → None

Create a bootable image of GRUB using grub2-mkimage.

Parameters

- **image_format** – Format of the image that is being created. See man(1) grub2-mkimage for a list of supported formats.
- **image_filename** – Location of the image that is being created.
- **modules** – List of GRUB modules to include into the image

Raises
subprocess.CalledProcessError – Error raised by `subprocess.run`.

cobbler.actions.mkloaders.**symlink**(*target: Path*, *link: Path*, *skip_existing: bool = False*) → None

Create a symlink LINK pointing to TARGET.

Parameters

- **target** – File/directory that the link will point to. The file/directory must exist.
- **link** – Filename for the link.
- **skip_existing** – Controls if existing links are skipped, defaults to False.

Raises

- **FileNotFoundError** – `target` is not an existing file.
- **FileExistsError** – `skip_existing` is False and `link` already exists.

cobbler.actions.replicate module

Replicate from a Cobbler master.

class cobbler.actions.replicate.**Replicate**(*api: CobblerAPI*)

Bases: `object`

This class contains the magic to replicate a Cobbler instance to another Cobbler instance.

add_objects_not_on_local(*obj_type: str*) → None

Add objects locally which are not present on the slave but on the master.

Parameters
obj_type –

generate_include_map() → None

Method that generates the information that is required to perform the replicate option.

link_distros() → None

Link a distro from its location into the web directory to make it available for usage.

remove_objects_not_on_master(*obj_type: str*) → None

Remove objects on this slave which are not on the master.

Parameters
obj_type – The type of object which should be synchronized.

replace_objects_newer_on_remote(*obj_type: str*) → None

Replace objects which are newer on the local slave then on the remote slave

Parameters
obj_type – The type of object to synchronize.

replicate_data() → None

Replicate the local and remote data to each another.

rsync_it(*from_path: str*, *to_path: str*, *object_type: Optional[str] = None*) → None

Rsync from a source to a destination with the rsync options Cobbler was configured with.

Parameters

- **from_path** – The source to rsync from.
- **to_path** – The destination to rsync to.
- **object_type** – If set to "repo" this will take the repo rsync options instead of the global ones.

run(*cobbler_master: Optional[str] = None*, *port: str = '80'*, *distro_patterns: Optional[str] = None*, *profile_patterns: Optional[str] = None*, *system_patterns: Optional[str] = None*, *repo_patterns: Optional[str] = None*, *image_patterns: Optional[str] = None*, *prune: bool = False*, *omit_data: bool = False*, *sync_all: bool = False*, *use_ssl: bool = False*) → None

Get remote profiles and distros and sync them locally

Parameters

- **cobbler_master** – The remote url of the master server.
- **port** – The remote port of the master server.
- **distro_patterns** – The pattern of distros to sync.
- **profile_patterns** – The pattern of profiles to sync.
- **system_patterns** – The pattern of systems to sync.
- **repo_patterns** – The pattern of repositories to sync.
- **image_patterns** – The pattern of images to sync.
- **prune** – If the local server should be pruned before coping stuff.
- **omit_data** – If the data behind images etc should be omitted or not.
- **sync_all** – If everything should be synced (then the patterns are useless) or not.
- **use_ssl** – If HTTPS or HTTP should be used.

cobbler.actions.report module

Report from a Cobbler master. FIXME: reinstante functionality for 2.0

class cobbler.actions.report.**Report**(*api:* CobblerAPI)

Bases: object

TODO

fielder(*structure: Dict[str, Any]*, *fields_list: List[str]*) → Dict[str, str]

Return data from a subset of fields of some item

Parameters

- **structure** – The item structure to report.
- **fields_list** – The list of fields which should be returned.

Returns

The same item with only the given subset of information.

print_formatted_data(*data: List[Dict[str, str]]*, *order: List[str]*, *report_type: str*, *noheaders: bool*) → None

Used for picking the correct format to output data as

Parameters

- **data** – The list of iteratable items for table output.
- **order** – The list of fields which are available in the table file.
- **noheaders** – Whether headers are printed to the output or not.
- **report_type** – The type of report which should be used.

reporting_csv(*info: List[Dict[str, str]]*, *order: List[Any]*, *noheaders: bool*) → str

Formats data on 'info' for csv output

Parameters

- **info** – The list of iteratable items for csv output.
- **order** – The list of fields which are available in the csv file.
- **noheaders** – Whether headers are printed to the output or not.

Returns

The string with the csv.

reporting_doku(*info: List[Dict[str, str]]*, *order: List[Any]*, *noheaders: bool*) → str

Formats data on 'info' for doku wiki table output

Parameters

- **info** – The list of iteratable items for table output.
- **order** – The list of fields which are available in the table file.
- **noheaders** – Whether headers are printed to the output or not.

Returns

The string with the generated table.

static reporting_list_names2(*collection:* Collection*[ITEM]*, *name: str*) → None

Prints a specific object in a collection.

Parameters

- **collection** – The collections object to print a collection from.
- **name** – The name of the collection to print.

reporting_mediawiki(*info: List[Dict[str, str]]*, *order: List[Any]*, *noheaders: bool*) → str

Formats data on 'info' for mediawiki table output

Parameters

- **info** – The list of iteratable items for table output.
- **order** – The list of fields which are available in the table file.
- **noheaders** – Whether headers are printed to the output or not.

Returns

The string with the generated table.

reporting_print_all_fields(*collection: Collection[ITEM]*, *report_name: str*, *report_type: str*, *report_noheaders: bool*) → None

Prints all fields in a collection as a table given the report type

Parameters

- **collection** – The collection to report.
- **report_name** – The name of the report.
- **report_type** – The type of report to give.
- **report_noheaders** – Report without the headers. (May be useful for machine parsing)

Returns

A report with all fields included pretty printed or machine readable.

static reporting_print_sorted(*collection: Collection[ITEM]*) → None

Prints all objects in a collection sorted by name

Parameters

collection – The collection to print.

reporting_print_x_fields(*collection: Collection[ITEM]*, *report_name: str*, *report_type: str*, *report_fields: str*, *report_noheaders: bool*) → None

Prints specific fields in a collection as a table given the report type

Parameters

- **collection** – The collection to report.
- **report_name** – The name of the report.
- **report_type** – The type of report to give.
- **report_fields** – The fields which should be included in the report.
- **report_noheaders** – Report without the headers. (May be useful for machine parsing)

reporting_trac(*info: List[Dict[str, str]]*, *order: List[Any]*, *noheaders: bool*) → str

Formats data on 'info' for trac wiki table output

Parameters

- **info** – The list of iteratable items for table output.
- **order** – The list of fields which are available in the table file.
- **noheaders** – Whether headers are printed to the output or not.

Returns

The string with the generated table.

run(*report_what: str = ''*, *report_name: str = ''*, *report_type: str = ''*, *report_fields: str = ''*, *report_noheaders: bool = False*) → None

Get remote profiles and distros and sync them locally

1. Handles original report output
2. Handles all fields of report outputs as table given a format
3. Handles specific fields of report outputs as table given a format

Parameters

- **report_what** – What should be reported. May be "all".
- **report_name** – The name of the report.
- **report_type** – The type of report to give.
- **report_fields** – The fields which should be included in the report.
- **report_noheaders** – Report without the headers. (May be useful for machine parsing)

cobbler.actions.reposync module

Builds out and synchronizes yum repo mirrors. Initial support for rsync, perhaps reposync coming later.

class cobbler.actions.reposync.**RepoSync**(*api: CobblerAPI*, *tries: int = 1*, *nofail: bool = False*)

Bases: object

Handles conversion of internal state to the tftpboot tree layout.

apt_sync(*repo: Repo*) → None

Handle copying of http:// and ftp:// debian repos.

Parameters

repo – The apt repository to sync.

create_local_file(*dest_path: str*, *repo: Repo*, *output: bool = True*) → str

Creates Yum config files for use by reposync

Two uses: (A) output=True, Create local files that can be used with yum on provisioned clients to make use of this mirror. (B) output=False, Create a temporary file for yum to feed into yum for mirroring

Parameters

- **dest_path** – The destination path to create the file at.
- **repo** – The repository object to create a file for.
- **output** – See described above.

Returns

The name of the file which was written.

createrepo_walker(*repo: Repo*, *dirname: str*, *fnames: Any*) → None

Used to run createrepo on a copied Yum mirror.

Parameters

- **repo** – The repository object to run for.
- **dirname** – The directory to run in.
- **fnames** – Not known what this is for.

gen_urlgrab_ssl_opts(*yumopts: Dict[str, Any]*) → Tuple[Optional[Tuple[Any, ...]], bool]

This function translates yum repository options into the appropriate options for python-requests

Parameters
yumopts – The options to convert.

Returns
A tuple with the cert and a boolean if it should be verified or not.

librepo_getinfo(*dirname: str*) → Dict[Any, Any]

Used to get records from a repomd.xml file of downloaded rpmmd repository.

Parameters
dirname – The local path of rpmmd repository.

Returns
The dict representing records from a repomd.xml file of rpmmd repository.

static reposync_cmd() → List[str]

Determine reposync command

Returns
The path to the reposync command. If dnf exists it is used instead of reposync.

rhn_sync(*repo:* Repo) → None

Handle mirroring of RHN repos.

Parameters
repo – The repo object to synchronize.

rsync_sync(*repo:* Repo) → None

Handle copying of rsync:// and rsync-over-ssh repos.

Parameters
repo – The repo to sync via rsync.

run(*name: Optional[str] = None*, *verbose: bool = True*) → None

Syncs the current repo configuration file with the filesystem.

Parameters

- **name** – The name of the repository to synchronize.
- **verbose** – If the action should be logged verbose or not.

sync(*repo:* Repo) → None

Conditionally sync a repo, based on type.

Parameters
repo – The repo to sync.

update_permissions(*repo_path: str*) → None

Verifies that permissions and contexts after an rsync are as expected. Sending proper rsync flags should prevent the need for this, though this is largely a safeguard.

Parameters
repo_path – The path to update the permissions of.

wget_sync(*repo:* Repo) → None

Handle mirroring of directories using wget

Parameters
repo – The repo object to sync via wget.

yum_sync(*repo:* Repo) → None

Handle copying of http:// and ftp:// yum repos.

Parameters

repo – The yum reporitory to sync.

cobbler.actions.reposync.**repo_walker**(*top: str*, *func: Callable[[Any, str, List[str]], None]*, *arg: Any*) → None

Directory tree walk with callback function.

For each directory in the directory tree rooted at top (including top itself, but excluding '.' and '..'), call func(arg, dirname, fnames). dirname is the name of the directory, and fnames a list of the names of the files and subdirectories in dirname (excluding '.' and '..'). func may modify the fnames list in-place (e.g. via del or slice assignment), and walk will only recurse into the subdirectories whose names remain in fnames; this can be used to implement a filter, or to impose a specific order of visiting. No semantics are defined for, or required of, arg, beyond that arg is always passed to func. It can be used, e.g., to pass a filename pattern, or a mutable object designed to accumulate statistics. Passing None for arg is common.

Parameters

- **top** – The directory that should be taken as root. The root dir will also be included in the processing.
- **func** – The function that should be executed.
- **arg** – The arguments for that function.

cobbler.actions.status module

Reports on automatic installation activity by examining the logs in /var/log/cobbler.

class cobbler.actions.status.**CobblerStatusReport**(*api:* CobblerAPI, *mode: str*)

Bases: object

TODO

catalog(*profile_or_system: str*, *name: str*, *ip_address: str*, *start_or_stop: str*, *timestamp: float*) → None

Add a system to `cobbler status`.

Parameters

- **profile_or_system** – This can be `system` or `profile`.
- **name** – The name of the object.
- **ip_address** – The ip of the system to watch.
- **start_or_stop** – This parameter may be `start` or `stop`
- **timestamp** – Timestamp as returned by `time.time()`

static collect_logfiles() → List[str]

Collects all installation logfiles from `/var/log/cobbler/`. This will also collect gzipped logfiles.

Returns

List of absolute paths that are matching the filepattern `install.log` or `install.log.x`, where x is a number equal or greater than zero.

get_printable_results() → str

Convert the status of Cobbler from a machine-readable form to human-readable.

Returns

A nice formatted representation of the results of `cobbler status`.

process_results() → Dict[Any, Any]

Look through all systems which were collected and update the status.

Returns
Return `ip_data` of the object.

run() → Union[Dict[Any, Any], str]

Calculate and print a automatic installation status report.

scan_logfiles() → None

Scan the installation log-files - starting with the oldest file.

class cobbler.actions.status.**InstallStatus**

Bases: `object`

Helper class that represents the current state of the installation of a system or profile.

cobbler.actions.sync module

Builds out filesystem trees/data based on the object tree. This is the code behind 'cobbler sync'.

class cobbler.actions.sync.**CobblerSync**(*api:* CobblerAPI, *verbose: bool = True*, *dhcp: Optional[DhcpManagerModule] = None*, *dns: Optional[DnsManagerModule] = None*, *tftpd: Optional[TftpManagerModule] = None*)

Bases: `object`

Handles conversion of internal state to the tftpboot tree layout

add_single_distro(*distro_obj:* Distro) → None

Sync adding a single distro.

Parameters
name – The name of the distribution.

add_single_image(*image_obj:* Image) → None

Sync adding a single image.

Parameters
name – The name of the image.

add_single_profile(*profile:* Profile, *rebuild_menu: bool = True*) → Optional[bool]

Sync adding a single profile.

Parameters

- **name** – The name of the profile.
- **rebuild_menu** – Whether to rebuild the grub/... menu or not.

Returns
`True` if this succeeded.

add_single_system(*system_obj:* System) → None

Sync adding a single system.

Parameters
name – The name of the system.

clean_link_cache()

All files which are linked into the cache will be deleted so the cache can be rebuild.

clean_trees()

Delete any previously built pxelinux.cfg tree and virt tree info and then create directories.

Note: for SELinux reasons, some information goes in `/tftpboot`, some in `/var/www/cobbler` and some must be duplicated in both. This is because PXE needs tftp, and automatic installation and Virt operations need http. Only the kernel and initrd images are duplicated, which is unfortunate, though SELinux won't let me give them two contexts, so symlinks are not a solution. *Otherwise* duplication is minimal.

remove_single_distro(*distro_obj:* Distro) → None

Sync removing a single distro.

Parameters
name – The name of the distribution.

remove_single_image(*image_obj:* Image) → None

Sync removing a single image.

Parameters
image_obj – The name of the image.

remove_single_menu(*rebuild_menu: bool = True*) → None

Sync removing a single menu.

Parameters
rebuild_menu – Whether to rebuild the grub/... menu or not.

remove_single_profile(*profile_obj:* Profile, *rebuild_menu: bool = True*) → None

Sync removing a single profile.

Parameters

- **name** – The name of the profile.
- **rebuild_menu** – Whether to rebuild the grub/... menu or not.

remove_single_system(*system_obj:* System) → None

Sync removing a single system.

Parameters
name – The name of the system.

rsync_gen() → None

Generate rsync modules of all repositories and distributions

Raises
OSError –

run() → None

Syncs the current configuration file with the config tree. Using the `Check().run_` functions previously is recommended

run_sync_systems(*systems: List[str]*)

Syncs the specific systems with the config tree.

sync_dhcp()

This calls write_dhcp and restarts the DHCP server.

update_system_netboot_status(*name: str*) → None

Update the netboot status of a system.

Parameters
name – The name of the system.

write_dhcp()

Write all files which are associated to DHCP.

Module contents

The action module is responsible for containing one Python module for each action which Cobbler offers. The code should never be dependent on another module or on other parts. An action should request the exact data it requires and nothing more.

8.1.2 cobbler.cobbler_collections package

Submodules

cobbler.cobbler_collections.collection module

This module contains the code for the abstract base collection that powers all the other collections.

class cobbler.cobbler_collections.collection.**Collection**(*collection_mgr:* CollectionManager)

Bases: Generic[ITEM]

Base class for any serializable list of things.

SEARCH_REKEY = {'boot_loader': 'boot_loaders', 'dhcp-tag': 'dhcp_tag', 'enable_gpxe': 'enable_ipxe', 'inherit': 'parent', 'ip': 'ip_address', 'kopts': 'kernel_options', 'kopts_post': 'kernel_options_post', 'mac': 'mac_address', 'netboot-enabled': 'netboot_enabled', 'virt-auto-boot': 'virt_auto_boot', 'virt-bridge': 'virt_bridge', 'virt-cpus': 'virt_cpus', 'virt-disk-driver': 'virt_disk_driver', 'virt-file-size': 'virt_file_size', 'virt-group': 'virt_group', 'virt-host': 'virt_host', 'virt-path': 'virt_path', 'virt-ram': 'virt_ram', 'virt-type': 'virt_type'}

add(*ref: ITEM*, *save: bool = False*, *with_copy: bool = False*, *with_triggers: bool = True*, *with_sync: bool = True*, *quick_pxe_update: bool = False*, *check_for_duplicate_names: bool = False*) → None

Add an object to the collection

Parameters

- **ref** – The reference to the object.
- **save** – If this is true then the objet is persisted on the disk.
- **with_copy** – Is a bit of a misnomer, but lots of internal add operations can run with "with_copy" as False. True means a real final commit, as if entered from the command line (or basically, by a user). With with_copy as False, the particular add call might just be being run during deserialization, in which case extra semantics around the add don't really apply. So, in that case, don't run any triggers and don't deal with any actual files.
- **with_sync** – If a sync should be triggered when the object is renamed.
- **with_triggers** – If triggers should be run when the object is added.
- **quick_pxe_update** – This decides if there should be run a quick or full update after the add was done.
- **check_for_duplicate_names** – If the name of an object should be unique or not.

Raises

- **TypError** – Raised in case `ref` is None.
- **ValueError** – Raised in case the name of `ref` is empty.

abstract static collection_type() → str

Returns the string key for the name of the collection (used by serializer etc)

abstract static collection_types() → str

Returns the string key for the plural name of the collection (used by serializer)

copy(*ref: ITEM*, *newname: str*)

Copy an object with a new name into the same collection.

Parameters

- **ref** – The reference to the object which should be copied.
- **newname** – The new name for the copied object.

property deserialize_running: bool

If set to `true`, then the collection items are currently being loaded from disk.

Getter
The deserialize_running for the collection.

Setter
The new deserialize_running value for the collection.

abstract factory_produce(*api: CobblerAPI*, *seed_data: Dict[str, Any]*) → ITEM

Must override in subclass. Factory_produce returns an Item object from dict.

Parameters

- **api** – The API to resolve all information with.
- **seed_data** – Unused Parameter in the base collection.

find(*name: str = ''*, *return_list: bool = False*, *no_errors: bool = False*, ***kargs: Union[str, int, bool, Dict[Any, Any], List[Any]]*) → Optional[Union[List[ITEM], ITEM]]

Return first object in the collection that matches all item='value' pairs passed, else return None if no objects can be found. When return_list is set, can also return a list. Empty list would be returned instead of None in that case.

Parameters

- **name** – The object name which should be found.
- **return_list** – If a list should be returned or the first match.
- **no_errors** – If errors which are possibly thrown while searching should be ignored or not.
- **kargs** – If name is present, this is optional, otherwise this dict needs to have at least a key with `name`. You may specify more keys to finetune the search.

Returns
The first item or a list with all matches.

Raises
ValueError – In case no arguments for searching were specified.

from_list(*_list: List[Dict[str, Any]]*) → None

Create all collection object items from `_list`.

Parameters
_list – The list with all item dictionaries.

get(*name: str*) → Optional[*Item*]

Return object with name in the collection

Parameters
name – The name of the object to retrieve from the collection.

Returns
The object if it exists. Otherwise, "None".

`get_names()` → List[str]

Return list of names in the collection.

Returns
list of names in the collection.

`property inmemory: bool`

If set to `true`, then all items of the collection are loaded into memory.

Getter
The inmemory for the collection.

Setter
The new inmemory value for the collection.

`property lite_sync: CobblerSync`

Provide a ready to use CobblerSync object.

Getter
Return the object that can update the filesystem state to a new one.

`abstract remove(name: str, with_delete: bool = True, with_sync: bool = True, with_triggers: bool = True, recursive: bool = False)` → None

Remove an item from collection. This method must be overridden in any subclass.

Parameters

- **name** – Item Name
- **with_delete** – sync and run triggers
- **with_sync** – sync to server file system
- **with_triggers** – run "on delete" triggers
- **recursive** – recursively delete children

Returns
NotImplementedError

`rename(ref: ITEM, newname: str, with_sync: bool = True, with_triggers: bool = True)`

Allows an object "ref" to be given a new name without affecting the rest of the object tree.

Parameters

- **ref** – The reference to the object which should be renamed.
- **newname** – The new name for the object.
- **with_sync** – If a sync should be triggered when the object is renamed.
- **with_triggers** – If triggers should be run when the object is renamed.

`to_list()` → List[Dict[str, Any]]

Serialize the collection

Returns
All elements of the collection as a list.

`to_string()` → str

Creates a printable representation of the collection suitable for reading by humans or parsing from scripts. Actually scripts would be better off reading the JSON in the cobbler_collections files directly.

Returns
The object as a string representation.

cobbler.cobbler_collections.distros module

Cobbler module that at runtime holds all distros in Cobbler.

class cobbler.cobbler_collections.distros.**Distros**(*collection_mgr:* CollectionManager)

Bases: *Collection*[*Distro*]

A distro represents a network bootable matched set of kernels and initrd files.

static collection_type() → str

Returns the string key for the name of the collection (used by serializer etc)

static collection_types() → str

Returns the string key for the plural name of the collection (used by serializer)

factory_produce(*api:* CobblerAPI, *seed_data: Dict[str, Any]*) → *distro.Distro*

Return a Distro forged from seed_data

Parameters

- **api** – Parameter is skipped.
- **seed_data** – Data to seed the object with.

Returns
The created object.

remove(*name: str*, *with_delete: bool = True*, *with_sync: bool = True*, *with_triggers: bool = True*, *recursive: bool = False*) → None

Remove element named 'name' from the collection

Raises
CX – In case any subitem (profiles or systems) would be orphaned. If the option `recursive` is set then the orphaned items would be removed automatically.

cobbler.cobbler_collections.images module

Cobbler module that at runtime holds all images in Cobbler.

class cobbler.cobbler_collections.images.**Images**(*collection_mgr:* CollectionManager)

Bases: *Collection*[*Image*]

A image instance represents a ISO or virt image we want to track and repeatedly install. It differs from a answer-file based installation.

static collection_type() → str

Returns the string key for the name of the collection (used by serializer etc)

static collection_types() → str

Returns the string key for the plural name of the collection (used by serializer)

factory_produce(*api:* CobblerAPI, *seed_data: Dict[str, Any]*)

Return a Distro forged from seed_data

Parameters

- **api** – Parameter is skipped.
- **seed_data** – Data to seed the object with.

Returns
The created object.

remove(*name: str*, *with_delete: bool = True*, *with_sync: bool = True*, *with_triggers: bool = True*, *recursive: bool = True*) → None

Remove element named 'name' from the collection

Raises
CX – In case object does not exist or it would orhan a system.

cobbler.cobbler_collections.manager module

Repository of the Cobbler object model

class cobbler.cobbler_collections.manager.**CollectionManager**(*api: CobblerAPI*)

Bases: object

Manages a definitive copy of all data cobbler_collections with weakrefs pointing back into the class so they can understand each other's contents.

deserialize() → None

Load all cobbler_collections from disk

Raises
CX – if there is an error in deserialization

deserialize_one_item(*obj: Item*) → Dict[str, Any]

Load a collection item from disk

Parameters
obj – collection item

distros() → *Distros*

Return the definitive copy of the Distros collection

get_items(*collection_type: str*) → Union[*Distros*, *Profiles*, *Systems*, *Repos*, *Images*, *Menus*, *Settings*]

Get a full collection of a single type.

Valid Values vor `collection_type` are: "distro", "profile", "repo", "image", "menu" and "settings".

Parameters
collection_type – The type of collection to return.

Returns
The collection if `collection_type` is valid.

Raises
CX – If the `collection_type` is invalid.

has_loaded = False

images() → *Images*

Return the definitive copy of the Images collection

menus() → *Menus*

Return the definitive copy of the Menus collection

profiles() → *Profiles*

Return the definitive copy of the Profiles collection

repos() → *Repos*

Return the definitive copy of the Repos collection

serialize() → None

Save all cobbler_collections to disk

serialize_delete(*collection: Collection[ITEM]*, *item: ITEM*) → None

Delete a collection item from disk

Parameters

- **collection** – collection
- **item** – collection item

serialize_delete_one_item(*item: ITEM*) → None

Save a collection item to disk

Parameters
item – collection item

serialize_item(*collection: Collection[ITEM]*, *item: ITEM*) → None

Save a collection item to disk

Deprecated - Use above serialize_one_item function instead collection param can be retrieved

Parameters

- **collection** – Collection
- **item** – collection item

serialize_one_item(*item: ITEM*) → None

Save a collection item to disk

Parameters
item – collection item

settings() → *Settings*

Return the definitive copy of the application settings

systems() → *Systems*

Return the definitive copy of the Systems collection

cobbler.cobbler_collections.menus module

Cobbler module that at runtime holds all menus in Cobbler.

class cobbler.cobbler_collections.menus.**Menus**(*collection_mgr: CollectionManager*)

Bases: *Collection*[*Menu*]

A menu represents an element of the hierarchical boot menu.

static collection_type() → str

Returns the string key for the name of the collection (used by serializer etc)

static collection_types() → str

Returns the string key for the plural name of the collection (used by serializer)

factory_produce(*api: CobblerAPI*, *seed_data: Dict[str, Any]*) → *Menu*

Return a Menu forged from seed_data

Parameters

- **api** – Parameter is skipped.
- **seed_data** – Data to seed the object with.

Returns
The created object.

remove(*name: str*, *with_delete: bool = True*, *with_sync: bool = True*, *with_triggers: bool = True*, *recursive: bool = False*) → None

Remove element named 'name' from the collection

Parameters

- **name** – The name of the menu
- **with_delete** – In case the deletion triggers are executed for this menu.
- **with_sync** – In case a Cobbler Sync should be executed after the action.
- **with_triggers** – In case the Cobbler Trigger mechanism should be executed.
- **recursive** – In case you want to delete all objects this menu references.

Raises

CX – Raised in case you want to delete a none existing menu.

cobbler.cobbler_collections.profiles module

Cobbler module that at runtime holds all profiles in Cobbler.

class cobbler.cobbler_collections.profiles.**Profiles**(*collection_mgr: CollectionManager*)

Bases: *Collection*[*Profile*]

A profile represents a distro paired with an automatic OS installation template file.

static collection_type() → str

Returns the string key for the name of the collection (used by serializer etc)

static collection_types() → str

Returns the string key for the plural name of the collection (used by serializer)

factory_produce(*api: CobblerAPI*, *seed_data: Dict[Any, Any]*)

Return a Distro forged from seed_data

remove(*name: str*, *with_delete: bool = True*, *with_sync: bool = True*, *with_triggers: bool = True*, *recursive: bool = False*)

Remove element named 'name' from the collection

Raises

CX – In case the name of the object was not given or any other descendant would be orphaned.

cobbler.cobbler_collections.repos module

Cobbler module that at runtime holds all repos in Cobbler.

class cobbler.cobbler_collections.repos.**Repos**(*collection_mgr: CollectionManager*)

Bases: *Collection*[*Repo*]

Repositories in Cobbler are way to create a local mirror of a yum repository. When used in conjunction with a mirrored distro tree (see "cobbler import"), outside bandwidth needs can be reduced and/or eliminated.

static collection_type() → str

Returns the string key for the name of the collection (used by serializer etc)

static collection_types() → str

Returns the string key for the plural name of the collection (used by serializer)

factory_produce(*api:* CobblerAPI, *seed_data:* *Dict[str, Any]*)

Return a Distro forged from seed_data

Parameters

- **api** – Parameter is skipped.
- **seed_data** – The data the object is initalized with.

Returns
The created repository.

remove(*name:* *str*, *with_delete:* *bool* = *True*, *with_sync:* *bool* = *True*, *with_triggers:* *bool* = *True*, *recursive:* *bool* = *False*)

Remove element named 'name' from the collection

Raises
CX – In case the object does not exist.

cobbler.cobbler_collections.systems module

Cobbler module that at runtime holds all systems in Cobbler.

class cobbler.cobbler_collections.systems.**Systems**(*collection_mgr:* CollectionManager)

Bases: *Collection*[*System*]

Systems are hostnames/MACs/IP names and the associated profile they belong to.

static collection_type() → str

Returns the string key for the name of the collection (used by serializer etc)

static collection_types() → str

Returns the string key for the plural name of the collection (used by serializer)

factory_produce(*api:* CobblerAPI, *seed_data:* *Dict[str, Any]*) → *System*

Return a Distro forged from seed_data

Parameters

- **api** – Parameter is skipped.
- **seed_data** – Data to seed the object with.

Returns
The created object.

remove(*name:* *str*, *with_delete:* *bool* = *True*, *with_sync:* *bool* = *True*, *with_triggers:* *bool* = *True*, *recursive:* *bool* = *False*) → None

Remove element named 'name' from the collection

Raises
CX – In case the name of the object was not given.

Module contents

The collections have the responsibility of ensuring the relational validity of the data present in Cobbler. Further they hold the data at runtime.

8.1.3 cobbler.items package

Submodules

cobbler.items.distro module

Cobbler module that contains the code for a Cobbler distro object.

Changelog:

Schema: From -> To

V3.4.0 (unreleased):

- **Added:**
  - `find_distro_path()`
  - `link_distro()`
- **Changed:**
  - Constructor: `kwargs` can now be used to seed the item during creation.
  - `children`: The property was moved to the base class.
  - `from_dict()`: The method was moved to the base class.

V3.3.4 (unreleased):

- No changes

V3.3.3:

- **Changed:**
  - `redhat_management_key`: Inherits from the settings again

V3.3.2:

- No changes

V3.3.1:

- No changes

V3.3.0:

- This release switched from pure attributes to properties (getters/setters).
- **Added:**
  - `from_dict()`
- **Moved to base class (Item):**
  - `ctime`: float
  - `depth`: int
  - `mtime`: float
  - `uid`: str
  - `kernel_options`: dict
  - `kernel_options_post`: dict
  - `autoinstall_meta`: dict
  - `boot_files`: list/dict
  - `template_files`: list/dict
  - `comment`: str

  - name: str
  - owners: list[str]
- **Changed:**
  - tree_build_time: str -> float
  - arch: str -> Union[list, str]
  - fetchable_files: list/dict? -> dict
  - boot_loader -> boot_loaders (rename)
- **Removed:**
  - get_fields()
  - get_parent
  - set_kernel() - Please use the property kernel
  - set_remote_boot_kernel() - Please use the property remote_boot_kernel
  - set_tree_build_time() - Please use the property tree_build_time
  - set_breed() - Please use the property breed
  - set_os_version() - Please use the property os_version
  - set_initrd() - Please use the property initrd
  - set_remote_boot_initrd() - Please use the property remote_boot_initrd
  - set_source_repos() - Please use the property source_repos
  - set_arch() - Please use the property arch
  - get_arch() - Please use the property arch
  - set_supported_boot_loaders() - Please use the property supported_boot_loaders. It is readonly.
  - set_boot_loader() - Please use the property boot_loader
  - set_redhat_management_key() - Please use the property redhat_management_key
  - get_redhat_management_key() - Please use the property redhat_management_key

V3.2.2:

- No changes

V3.2.1:

- **Added:**
  - kickstart: Resolves as a proxy to autoinstall

V3.2.0:

- No changes

V3.1.2:

- **Added:**
  - remote_boot_kernel: str
  - remote_grub_kernel: str
  - remote_boot_initrd: str
  - remote_grub_initrd: str

V3.1.1:

- No changes

V3.1.0:

- **Added:**
  - `get_arch()`

V3.0.1:

- File was moved from `cobbler/item_distro.py` to `cobbler/items/distro.py`.

V3.0.0:

- **Added:**
  - `boot_loader`: Union[str, inherit]
- **Changed:**
  - rename: `ks_meta` -> `autoinstall_meta`
  - `redhat_management_key`: Union[str, inherit] -> str
- **Removed:**
  - `redhat_management_server`: Union[str, inherit]

V2.8.5:

- Inital tracking of changes for the changelog.
- **Added:**
  - `name`: str
  - `ctime`: float
  - `mtime`: float
  - `uid`: str
  - `owners`: Union[list, SETTINGS:default_ownership]
  - `kernel`: str
  - `initrd`: str
  - `kernel_options`: dict
  - `kernel_options_post`: dict
  - `ks_meta`: dict
  - `arch`: str
  - `breed`: str
  - `os_version`: str
  - `source_repos`: list
  - `depth`: int
  - `comment`: str
  - `tree_build_time`: str
  - `mgmt_classes`: list
  - `boot_files`: list/dict?
  - `fetchable_files`: list/dict?
  - `template_files`: list/dict?
  - `redhat_management_key`: Union[str, inherit]

- redhat_management_server: Union[str, inherit]

class cobbler.items.distro.**Distro**(*api:* CobblerAPI, **args:* *Any*, ***kwargs:* *Any*)

Bases: *Item*

A Cobbler distribution object

COLLECTION_TYPE = 'distro'

TYPE_NAME = 'distro'

property arch

The field is mainly relevant to PXE provisioning.

Using an alternative distro type allows for dhcpd.conf templating to "do the right thing" with those systems – this also relates to bootloader configuration files which have different syntax for different distro types (because of the bootloaders).

This field is named "arch" because mainly on Linux, we only care about the architecture, though if (in the future) new provisioning types are added, an arch value might be something like "bsd_x86".

Returns
Return the current architecture.

property boot_loaders: List[str]

All boot loaders for which Cobbler generates entries for.

Note: This property can be set to <<inherit>>.

Getter
The bootloaders.

Setter
Validates this against the list of well-known bootloaders and raises a `TypeError` or `ValueError` in case the validation goes south.

property breed: str

The repository system breed. This decides some defaults for most actions with a repo in Cobbler.

Getter
The breed detected.

Setter
May raise a `ValueError` or `TypeError` in case the given value is wrong.

check_if_valid()

Check if a distro object is valid. If invalid an exception is raised.

find_distro_path()

This returns the absolute path to the distro under the `distro_mirror` directory. If that directory doesn't contain the kernel, the directory of the kernel in the distro is returned.

Returns
The path to the distribution files.

property initrd: str

Specifies an initrd image. Path search works as in set_kernel. File must be named appropriately.

Getter
The current path to the initrd.

Setter
May raise a `TypeError` or `ValueError` in case the validation is not successful.

property kernel: str

Specifies a kernel. The kernel parameter is a full path, a filename in the configured kernel directory or a directory path that would contain a selectable kernel. Kernel naming conventions are checked, see docs in the utils module for `find_kernel`.

Getter
The last successfully validated kernel path.

Setter
May raise a `ValueError` or `TypeError` in case of validation errors.

link_distro()

Link a Cobbler distro from its source into the web directory to make it reachable from the outside.

make_clone()

Clone a distro object.

Returns
The cloned object. Not persisted on the disk or in a database.

property os_version: str

The operating system version which the image contains.

Getter
The sanitized operating system version.

Setter
Accepts a str which will be validated against the `distro_signatures.json`.

property parent

Distros don't have parent objects.

property redhat_management_key: str

Get the redhat management key. This is probably only needed if you have spacewalk, uyuni or SUSE Manager running.

Note: This property can be set to <<inherit>>.

Returns
The key as a string.

property remote_boot_initrd: str

URL to a remote initrd. If the bootloader supports this feature, it directly tries to retrieve the initrd and boot it. (grub supports tftp and http protocol and server must be an IP).

Getter
Returns the current remote URL to boot from.

Setter
Raises a `TypeError` or `ValueError` in case the provided value was not correct.

property remote_boot_kernel: str

URL to a remote kernel. If the bootloader supports this feature, it directly tries to retrieve the kernel and boot it. (grub supports tftp and http protocol and server must be an IP).

Getter
Returns the current remote URL to boot from.

Setter
Raises a `TypeError` or `ValueError` in case the provided value was not correct.

property remote_grub_initrd: str

This is tied to the remote_boot_initrd property. It contains the URL of that field in a format which grub can use directly.

Getter
The computed URL from remote_boot_initrd.

property remote_grub_kernel: str

This is tied to the remote_boot_kernel property. It contains the URL of that field in a format which grub can use directly.

Getter
The computed URL from remote_boot_kernel.

property source_repos: List[Any]

A list of http:// URLs on the Cobbler server that point to yum configuration files that can be used to install core packages. Use by cobbler import only.

Getter
The source repos used.

Setter
The new list of source repos to use.

property supported_boot_loaders: List[str]

Some distributions, particularly on powerpc, can only be netbooted using specific bootloaders.

Returns
The bootloaders which are available for being set.

property tree_build_time: float

Represents the import time of the distro. If not imported, this field is not meaningful.

Getter
Setter

cobbler.items.image module

Cobbler module that contains the code for a Cobbler image object.

Changelog:

V3.4.0 (unreleased):

- **Added:**
  - display_name
- **Changed:**
  - Constructor: kwargs can now be used to seed the item during creation.
  - autoinstall: Restored inheritance of the property.
  - children: The proqperty was moved to the base class.
  - from_dict(): The method was moved to the base class.
  - virt_disk_driver: Restored inheritance of the property.
  - virt_ram: Restored inheritance of the property.
  - virt_type: Restored inheritance of the property.
  - virt_bridge: Restored inheritance of the property.

V3.3.4 (unreleased):

- No changes

V3.3.3:

- **Added:**
  - children
- **Changes:**
  - virt_file_size: Inherits from the settings again
  - boot_loaders: Inherits from the settings again

V3.3.2:

- No changes

V3.3.1:

- No changes

V3.3.0:

- This release switched from pure attributes to properties (getters/setters).
- **Added:**
  - boot_loaders: list
  - menu: str
  - supported_boot_loaders: list
  - from_dict()
- **Moved to parent class (Item):**
  - ctime: float
  - mtime: float
  - depth: int
  - parent: str
  - uid: str
  - comment: str
  - name: str
- **Removed:**
  - get_fields()
  - get_parent()
  - set_arch() - Please use the arch property.
  - set_autoinstall() - Please use the autoinstall property.
  - set_file() - Please use the file property.
  - set_os_version() - Please use the os_version property.
  - set_breed() - Please use the breed property.
  - set_image_type() - Please use the image_type property.
  - set_virt_cpus() - Please use the virt_cpus property.
  - set_network_count() - Please use the network_count property.
  - set_virt_auto_boot() - Please use the virt_auto_boot property.
  - set_virt_file_size() - Please use the virt_file_size property.
  - set_virt_disk_driver() - Please use the virt_disk_driver property.

  - `set_virt_ram()` - Please use the `virt_ram` property.
  - `set_virt_type()` - Please use the `virt_type` property.
  - `set_virt_bridge()` - Please use the `virt_bridge` property.
  - `set_virt_path()` - Please use the `virt_path` property.
  - `get_valid_image_types()`
- **Changes:**
  - `arch`: str -> enums.Archs
  - `autoinstall`: str -> enums.VALUE_INHERITED
  - `image_type`: str -> enums.ImageTypes
  - `virt_auto_boot`: Union[bool, SETTINGS:virt_auto_boot] -> bool
  - `virt_bridge`: Union[str, SETTINGS:default_virt_bridge] -> str
  - `virt_disk_driver`: Union[str, SETTINGS:default_virt_disk_driver] -> enums.VirtDiskDrivers
  - `virt_file_size`: Union[float, SETTINGS:default_virt_file_size] -> float
  - `virt_ram`: Union[int, SETTINGS:default_virt_ram] -> int
  - `virt_type`: Union[str, SETTINGS:default_virt_type] -> enums.VirtType

V3.2.2:

- No changes

V3.2.1:

- **Added:**
  - `kickstart`: Resolves as a proxy to `autoinstall`

V3.2.0:

- No changes

V3.1.2:

- No changes

V3.1.1:

- No changes

V3.1.0:

- No changes

V3.0.1:

- No changes

V3.0.0:

- **Added:**
  - `set_autoinstall()`
- **Changes:**
  - Rename: `kickstart` -> `autoinstall`
- **Removed:**
  - `set_kickstart()` - Please use `set_autoinstall()`

V2.8.5:

- Inital tracking of changes for the changelog.
- **Added:**
  - ctime: float
  - depth: int
  - mtime: float
  - parent: str
  - uid: str
  - arch: str
  - kickstart: str
  - breed: str
  - comment: str
  - file: str
  - image_type: str
  - name: str
  - network_count: int
  - os_version: str
  - owners: Union[list, SETTINGS:default_ownership]
  - virt_auto_boot: Union[bool, SETTINGS:virt_auto_boot]
  - virt_bridge: Union[str, SETTINGS:default_virt_bridge]
  - virt_cpus: int
  - virt_disk_driver: Union[str, SETTINGS:default_virt_disk_driver]
  - virt_file_size: Union[float, SETTINGS:default_virt_file_size]
  - virt_path: str
  - virt_ram: Union[int, SETTINGS:default_virt_ram]
  - virt_type: Union[str, SETTINGS:default_virt_type]

class cobbler.items.image.**Image**(*api:* CobblerAPI, **args:* *Any*, ***kwargs:* *Any*)

Bases: *Item*

A Cobbler Image. Tracks a virtual or physical image, as opposed to a answer file (autoinst) led installation.

COLLECTION_TYPE = 'image'

TYPE_NAME = 'image'

property arch: ***Archs***

Represents the architecture the image has. If deployed to a physical host this should be enforced, a virtual image may be deployed on a host with any architecture.

Getter
The current architecture. Default is X86_64.

Setter
Should be of the enum type or str. May raise an exception in case the architecture is not known to Cobbler.

property autoinstall: str

Property for the automatic installation file path, this must be a local file.

It may not make sense for images to have automatic installation templates. It really doesn't. However if the image type is 'iso' koan can create a virtual floppy and shove an answer file on it, to script an installation. This may not be a automatic installation template per se, it might be a Windows answer file (SIF) etc.

This property can inherit from a parent. Which is actually the default value.

Getter
The path relative to the template directory.

Setter
The location of the template relative to the template base directory.

property boot_loaders: List[str]

Represents the boot loaders which are able to boot this image.

Getter
The bootloaders. May be an emtpy list.

Setter
A list with the supported boot loaders for this image.

property breed: str

The operating system breed.

Getter
Returns the current breed.

Setter
When setting this it is validated against the `distro_signatures.json` file.

property display_name: str

Returns the display name.

Getter
Returns the display name for the boot menu.

Setter
Sets the display name for the boot menu.

property file: str

Stores the image location. This should be accessible on all nodes that need to access it.

Format: can be one of the following: * username:password@hostname:/path/to/the/filename.ext * username@hostname:/path/to/the/filename.ext * hostname:/path/to/the/filename.ext * /path/to/the/filename.ext

Getter
The path to the image location or an emtpy string.

Setter
May raise a TypeError or SyntaxError in case the validation of the location fails.

property image_type: *ImageTypes*

Indicates what type of image this is. direct = something like "memdisk", physical only iso = a bootable ISO that pxe's or can be used for virt installs, virtual only virt-clone = a cloned virtual disk (FIXME: not yet supported), virtual only memdisk = hdd image (physical only)

Getter
The enum type value of the image type.

Setter
Accepts str like and enum type values and raises a TypeError or ValueError in the case of a problem.

make_clone()

Clone this image object. Please manually adjust all value yourself to make the cloned object unique.

Returns
The cloned instance of this object.

property menu: str

Property to represent the menu which this image should be put into.

Getter
The name of the menu or an emtpy str.

Setter
Should only be the name of the menu not the object. May raise CX in case the menu does not exist.

property network_count: int

Represents the number of virtual NICs this image has.

Deprecated since version 3.3.0: This is nowhere used in the project and will be removed in a future release.

Getter
The number of networks.

Setter
Raises a TypeError in case the value is not an int.

property os_version: str

The operating system version which the image contains.

Getter
The sanitized operating system version.

Setter
Accepts a str which will be validated against the distro_signatures.json.

property supported_boot_loaders: List[str]

Read only property which represents the subset of settable bootloaders.

Getter
The bootloaders which are available for being set.

property virt_auto_boot: bool

Whether the VM should be booted when booting the host or not.

Getter
True means autoboot is enabled, otherwise VM is not booted automatically.

Setter
The new state for the property.

property virt_bridge: str

The name of the virtual bridge used for networking.

> **Warning:** The new validation for the setter is not working. Thus the inheritance from the settings is broken.

Getter
The name of the bridge.

Setter
The new name of the bridge. If set to an empty str, it will be taken from the settings.

property virt_cpus: *int*

The amount of vCPU cores used in case the image is being deployed on top of a VM host.

Getter
The cores used.

Setter
The new number of cores.

property virt_disk_driver: *VirtDiskDrivers*

The type of disk driver used for storing the image.

Getter
The enum type representation of the disk driver.

Setter
May be a `str` with the name of the disk driver or from the enum type directly.

property virt_file_size: *float*

The size of the image and thus the usable size for the guest.

> **Warning:** There is a regression which makes the usage of multiple disks not possible right now. This will be fixed in a future release.

Getter
The size of the image(s) in GB.

Setter
The float with the new size in GB.

property virt_path: *str*

Represents the location where the image for the VM is stored.

Getter
The path.

Setter
Is being validated for being a reasonable path. If yes is set, otherwise ignored.

property virt_ram: *int*

The amount of RAM given to the guest in MB.

Getter
The amount of RAM currently assigned to the image.

Setter
The new amount of ram. Must be an integer.

property virt_type: *VirtType*

The type of image used.

Getter
The value of the virtual machine.

Setter
May be of the enum type or a str which is then converted to the enum type.

cobbler.items.item module

Cobbler module that contains the code for a generic Cobbler item.

Changelog:

V3.4.0 (unreleased):

- **(Re-)Added Cache implementation with the following new methods and properties:**
  - `cache`
  - `inmemery`
  - `clean_cache()`
- **Overhauled the parent/child system:**
  - `children` is now inside `item.py`.
  - `tree_walk()` was added.
  - `logical_parent` was added.
  - `get_parent()` was added which returns the internal reference that is used to return the object of the `parent` property.
- **Removed:**
  - mgmt_classes
  - mgmt_parameters

V3.3.4 (unreleased):

- No changes

V3.3.3:

- **Added:**
  - `grab_tree`

V3.3.2:

- No changes

V3.3.1:

- No changes

V3.3.0:

- This release switched from pure attributes to properties (getters/setters).
- **Added:**
  - `depth`: int
  - `comment`: str
  - `owners`: Union[list, str]
  - `mgmt_classes`: Union[list, str]
  - `mgmt_classes`: Union[dict, str]
  - `conceptual_parent`: Union[distro, profile]
- **Removed:**
  - collection_mgr: collection_mgr
  - **Remove unreliable caching:**
    - `get_from_cache()`

        * set_cache()
        * remove_from_cache()
    - **Changed:**
        - Constructor: Takes an instance of CobblerAPI instead of CollectionManager.
        - children: dict -> list
        - ctime: int -> float
        - mtime: int -> float
        - uid: str
        - kernel_options: dict -> Union[dict, str]
        - kernel_options_post: dict -> Union[dict, str]
        - autoinstall_meta: dict -> Union[dict, str]
        - fetchable_files: dict -> Union[dict, str]
        - boot_files: dict -> Union[dict, str]

V3.2.2:

- No changes

V3.2.1:

- No changes

V3.2.0:

- No changes

V3.1.2:

- No changes

V3.1.1:

- No changes

V3.1.0:

- No changes

V3.0.1:

- No changes

V3.0.0:

- **Added:**
    - collection_mgr: collection_mgr
    - kernel_options: dict
    - kernel_options_post: dict
    - autoinstall_meta: dict
    - fetchable_files: dict
    - boot_files: dict
    - template_files: dict
    - name: str
    - last_cached_mtime: int
- **Changed:**

- Rename: `cached_datastruct` -> `cached_dict`

- **Removed:**
  - `config`

V2.8.5:

- **Added:**
  - `config`: ?
  - `settings`: settings
  - `is_subobject`: bool
  - `parent`: Union[distro, profile]
  - `children`: dict
  - `log_func`: collection_mgr.api.log
  - `ctime`: int
  - `mtime`: int
  - `uid`: str
  - `last_cached_mtime`: int
  - `cached_datastruct`: str

class cobbler.items.item.**Item**(*api:* CobblerAPI, *is_subobject: bool = False*, ***kwargs: Any*)

Bases: `object`

An Item is a serializable thing that can appear in a Collection

COLLECTION_TYPE = 'generic'

LOGICAL_INHERITANCE: Dict[str, Tuple[List[Tuple[str, str]], List[Tuple[str, str]]]] = {'distro': ([], [('profile', 'distro')]), 'image': ([], [('system', 'image')]), 'profile': ([('distro', 'distro')], [('system', 'profile')]), 'system': ([('image', 'image'), ('profile', 'profile')], [])}

TYPE_DEPENDENCIES: Dict[str, List[Tuple[str, str]]] = {'distro': [('profile', 'distro')], 'image': [('system', 'image')], 'menu': [('menu', 'parent'), ('image', 'menu'), ('profile', 'menu')], 'profile': [('profile', 'parent'), ('system', 'profile')], 'repo': [('profile', 'repos')], 'system': []}

TYPE_NAME = 'generic'

property autoinstall_meta: Dict[Any, Any]

A comma delimited list of key value pairs, like 'a=b,c=d,e=f' or a dict. The meta tags are used as input to the templating system to preprocess automatic installation template files.

Note: This property can be set to <<inherit>>.

Getter
The metadata or an empty dict.

Setter
Accepts anything which can be split by *input_string_or_dict()*.

property boot_files: Dict[Any, Any]

Files copied into tftpboot beyond the kernel/initrd

Getter
The dictionary with name-path key-value pairs.

Setter
A dict. If not a dict must be a str which is split by *input_string_or_dict()*. Raises TypeError otherwise.

property cache: ItemCache

Gettinging the ItemCache oject.

Note: This is a read only property.

Getter
This is the ItemCache oject.

check_if_valid() → None

Raise exceptions if the object state is inconsistent.

Raises
CX – In case the name of the item is not set.

property children: List[ITEM_UNION]

The list of logical children of any depth. :getter: An empty list in case of items which don't have logical children. :setter: Replace the list of children completely with the new provided one.

clean_cache(*name: Optional[str] = None*)

Clearing the Item cache.

Parameters

- **obj** – The object whose modification invalidates the dict cache. Can be Item, Settings or SIGNATURE_CACHE.
- **name** – The name of Item attribute or None.

property comment: str

For every object you are able to set a unique comment which will be persisted on the object.

Getter
The comment or an emtpy string.

Setter
The new comment for the item.

property ctime: float

Property which represents the creation time of the object.

Getter
The float which can be passed to Python time stdlib.

Setter
Should only be used by the Cobbler Item Factory.

property depth: int

This represents the logical depth of an object in the category of the same items. Important for the order of loading items from the disk and other related features where the alphabetical order is incorrect for sorting.

Getter
The logical depth of the object.

Setter
The new int for the logical object-depth.

property descendants: List[ITEM_UNION]

Get objects that depend on this object, i.e. those that would be affected by a cascading delete, etc.

Note: This is a read only property.

Getter
This is a list of all descendants. May be empty if none exist.

deserialize() → None

Deserializes the object itself and, if necessary, recursively all the objects it depends on.

dump_vars(*formatted_output: bool = True*, *remove_dicts: bool = False*) → Union[Dict[str, Any], str]

Dump all variables.

Parameters

- **formatted_output** – Whether to format the output or not.
- **remove_dicts** – If True the dictionaries will be put into str form.

Returns
The raw or formatted data.

property fetchable_files: Dict[Any, Any]

A comma seperated list of `virt_name=path_to_template` that should be fetchable via tftp or a webserver

---

Note: This property can be set to <<inherit>>.

Getter
The dictionary with name-path key-value pairs.

Setter
A dict. If not a dict must be a str which is split by *input_string_or_dict()*. Raises `TypeError` otherwise.

find_match(*kwargs: Dict[str, Any]*, *no_errors: bool = False*) → bool

Find from a given dict if the item matches the kv-pairs.

Parameters

- **kwargs** – The dict to match for in this item.
- **no_errors** – How strict this matching is.

Returns
True if matches or False if the item does not match.

find_match_single_key(*data: Dict[str, Any]*, *key: str*, *value: Any*, *no_errors: bool = False*) → bool

Look if the data matches or not. This is an alternative for `find_match()`.

Parameters

- **data** – The data to search through.
- **key** – The key to look for int the item.
- **value** – The value for the key.
- **no_errors** – How strict this matching is.

Returns
Whether the data matches or not.

`from_dict`(*dictionary: Dict[Any, Any]*) → None

Modify this object to take on values in `dictionary`.

Parameters
`dictionary` – This should contain all values which should be updated.

Raises

- `AttributeError` – In case during the process of setting a value for an attribute an error occurred.
- `KeyError` – In case there were keys which could not be set in the item dictionary.

`get_conceptual_parent`() → Optional[ITEM_UNION]

The parent may just be a superclass for something like a subprofile. Get the first parent of a different type.

Returns
The first item which is conceptually not from the same type.

`property get_parent: str`

This method returns the name of the parent for the object. In case there is not parent this return empty string.

`grab_tree`() → List[Union[*Item*, *Settings*]]

Climb the tree and get every node.

Returns
The list of items with all parents from that object upwards the tree. Contains at least the item itself and the settings of Cobbler.

`property inmemory: bool`

If set to `false`, only the Item name is in memory. The rest of the Item's properties can be retrieved either on demand or as a result of the `load_items` background task.

Getter
The inmemory for the item.

Setter
The new inmemory value for the object. Should only be used by the Cobbler serializers.

`property is_subobject: bool`

Weather the object is a subobject of another object or not.

Getter
True in case the object is a subobject, False otherwise.

Setter
Sets the value. If this is not a bool, this will raise a `TypeError`.

`property kernel_options: Dict[Any, Any]`

Kernel options are a space delimited list, like 'a=b c=d e=f g h i=j' or a dict.

Note: This property can be set to <<inherit>>.

Getter
The parsed kernel options.

Setter
The new kernel options as a space delimited list. May raise `ValueError` in case of parsing problems.

property kernel_options_post: Dict[str, Any]

Post kernel options are a space delimited list, like 'a=b c=d e=f g h i=j' or a dict.

Note: This property can be set to <<inherit>>.

Getter
The dictionary with the parsed values.

Setter
Accepts str in above mentioned format or directly a dict.

property logical_parent: Any

This property contains the name of the logical parent of an object. In case there is not parent this return None.

Getter
Returns the parent object or None if it can't be resolved via the Cobbler API.

Setter
The name of the new logical parent.

abstract make_clone() → ITEM

Must be defined in any subclass

property mtime: float

Represents the last modification time of the object via the API. This is not updated automagically.

Getter
The float which can be fed into a Python time object.

Setter
The new time something was edited via the API.

property name: str

Property which represents the objects name.

Getter
The name of the object.

Setter
Updating this has broad implications. Please try to use the `rename()` functionality from the corresponding collection.

property owners: List[Any]

This is a feature which is related to the ownership module of Cobbler which gives only specific people access to specific records. Otherwise this is just a cosmetic feature to allow assigning records to specific users.

Warning: This is never validated against a list of existing users. Thus you can lock yourself out of a record.

Note: This property can be set to <<inherit>>.

Getter
Return the list of users which are currently assigned to the record.

Setter
The list of people which should be new owners. May lock you out if you are using the ownership authorization module.

property parent: Optional[Union[*System*, *Profile*, *Distro*, *Menu*]]

This property contains the name of the parent of an object. In case there is not parent this return None.

Getter
Returns the parent object or None if it can't be resolved via the Cobbler API.

Setter
The name of the new logical parent.

serialize() → Dict[str, Any]

This method is a proxy for *to_dict()* and contains additional logic for serialization to a persistent location.

Returns
The dictionary with the information for serialization.

sort_key(*sort_fields: List[Any]*)

Convert the item to a dict and sort the data after specific given fields.

Parameters
sort_fields – The fields to sort the data after.

Returns
The sorted data.

property template_files: Dict[Any, Any]

File mappings for built-in configuration management

Getter
The dictionary with name-path key-value pairs.

Setter
A dict. If not a dict must be a str which is split by *input_string_or_dict()*. Raises `TypeError` otherwise.

to_dict(*resolved: bool = False*) → Dict[Any, Any]

This converts everything in this object to a dictionary.

Parameters
resolved – If this is True, Cobbler will resolve the values to its final form, rather than give you the objects raw value.

Returns
A dictionary with all values present in this object.

tree_walk() → List[ITEM_UNION]

Get all children related by parent/child relationship. :return: The list of children objects.

property uid: str

The uid is the internal unique representation of a Cobbler object. It should never be used twice, even after an object was deleted.

Getter
The uid for the item. Should be unique across a running Cobbler instance.

Setter
The new uid for the object. Should only be used by the Cobbler Item Factory.

cobbler.items.menu module

Cobbler module that contains the code for a Cobbler menu object.

Changelog:

V3.4.0 (unreleased):

- **Changes:**
  - Constructor: `kwargs` can now be used to seed the item during creation.
  - `children`: The property was moved to the base class.
  - `parent`: The property was moved to the base class.
  - `from_dict()`: The method was moved to the base class.

V3.3.4 (unreleased):

- No changes

V3.3.3:

- **Changed:**
  - `check_if_valid()`: Now present in base class.

V3.3.2:

- No changes

V3.3.1:

- No changes

V3.3.0:

- Inital version of the item type.
- **Added:**
  - display_name: str

`class cobbler.items.menu.Menu(api: CobblerAPI, *args: Any, **kwargs: Any)`

Bases: *Item*

A Cobbler menu object.

`COLLECTION_TYPE = 'menu'`

`TYPE_NAME = 'menu'`

`property display_name: str`

Returns the display name.

Getter
Returns the display name for the boot menu.

Setter
Sets the display name for the boot menu.

`make_clone() → Menu`

Clone this file object. Please manually adjust all value yourself to make the cloned object unique.

Returns
The cloned instance of this object.

cobbler.items.profile module

Cobbler module that contains the code for a Cobbler profile object.

Changelog:

V3.4.0 (unreleased):

- **Changes:**
  - Constructor: `kwargs` can now be used to seed the item during creation.
  - `children`: The property was moved to the base class.
  - `parent`: The property was moved to the base class.
  - `from_dict()`: The method was moved to the base class.

V3.3.4 (unreleased):

- No changes

V3.3.3:

- **Changed:**
  - `next_server_v4`: str -> enums.VALUE_INHERITED
  - `next_server_v6`: str -> enums.VALUE_INHERITED
  - `virt_bridge`: str -> enums.VALUE_INHERITED
  - `virt_file_size`: int -> enums.VALUE_INHERITED
  - `virt_ram`: int -> enums.VALUE_INHERITED

V3.3.2:

- No changes

V3.3.1:

- No changes

V3.3.0:

- This release switched from pure attributes to properties (getters/setters).
- **Added:**
  - `boot_loaders`: Union[list, str]
  - `enable_ipxe`: bool
  - `next_server_v4`: str
  - `next_server_v6`: str
  - `menu`: str
  - `from_dict()`
- **Removed:**
  - `enable_gpxe`: Union[bool, SETTINGS:enable_gpxe]
  - `next_server`: Union[str, inherit]
  - `get_fields()`
  - `get_parent()`: Please use the property `parent` instead
  - `set_parent()`: Please use the property `parent` instead
  - `set_distro()`: Please use the property `distro` instead
  - `set_name_servers()`: Please use the property `name_servers` instead

  - `set_name_servers_search()`: Please use the property `name_servers_search` instead
  - `set_proxy()`: Please use the property `proxy` instead
  - `set_enable_gpxe()`: Please use the property `enable_gpxe` instead
  - `set_enable_menu()`: Please use the property `enable_menu` instead
  - `set_dhcp_tag()`: Please use the property `dhcp_tag` instead
  - `set_server()`: Please use the property `server` instead
  - `set_next_server()`: Please use the property `next_server` instead
  - `set_filename()`: Please use the property `filename` instead
  - `set_autoinstall()`: Please use the property `autoinstall` instead
  - `set_virt_auto_boot()`: Please use the property `virt_auto_boot` instead
  - `set_virt_cpus()`: Please use the property `virt_cpus` instead
  - `set_virt_file_size()`: Please use the property `virt_file_size` instead
  - `set_virt_disk_driver()`: Please use the property `virt_disk_driver` instead
  - `set_virt_ram()`: Please use the property `virt_ram` instead
  - `set_virt_type()`: Please use the property `virt_type` instead
  - `set_virt_bridge()`: Please use the property `virt_bridge` instead
  - `set_virt_path()`: Please use the property `virt_path` instead
  - `set_repos()`: Please use the property `repos` instead
  - `set_redhat_management_key()`: Please use the property `redhat_management_key` instead
  - `get_redhat_management_key()`: Please use the property `redhat_management_key` instead
  - `get_arch()`: Please use the property `arch` instead
- **Changed:**
  - `autoinstall`: Union[str, SETTINGS:default_kickstart] -> enums.VALUE_INHERITED
  - `enable_menu`: Union[bool, SETTINGS:enable_menu] -> bool
  - `name_servers`: Union[list, SETTINGS:default_name_servers] -> list
  - `name_servers_search`: Union[list, SETTINGS:default_name_servers_search] -> list
  - `filename`: Union[str, inherit] -> str
  - `proxy`: Union[str, SETTINGS:proxy_url_int] -> enums.VALUE_INHERITED
  - `redhat_management_key`: Union[str, inherit] -> enums.VALUE_INHERITED
  - `server`: Union[str, inherit] -> enums.VALUE_INHERITED
  - `virt_auto_boot`: Union[bool, SETTINGS:virt_auto_boot] -> bool
  - `virt_bridge`: Union[str, SETTINGS:default_virt_bridge] -> str
  - `virt_cpus`: int -> Union[int, str]
  - `virt_disk_driver`: Union[str, SETTINGS:default_virt_disk_driver] -> enums.VirtDiskDrivers
  - `virt_file_size`: Union[int, SETTINGS:default_virt_file_size] -> int
  - `virt_ram`: Union[int, SETTINGS:default_virt_ram] -> int
  - `virt_type`: Union[str, SETTINGS:default_virt_type] -> enums.VirtType

- boot_files: list/dict? -> enums.VALUE_INHERITED
- fetchable_files: dict -> enums.VALUE_INHERITED
- autoinstall_meta: dict -> enums.VALUE_INHERITED
- kernel_options: dict -> enums.VALUE_INHERITED
- kernel_options_post: dict -> enums.VALUE_INHERITED
- mgmt_classes: list -> enums.VALUE_INHERITED
- mgmt_parameters: Union[str, inherit] -> enums.VALUE_INHERITED

(mgmt_classes parameter has a duplicate)

V3.2.2:

- No changes

V3.2.1:

- **Added:**
  - kickstart: Resolves as a proxy to autoinstall

V3.2.0:

- No changes

V3.1.2:

- **Added:**
  - filename: Union[str, inherit]

V3.1.1:

- No changes

V3.1.0:

- **Added:**
  - get_arch()

V3.0.1:

- File was moved from cobbler/item_profile.py to cobbler/items/profile.py.

V3.0.0:

- **Added:**
  - next_server: Union[str, inherit]
- **Changed:**
  - Renamed: kickstart -> autoinstall
  - Renamed: ks_meta -> autoinstall_meta
  - autoinstall: Union[str, SETTINGS:default_kickstart] -> Union[str, SETTINGS:default_autoinstall]
  - set_kickstart(): Renamed to set_autoinstall()
- **Removed:**
  - redhat_management_server: Union[str, inherit]
  - template_remote_kickstarts: Union[bool, SETTINGS:template_remote_kickstarts]
  - set_redhat_management_server()
  - set_template_remote_kickstarts()

V2.8.5:

- Inital tracking of changes for the changelog.
- **Added**
  - `ctime`: int
  - `depth`: int
  - `mtime`: int
  - `uid`: str
  - `kickstart`: Union[str, SETTINGS:default_kickstart]
  - `ks_meta`: dict
  - `boot_files`: list/dict?
  - `comment`: str
  - `dhcp_tag`: str
  - `distro`: str
  - `enable_gpxe`: Union[bool, SETTINGS:enable_gpxe]
  - `enable_menu`: Union[bool, SETTINGS:enable_menu]
  - `fetchable_files`: dict
  - `kernel_options`: dict
  - `kernel_options_post`: dict
  - `mgmt_classes`: list
  - `mgmt_parameters`: Union[str, inherit]
  - `name`: str
  - `name_servers`: Union[list, SETTINGS:default_name_servers]
  - `name_servers_search`: Union[list, SETTINGS:default_name_servers_search]
  - `owners`: Union[list, SETTINGS:default_ownership]
  - `parent`: str
  - `proxy`: Union[str, SETTINGS:proxy_url_int]
  - `redhat_management_key`: Union[str, inherit]
  - `redhat_management_server`: Union[str, inherit]
  - `template_remote_kickstarts`: Union[bool, SETTINGS:template_remote_kickstarts]
  - `repos`: list
  - `server`: Union[str, inherit]
  - `template_files`: dict
  - `virt_auto_boot`: Union[bool, SETTINGS:virt_auto_boot]
  - `virt_bridge`: Union[str, SETTINGS:default_virt_bridge]
  - `virt_cpus`: int
  - `virt_disk_driver`: Union[str, SETTINGS:default_virt_disk_driver]
  - `virt_file_size`: Union[int, SETTINGS:default_virt_file_size]
  - `virt_path`: str
  - `virt_ram`: Union[int, SETTINGS:default_virt_ram]

- virt_type: Union[str, SETTINGS:default_virt_type]

class cobbler.items.profile.**Profile**(*api:* CobblerAPI, **args:* *Any*, ***kwargs:* *Any*)

Bases: *Item*

A Cobbler profile object.

COLLECTION_TYPE = 'profile'

TYPE_NAME = 'profile'

property arch: Optional[*Archs*]

This represents the architecture of a profile. It is read only.

Getter
None or the parent architecture.

property autoinstall: str

Represents the automatic OS installation template file path, this must be a local file.

Getter
Either the inherited name or the one specific to this profile.

Setter
The name of the new autoinstall template is validated. The path should come in the format of a str.

property boot_loaders: List[str]

This represents all boot loaders for which Cobbler will try to generate bootloader configuration for.

Note: This property can be set to <<inherit>>.

Getter
The bootloaders.

Setter
The new bootloaders. Will be validates against a list of well known ones.

check_if_valid()

Check if the profile is valid. This checks for an existing name and a distro as a conceptual parent.

Raises
CX – In case the distro or name is not present.

property dhcp_tag: str

Represents the VLAN tag the DHCP Server is in/answering to.

Getter
The VLAN tag or nothing if a system with the profile should not be in a VLAN.

Setter
The new VLAN tag.

property display_name: str

Returns the display name.

Getter
Returns the display name for the boot menu.

Setter
Sets the display name for the boot menu.

property distro: Optional[*Distro*]

The parent distro of a profile. This is not representing the Distro but the id of it.

This is a required property, if saved to the disk, with the exception if this is a subprofile.

Returns
The distro object or None.

property enable_ipxe: bool

Sets whether or not the profile will use iPXE for booting.

Getter
If set to inherit then this returns the parent value, otherwise it returns the real value.

Setter
May throw a `TypeError` in case the new value cannot be cast to `bool`.

property enable_menu: bool

Sets whether or not the profile will be listed in the default PXE boot menu. This is pretty forgiving for YAML's sake.

Getter
The value resolved from the defaults or the value specific to the profile.

Setter
May raise a `TypeError` in case the boolean could not be converted.

property filename: str

The filename which is fetched by the client from TFTP.

If the filename is set to <<inherit>> and there is no parent profile then it will be set to an empty string.

Getter
Either the default/inherited one, or the one specific to this profile.

Setter
The new filename which is fetched on boot. May raise a `TypeError` when the wrong type was given.

find_match_single_key(*data: Dict[str, Any]*, *key: str*, *value: Any*, *no_errors: bool = False*) → bool

Look if the data matches or not. This is an alternative for `find_match()`.

Parameters

- **data** – The data to search through.
- **key** – The key to look for int the item.
- **value** – The value for the key.
- **no_errors** – How strict this matching is.

Returns
Whether the data matches or not.

make_clone()

Clone this file object. Please manually adjust all value yourself to make the cloned object unique.

Returns
The cloned instance of this object.

property menu: str

Property to represent the menu which this image should be put into.

Getter
The name of the menu or an emtpy str.

Setter
Should only be the name of the menu not the object. May raise `CX` in case the menu does not exist.

property name_servers: List[Any]

Represents the list of nameservers to set for the profile.

Getter
The nameservers.

Setter
Comma delimited `str` or list with the nameservers.

property name_servers_search: List[Any]

Represents the list of DNS search paths.

Getter
The list of DNS search paths.

Setter
Comma delimited `str` or list with the nameservers search paths.

property next_server_v4: str

Represents the next server for IPv4.

Getter
The IP for the next server.

Setter
May raise a `TypeError` if the new value is not of type `str`.

property next_server_v6: str

Represents the next server for IPv6.

Getter
The IP for the next server.

Setter
May raise a `TypeError` if the new value is not of type `str`.

property proxy: str

Override the default external proxy which is used for accessing the internet.

Getter
Returns the default one or the specific one for this repository.

Setter
May raise a `TypeError` in case the wrong value is given.

property redhat_management_key: str

Getter of the redhat management key of the profile or it's parent.

Note: This property can be set to <<inherit>>.

Getter
Returns the redhat_management_key of the profile.

Setter
May raise a `TypeError` in case of a validation error.

property repos: Union[str, List[str]]

The repositories to add once the system is provisioned.

Getter
The names of the repositories the profile has assigned.

Setter
The new names of the repositories for the profile. Validated against existing repositories.

property server: str

Represents the hostname the Cobbler server is reachable by a client.

Note: This property can be set to <<inherit>>.

Getter
The hostname of the Cobbler server.

Setter
May raise a TypeError in case the new value is not a str.

property virt_auto_boot: bool

Whether the VM should be booted when booting the host or not.

Note: This property can be set to <<inherit>>.

Getter
True means autoboot is enabled, otherwise VM is not booted automatically.

Setter
The new state for the property.

property virt_bridge: str

Represents the name of the virtual bridge to use.

Note: This property can be set to <<inherit>>.

Getter
Either the default name for the bridge or the specific one for this profile.

Setter
The new name. Does not overwrite the default one.

property virt_cpus: int

The amount of vCPU cores used in case the image is being deployed on top of a VM host.

Getter
The cores used.

Setter
The new number of cores.

property virt_disk_driver: *VirtDiskDrivers*

The type of disk driver used for storing the image.

Note: This property can be set to <<inherit>>.

Getter
The enum type representation of the disk driver.

Setter
May be a `str` with the name of the disk driver or from the enum type directly.

property virt_file_size: *float*

The size of the image and thus the usable size for the guest.

> **Warning:** There is a regression which makes the usage of multiple disks not possible right now. This will be fixed in a future release.

Note: This property can be set to <<inherit>>.

Getter
The size of the image(s) in GB.

Setter
The float with the new size in GB.

property virt_path: *str*

The path to the place where the image will be stored.

Getter
The path to the image.

Setter
The new path for the image.

property virt_ram: *int*

The amount of RAM given to the guest in MB.

Note: This property can be set to <<inherit>>.

Getter
The amount of RAM currently assigned to the image.

Setter
The new amount of ram. Must be an integer.

property virt_type: *VirtType*

The type of image used.

Note: This property can be set to <<inherit>>.

Getter
The value of the virtual machine.

Setter
May be of the enum type or a str which is then converted to the enum type.

cobbler.items.repo module

Cobbler module that contains the code for a Cobbler repo object.

Changelog:

V3.4.0 (unreleased):

- **Changed:**
  - Constructor: `kwargs` can now be used to seed the item during creation.
  - `children`: The property was moved to the base class.
  - `from_dict()`: The method was moved to the base class.

V3.3.4 (unreleased):

- No changes

V3.3.3:

- No changes

V3.3.2:

- No changes

V3.3.1:

- No changes

V3.3.0:

- This release switched from pure attributes to properties (getters/setters).
- **Added:**
  - os_version: str
  - `from_dict()`
- **Moved to base class (Item):**
  - `ctime`: float
  - `depth`: float
  - `mtime`: float
  - `parent`: str
  - `uid`: str
  - `comment`: str
  - `name`: str
  - `owners`: Union[list, SETTINGS:default_ownership]
- **Changes:**
  - `breed`: str -> enums.RepoBreeds
  - `arch`: str -> enums.RepoArchs
  - `rsyncopts`: dict/str? -> dict
  - `mirror_type`: str -> enums.MirrorType
  - `apt_components`: list/str? -> list
  - `apt_dists`: list/str? -> list
  - `createrepo_flags`: Union[dict, inherit] -> enums.VALUE_INHERITED
  - `proxy`: Union[str, inherit] -> enums.VALUE_INHERITED

V3.2.2:

- No changes

V3.2.1:

- **Added:**
  - `mirror_type`: str
  - `set_mirror_type()`

V3.2.0:

- **Added:**
  - `rsyncopts`: dict/str
  - `set_rsyncopts()`

V3.1.2:

- No changes

V3.1.1:

- No changes

V3.1.0:

- **Changed:**
  - `arch`: New valid value `s390x` as an architecture.

V3.0.1:

- File was moved from `cobbler/item_repo.py` to `cobbler/items/repo.py`.

V3.0.0:

- **Changes:**
  - `proxy`: Union[str, inherit, SETTINGS:proxy_url_ext] -> Union[str, inherit]

V2.8.5:

- Inital tracking of changes for the changelog.
- **Added:**
  - `ctime`: float
  - `depth`: float
  - `mtime`: float
  - `parent`: str
  - `uid`: str
  - `apt_components`: list/str?
  - `apt_dists`: list/str?
  - `arch`: str
  - `breed`: str
  - `comment`: str
  - `createrepo_flags`: Union[dict, inherit]
  - `environment`: dict
  - `keep_updated`: bool
  - `mirror`: str

- mirror_locally: bool
- name: str
- owners: Union[list, SETTINGS:default_ownership]
- priority: int
- proxy: Union[str, inherit, SETTINGS:proxy_url_ext]
- rpm_list: list
- yumopts: dict

class cobbler.items.repo.**Repo**(*api:* CobblerAPI, **args: Any*, ***kwargs: Any*)

Bases: *Item*

A Cobbler repo object.

COLLECTION_TYPE = 'repo'

TYPE_NAME = 'repo'

property apt_components: List[str]

Specify the section of Debian to mirror. Defaults to "main,contrib,non-free,main/debian-installer".

Getter
If empty the default is used.

Setter
May be a comma delimited `str` or a real `list`.

property apt_dists: List[str]

This decides which installer images are downloaded. For more information please see: https://www.debian.org/CD/mirroring/index.html or the manpage of `debmirror`.

Getter
Per default no images are mirrored.

Setter
Either a comma delimited `str` or a real `list`.

property arch: *RepoArchs*

Override the arch used for reposync

Getter
The repo arch enum object.

Setter
May throw a `ValueError` or `TypeError` in case the conversion of the value is unsuccessful.

property breed: *RepoBreeds*

The repository system breed. This decides some defaults for most actions with a repo in Cobbler.

Getter
The breed detected.

Setter
May raise a `ValueError` or `TypeError` in case the given value is wrong.

check_if_valid() → None

Checks if the object is valid. Currently checks for name and mirror to be present.

Raises
CX – In case the name or mirror is missing.

property createrepo_flags: str

Flags passed to createrepo when it is called. Common flags to use would be `-c cache` or `-g comps.xml` to generate group information.

Note: This property can be set to <<inherit>>.

Getter
The createrepo_flags to apply to the repo.

Setter
The new flags. May raise a `TypeError` in case the options are not a `str`.

property environment: Dict[Any, Any]

Yum can take options from the environment. This puts them there before each reposync.

Getter
The options to be attached to the environment.

Setter
May raise a `ValueError` in case the data provided is not parsable.

property keep_updated: bool

This allows the user to disable updates to a particular repo for whatever reason.

Getter
True in case the repo is updated automatically and False otherwise.

Setter
Is auto-converted to a bool via multiple types. Raises a `TypeError` if this was not possible.

make_clone() → *Repo*

Clone this file object. Please manually adjust all value yourself to make the cloned object unique.

Returns
The cloned instance of this object.

property mirror: str

A repo is (initially, as in right now) is something that can be rsynced. reposync/repotrack integration over HTTP might come later.

Getter
The mirror uri.

Setter
May raise a `TypeError` in case we run into

property mirror_locally: bool

If this property is set to `True` then all content of the source is mirrored locally. This may take up a lot of disk space.

Getter
Whether the mirror is locally available or not.

Setter
Raises a `TypeError` in case after the conversion of the value is not of type `bool`.

property mirror_type: *MirrorType*

Override the mirror_type used for reposync

Getter
The mirror type. Is one of the predefined ones.

Setter
Hand over a str or enum type value to this. May raise `TypeError` or `ValueError` in case there are conversion or type problems.

`property os_version: str`

The operating system version which is compatible with this repository.

Getter
The os version.

Setter
The version as a `str`.

`property priority: int`

Set the priority of the repository. Only works if host is using priorities plugin for yum.

Getter
The priority of the repo.

Setter
A number between 1 & 99. May raise otherwise `TypeError` or `ValueError`.

`property proxy: str`

Override the default external proxy which is used for accessing the internet.

Note: This property can be set to <<inherit>>.

Getter
Returns the default one or the specific one for this repository.

Setter
May raise a `TypeError` in case the wrong value is given.

`property rpm_list: List[str]`

Rather than mirroring the entire contents of a repository (Fedora Extras, for instance, contains games, and we probably don't want those), make it possible to list the packages one wants out of those repos, so only those packages and deps can be mirrored.

Getter
The list of packages to be mirrored.

Setter
May be a space delimited list or a real one.

`property rsyncopts: Dict[Any, Any]`

Options for `rsync` when being used for repo management.

Getter
The options to apply to the generated ones.

Setter
A str or dict to replace the old options with. If the str can't be parsed we throw a `ValueError`.

`property yumopts: Dict[Any, Any]`

Options for the yum tool. Should be presented in the same way as the `kernel_options`.

Getter
The dict with the parsed options.

Setter
Either the dict or a str which is then parsed. If parsing is unsuccessful then a ValueError is raised.

cobbler.items.system module

All code belonging to Cobbler systems. This includes network interfaces.

Changelog (NetworkInterface):

V3.4.0 (unreleased):

- **Changes:**
  - Constructor: `kwargs` can now be used to seed the item during creation.
  - `virt_type`: str - Inheritable; One of "qemu", "kvm", "xenpv", "xenfv", "vmware", "vmwarew", "openvz" or "auto".

V3.3.4 (unreleased):

- No changes

V3.3.3:

- **Changed:**
  - `to_dict()`: Accepts new parameter `resolved`
  - `virt_bridge`: Can now be set to `<<inherit>>` to get its value from the settings key `default_virt_bridge`

V3.3.2:

- No changes

V3.3.1:

- No changes

V3.3.0:

- This release switched from pure attributes to properties (getters/setters).
- **Added:**
  - `NetworkInterface` is now a class.
  - Serialization still happens inside the system collection.
  - Properties have been used.

V3.2.2:

- No changes

V3.2.1:

- No changes

V3.2.0:

- No changes

V3.1.2:

- No changes

V3.1.1:

- No changes

V3.1.0:

- No changes

V3.0.1:

- No changes

V3.0.0:

- Field defintions now split of `System` class

V2.8.5:

- Inital tracking of changes for the changelog.
- Field definitions part of `System` class
- **Added:**
  - `mac_address`: str
  - `connected_mode`: bool
  - `mtu`: str
  - `ip_address`: str
  - `interface_type`: str - One of "na", "bond", "bond_slave", "bridge", bridge_slave", "bonded_bridge_slave", "infiniband"
  - `interface_master`: str
  - `bonding_opts`: str
  - `bridge_opts`: str
  - `management`: bool
  - `static`: bool
  - `netmask`: str
  - `if_gateway`: str
  - `dhcp_tag`: str
  - `dns_name`: str
  - `static_routes`: List[str]
  - `virt_bridge`: str
  - `ipv6_address`: str
  - `ipv6_prefix`: str
  - `ipv6_secondaries`: List[str]
  - `ipv6_mtu`: str
  - `ipv6_static_routes`: List[str]
  - `ipv6_default_gateway`: str
  - `cnames`: List[str]

Changelog (System):

V3.4.0 (unreleased):

- **Added:**
  - `display_name`: str
- **Changes:**
  - Constructor: `kwargs` can now be used to seed the item during creation.
  - `from_dict()`: The method was moved to the base class.
  - `parent`: The property was moved to the base class.

V3.3.4 (unreleased):

- **Changed:**
  - The network interface `default` is not created on object creation.

V3.3.3:

- **Changed:**
  - `boot_loaders`: Can now be set to <<inherit>>
  - `next_server_v4`: Can now be set to <<inhertit>>
  - `next_server_v6`: Can now be set to <<inhertit>>
  - `virt_cpus`: Can now be set to <<inhertit>>
  - `virt_file_size`: Can now be set to <<inhertit>>
  - `virt_disk_driver`: Can now be set to <<inhertit>>
  - `virt_auto_boot`: Can now be set to <<inhertit>>
  - `virt_ram`: Can now be set to <<inhertit>>
  - `virt_type`: Can now be set to <<inhertit>>
  - `virt_path`: Can now be set to <<inhertit>>

V3.3.2:

- No changes

V3.3.1:

- **Changed:**
  - `serial_device`: Default value is now `-1`

V3.3.0:

- This release switched from pure attributes to properties (getters/setters).
- **Added:**
  - `next_server_v4`
  - `next_server_v6`
- **Changed:**
  - `virt_*`: Cannot be set to inherit anymore
  - `enable_gpxe`: Renamed to `enable_ipxe`
- **Removed:**
  - `get_fields()`
  - `next_server` - Please use one of `next_server_v4` or `next_server_v6`
  - `set_boot_loader()` - Moved to `boot_loader` property
  - `set_server()` - Moved to `server` property
  - `set_next_server()` - Moved to `next_server` property
  - `set_filename()` - Moved to `filename` property
  - `set_proxy()` - Moved to `proxy` property
  - `set_redhat_management_key()` - Moved to `redhat_management_key` property
  - `get_redhat_management_key()` - Moved to `redhat_management_key` property
  - `set_dhcp_tag()` - Moved to `NetworkInterface` class property `dhcp_tag`
  - `set_cnames()` - Moved to `NetworkInterface` class property `cnames`

- set_status() - Moved to status property
- set_static() - Moved to NetworkInterface class property static
- set_management() - Moved to NetworkInterface class property management
- set_dns_name() - Moved to NetworkInterface class property dns_name
- set_hostname() - Moved to hostname property
- set_ip_address() - Moved to NetworkInterface class property ip_address
- set_mac_address() - Moved to NetworkInterface class property mac_address
- set_gateway() - Moved to gateway property
- set_name_servers() - Moved to name_servers property
- set_name_servers_search() - Moved to name_servers_search property
- set_netmask() - Moved to NetworkInterface class property netmask
- set_if_gateway() - Moved to NetworkInterface class property if_gateway
- set_virt_bridge() - Moved to NetworkInterface class property virt_bridge
- set_interface_type() - Moved to NetworkInterface class property interface_type
- set_interface_master() - Moved to NetworkInterface class property interface_master
- set_bonding_opts() - Moved to NetworkInterface class property bonding_opts
- set_bridge_opts() - Moved to NetworkInterface class property bridge_opts
- set_ipv6_autoconfiguration() - Moved to ipv6_autoconfiguration property
- set_ipv6_default_device() - Moved to ipv6_default_device property
- set_ipv6_address() - Moved to NetworkInterface class property ipv6_address
- set_ipv6_prefix() - Moved to NetworkInterface class property ipv6_prefix
- set_ipv6_secondaries() - Moved to NetworkInterface class property ipv6_secondaries
- set_ipv6_default_gateway() - Moved to NetworkInterface class property ipv6_default_gateway
- set_ipv6_static_routes() - Moved to NetworkInterface class property ipv6_static_routes
- set_ipv6_mtu() - Moved to NetworkInterface class property ipv6_mtu
- set_mtu() - Moved to NetworkInterface class property mtu
- set_connected_mode() - Moved to NetworkInterface class property connected_mode
- set_enable_gpxe() - Moved to enable_gpxe property
- set_profile() - Moved to profile property
- set_image() - Moved to image property
- set_virt_cpus() - Moved to virt_cpus property
- set_virt_file_size() - Moved to virt_file_size property
- set_virt_disk_driver() - Moved to virt_disk_driver property
- set_virt_auto_boot() - Moved to virt_auto_boot property
- set_virt_pxe_boot() - Moved to virt_pxe_boot property
- set_virt_ram() - Moved to virt_ram property

  - set_virt_type() - Moved to virt_type property
  - set_virt_path() - Moved to virt_path property
  - set_netboot_enabled() - Moved to netboot_enabled property
  - set_autoinstall() - Moved to autoinstall property
  - set_power_type() - Moved to power_type property
  - set_power_identity_file() - Moved to power_identity_file property
  - set_power_options() - Moved to power_options property
  - set_power_user() - Moved to power_user property
  - set_power_pass() - Moved to power_pass property
  - set_power_address() - Moved to power_address property
  - set_power_id() - Moved to power_id property
  - set_repos_enabled() - Moved to repos_enabled property
  - set_serial_device() - Moved to serial_device property
  - set_serial_baud_rate() - Moved to serial_baud_rate property

V3.2.2:

- No changes

V3.2.1:

- **Added:**
  - kickstart: Resolves as a proxy to autoinstall

V3.2.0:

- No changes

V3.1.2:

- **Added:**
  - filename: str - Inheritable
  - set_filename()

V3.1.1:

- No changes

V3.1.0:

- No changes

V3.0.1:

- File was moved from cobbler/item_system.py to cobbler/items/system.py.

V3.0.0:

- Field definitions for network interfaces moved to own FIELDS array
- **Added:**
  - boot_loader: str - Inheritable
  - next_server: str - Inheritable
  - power_options: str
  - power_identity_file: str
  - serial_device: int

  - serial_baud_rate: int - One of "", "2400", "4800", "9600", "19200", "38400", "57600", "115200"
  - set_next_server()
  - set_serial_device()
  - set_serial_baud_rate()
  - get_config_filename()
  - set_power_identity_file()
  - set_power_options()
- **Changed:**
  - kickstart: Renamed to autoinstall
  - ks_meta: Renamed to autoinstall_meta
  - from_datastruct: Renamed to from_dict()
  - set_kickstart(): Renamed to set_autoinstall()
- **Removed:**
  - redhat_management_server
  - set_ldap_enabled()
  - set_monit_enabled()
  - set_template_remote_kickstarts()
  - set_redhat_management_server()
  - set_name()

V2.8.5:

- Inital tracking of changes for the changelog.
- Network interface defintions part of this class
- **Added:**
  - name: str
  - uid: str
  - owners: List[str] - Inheritable
  - profile: str - Name of the profile
  - image: str - Name of the image
  - status: str - One of "", "development", "testing", "acceptance", "production"
  - kernel_options: Dict[str, Any]
  - kernel_options_post: Dict[str, Any]
  - ks_meta: Dict[str, Any]
  - enable_gpxe: bool - Inheritable
  - proxy: str - Inheritable
  - netboot_enabled: bool
  - kickstart: str - Inheritable
  - comment: str
  - depth: int

- server: str - Inheritable
- virt_path: str - Inheritable
- virt_type: str - Inheritable; One of "xenpv", "xenfv", "qemu", "kvm", "vmware", "openvz"
- virt_cpus: int - Inheritable
- virt_file_size: float - Inheritable
- virt_disk_driver: str - Inheritable; One of "<<inherit>>", "raw", "qcow", "qcow2", "aio", "vmdk", "qed"
- virt_ram: int - Inheritable
- virt_auto_boot: bool - Inheritable
- virt_pxe_boot: bool
- ctime: float
- mtime: float
- power_type: str - Default loaded from settings key power_management_default_type
- power_address: str
- power_user: str
- power_pass: str
- power_id: str
- hostname: str
- gateway: str
- name_servers: List[str]
- name_servers_search: List[str]
- ipv6_default_device: str
- ipv6_autoconfiguration: bool
- mgmt_classes: List[Any] - Inheritable
- mgmt_parameters: str - Inheritable
- boot_files: Dict[str, Any]/List (Not reverse engineeriable) - Inheritable
- fetchable_files: Dict[str, Any] - Inheritable
- template_files: Dict[str, Any] - Inheritable
- redhat_management_key: str - Inheritable
- redhat_management_server: str - Inheritable
- template_remote_kickstarts: bool - Default loaded from settings key template_remote_kickstarts
- repos_enabled: bool
- ldap_enabled: - bool
- ldap_type: str - Default loaded from settings key ldap_management_default_type
- monit_enabled: bool

class cobbler.items.system.**NetworkInterface**(*api:* CobblerAPI, **args: Any*, ***kwargs: Any*)

Bases: object

A subobject of a Cobbler System which represents the network interfaces

property bonding_opts: str

bonding_opts property.

Getter
Returns the value for `bonding_opts`.

Setter
Sets the value for the property `bonding_opts`.

property bridge_opts: str

bridge_opts property.

Getter
Returns the value for `bridge_opts`.

Setter
Sets the value for the property `bridge_opts`.

property cnames: List[str]

cnames property.

Getter
Returns the value for `cnames`.

Setter
Sets the value for the property `cnames`.

property connected_mode: bool

connected_mode property.

Getter
Returns the value for `connected_mode`.

Setter
Sets the value for the property `connected_mode`.

deserialize(*interface_dict: Dict[str, Any]*)

This is currently a proxy for *from_dict()* .

Parameters
interface_dict – The dictionary with the data to deserialize.

property dhcp_tag: str

dhcp_tag property.

Getter
Returns the value for `dhcp_tag`.

Setter
Sets the value for the property `dhcp_tag`.

property dns_name: str

dns_name property.

Getter
Returns the value for ``dns_name`.

Setter
Sets the value for the property `dns_name`.

from_dict(*dictionary: Dict[str, Any]*)

Initializes the object with attributes from the dictionary.

Parameters
dictionary – The dictionary with values.

property if_gateway: str

if_gateway property.

Getter
Returns the value for `if_gateway`.

Setter
Sets the value for the property `if_gateway`.

property interface_master: str

interface_master property.

Getter
Returns the value for `interface_master`.

Setter
Sets the value for the property `interface_master`.

property interface_type: *NetworkInterfaceType*

interface_type property.

Getter
Returns the value for `interface_type`.

Setter
Sets the value for the property `interface_type`.

property ip_address: str

ip_address property.

Getter
Returns the value for `ip_address`.

Setter
Sets the value for the property `ip_address`.

property ipv6_address: str

ipv6_address property.

Getter
Returns the value for `ipv6_address`.

Setter
Sets the value for the property `ipv6_address`.

property ipv6_default_gateway: str

ipv6_default_gateway property.

Getter
Returns the value for `ipv6_default_gateway`.

Setter
Sets the value for the property `ipv6_default_gateway`.

property ipv6_mtu: str

ipv6_mtu property.

Getter
Returns the value for `ipv6_mtu`.

Setter
Sets the value for the property `ipv6_mtu`.

property ipv6_prefix: str

ipv6_prefix property.

Getter
Returns the value for `ipv6_prefix`.

Setter
Sets the value for the property `ipv6_prefix`.

property ipv6_secondaries: List[str]

ipv6_secondaries property.

Getter
Returns the value for `ipv6_secondaries`.

Setter
Sets the value for the property `ipv6_secondaries`.

property ipv6_static_routes: List[str]

ipv6_static_routes property.

Getter
Returns the value for `ipv6_static_routes`.

Setter
Sets the value for the property *ipv6_static_routes `*.

property mac_address: str

mac_address property.

Getter
Returns the value for `mac_address`.

Setter
Sets the value for the property `mac_address`.

property management: bool

management property.

Getter
Returns the value for `management`.

Setter
Sets the value for the property `management`.

modify_interface(*_dict: Dict[str, Any]*)

Modify the interface

Parameters
_dict – The dict with the parameter.

property mtu: str

mtu property.

Getter
Returns the value for `mtu`.

Setter
Sets the value for the property `mtu`.

property netmask: str

netmask property.

Getter
Returns the value for `netmask`.

Setter
Sets the value for the property `netmask`.

serialize() → Dict[str, Any]

This method is a proxy for *to_dict()* and contains additional logic for serialization to a persistent location.

Returns
The dictionary with the information for serialization.

property static: bool

static property.

Getter
Returns the value for `static`.

Setter
Sets the value for the property `static`.

property static_routes: List[str]

static_routes property.

Getter
Returns the value for `static_routes`.

Setter
Sets the value for the property `static_routes`.

to_dict(*resolved: bool = False*) → Dict[str, Any]

This converts everything in this object to a dictionary.

Parameters
resolved – If this is True, Cobbler will resolve the values to its final form, rather than give you the objects raw value.

Returns
A dictionary with all values present in this object.

property virt_bridge: str

virt_bridge property. If set to <<inherit>> this will read the value from the setting "default_virt_bridge".

Getter
Returns the value for `virt_bridge`.

Setter
Sets the value for the property `virt_bridge`.

class cobbler.items.system.**System**(*api: CobblerAPI*, **args: Any*, ***kwargs: Any*)

Bases: *Item*

A Cobbler system object.

COLLECTION_TYPE = 'system'

TYPE_NAME = 'system'

property autoinstall: str

autoinstall property.

Getter
Returns the value for `autoinstall`.

Setter
Sets the value for the property `autoinstall`.

property boot_loaders: List[str]

boot_loaders property.

Note: This property can be set to <<inherit>>.

Getter
Returns the value for boot_loaders.

Setter
Sets the value for the property boot_loaders.

check_if_valid()

Checks if the current item passes logical validation.

Raises
CX – In case name is missing. Additionally either image or profile is required.

delete_interface(*name: Union[str, Dict[Any, Any]]*) → None

Used to remove an interface.

Raises
TypeError – If the name of the interface is not of type str or dict.

property display_name: str

Returns the display name.

Getter
Returns the display name for the boot menu.

Setter
Sets the display name for the boot menu.

property enable_ipxe: bool

enable_ipxe property.

Note: This property can be set to <<inherit>>.

Getter
Returns the value for enable_ipxe.

Setter
Sets the value for the property enable_ipxe.

property filename: str

filename property.

Getter
Returns the value for filename.

Setter
Sets the value for the property filename.

property gateway

gateway property.

Getter
Returns the value for gateway.

Setter
Sets the value for the property gateway.

get_config_filename(*interface: str*, *loader: Optional[str] = None*) → Optional[str]

The configuration file for each system pxe uses is either a form of the MAC address or the hex version or the IP address. If none of that is available, just use the given name, though the name given will be unsuitable for PXE configuration (For this, check system.is_management_supported()). This same file is used to store system config information in the Apache tree, so it's still relevant.

Parameters

- **interface** – Name of the interface.
- **loader** – Bootloader type.

get_ip_address(*interface: str*) → str

Get the IP address for the given interface.

Parameters
interface – The name of the interface to get the IP address of.

get_mac_address(*interface: str*)

Get the mac address, which may be implicit in the object name or explicit with –mac-address. Use the explicit location first.

Parameters
interface – The name of the interface to get the MAC of.

property hostname: str

hostname property.

Getter
Returns the value for `hostname`.

Setter
Sets the value for the property `hostname`.

property image: str

image property.

Getter
Returns the value for `image`.

Setter
Sets the value for the property `image`.

property interfaces: Dict[str, *NetworkInterface*]

Represents all interfaces owned by the system.

Getter
The interfaces present. Has at least the `default` one.

Setter
Accepts not only the correct type but also a dict with dicts which will then be converted by the setter.

property ipv6_autoconfiguration: bool

ipv6_autoconfiguration property.

Getter
Returns the value for `ipv6_autoconfiguration`.

Setter
Sets the value for the property `ipv6_autoconfiguration`.

property ipv6_default_device: str

ipv6_default_device property.

Getter
Returns the value for `ipv6_default_device`.

Setter
Sets the value for the property `ipv6_default_device`.

`is_management_supported`(*cidr_ok: bool = True*) → bool

Can only add system PXE records if a MAC or IP address is available, else it's a koan only record.

Parameters
cidr_ok – Deprecated parameter which is not used anymore.

`make_clone`()

Must be defined in any subclass

`modify_interface`(*interface_values: Dict[str, Any]*)

Modifies a magic interface dictionary in the form of: {"macaddress-eth0" : "aa:bb:cc:dd:ee:ff"}

`property name_servers: List[str]`

name_servers property. FIXME: Differentiate between IPv4/6

Getter
Returns the value for `name_servers`.

Setter
Sets the value for the property `name_servers`.

`property name_servers_search: List[str]`

name_servers_search property.

Getter
Returns the value for `name_servers_search`.

Setter
Sets the value for the property `name_servers_search`.

`property netboot_enabled: bool`

netboot_enabled property.

Getter
Returns the value for `netboot_enabled`.

Setter
Sets the value for the property `netboot_enabled`.

`property next_server_v4: str`

next_server_v4 property.

Note: This property can be set to <<inherit>>.

Getter
Returns the value for `next_server_v4`.

Setter
Sets the value for the property `next_server_v4`.

`property next_server_v6: str`

next_server_v6 property.

Note: This property can be set to <<inherit>>.

Getter
Returns the value for `next_server_v6`.

Setter
Sets the value for the property `next_server_v6`.

property power_address: str

power_address property.

Getter
Returns the value for `power_address`.

Setter
Sets the value for the property `power_address`.

property power_id: str

power_id property.

Getter
Returns the value for `power_id`.

Setter
Sets the value for the property `power_id`.

property power_identity_file: str

power_identity_file property.

Getter
Returns the value for `power_identity_file`.

Setter
Sets the value for the property `power_identity_file`.

property power_options: str

power_options property.

Getter
Returns the value for `power_options`.

Setter
Sets the value for the property `power_options`.

property power_pass: str

power_pass property.

Getter
Returns the value for `power_pass`.

Setter
Sets the value for the property `power_pass`.

property power_type: str

power_type property.

Getter
Returns the value for `power_type`.

Setter
Sets the value for the property `power_type`.

property power_user: str

power_user property.

Getter
Returns the value for `power_user`.

Setter
Sets the value for the property `power_user`.

property profile: str

profile property.

Getter
Returns the value for profile.

Setter
Sets the value for the property profile.

property proxy: str

proxy property. This corresponds per default to the setting``proxy_url_int``.

Note: This property can be set to <<inherit>>.

Getter
Returns the value for proxy.

Setter
Sets the value for the property proxy.

property redhat_management_key: str

redhat_management_key property.

Note: This property can be set to <<inherit>>.

Getter
Returns the value for redhat_management_key.

Setter
Sets the value for the property redhat_management_key.

rename_interface(*old_name: str*, *new_name: str*)

Used to rename an interface.

Raises

- TypeError – In case on of the params was not a str.
- ValueError – In case the name for the old interface does not exist or the new name does.

property repos_enabled: bool

repos_enabled property.

Getter
Returns the value for repos_enabled.

Setter
Sets the value for the property repos_enabled.

property serial_baud_rate: *BaudRates*

serial_baud_rate property. The value “disabled” will disable the functionality completely.

Getter
Returns the value for serial_baud_rate.

Setter
Sets the value for the property serial_baud_rate.

property serial_device: int

serial_device property. "-1" disables the serial device functionality completely.

Getter
Returns the value for `serial_device`.

Setter
Sets the value for the property `serial_device`.

property server: str

server property.

Note: This property can be set to <<inherit>>.

Getter
Returns the value for `server`.

Setter
Sets the value for the property `server`.

property status: str

status property.

Getter
Returns the value for `status`.

Setter
Sets the value for the property `status`.

property virt_auto_boot: bool

virt_auto_boot property.

Note: This property can be set to <<inherit>>.

Getter
Returns the value for `virt_auto_boot`.

Setter
Sets the value for the property `virt_auto_boot`.

property virt_cpus: int

virt_cpus property.

Note: This property can be set to <<inherit>>.

Getter
Returns the value for `virt_cpus`.

Setter
Sets the value for the property `virt_cpus`.

property virt_disk_driver: *VirtDiskDrivers*

virt_disk_driver property.

Note: This property can be set to <<inherit>>.

Getter
Returns the value for `virt_disk_driver`.

Setter
Sets the value for the property `virt_disk_driver`.

property virt_file_size: *float*

virt_file_size property.

Note: This property can be set to <<inherit>>.

Getter
Returns the value for `virt_file_size`.

Setter
Sets the value for the property `virt_file_size`.

property virt_path: *str*

virt_path property.

Note: This property can be set to <<inherit>>.

Getter
Returns the value for `virt_path`.

Setter
Sets the value for the property `virt_path`.

property virt_pxe_boot: *bool*

virt_pxe_boot property.

Getter
Returns the value for `virt_pxe_boot`.

Setter
Sets the value for the property `virt_pxe_boot`.

property virt_ram: *int*

virt_ram property.

Note: This property can be set to <<inherit>>.

Getter
Returns the value for `virt_ram`.

Setter
Sets the value for the property `virt_ram`.

property virt_type: *VirtType*

virt_type property.

Note: This property can be set to <<inherit>>.

Getter
Returns the value for `virt_type`.

Setter
Sets the value for the property `virt_type`.

Module contents

This package contains all data storage classes. The classes are responsible for ensuring that types of the properties are correct but not for logical checks. The classes should be as stupid as possible. Further they are responsible for returning the logic for serializing and deserializing themselves.

Cobbler has a concept of inheritance where an attribute/a property may have the value `<<inherit>>`. This then takes over the value of the parent item with the exception of dictionaries. Values that are of type `dict` are always implicitly inherited, to remove a key-value pair from the dictionary in the inheritance chain prefix the key with `!`.

8.1.4 cobbler.modules package

Subpackages

cobbler.modules.authentication package

Submodules

cobbler.modules.authentication.configfile module

Authentication module that uses /etc/cobbler/auth.conf Choice of authentication module is in /etc/cobbler/modules.conf

`cobbler.modules.authentication.configfile.`**`authenticate`**(*api_handle:* CobblerAPI, *username:* *str*, *password:* *str*) → bool

Validate a username/password combo.

Thanks to https://trac.edgewall.org/ticket/845 for supplying the algorithm info.

Parameters

- **`api_handle`** – Unused in this implementation.
- **`username`** – The username to log in with. Must be contained in /etc/cobbler/users.digest
- **`password`** – The password to log in with. Must be contained hashed in /etc/cobbler/users.digest

Returns
A boolean which contains the information if the username/password combination is correct.

`cobbler.modules.authentication.configfile.`**`hashfun`**(*api:* CobblerAPI, *text:* *str*) → str

Converts a str object to a hash which was configured in modules.conf of the Cobbler settings.

Parameters

- **`api`** – CobblerAPI
- **`text`** – The text to hash.

Returns
The hash of the text. This should output the same hash when entered the same text.

`cobbler.modules.authentication.configfile.`**`register`**() → str

The mandatory Cobbler module registration hook.

cobbler.modules.authentication.denyall module

Authentication module that denies everything. Used to disable the WebUI by default.

cobbler.modules.authentication.denyall.**authenticate**(*api_handle: CobblerAPI*, *username: str*, *password: str*) → bool

Validate a username/password combo, always returning false.

Returns
False

cobbler.modules.authentication.denyall.**register**() → str

The mandatory Cobbler module registration hook.

cobbler.modules.authentication.ldap module

Authentication module that uses ldap Settings in /etc/cobbler/authn_ldap.conf Choice of authentication module is in /etc/cobbler/modules.conf

cobbler.modules.authentication.ldap.**authenticate**(*api_handle: CobblerAPI*, *username: str*, *password: str*) → bool

Validate an LDAP bind, returning whether the authentication was successful or not.

Parameters

- **api_handle** – The api instance to resolve settings.
- **username** – The username to authenticate.
- **password** – The password to authenticate.

Returns
True if the ldap server authentication was a success, otherwise false.

Raises
CX – Raised in case the LDAP search bind credentials are missing in the settings.

cobbler.modules.authentication.ldap.**register**() → str

The mandatory Cobbler module registration hook.

Returns
Always “authn”

cobbler.modules.authentication.pam module

Authentication module that uses /etc/cobbler/auth.conf Choice of authentication module is in /etc/cobbler/modules.conf

PAM python code based on the pam_python code created by Chris AtLee: https://atlee.ca/software/pam/

#———————————————— pam_python (c) 2007 Chris AtLee <chris@atlee.ca> Licensed under the MIT license: https://www.opensource.org/licenses/mit-license.php

PAM module for python

Provides an authenticate function that will allow the caller to authenticate a user against the Pluggable Authentication Modules (PAM) on the system.

Implemented using ctypes, so no compilation is necessary.

class cobbler.modules.authentication.pam.**PamConv**

Bases: Structure

wrapper class for pam_conv structure

appdata_ptr

Structure/Union member

conv

Structure/Union member

class cobbler.modules.authentication.pam.**PamHandle**

Bases: Structure

wrapper class for pam_handle_t

handle

Structure/Union member

class cobbler.modules.authentication.pam.**PamMessage**

Bases: Structure

wrapper class for pam_message structure

msg

Structure/Union member

msg_style

Structure/Union member

class cobbler.modules.authentication.pam.**PamResponse**

Bases: Structure

wrapper class for pam_response structure

resp

Structure/Union member

resp_retcode

Structure/Union member

cobbler.modules.authentication.pam.**authenticate**(*api_handle:* CobblerAPI, *username:* *str*, *password:* *str*) → bool

Validate PAM authentication, returning whether the authentication was successful or not.

Parameters

- **api_handle** – Used for resolving the pam service name and getting the Logger.
- **username** – The username to log in with.
- **password** – The password to log in with.

Returns

True if the given username and password authenticate for the given service. Otherwise False

cobbler.modules.authentication.pam.**register**() → str

The mandatory Cobbler module registration hook.

cobbler.modules.authentication.passthru module

Authentication module that defers to Apache and trusts what Apache trusts.

cobbler.modules.authentication.passthru.**authenticate**(*api_handle:* CobblerAPI, *username:* *str*, *password:* *str*) → bool

Validate a username/password combo. Uses cobbler_auth_helper

Parameters

- **api_handle** – This parameter is not used currently.

- **username** – This parameter is not used currently.
- **password** – This should be the internal Cobbler secret.

Returns
True if the password is the secret, otherwise false.

cobbler.modules.authentication.passthru.**register**() → str

The mandatory Cobbler module registration hook.

Returns
Always "authn"

cobbler.modules.authentication.spacewalk module

Authentication module that uses Spacewalk's auth system. Any org_admin or kickstart_admin can get in.

cobbler.modules.authentication.spacewalk.**authenticate**(*api_handle: CobblerAPI*, *username: str*, *password: str*) → bool

Validate a username/password combo. This will pass the username and password back to Spacewalk to see if this authentication request is valid.

See also: https://github.com/uyuni-project/uyuni/blob/c9b7285117822af96c223cb0b6e0ae96ec7f0837/java/code/src/com/redhat/rhn/frontend/xmlrpc/auth/AuthHandler.java#L107

Parameters

- **api_handle** – The api instance to retrieve settings of.
- **username** – The username to authenticate against spacewalk/uyuni/SUSE Manager
- **password** – The password to authenticate against spacewalk/uyuni/SUSE Manager

Returns
True if it succeeded, False otherwise.

Raises
CX – Raised in case `api_handle` is missing.

cobbler.modules.authentication.spacewalk.**register**() → str

The mandatory Cobbler module registration hook.

Module contents

This module represents all Cobbler methods of authentication. All present modules may be used through the configuration file `modules.conf` normally found at `/etc/cobbler/`.

In the following the specification of an authentication module is given:

1. The name of the only public method - except the generic `register()` method - must be `authenticate`
2. The attributes are - in exactly this order: `api_handle`, `username`, `password`
3. The username and password both must be of type `str`.
4. The `api_handle` must be the main `CobblerAPI` instance.
5. The return value of the module must be a `bool`.
6. The method should only return `True` in case the authentication is successful.
7. Errors should result in the return of `False` and a log message to the standard Python logger obtioned via `logging.getLogger()`.
8. The return value of `register()` must be `authn`.

The list of currently known authentication modules is:

- authentication.configfile
- authentication.denyall
- authentication.ldap
- authentication.pam
- authentication.passthru
- authentication.spacewalk

cobbler.modules.authorization package

Submodules

cobbler.modules.authorization.allowall module

Authorization module that allows everything, which is the default for new Cobbler installs.

cobbler.modules.authorization.allowall.**authorize**(*api_handle: CobblerAPI*, *user: str*, *resource: str*, *arg1: Any = None*, *arg2: Any = None*) → int

Validate a user against a resource. NOTE: acls are not enforced as there is no group support in this module

Parameters

- **api_handle** – This parameter is not used currently.
- **user** – This parameter is not used currently.
- **resource** – This parameter is not used currently.
- **arg1** – This parameter is not used currently.
- **arg2** – This parameter is not used currently.

Returns

Always 1

cobbler.modules.authorization.allowall.**register**() → str

The mandatory Cobbler module registration hook.

Returns

Always "authz"

cobbler.modules.authorization.configfile module

Authorization module that allow users listed in /etc/cobbler/users.conf to be permitted to access resources. For instance, when using authz_ldap, you want to use authn_configfile, not authz_allowall, which will most likely NOT do what you want.

cobbler.modules.authorization.configfile.**authorize**(*api_handle: CobblerAPI*, *user: str*, *resource: str*, *arg1: Any = None*, *arg2: Any = None*) → int

Validate a user against a resource. All users in the file are permitted by this module.

Parameters

- **api_handle** – This parameter is not used currently.
- **user** – The user to authorize.
- **resource** – This parameter is not used currently.

- **arg1** – This parameter is not used currently.
- **arg2** – This parameter is not used currently.

Returns
"0" if no authorized, "1" if authorized.

cobbler.modules.authorization.configfile.**register**() → str

The mandatory Cobbler module registration hook.

Returns
Always "authz".

cobbler.modules.authorization.ownership module

Authorization module that allow users listed in /etc/cobbler/users.conf to be permitted to access resources, with the further restriction that Cobbler objects can be edited to only allow certain users/groups to access those specific objects.

cobbler.modules.authorization.ownership.**authorize**(*api_handle: CobblerAPI*, *user: str*, *resource: str*, *arg1: Optional[str] = None*, *arg2: Any = None*) → int

Validate a user against a resource. All users in the file are permitted by this module.

Parameters

- **api_handle** – The api to resolve required information.
- **user** – The user to authorize to the resource.
- **resource** – The resource the user is asking for access. This is something abstract like a remove operation.
- **arg1** – This is normally the name of the specific object in question.
- **arg2** – This parameter is pointless currently. Reserved for future code.

Returns
1 if okay, otherwise `0`.

cobbler.modules.authorization.ownership.**register**() → str

The mandatory Cobbler module registration hook.

Returns
Always "authz"

Module contents

This module represents all Cobbler methods of authorization. All present modules may be used through the configuration file `modules.conf` normally found at `/etc/cobbler/`.

In the following the specification of an authorization module is given:

1. The name of the only public method - except the generic `register()` method - must be `authorize`
2. The attributes are - in exactly that order: `api_handle`, `user`, `resource`, `arg1`, `arg2`
3. The `api_handle` must be the main `CobblerAPI` instance.
4. The `user` and `resource` attribute must be of type `str`.
5. The attributes `arg1` and `arg2` are reserved for the individual use of your authorization module and may have any type and form your desire.
6. The method must return an integer in all cases.

7. The method should return `1` for success and `0` for an authorization failure.
8. Additional codes can be defined, however they should be documented in the module description.
9. The values of additional codes should be positive integers.
10. Errors should result in the return of `-1` and a log message to the standard Python logger obtioned via `logging.getLogger()`.
11. The return value of `register()` must be `authz`.

cobbler.modules.installation package

Submodules

cobbler.modules.installation.post_log module

Cobbler Module Trigger that will mark a system as installed in `cobbler status`.

`cobbler.modules.installation.post_log.`**`register`**`()` → str

The mandatory Cobbler module registration hook.

`cobbler.modules.installation.post_log.`**`run`**`(`*api:* CobblerAPI, *args:* *List[str]*`)` → int

The method runs the trigger, meaning this logs that an installation has ended.

The list of args should have three elements:

- 0: system or profile
- 1: the name of the system or profile
- 2: the ip or a "?"

Parameters

- **`api`** – This parameter is unused currently.
- **`args`** – An array of three elements. Type (system/profile), name and ip. If no ip is present use a ?.

Returns

Always 0

cobbler.modules.installation.post_power module

Post install trigger for Cobbler to power cycle the guest if needed

`class` `cobbler.modules.installation.post_power.`**`RebootSystemThread`**`(`*api:* CobblerAPI, *target:* System`)`

Bases: `Thread`

TODO

`run``()` → None

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

`cobbler.modules.installation.post_power.`**`register`**`()` → str

The mandatory Cobbler module registration hook.

cobbler.modules.installation.post_power.**run**(*api:* CobblerAPI, *args: List[str]*) → int

Obligatory trigger hook.

Parameters

- **api** – The api to resolve information with.
- **args** – This is an array containing two objects. 0: The str "system". All other content will result in an early exit of the trigger. 1: The name of the target system.

Returns

0 on success.

cobbler.modules.installation.post_puppet module

This module signs newly installed client puppet certificates if the puppet master server is running on the same machine as the Cobbler server.

Based on: https://www.ithiriel.com/content/2010/03/29/writing-install-triggers-cobbler

cobbler.modules.installation.post_puppet.**register**() → str

The mandatory Cobbler module registration hook.

cobbler.modules.installation.post_puppet.**run**(*api:* CobblerAPI, *args: List[str]*) → int

The obligatory Cobbler modules hook.

Parameters

- **api** – The api to resolve all information with.
- **args** – This is an array with two items. The first must be `system`, if the value is different we do an early and the second is the name of this system or profile.

Returns

0 or nothing.

cobbler.modules.installation.post_report module

Post install trigger for Cobbler to send out a pretty email report that contains target information.

cobbler.modules.installation.post_report.**register**() → str

The mandatory Cobbler module registration hook.

Returns

Always `/var/lib/cobbler/triggers/install/post/*`.

cobbler.modules.installation.post_report.**run**(*api:* CobblerAPI, *args: List[str]*) → int

This is the mandatory Cobbler module run trigger hook.

Parameters

- **api** – The api to resolve information with.
- **args** – This is an array with three elements. 0: "system" or "profile" 1: name of target or profile 2: ip or "?"

Returns

0 or 1.

Raises

CX – Raised if the blender result is empty.

cobbler.modules.installation.pre_clear_anamon_logs module

Cobbler Module Trigger that will clear the anamon logs.

cobbler.modules.installation.pre_clear_anamon_logs.**register**() → str

This pure python trigger acts as if it were a legacy shell-trigger, but is much faster. The return of this method indicates the trigger type.

Returns
Always `/var/lib/cobbler/triggers/install/pre/*`

cobbler.modules.installation.pre_clear_anamon_logs.**run**(*api:* CobblerAPI, *args: List[str]*) → int

The list of args should have one element:

- 1: the name of the system or profile

Parameters

- **api** – The api to resolve metadata with.
- **args** – This should be a list as described above.

Returns
"0" on success.

Raises
CX – Raised in case of missing arguments.

cobbler.modules.installation.pre_log module

TODO

cobbler.modules.installation.pre_log.**register**() → str

This pure python trigger acts as if it were a legacy shell-trigger, but is much faster. The return of this method indicates the trigger type.

Returns
Always */var/lib/cobbler/triggers/install/pre/**

cobbler.modules.installation.pre_log.**run**(*api:* CobblerAPI, *args: List[str]*) → int

The method runs the trigger, meaning this logs that an installation has started.

The list of args should have three elements:

- 0: system or profile
- 1: the name of the system or profile
- 2: the ip or a "?"

Parameters

- **api** – This parameter is currently unused.
- **args** – Already described above.

Returns
A "0" on success.

cobbler.modules.installation.pre_puppet module

This module removes puppet certs from the puppet master prior to reinstalling a machine if the puppet master is running on the Cobbler server.

Based on: https://www.ithiriel.com/content/2010/03/29/writing-install-triggers-cobbler

cobbler.modules.installation.pre_puppet.**register**() → str

This pure python trigger acts as if it were a legacy shell-trigger, but is much faster. The return of this method indicates the trigger type.

Returns
Always */var/lib/cobbler/triggers/install/pre/**

cobbler.modules.installation.pre_puppet.**run**(*api: CobblerAPI*, *args: List[str]*) → int

This method runs the trigger, meaning in this case that old puppet certs are automatically removed via puppetca.

The list of args should have two elements:

- 0: system or profile
- 1: the name of the system or profile

Parameters

- **api** – The api to resolve external information with.
- **args** – Already described above.

Returns
"0" on success. If unsuccessful this raises an exception.

Module contents

This module contains Python triggers for Cobbler. With Cobbler one is able to add custom actions and commands after many events happening in Cobbler. The Python modules presented here are an example of what can be done after certain events. Custom triggers may be added in any language as long as Cobbler is allowed to execute them. If implemented in Python they need to follow the following specification:

- Expose a method called `register()` which returns a `str` and returns the path of the trigger in the filesystem.
- Expose a method called `run(api, args)` of type `int`. The integer would represent the exit status of an e.g. shell script. Thus 0 means success and anything else a failure.

cobbler.modules.managers package

Submodules

cobbler.modules.managers.bind module

This is some of the code behind 'cobbler sync'.

class cobbler.modules.managers.bind.**MetadataZoneHelper**(*forward_zones: List[str]*, *reverse_zones: List[Tuple[str, str]]*, *zone_include: str*)

Bases: `object`

Helper class to hold data for template rendering of named config files.

cobbler.modules.managers.bind.**get_manager**(*api:* CobblerAPI) → _BindManager

This returns the object to manage a BIND server located locally on the Cobbler server.

Parameters
api – The API to resolve all information with.

Returns
The BindManger object to manage bind with.

cobbler.modules.managers.bind.**register**() → str

The mandatory Cobbler module registration hook.

cobbler.modules.managers.dnsmasq module

This is some of the code behind 'cobbler sync'.

cobbler.modules.managers.dnsmasq.**get_manager**(*api:* CobblerAPI) → _DnsmasqManager

Creates a manager object to manage a dnsmasq server.

Parameters
api – The API to resolve all information with.

Returns
The object generated from the class.

cobbler.modules.managers.dnsmasq.**register**() → str

The mandatory Cobbler modules registration hook.

Returns
Always "manage".

cobbler.modules.managers.genders module

Cobbler Module that manages the cluster configuration tool from CHAOS. For more information please see: GitHub - chaos/genders

cobbler.modules.managers.genders.**register**() → str

We should run anytime something inside of Cobbler changes.

Returns
Always `/var/lib/cobbler/triggers/change/*`

cobbler.modules.managers.genders.**run**(*api:* CobblerAPI, *args:* *Any*) → int

Mandatory Cobbler trigger hook.

Parameters

- **api** – The api to resolve information with.
- **args** – For this implementation unused.

Returns
`0` or `1`, depending on the outcome of the operation.

cobbler.modules.managers.genders.**write_genders_file**(*config:* CobblerAPI, *profiles_genders:* *Dict[str, str]*, *distros_genders:* *Dict[str, str]*, *mgmtcls_genders:* *Dict[str, str]*)

Genders file is over-written when `manage_genders` is set in our settings.

Parameters

- **config** – The API instance to template the data with.
- **profiles_genders** – The profiles which should be included.

- **distros_genders** – The distros which should be included.
- **mgmtcls_genders** – The management classes which should be included.

Raises
OSError – Raised in case the template could not be read.

cobbler.modules.managers.import_signatures module

cobbler.modules.managers.in_tftpd module

This is some of the code behind 'cobbler sync'.

cobbler.modules.managers.in_tftpd.**get_manager**(*api:* CobblerAPI) → _InTftpdManager

Creates a manager object to manage an in_tftp server.

Parameters
api – The API which holds all information in the current Cobbler instance.

Returns
The object to manage the server with.

cobbler.modules.managers.in_tftpd.**register**() → str

The mandatory Cobbler module registration hook.

cobbler.modules.managers.isc module

This is some of the code behind 'cobbler sync'.

cobbler.modules.managers.isc.**get_manager**(*api:* CobblerAPI) → _IscManager

Creates a manager object to manage an isc dhcp server.

Parameters
api – The API which holds all information in the current Cobbler instance.

Returns
The object to manage the server with.

cobbler.modules.managers.isc.**register**() → str

The mandatory Cobbler module registration hook.

cobbler.modules.managers.ndjbdns module

This is some of the code behind 'cobbler sync'.

cobbler.modules.managers.ndjbdns.**get_manager**(*api:* CobblerAPI) → _NDjbDnsManager

Creates a manager object to manage an isc dhcp server.

Parameters
api – The API which holds all information in the current Cobbler instance.

Returns
The object to manage the server with.

cobbler.modules.managers.ndjbdns.**register**() → str

The mandatory Cobbler module registration hook.

Module contents

This module contains extensions for services Cobbler is managing. The services are restarted via the `service` command or alternatively through the server executables directly. Cobbler does not announce the restarts but is expecting to be allowed to do this on its own at any given time. Thus all services managed by Cobbler should not be touched by any other tool or administrator.

class cobbler.modules.managers.**DhcpManagerModule**(*api:* CobblerAPI)

Bases: *ManagerModule*

TODO

abstract sync_dhcp() → None

TODO

class cobbler.modules.managers.**DnsManagerModule**(*api:* CobblerAPI)

Bases: *ManagerModule*

TODO

abstract regen_hosts() → None

TODO

class cobbler.modules.managers.**ManagerModule**(*api:* CobblerAPI)

Bases: object

Base class for Manager modules located in `modules/manager/*.py`

These are typically but not necessarily used to manage systemd services. Enabling can be done via settings `manage_*` (e.g. `manage_dhcp`) and `restart_*` (e.g. `restart_dhcp`). Different modules could manage the same functionality as dhcp can be managed via isc.py or dnsmasq.py (compare with `/etc/cobbler/modules.py`).

regen_ethers() → None

ISC/BIND doesn't use this. It is there for compatibility reasons with other managers.

restart_service() → int

Write module specific config files. E.g. dhcp manager would write `/etc/dhcpd.conf` here

sync() → int

This syncs the manager's server (systemd service) with it's new config files. Basically this restarts the service to apply the changes.

Returns
Integer return value of restart_service - 0 on success

static what() → str

Static method to identify the manager module. Must be overwritten by the inheriting class

write_configs() → None

Write module specific config files. E.g. dhcp manager would write `/etc/dhcpd.conf` here

class cobbler.modules.managers.**TftpManagerModule**(*api:* CobblerAPI)

Bases: *ManagerModule*

TODO

abstract add_single_distro(*distro:* Distro) → None

TODO

Parameters
distro – TODO

abstract sync_single_system(*system:* System, *menu_items: Optional[Dict[str, Union[str, Dict[str, str]]]] = None*) → None

TODO

Parameters

- **system** – TODO
- **menu_items** – TODO

abstract sync_systems(*systems: List[str]*, *verbose: bool = True*) → None

TODO

Parameters

- **systems** – TODO
- **verbose** – TODO

abstract write_boot_files() → int

TODO

cobbler.modules.serializers package

Submodules

cobbler.modules.serializers.file module

Cobbler's file-based object serializer. As of 9/2014, this is Cobbler's default serializer and the most stable one. It uses multiple JSON files in /var/lib/cobbler/collections/distros, profiles, etc

class cobbler.modules.serializers.file.**FileSerializer**(*api:* CobblerAPI)

Bases: *StorageBase*

TODO

deserialize(*collection:* Collection*[ITEM]*, *topological: bool = True*) → None

Load a collection from disk.

Parameters

- **collection** – The Cobbler collection to know the type of the item.
- **topological** – Sort collection based on each items' depth attribute in the list of collection items. This ensures properly ordered object loading from disk with objects having parent/child relationships, i.e. profiles/subprofiles. See cobbler/items/item.py

deserialize_item(*collection_type: str*, *name: str*) → Dict[str, Any]

Get a collection item from disk and parse it into an object.

Parameters

- **collection_type** – The collection type to fetch.
- **name** – collection Item name

Returns

Dictionary of the collection item.

deserialize_raw(*collection_type: str*) → List[Dict[str, Any]]

Read the collection from the disk or read the settings file.

Parameters

collection_type – The collection type to read.

Returns
The list of collection dicts or settings dict.

serialize(*collection:* Collection*[ITEM]*) → None
Save a collection to disk

Parameters
collection – The collection to serialize.

serialize_delete(*collection:* Collection*[ITEM]*, *item: ITEM*) → None
Delete a collection item from disk.

Parameters
- **collection** – collection
- **item** – collection item

serialize_item(*collection:* Collection*[ITEM]*, *item: ITEM*) → None
Save a collection item to disk

Parameters
- **collection** – The Cobbler collection to know the type of the item.
- **item** – The collection item to serialize.

cobbler.modules.serializers.file.**register**() → str
The mandatory Cobbler module registration hook.

cobbler.modules.serializers.file.**storage_factory**(*api:* CobblerAPI) → *FileSerializer*
TODO

cobbler.modules.serializers.file.**what**() → str
Module identification function

cobbler.modules.serializers.mongodb module

Cobbler's Mongo database based object serializer.

class cobbler.modules.serializers.mongodb.**MongoDBSerializer**(*api:* CobblerAPI)
Bases: *StorageBase*

TODO

deserialize(*collection:* Collection*[ITEM]*, *topological: bool = True*) → None
Load a collection from disk.

Parameters
- **collection** – The Cobbler collection to know the type of the item.
- **topological** – Sort collection based on each items' depth attribute in the list of collection items. This ensures properly ordered object loading from disk with objects having parent/child relationships, i.e. profiles/subprofiles. See cobbler/items/item.py

deserialize_item(*collection_type: str*, *name: str*) → Dict[str, Any]
Get a collection item from database.

Parameters
- **collection_type** – The collection type to fetch.
- **name** – collection Item name

Returns
Dictionary of the collection item.

deserialize_raw(*collection_type: str*) → Union[List[Optional[Dict[str, Any]]], Dict[str, Any]]

Read the collection from the disk or read the settings file.

Parameters
collection_type – The collection type to read.

Returns
The list of collection dicts or settings dict.

serialize(*collection:* Collection*[ITEM]*) → None

Save a collection to disk

Parameters
collection – The collection to serialize.

serialize_delete(*collection:* Collection*[ITEM]*, *item: ITEM*) → None

Delete a collection item from disk.

Parameters

- **collection** – collection
- **item** – collection item

serialize_item(*collection:* Collection*[ITEM]*, *item: ITEM*) → None

Save a collection item to disk

Parameters

- **collection** – The Cobbler collection to know the type of the item.
- **item** – The collection item to serialize.

cobbler.modules.serializers.mongodb.**register**() → str

The mandatory Cobbler module registration hook.

cobbler.modules.serializers.mongodb.**storage_factory**(*api:* CobblerAPI) → *MongoDBSerializer*

TODO

cobbler.modules.serializers.mongodb.**what**() → str

Module identification function

cobbler.modules.serializers.sqlite module

Cobbler's SQLite database based object serializer.

class cobbler.modules.serializers.sqlite.**SQLiteSerializer**(*api:* CobblerAPI)

Bases: *StorageBase*

Each collection is stored in a separate table named distros, profiles, etc. Tables are created on demand, when the first object of this type is written.

TABLE name // name from collection.collection_types() (

name TEXT PRIMARY KEY, // name from item.name item TEXT // JSON representation of an object

)

deserialize(*collection:* Collection*[ITEM]*, *topological: bool = True*) → None

Load a collection from disk.

Parameters

- **collection** – The Cobbler collection to know the type of the item.

- **topological** – Sort collection based on each items' depth attribute in the list of collection items. This ensures properly ordered object loading from disk with objects having parent/child relationships, i.e. profiles/subprofiles. See cobbler/items/item.py

deserialize_item(*collection_type: str*, *name: str*) → Dict[str, Any]

Get a collection item from disk and parse it into an object.

Parameters

- **collection_type** – The collection type to deserialize.
- **item_name** – The collection item name to deserialize.

Returns
Dictionary of the collection item.

deserialize_raw(*collection_type: str*) → Union[List[Optional[Dict[str, Any]]], Dict[str, Any]]

Read the collection from the table or read the settings file.

Parameters
collection_type – The collection type to read.

Returns
The list of collection dicts or settings dict.

serialize(*collection:* Collection*[ITEM]*) → None

Save a collection to disk

Parameters
collection – The collection to serialize.

serialize_delete(*collection:* Collection*[ITEM]*, *item: ITEM*) → None

Delete a collection item from table

Parameters

- **collection** – The Cobbler collection to know the type of the item.
- **item** – The collection item to delete.

serialize_item(*collection:* Collection*[ITEM]*, *item: ITEM*) → None

Save a collection item to table

Parameters

- **collection** – The Cobbler collection to know the type of the item.
- **item** – The collection item to serialize.

cobbler.modules.serializers.sqlite.**register**() → str

The mandatory Cobbler module registration hook.

cobbler.modules.serializers.sqlite.**storage_factory**(*api:* CobblerAPI) → *SQLiteSerializer*

TODO

cobbler.modules.serializers.sqlite.**what**() → str

Module identification function

Module contents

This module contains code to persist the in memory state of Cobbler on a target. The name of the target should be the name of the Python file. Cobbler is currently only tested against the file serializer.

class cobbler.modules.serializers.**StorageBase**(*api:* CobblerAPI)

Bases: object

TODO

deserialize(*collection:* Collection*[ITEM]*, *topological:* *bool* = *True*) → None

Load a collection from disk.

Parameters

- **collection** – The Cobbler collection to know the type of the item.
- **topological** – Sort collection based on each items' depth attribute in the list of collection items. This ensures properly ordered object loading from disk with objects having parent/child relationships, i.e. profiles/subprofiles. See cobbler/items/item.py

deserialize_item(*collection_type:* *str*, *name:* *str*) → Dict[str, Any]

Get a collection item from disk and parse it into an object.

Parameters

- **collection_type** – The collection type to deserialize.
- **item_name** – The collection item name to deserialize.

Returns
Dictionary of the collection item.

deserialize_raw(*collection_type:* *str*) → Union[List[Optional[Dict[str, Any]]], Dict[str, Any]]

Read the collection from the disk or read the settings file.

Parameters
collection_type – The collection type to read.

Returns
The list of collection dicts or settings dict.

serialize(*collection:* Collection*[ITEM]*) → None

Save a collection to disk

Parameters
collection – The collection to serialize.

serialize_delete(*collection:* Collection*[ITEM]*, *item:* *ITEM*) → None

Delete a collection item from disk.

Parameters

- **collection** – collection
- **item** – collection item

serialize_item(*collection:* Collection*[ITEM]*, *item:* *ITEM*) → None

Save a collection item to disk

Parameters

- **collection** – The Cobbler collection to know the type of the item.
- **item** – The collection item to serialize.

cobbler.modules.serializers.**register**() → str

TODO

cobbler.modules.serializers.**storage_factory**(*api:* CobblerAPI) → *StorageBase*

TODO

cobbler.modules.serializers.**what**() → str

TODO

Submodules

cobbler.modules.nsupdate_add_system_post module

Replace (or remove) records in DNS zone for systems created (or removed) by Cobbler

cobbler.modules.nsupdate_add_system_post.**nslog**(*msg: str*) → None

Log a message to the logger.

Parameters
msg – The message to log.

cobbler.modules.nsupdate_add_system_post.**register**() → str

This method is the obligatory Cobbler registration hook.

Returns
The trigger name or an empty string.

cobbler.modules.nsupdate_add_system_post.**run**(*api:* CobblerAPI, *args: List[Any]*)

This method executes the trigger, meaning in this case that it updates the dns configuration.

Parameters

- **api** – The api to read metadata from.
- **args** – Metadata to log.

Returns
"0" on success or a skipped task. If the task failed or problems occurred then an exception is raised.

cobbler.modules.nsupdate_delete_system_pre module

Replace (or remove) records in DNS zone for systems created (or removed) by Cobbler

cobbler.modules.nsupdate_delete_system_pre.**nslog**(*msg: str*) → None

Log a message to the logger.

Parameters
msg – The message to log.

cobbler.modules.nsupdate_delete_system_pre.**register**() → str

This method is the obligatory Cobbler registration hook.

Returns
The trigger name or an empty string.

cobbler.modules.nsupdate_delete_system_pre.**run**(*api:* CobblerAPI, *args: List[Any]*)

This method executes the trigger, meaning in this case that it updates the dns configuration.

Parameters

- **api** – The api to read metadata from.
- **args** – Metadata to log.

Returns
"0" on success or a skipped task. If the task failed or problems occurred then an exception is raised.

cobbler.modules.scm_track module

Cobbler Trigger Module that puts the content of the Cobbler data directory under version control. Depending on `scm_track_mode` in the settings, this can either be git or Mercurial.

`cobbler.modules.scm_track.`**`register`**`()` → str

This pure python trigger acts as if it were a legacy shell-trigger, but is much faster. The return of this method indicates the trigger type :return: Always: `/var/lib/cobbler/triggers/change/*`

`cobbler.modules.scm_track.`**`run`**`(`*api:* CobblerAPI, *args:* *Any*`)`

Runs the trigger, meaning in this case track any changed which happen to a config or data file.

Parameters

- **`api`** – The api instance of the Cobbler server. Used to look up if scm_track_enabled is true.
- **`args`** – The parameter is currently unused for this trigger.

Returns

0 on success, otherwise an exception is risen.

cobbler.modules.sync_post_restart_services module

Restarts the DHCP and/or DNS after a Cobbler sync to apply changes to the configuration files.

`cobbler.modules.sync_post_restart_services.`**`register`**`()` → str

This pure python trigger acts as if it were a legacy shell-trigger, but is much faster. The return of this method indicates the trigger type

Returns

Always `/var/lib/cobbler/triggers/sync/post/*`

`cobbler.modules.sync_post_restart_services.`**`run`**`(`*api:* CobblerAPI, *args:* *List[str]*`)` → int

Run the trigger via this method, meaning in this case that depending on the settings dns and/or dhcp services are restarted.

Parameters

- **`api`** – The api to resolve settings.
- **`args`** – This parameter is not used currently.

Returns

The return code of the service restarts.

cobbler.modules.sync_post_wingen module

Create Windows boot files

To create Windows boot files, files are used that must be extracted from the distro. The `cobbler import`" command extracts the required files and places them where the given trigger expects them to be found.

To create boot files per profile/system, the trigger uses the following metadata from `--autoinstall-meta`:

- **`kernel` - the name of the bootstrap file for profile/system, can be:**
  - any filename, in the case of PXE boot without using `wimboot` which is not the same as the filename for other profiles/systems of that distro. The trigger creates it from a copy of `pxeboot.n12` by replacing the `bootmgr.exe` string in the binary copy with the `bootmgr` metadata value. In the case of Windows XP/2003, it replaces the `NTLDR` string.
  - in case of PXE boot using `wimboot`, specify the path to `wimboot` in the file system, e.g `/var/lib/tftpboot/wimboot`

  - in case of iPXE boot using `wimboot`, specify the path to `wimboot` in the file system or any url that supports iPXE, e.g `http://@@http_server@@/cobbler/images/@@distro_name@@/wimboot`

- `bootmgr` - filename of the Boot Manager for the profile/system. The trigger creates it by copying `bootmgr.exe` and replacing the `BCD` string in the binary copy with the string specified in the `bcd` metadata parameter. The filename must be exactly 11 characters long, e.g. `bootmg1.exe`, `bootmg2.exe`, .. and not match the names for other profiles/systems of the same distro. For Windows XP/2003, `setupldr.exe` is used as the Boot Manager and the string `winnt.sif` is replaced in its copy.
- `bcd` - The name of the Windows Boot Configuration Data (BCD) file for the profile/system. Must be exactly 3 characters and not the same as names for other profiles/systems on the same distro, e.g. `000`, `001`, etc.
- `winpe` - The name of the Windows PE image file for the profile/system. The trigger copies it from the distro and replaces the `/Windows/System32/startnet.cmd` file in it with the one created from the `startnet.template` template. Filenames must be unique per the distro.
- `answerfile` - the name of the answer file for the Windows installation, e.g. `autounattend01.xml` or`` win01.sif`` for Windows XP/2003. The trigger creates the answerfile from the `answerfile.template`. Filenames must be unique per the distro.
- `post_install_script` - The name of the post-installation script file that will be run after Windows is installed. To run a script, its filename is substituted into the answerfile template. Any valid Windows commands can be used in the script, but its usual purpose is to download and run the script for the profile from `http://@@http_server@@/cblr/svc/op/autoinstall/profile/@@profile_name@@`, for this the script is passed profile name as parameter . The post-installation script is created by a trigger from the `post_inst_cmd.template` template in the `sources/$OEM$/$1` distro directory only if it exists. The Windows Installer copies the contents of this directory to the target host during installation.
- any other key/value pairs that can be used in `startnet.template`, `answerfile.template`, `post_inst_cmd.template` templates

cobbler.modules.sync_post_wingen.**bcdedit**(*orig_bcd: str*, *new_bcd: str*, *wim: str*, *sdi: str*, *startoptions: Optional[str] = None*)

Create new Windows Boot Configuration Data (BCD) based on Microsoft BCD extracted from a WIM image.

Parameters

- **orig_bcd** – Path to the original BCD
- **new_bcd** – Path to the new customized BCD
- **wim** – Path to the WIM image
- **sdi** – Path to the System Deployment Image (SDI)
- **startoptions** – Other BCD options

Returns

cobbler.modules.sync_post_wingen.**register**() → Optional[str]

This pure python trigger acts as if it were a legacy shell-trigger, but is much faster. The return of this method indicates the trigger type :return: Always `/var/lib/cobbler/triggers/sync/post/*`

cobbler.modules.sync_post_wingen.**run**(*api: CobblerAPI*, *args: Any*)

Runs the trigger, meaning in this case creates Windows boot files.

Parameters

- **api** – The api instance of the Cobbler server. Used to look up if windows_enabled is true.
- **args** – The parameter is currently unused for this trigger.

Returns
0 on success, otherwise an exception is risen.

Module contents

This part of Cobbler may be utilized by any plugins which are extending Cobbler and core code which can be exchanged through the `modules.conf` file.

A Cobbler module is loaded if it has a method called `register()`. The method must return a `str` which represents the module category.

8.1.5 cobbler.settings package

Subpackages

cobbler.settings.migrations package

Submodules

cobbler.settings.migrations.V2_8_5 module

Migration from V2.x.x to V2.8.5

`cobbler.settings.migrations.V2_8_5.`**`migrate`**`(`*settings: Dict[str, Any]*`)` → Dict[str, Any]

Migration of the settings `settings` to the V2.8.5 settings

Parameters
`settings` – The settings dict to migrate

Returns
The migrated dict

`cobbler.settings.migrations.V2_8_5.`**`normalize`**`(`*settings: Dict[str, Any]*`)` → Dict[str, Any]

If data in `settings` is valid the validated data is returned.

Parameters
`settings` – The settings dict to validate.

Returns
The validated dict.

`cobbler.settings.migrations.V2_8_5.`**`validate`**`(`*settings: Dict[str, Any]*`)` → bool

Checks that a given settings dict is valid according to the reference schema `schema`.

Parameters
`settings` – The settings dict to validate.

Returns
True if valid settings dict otherwise False.

cobbler.settings.migrations.V3_0_0 module

Migration from V2.8.5 to V3.0.0

cobbler.settings.migrations.V3_0_0.**migrate**(*settings: Dict[str, Any]*) → Dict[str, Any]

Migration of the settings `settings` to the V3.0.0 settings

Parameters
settings – The settings dict to migrate

Returns
The migrated dict

cobbler.settings.migrations.V3_0_0.**normalize**(*settings: Dict[str, Any]*) → Dict[str, Any]

If data in `settings` is valid the validated data is returned.

Parameters
settings – The settings dict to validate.

Returns
The validated dict.

cobbler.settings.migrations.V3_0_0.**validate**(*settings: Dict[str, Any]*) → bool

Checks that a given settings dict is valid according to the reference schema `schema`.

Parameters
settings – The settings dict to validate.

Returns
True if valid settings dict otherwise False.

cobbler.settings.migrations.V3_0_1 module

Migration from V3.0.0 to V3.0.1

cobbler.settings.migrations.V3_0_1.**migrate**(*settings: Dict[str, Any]*) → Dict[str, Any]

Migration of the settings `settings` to the V3.0.1 settings

Parameters
settings – The settings dict to migrate

Returns
The migrated dict

cobbler.settings.migrations.V3_0_1.**normalize**(*settings: Dict[str, Any]*) → Dict[str, Any]

If data in `settings` is valid the validated data is returned.

Parameters
settings – The settings dict to validate.

Returns
The validated dict.

cobbler.settings.migrations.V3_0_1.**validate**(*settings: Dict[str, Any]*) → bool

Checks that a given settings dict is valid according to the reference schema `schema`.

Parameters
settings – The settings dict to validate.

Returns
True if valid settings dict otherwise False.

cobbler.settings.migrations.V3_1_0 module

Migration from V3.0.1 to V3.1.0

cobbler.settings.migrations.V3_1_0.**migrate**(*settings: Dict[str, Any]*) → Dict[str, Any]

Migration of the settings `settings` to the V3.1.0 settings

Parameters
settings – The settings dict to migrate

Returns
The migrated dict

cobbler.settings.migrations.V3_1_0.**normalize**(*settings: Dict[str, Any]*) → Dict[str, Any]

If data in `settings` is valid the validated data is returned.

Parameters
settings – The settings dict to validate.

Returns
The validated dict.

cobbler.settings.migrations.V3_1_0.**validate**(*settings: Dict[str, Any]*) → bool

Checks that a given settings dict is valid according to the reference schema `schema`.

Parameters
settings – The settings dict to validate.

Returns
True if valid settings dict otherwise False.

cobbler.settings.migrations.V3_1_1 module

Migration from V3.1.0 to V3.1.1

cobbler.settings.migrations.V3_1_1.**migrate**(*settings: Dict[str, Any]*) → Dict[str, Any]

Migration of the settings `settings` to the V3.1.1 settings

Parameters
settings – The settings dict to migrate

Returns
The migrated dict

cobbler.settings.migrations.V3_1_1.**normalize**(*settings: Dict[str, Any]*) → Dict[str, Any]

If data in `settings` is valid the validated data is returned.

Parameters
settings – The settings dict to validate.

Returns
The validated dict.

cobbler.settings.migrations.V3_1_1.**validate**(*settings: Dict[str, Any]*) → bool

Checks that a given settings dict is valid according to the reference schema `schema`.

Parameters
settings – The settings dict to validate.

Returns
True if valid settings dict otherwise False.

cobbler.settings.migrations.V3_1_2 module

Migration from V3.1.1 to V3.1.2

cobbler.settings.migrations.V3_1_2.**migrate**(*settings: Dict[str, Any]*) → Dict[str, Any]

Migration of the settings `settings` to the V3.1.2 settings

Parameters
settings – The settings dict to migrate

Returns
The migrated dict

cobbler.settings.migrations.V3_1_2.**normalize**(*settings: Dict[str, Any]*) → Dict[str, Any]

If data in `settings` is valid the validated data is returned.

Parameters
settings – The settings dict to validate.

Returns
The validated dict.

cobbler.settings.migrations.V3_1_2.**validate**(*settings: Dict[str, Any]*) → bool

Checks that a given settings dict is valid according to the reference schema `schema`.

Parameters
settings – The settings dict to validate.

Returns
True if valid settings dict otherwise False.

cobbler.settings.migrations.V3_2_0 module

Migration from V3.1.2 to V3.2.0

cobbler.settings.migrations.V3_2_0.**migrate**(*settings: Dict[str, Any]*) → Dict[str, Any]

Migration of the settings `settings` to the V3.2.0 settings

Parameters
settings – The settings dict to migrate

Returns
The migrated dict

cobbler.settings.migrations.V3_2_0.**normalize**(*settings: Dict[str, Any]*) → Dict[str, Any]

If data in `settings` is valid the validated data is returned.

Parameters
settings – The settings dict to validate.

Returns
The validated dict.

cobbler.settings.migrations.V3_2_0.**validate**(*settings: Dict[str, Any]*) → bool

Checks that a given settings dict is valid according to the reference schema `schema`.

Parameters
settings – The settings dict to validate.

Returns
True if valid settings dict otherwise False.

cobbler.settings.migrations.V3_2_1 module

Migration from V3.2.0 to V3.2.1

cobbler.settings.migrations.V3_2_1.**migrate**(*settings: Dict[str, Any]*) → Dict[str, Any]

Migration of the settings `settings` to the V3.2.1 settings

Parameters
settings – The settings dict to migrate

Returns
The migrated dict

cobbler.settings.migrations.V3_2_1.**normalize**(*settings: Dict[str, Any]*) → Dict[str, Any]

If data in `settings` is valid the validated data is returned.

Parameters
settings – The settings dict to validate.

Returns
The validated dict.

cobbler.settings.migrations.V3_2_1.**validate**(*settings: Dict[str, Any]*) → bool

Checks that a given settings dict is valid according to the reference schema `schema`.

Parameters
settings – The settings dict to validate.

Returns
True if valid settings dict otherwise False.

cobbler.settings.migrations.V3_3_0 module

Migration from V3.2.1 to V3.3.0

cobbler.settings.migrations.V3_3_0.**migrate**(*settings: Dict[str, Any]*) → Dict[str, Any]

Migration of the settings `settings` to version V3.3.0 settings

Parameters
settings – The settings dict to migrate

Returns
The migrated dict

cobbler.settings.migrations.V3_3_0.**migrate_cobbler_collections**(*collections_dir: str*) → None

Manipulate the main Cobbler stored collections and migrate deprecated settings to work with newer Cobbler versions.

Parameters
collections_dir – The directory of Cobbler where the collections files are.

cobbler.settings.migrations.V3_3_0.**normalize**(*settings: Dict[str, Any]*) → Dict[str, Any]

If data in `settings` is valid the validated data is returned.

Parameters
settings – The settings dict to validate.

Returns
The validated dict.

cobbler.settings.migrations.V3_3_0.**validate**(*settings: Dict[str, Any]*) → bool

Checks that a given settings dict is valid according to the reference V3.3.0 schema `schema`.

Parameters
settings – The settings dict to validate.

Returns
True if valid settings dict otherwise False.

cobbler.settings.migrations.V3_3_1 module

Migration from V3.3.0 to V3.3.1

cobbler.settings.migrations.V3_3_1.**migrate**(*settings: Dict[str, Any]*) → Dict[str, Any]
Migration of the settings `settings` to version V3.3.1 settings

Parameters
settings – The settings dict to migrate

Returns
The migrated dict

cobbler.settings.migrations.V3_3_1.**normalize**(*settings: Dict[str, Any]*) → Dict[str, Any]
If data in `settings` is valid the validated data is returned.

Parameters
settings – The settings dict to validate.

Returns
The validated dict.

cobbler.settings.migrations.V3_3_1.**validate**(*settings: Dict[str, Any]*) → bool
Checks that a given settings dict is valid according to the reference V3.3.1 schema `schema`.

Parameters
settings – The settings dict to validate.

Returns
True if valid settings dict otherwise False.

cobbler.settings.migrations.V3_3_2 module

Migration from V3.3.1 to V3.3.2

cobbler.settings.migrations.V3_3_2.**migrate**(*settings: Dict[str, Any]*) → Dict[str, Any]
Migration of the settings `settings` to version V3.3.1 settings

Parameters
settings – The settings dict to migrate

Returns
The migrated dict

cobbler.settings.migrations.V3_3_2.**normalize**(*settings: Dict[str, Any]*) → Dict[str, Any]
If data in `settings` is valid the validated data is returned.

Parameters
settings – The settings dict to validate.

Returns
The validated dict.

cobbler.settings.migrations.V3_3_2.**validate**(*settings: Dict[str, Any]*) → bool
Checks that a given settings dict is valid according to the reference V3.3.1 schema `schema`.

Parameters
settings – The settings dict to validate.

Returns
True if valid settings dict otherwise False.

cobbler.settings.migrations.V3_3_3 module

Migration from V3.3.2 to V3.3.3

cobbler.settings.migrations.V3_3_3.**migrate**(*settings: Dict[str, Any]*) → Dict[str, Any]

Migration of the settings `settings` to version V3.3.1 settings

Parameters
settings – The settings dict to migrate

Returns
The migrated dict

cobbler.settings.migrations.V3_3_3.**normalize**(*settings: Dict[str, Any]*) → Dict[str, Any]

If data in `settings` is valid the validated data is returned.

Parameters
settings – The settings dict to validate.

Returns
The validated dict.

cobbler.settings.migrations.V3_3_3.**validate**(*settings: Dict[str, Any]*) → bool

Checks that a given settings dict is valid according to the reference V3.3.1 schema `schema`.

Parameters
settings – The settings dict to validate.

Returns
True if valid settings dict otherwise False.

cobbler.settings.migrations.V3_4_0 module

Migration from V3.3.1 to V3.3.2

cobbler.settings.migrations.V3_4_0.**migrate**(*settings: Dict[str, Any]*) → Dict[str, Any]

Migration of the settings `settings` to version V3.4.0 settings

Parameters
settings – The settings dict to migrate

Returns
The migrated dict

cobbler.settings.migrations.V3_4_0.**migrate_cobbler_collections**(*collections_dir: str*) → None

Manipulate the main Cobbler stored collections and migrate deprecated settings to work with newer Cobbler versions.

Parameters
collections_dir – The directory of Cobbler where the collections files are.

cobbler.settings.migrations.V3_4_0.**normalize**(*settings: Dict[str, Any]*) → Dict[str, Any]

If data in `settings` is valid the validated data is returned.

Parameters
settings – The settings dict to validate.

Returns
The validated dict.

cobbler.settings.migrations.V3_4_0.**validate**(*settings: Dict[str, Any]*) → bool

Checks that a given settings dict is valid according to the reference V3.4.0 schema `schema`.

Parameters
settings – The settings dict to validate.

Returns
True if valid settings dict otherwise False.

cobbler.settings.migrations.helper module

Helper module which contains shared logic for adjusting the settings.

class cobbler.settings.migrations.helper.**Setting**(*location: Union[str, List[str]]*, *value: Any*)

Bases: object

Specifies a setting object

property key_name: str

Returns the location.

static split_str_location(*location: str*) → List[str]

Split the given location at "." Necessary for nesting in our settings file

Parameters
location – Can be "manage.dhcp_v4" or "restart.dhcp_v4" for example.

cobbler.settings.migrations.helper.**backup_dir**(*dir_path: str*) → None

Copies the directory tree and adds a suffix ".backup.XXXXXXXXX" to it.

Parameters
dir_path – The full path to the directory which should be backed up.

Raises
FileNotFoundError – In case the path specified was not existing.

cobbler.settings.migrations.helper.**key_add**(*new:* Setting, *settings: Dict[str, Any]*) → None

Add a new settings key.

Parameters

- **new** – The new setting to add.
- **settings** – [description]

cobbler.settings.migrations.helper.**key_delete**(*delete: str*, *settings: Dict[str, Any]*) → None

Deletes a given setting

Parameters

- **delete** – The name of the setting to be deleted.
- **settings** – The settings dict where the key should be deleted.

cobbler.settings.migrations.helper.**key_drop_if_default**(*settings: Dict[str, Any]*, *defaults: Dict[str, Any]*) → Dict[str, Any]

Drop all keys which values are identical to the default ones.

Parameters

- **settings** – The current settings read from an external source
- **defaults** – The full settings with default values

cobbler.settings.migrations.helper.**key_get**(*key: str*, *settings: Dict[str, Any]*) → *Setting*

Get a key from the settings

Parameters

- **key** – The key to get in the form "a.b.c"
- **settings** – The dict to operate on.

Returns
The desired key from the settings dict

cobbler.settings.migrations.helper.**key_move**(*move:* Setting, *new_location: List[str]*, *settings: Dict[str, Any]*) → None

Delete the old setting and create a new key at `new_location`

Parameters

- **move** – The name of the old key which should be moved.
- **new_location** – The location of the new key
- **settings** – The dict to operate on.

cobbler.settings.migrations.helper.**key_rename**(*old_name:* Setting, *new_name: str*, *settings: Dict[str, Any]*) → None

Wrapper for key_move()

Parameters

- **old_name** – The old name
- **new_name** – The new name
- **settings** –

cobbler.settings.migrations.helper.**key_set_value**(*new:* Setting, *settings: Dict[str, Any]*) → None

Change the value of a setting.

Parameters

- **new** – A Settings object with the new information.
- **settings** – The settings dict.

Module contents

The name of the migration file is the target version. One migration should update from version x to x + 1, where X is any Cobbler version and the migration updates to any next version (e.g. 3.2.1 to 3.3.0). The validation of the current version is in the file with the name of the version.

class cobbler.settings.migrations.**CobblerVersion**(*major: int = 0*, *minor: int = 0*, *patch: int = 0*)

Bases: `object`

Specifies a Cobbler Version

cobbler.settings.migrations.**auto_migrate**(*yaml_dict: Dict[str, Any]*, *settings_path: Path*, *ignore_keys: Optional[List[str]] = None*) → Dict[str, Any]

Auto migration to the most recent version.

Parameters

- **yaml_dict** – The settings dict to migrate.
- **ignore_keys** – The list of ignore keys to exclude from auto migration.
- **settings_path** – The path of the settings dict.

Returns
The migrated dict.

cobbler.settings.migrations.**discover_migrations**(*path: str = '/home/docs/checkouts/readthedocs.org/user_builds/cobbler/checkou*) → None

Discovers the migration module for each Cobbler version and loads it if it is valid according to certain conditions:

- the module must contain the following methods: validate(), normalize(), migrate()
- those version must have a certain signature

Parameters
path – The path of the migration modules, defaults to migrations_path

cobbler.settings.migrations.**filter_settings_to_validate**(*settings: Dict[str, Any]*, *ignore_keys: Optional[List[str]] = None*) → Tuple[Dict[str, Any], Dict[str, Any]]

Separate settings to validate from the ones to exclude from validation according to "ignore_keys" parameter and "extra_settings_list" setting value.

Parameters

- **settings** – The settings dict to validate.
- **ignore_keys** – The list of ignore keys to exclude from validation.

Return data
The filtered settings to validate

Return data_to_exclude
The settings that were excluded from the validation

cobbler.settings.migrations.**get_installed_version**(*filepath: Union[str, Path] = '/etc/cobbler/version'*) → *CobblerVersion*

Retrieve the current Cobbler version. Normally it can be read from /etc/cobbler/version

Parameters
filepath – The filepath of the version file, defaults to "/etc/cobbler/version"

cobbler.settings.migrations.**get_schema**(*version:* CobblerVersion) → Schema

Returns a schema to a given Cobbler version

Parameters
version – The Cobbler version object

Returns
The schema of the Cobbler version

cobbler.settings.migrations.**get_settings_file_version**(*yaml_dict: Dict[str, Any]*, *ignore_keys: Optional[List[str]] = None*) → *CobblerVersion*

Return the correspondig version of the given settings dict.

Parameters

- **yaml_dict** – The settings dict to get the version from.
- **ignore_keys** – The list of ignore keys to exclude from validation.

Returns
The discovered Cobbler Version or `EMPTY_VERSION`

cobbler.settings.migrations.**migrate**(*yaml_dict: Dict[str, Any]*, *settings_path: Path*, *old:* CobblerVersion *= CobblerVersion(major=0, minor=0, patch=0)*, *new:* CobblerVersion *= CobblerVersion(major=0, minor=0, patch=0)*, *ignore_keys: Optional[List[str]] = None*) → Dict[str, Any]

Migration to a specific version. If no old and new version is supplied it will call `auto_migrate()`.

Parameters

- **yaml_dict** – The settings dict to migrate.
- **settings_path** – The path of the settings dict.
- **old** – The version to migrate from, defaults to EMPTY_VERSION.
- **new** – The version to migrate to, defaults to EMPTY_VERSION.
- **ignore_keys** – The list of settings ot be excluded from migration.

Raises
ValueError – Raised if attempting to downgraade.

Returns
The migrated dict.

cobbler.settings.migrations.**normalize**(*settings: Dict[str, Any]*, *ignore_keys: Optional[List[str]] = None*) → Dict[str, Any]

If data in `settings` is valid the validated data is returned.

Parameters

- **settings** – The settings dict to validate.
- **ignore_keys** – The list of settings ot be excluded from normalization.

Returns
The validated dict.

cobbler.settings.migrations.**validate**(*settings: Dict[str, Any]*, *settings_path: Path*, *ignore_keys: Optional[List[str]] = None*) → bool

Wrapper function for the validate() methods of the individual migration modules.

Parameters

- **settings** – The settings dict to validate.
- **settings_path** – TODO: not used at the moment
- **ignore_keys** – The list of settings ot be excluded from validation.

Returns
True if settings are valid, otherwise False.

Module contents

Cobbler app-wide settings

class cobbler.settings.**Settings**

Bases: object

This class contains all app-wide settings of Cobbler. It should only exist once in a Cobbler instance.

static collection_type() → str

This is a hardcoded string which represents the collection type.

Returns
"setting"

static collection_types() → str

return the collection plural name

from_dict(*new_values: Dict[str, Any]*) → Optional[*Settings*]

Modify this object to load values in dictionary. If the handed dict would lead to an invalid object it is silently discarded.

Warning: If the dict from the args has not all settings included Cobbler may behave unexpectedly.

Parameters
new_values – The dictionary with settings to replace.

Returns
Returns the settings instance this method was called from.

`is_valid`() → bool

Silently drops all errors and returns `True` when everything is valid.

Returns
If this settings object is valid this returns true. Otherwise false.

`save`(*filepath: str = '/etc/cobbler/settings.yaml'*, *ignore_keys: Optional[List[str]] = None*) → None

Saves the settings to the disk. :param filepath: This sets the path of the settingsfile to write. :param ignore_keys: The list of ignore keys to exclude from migration.

`to_dict`(*resolved: bool = False*) → Dict[str, Any]

Return an easily serializable representation of the config.

Deprecated since version 3.2.1: Use `obj.__dict__` directly please. Will be removed with 3.3.0

Parameters
resolved – Present for the compatibility with the Cobbler collections.

Returns
The dict with all user settings combined with settings which are left to the default.

`to_string`() → str

Returns the kernel options as a string.

Returns
The multiline string with the kernel options.

cobbler.settings.`migrate`(*yaml_dict: Dict[str, Any]*, *settings_path: Path*, *ignore_keys: Optional[List[str]] = None*) → Dict[str, Any]

Migrates the current settings

Parameters

- **yaml_dict** – The settings dict
- **settings_path** – The settings path
- **ignore_keys** – The list of ignore keys to exclude from migration.

Returns
The migrated settings

cobbler.settings.`read_settings_file`(*filepath: str = '/etc/cobbler/settings.yaml'*, *ignore_keys: Optional[List[str]] = None*) → Dict[str, Any]

Utilizes `read_yaml_file()`. If the read settings file is invalid in the context of Cobbler we will return an empty dictionary.

Parameters

- **filepath** – The path to the settings file.
- **ignore_keys** – The list of ignore keys to exclude from validation.

Raises

- **SchemaMissingKeyError** – In case keys are minssing.
- **SchemaWrongKeyError** – In case keys are not listed in the schema.

- **SchemaError** – In case the schema is wrong.

Returns
A dictionary with the settings. As a word of caution: This may not represent a correct settings object, it will only contain a correct YAML representation.

cobbler.settings.**read_yaml_file**(*filepath: str = '/etc/cobbler/settings.yaml'*) → Dict[str, Any]

Reads settings files from `filepath` and saves the content in a dictionary.

Parameters
filepath – Settings file path, defaults to "/ect/cobbler/settings.yaml"

Raises

- **FileNotFoundError** – In case file does not exist or is a directory.
- **yaml.YAMLError** – In case the file is not a valid YAML file.

Returns
The aggregated dict of all settings.

cobbler.settings.**update_settings_file**(*data: Dict[str, Any]*, *filepath: str = '/etc/cobbler/settings.yaml'*, *ignore_keys: Optional[List[str]] = None*) → bool

Write data handed to this function into the settings file of Cobbler. This function overwrites the existing content. It will only write valid settings. If you are trying to save invalid data this will raise a SchemaException described in `cobbler.settings.validate()`.

Parameters

- **data** – The data to put into the settings file.
- **filepath** – This sets the path of the settingsfile to write.
- **ignore_keys** – The list of ignore keys to exclude from validation.

Returns
True if the action succeeded. Otherwise return False.

cobbler.settings.**validate_settings**(*settings_content: Dict[str, Any]*, *ignore_keys: Optional[List[str]] = None*) → Dict[str, Any]

This function performs logical validation of our loaded YAML files. This function will: - Perform type validation on all values of all keys. - Provide defaults for optional settings. :param settings_content: The dictionary content from the YAML file. :param ignore_keys: The list of ignore keys to exclude from validation. :raises SchemaError: In case the data given is invalid. :return: The Settings of Cobbler which can be safely used inside this instance.

8.1.6 cobbler.utils package

Submodules

cobbler.utils.event module

This module contains logic to support the events Cobbler generates in its XML-RPC API.

class cobbler.utils.event.**CobblerEvent**(*name: str = ''*, *statetime: float = 0.0*)

Bases: `object`

This is a small helper class that represents an event in Cobbler.

property **event_id**: str

Read only property to retrieve the internal ID of the event.

property **name**: str

Read only property to retrieve the human-readable name of the event.

cobbler.utils.filesystem_helpers module

TODO

cobbler.utils.filesystem_helpers.**cachefile**(*src: str*, *dst: str*) → None

Copy a file into a cache and link it into place. Use this with caution, otherwise you could end up copying data twice if the cache is not on the same device as the destination.

Parameters

- **src** – The sourcefile for the copy action.
- **dst** – The destination for the copy action.

cobbler.utils.filesystem_helpers.**copyfile**(*src: str*, *dst: str*, *symlink: bool = False*) → None

Copy a file from source to the destination.

Parameters

- **src** – The source file. This may also be a folder.
- **dst** – The destination for the file or folder.
- **symlink** – If instead of a copy, a symlink is okay, then this may be set explicitly to "True".

Raises

OSError – Raised in case `src` could not be read.

cobbler.utils.filesystem_helpers.**copyfileimage**(*src: str*, *image_location: str*, *dst: str*) → None

Copy a file from source to the destination in the image.

Parameters

- **src** – The source file.
- **image_location** – The location of the image.
- **dst** – The destination for the file.

cobbler.utils.filesystem_helpers.**copyremotefile**(*src: str*, *dst1: str*, *api: Optional[*CobblerAPI*] = None*) → None

Copys a file from a remote place to the local destionation.

Parameters

- **src** – The remote file URI.
- **dst1** – The copy destination on the local filesystem.
- **api** – This parameter is not used currently.

Raises

OSError – Raised in case an error occurs when fetching or writing the file.

cobbler.utils.filesystem_helpers.**create_json_database_dirs**(*api:* CobblerAPI) → None

Creates the database directories for the file serializer

Parameters

api – CobblerAPI

cobbler.utils.filesystem_helpers.**create_tftpboot_dirs**(*api:* CobblerAPI) → None

Create directories for tftpboot images

Parameters

api – CobblerAPI

cobbler.utils.filesystem_helpers.**create_trigger_dirs**(*api:* CobblerAPI) → None

Creates the directories that the user/admin can fill with dynamically executed scripts.

Parameters
api – CobblerAPI

cobbler.utils.filesystem_helpers.**create_web_dirs**(*api:* CobblerAPI) → None

Create directories for HTTP content

Parameters
api – CobblerAPI

cobbler.utils.filesystem_helpers.**hashfile**(*file_name: str*, *lcache: Optional[Path] = None*) → Optional[str]

Returns the sha1sum of the file

Parameters

- **file_name** – The file to get the sha1sum of.
- **lcache** – This is a directory where Cobbler would store its `link_cache.json` file to speed up the return of the hash. The hash looked up would be checked against the Cobbler internal mtime of the object.

Returns
The sha1 sum or None if the file doesn't exist.

cobbler.utils.filesystem_helpers.**is_safe_to_hardlink**(*src: str*, *dst: str*, *api:* CobblerAPI) → bool

Determine if it is safe to hardlink a file to the destination path.

Parameters

- **src** – The hardlink source path.
- **dst** – The hardlink target path.
- **api** – The api-instance to resolve needed information with.

Returns
True if selinux is disabled, the file is on the same device, the source in not a link, and it is not a remote path. If selinux is enabled the functions still may return true if the object is a kernel or initrd. Otherwise returns False.

cobbler.utils.filesystem_helpers.**linkfile**(*api:* CobblerAPI, *src: str*, *dst: str*, *symlink_ok: bool = False*, *cache: bool = True*) → None

Attempt to create a link dst that points to src. Because file systems suck we attempt several different methods or bail to just copying the file.

Parameters

- **api** – This parameter is needed to check if a file can be hardlinked. This method fails if this parameter is not present.
- **src** – The source file.
- **dst** – The destination for the link.
- **symlink_ok** – If it is okay to just use a symbolic link.
- **cache** – If it is okay to use a cached file instead of the real one.

Raises
CX – Raised in case the API is not given.

cobbler.utils.filesystem_helpers.**mkdir**(*path: str*, *mode: int = 493*) → None

Create directory with a given mode.

Parameters

- **path** – The path to create the directory at.
- **mode** – The mode to create the directory with.

Raises
CX – Raised in case creating the directory fails with something different from error code 17 (directory already exists).

cobbler.utils.filesystem_helpers.**mkdirimage**(*path: Path*, *image_location: str*) → None

Create a directory in an image.

Parameters

- **path** – The path to create the directory at.
- **image_location** – The location of the image.

cobbler.utils.filesystem_helpers.**path_tail**(*apath: str*, *bpath: str*) → str

Given two paths (B is longer than A), find the part in B not in A

Parameters

- **apath** – The first path.
- **bpath** – The second path.

Returns
If the paths are not starting at the same location this function returns an empty string.

cobbler.utils.filesystem_helpers.**rmfile**(*path: str*) → None

Delete a single file.

Parameters
path – The file to delete.

cobbler.utils.filesystem_helpers.**rmglob_files**(*path: str*, *glob_pattern: str*) → None

Deletes all files in `path` with `glob_pattern` with the help of `rmfile()`.

Parameters

- **path** – The folder of the files to remove.
- **glob_pattern** – The glob pattern for the files to remove in `path`.

cobbler.utils.filesystem_helpers.**rmtree**(*path: str*) → None

Delete a complete directory or just a single file.

Parameters
path – The directory or folder to delete.

Raises
CX – Raised in case `path` does not exist.

cobbler.utils.filesystem_helpers.**rmtree_contents**(*path: str*) → None

Delete the content of a folder with a glob pattern.

Parameters
path – This parameter presents the glob pattern of what should be deleted.

cobbler.utils.filesystem_helpers.**safe_filter**(*var: Optional[str]*) → None

This function does nothing if the argument does not find any semicolons or two points behind each other.

Parameters
var – This parameter shall not be None or have “..”/”;” at the end.

Raises
CX – In case any `..` or `/` is found in `var`.

cobbler.utils.filesystem_helpers.**sha1_file**(*file_path: Union[str, Path]*, *buffer_size: int = 65536*) → str

This function is emulating the functionality of the sha1sum tool.

Parameters

- **file_path** – The path to the file that should be hashed.
- **buffer_size** – The buffer-size that should be used to hash the file.

Returns
The SHA1 hash as sha1sum would return it.

cobbler.utils.input_converters module

TODO

cobbler.utils.input_converters.**input_boolean**(*value: Union[str, bool, int]*) → bool

Convert a str to a boolean. If this is not possible or the value is false return false.

Parameters
value – The value to convert to boolean.

Returns
True if the value is in the following list, otherwise false: "true", "1", "on", "yes", "y" .

cobbler.utils.input_converters.**input_int**(*value: Union[str, int, float]*) → int

Convert a value to integer.

Parameters
value – The value to convert.

Raises
TypeError – In case after the attempted conversion we still don't have an int.

Returns
The integer if the conversion was successful.

cobbler.utils.input_converters.**input_string_or_dict**(*options: Union[str, List[Any], Dict[Any, Any]]*, *allow_multiples: bool = True*) → Union[str, Dict[Any, Any]]

Older Cobbler files stored configurations in a flat way, such that all values for strings. Newer versions of Cobbler allow dictionaries. This function is used to allow loading of older value formats so new users of Cobbler aren't broken in an upgrade.

Parameters

- **options** – The str or dict to convert.
- **allow_multiples** – True (default) to allow multiple identical keys, otherwise set this false explicitly.

Returns
A dict or the value <<inherit>> in case it is the only content of `options`.

Raises
TypeError – Raised in case the input type is wrong.

cobbler.utils.input_converters.**input_string_or_dict_no_inherit**(*options: Union[str, List[Any], Dict[Any, Any]]*, *allow_multiples: bool = True*) → Dict[Any, Any]

See *input_string_or_dict()*

cobbler.utils.input_converters.**input_string_or_list**(*options: Optional[Union[str, List[Any]]]*) → Union[List[Any], str]

Accepts a delimited list of stuff or a list, but always returns a list.

Parameters
options – The object to split into a list.

Returns
str when this functions get's passed <<inherit>>. if option is delete then an empty list is returned. Otherwise, this function tries to return the arg option or tries to split it into a list.

Raises
TypeError – In case the type of options was neither None, str or list.

cobbler.utils.input_converters.**input_string_or_list_no_inherit**(*options: Optional[Union[str, List[Any]]]*) → List[Any]

Accepts a delimited list of stuff or a list, but always returns a list.

Parameters
options – The object to split into a list.

Returns
If option is delete, None (object not literal) or an empty str, then an empty list is returned. Otherwise, this function tries to return the arg option or tries to split it into a list.

Raises
TypeError – In case the type of options was neither None, str or list.

cobbler.utils.mtab module

We cache the contents of /etc/mtab. The following module is used to keep our cache in sync.

class cobbler.utils.mtab.**MntEntObj**(*input_data: Optional[str] = None*)

Bases: object

TODO

mnt_dir = None

mnt_freq = 0

mnt_fsname = None

mnt_opts = None

mnt_passno = 0

mnt_type = None

cobbler.utils.mtab.**get_file_device_path**(*fname: str*) → Tuple[Optional[str], str]

What this function attempts to do is take a file and return:

- the device the file is on
- the path of the file relative to the device.

For example:
/boot/vmlinuz -> (/dev/sda3, /vmlinuz) /boot/efi/efi/redhat/elilo.conf -> (/dev/cciss0, /elilo.conf) /etc/fstab -> (/dev/sda4, /etc/fstab)

Parameters
fname – The filename to split up.

Returns
A tuple containing the device and relative filename.

cobbler.utils.mtab.**get_mtab**(*mtab: str = '/etc/mtab'*, *vfstype: bool = False*) → List[*MntEntObj*]

Get the list of mtab entries. If a custom mtab should be read then the location can be overridden via a parameter.

Parameters

- **mtab** – The location of the mtab. Argument can be omitted if the mtab is at its default location.
- **vfstype** – If this is True, then all filesystems which are nfs are returned. Otherwise this returns all mtab entries.

Returns

The list of requested mtab entries.

cobbler.utils.mtab.**is_remote_file**(*file: str*) → bool

This function is trying to detect if the file in the argument is remote or not.

Parameters

file – The filepath to check.

Returns

If remote True, otherwise False.

cobbler.utils.process_management module

TODO

cobbler.utils.process_management.**is_service**() → bool

Return whether this system uses service.

This method currently checks if the path `/usr/sbin/service` exists.

cobbler.utils.process_management.**is_supervisord**() → bool

Return whether this system uses supervisord.

This method currently checks if there is a running supervisord instance on `localhost`.

cobbler.utils.process_management.**is_systemd**() → bool

Return whether this system uses systemd.

This method currently checks if the path `/usr/lib/systemd/systemd` exists.

cobbler.utils.process_management.**service_restart**(*service_name: str*) → int

Restarts a daemon service independent of the underlining process manager. Currently, supervisord, systemd and SysV are supported. Checks which manager is present is done in the order just described.

Parameters

service_name – The name of the service

Returns

If the system is SystemD or SysV based the return code of the restart command.

cobbler.utils.signatures module

TODO

cobbler.utils.signatures.**get_supported_distro_boot_loaders**(*item: Union[Distro, Image]*, *api_handle: Optional[CobblerAPI] = None*) → List[str]

This is trying to return you the list of known bootloaders if all resorts fail. Otherwise this returns a list which contains only the subset of bootloaders which are available by the distro in the argument.

Parameters

- **distro** – The distro to check for.
- **api_handle** – The api instance to resolve metadata and settings from.

Returns

The list of bootloaders or a dict of well known bootloaders.

cobbler.utils.signatures.**get_valid_archs**() → List[str]

Return a list of valid architectures found in the import signatures

Returns

All architectures which are known to Cobbler according to the signature cache.

cobbler.utils.signatures.**get_valid_breeds**() → List[str]

Return a list of valid breeds found in the import signatures

cobbler.utils.signatures.**get_valid_os_versions**() → List[str]

Return a list of valid os-versions found in the import signatures

Returns

All operating system versions which are known to Cobbler according to the signature cache.

cobbler.utils.signatures.**get_valid_os_versions_for_breed**(*breed: str*) → List[str]

Return a list of valid os-versions for the given breed

Parameters

breed – The operating system breed to check for.

Returns

All operating system version which are known to Cobbler according to the signature cache filtered by a os-breed.

cobbler.utils.signatures.**load_signatures**(*filename: str*, *cache: bool = True*) → None

Loads the import signatures for distros.

Parameters

- **filename** – Loads the file with the given name.
- **cache** – If the cache should be set with the newly read data.

cobbler.utils.thread module

This module is responsible for managing the custom common threading logic Cobbler has.

class cobbler.utils.thread.**CobblerThread**(*event_id: str*, *remote: CobblerXMLRPCInterface*, *options: Dict[str, Any]*, *task_name: str*, *api: CobblerAPI*, *run: Callable[[CobblerThread], None]*, *on_done: Optional[Callable[[CobblerThread], None]] = None*)

Bases: Thread

This is a custom thread that has a custom logger as well as logic to execute Cobbler triggers.

run() → None

Run the thread.

Returns
The return code of the action. This may a boolean or a Linux return code.

Module contents

Misc heavy lifting functions for Cobbler

cobbler.utils.**blender**(*api_handle: CobblerAPI*, *remove_dicts: bool*, *root_obj: ITEM_UNION*) → Dict[str, Any]

Combine all of the data in an object tree from the perspective of that point on the tree, and produce a merged dictionary containing consolidated data.

Parameters

- **api_handle** – The api to use for collecting the information to blender the item.
- **remove_dicts** – Boolean to decide whether dicts should be converted.
- **root_obj** – The object which should act as the root-node object.

Returns
A dictionary with all the information from the root node downwards.

cobbler.utils.**cheetah_exc**(*exc: Exception*) → str

Converts an exception thrown by Cheetah3 into a custom error message.

Parameters
exc – The exception to convert.

Returns
The string representation of the Cheetah3 exception.

cobbler.utils.**command_existing**(*cmd: str*) → bool

This takes a command which should be known to the system and checks if it is available.

Parameters
cmd – The executable to check

Returns
If the binary does not exist `False`, otherwise `True`.

cobbler.utils.**compare_versions_gt**(*ver1: str*, *ver2: str*) → bool

Compares versions like "0.9.3" with each other and decides if ver1 is greater than ver2.

Parameters

- **ver1** – The first version.
- **ver2** – The second version.

Returns
True if ver1 is greater, otherwise False.

cobbler.utils.**dhcp_service_name**() → str

Determine the dhcp service which is different on various distros. This is currently a hardcoded detection.

Returns
This will return one of the following names: "dhcp3-server", "isc-dhcp-server", "dhcpd"

cobbler.utils.**dhcpconf_location**(*protocol: DHCP*, *filename: str = 'dhcpd.conf'*) → str

This method returns the location of the dhcpd.conf file.

Parameters

- **protocol** – The DHCP protocol version (v4/v6) that is used.
- **filename** – The filename of the DHCP configuration file.

Raises
AttributeError – If the protocol is not v4/v6.

Returns
The path possibly used for the dhcpd.conf file.

cobbler.utils.**dict_annihilate**(*dictionary: Dict[Any, Any]*) → None

Annihilate entries marked for removal. This method removes all entries with key names starting with "!". If a `dictionary` contains keys "!xxx" and "xxx", then both will be removed.

Parameters
dictionary – A dictionary to clean up.

cobbler.utils.**dict_removals**(*results: Dict[Any, Any]*, *subkey: str*) → None

Remove entries from a dictionary starting with a "!".

Parameters
- **results** – The dictionary to search in
- **subkey** – The subkey to search through.

cobbler.utils.**dict_to_string**(*_dict: Dict[Any, Any]*) → Union[str, Dict[Any, Any]]

Convert a dictionary to a printable string. Used primarily in the kernel options string and for some legacy stuff where koan expects strings (though this last part should be changed to dictionaries)

A KV-Pair is joined with a "=". Values are enclosed in single quotes.

Parameters
_dict – The dictionary to convert to a string.

Returns
The string which was previously a dictionary.

cobbler.utils.**die**(*msg: str*) → None

This method let's Cobbler crash with an exception. Log the exception once in the per-task log or the main log if this is not a background op.

Parameters
msg – The message to send for raising the exception

Raises
CX – Raised in all cases with `msg`.

cobbler.utils.**file_is_remote**(*file_location: str*) → bool

Returns true if the file is remote and referenced via a protocol we support.

Parameters
file_location – The URI to check.

Returns
True if the URI is http, https or ftp. Otherwise false.

cobbler.utils.**filelock**(*lock_file: str*)

Context manager to acquire a file lock and release it afterwards

Parameters
lock_file – Path to the file lock to acquire

Raises
OSError – Raised in case of unexpect error acquiring file lock.

cobbler.utils.**find_highest_files**(*directory: str*, *unversioned: str*, *regex: Pattern[str]*) → str

Find the highest numbered file (kernel or initrd numbering scheme) in a given directory that matches a given pattern. Used for auto-booting the latest kernel in a directory.

Parameters

- **directory** – The directory to search in.
- **unversioned** – The base filename which also acts as a last resort if no numbered files are found.
- **regex** – The regex to search for.

Returns
The file with the highest number or an empty string.

cobbler.utils.**find_initrd**(*path: str*) → Optional[str]

Given a directory or a filename, see if the path can be made to resolve into an intird, return that full path if possible.

Parameters
path – The path to check for initrd files.

Returns
None or the path to the found initrd.

cobbler.utils.**find_kernel**(*path: str*) → str

Given a filename, find if the path can be made to resolve into a kernel, and return that full path if possible.

Parameters
path – The path to check for a kernel.

Returns
path if at the specified location a possible match for a kernel was found, otherwise an empty string.

cobbler.utils.**find_matching_files**(*directory: str*, *regex: Pattern[str]*) → List[str]

Find all files in a given directory that match a given regex. Can't use glob directly as glob doesn't take regexen. The search does not include subdirectories.

Parameters

- **directory** – The directory to search in.
- **regex** – The regex to apply to the found files.

Returns
An array of files which apply to the regex.

cobbler.utils.**flatten**(*data: Dict[str, Any]*) → Optional[Dict[str, Any]]

Convert certain nested dicts to strings. This is only really done for the ones koan needs as strings this should not be done for everything

Parameters
data – The dictionary in which various keys should be converted into a string.

Returns
None (if data is None) or the flattened string.

cobbler.utils.**get_exc**(*exc: Exception*, *full: bool = True*) → str

This tries to analyze if an exception comes from Cobbler and potentially enriches or shortens the exception.

Parameters

- **exc** – The exception which should be analyzed.
- **full** – If the full exception should be returned or only the most important information.

Returns
The exception which has been converted into a string which then can be logged easily.

cobbler.utils.**get_family**() → str

Get family of running operating system.

Family is the base Linux distribution of a Linux distribution, with a set of common parents.

Returns
May be "redhat", "debian" or "suse" currently. If none of these are detected then just the distro name is returned.

cobbler.utils.**get_host_ip**(*ip_address: str*, *shorten: bool = True*) → str

Return the IP encoding needed for the TFTP boot tree.

Parameters

- **ip_address** – The IP address to pretty print.
- **shorten** – Whether the IP-Address should be shortened or not.

Returns
The IP encoded as a hexadecimal value.

cobbler.utils.**get_random_mac**(*api_handle: CobblerAPI*, *virt_type: str = 'kvm'*) → str

Generate a random MAC address.

The code of this method was taken from xend/server/netif.py

Parameters

- **api_handle** – The main Cobbler api instance.
- **virt_type** – The virtualization provider. Currently possible is 'vmware', 'xen', 'qemu', 'kvm'.

Returns
MAC address string

Raises
CX – Raised in case unsupported `virt_type` given.

cobbler.utils.**get_shared_secret**() → Union[str, int]

The 'web.ss' file is regenerated each time cobblerd restarts and is used to agree on shared secret interchange between the web server and cobblerd, and also the CLI and cobblerd, when username/password access is not required. For the CLI, this enables root users to avoid entering username/pass if on the Cobbler server.

Returns
The Cobbler secret which enables full access to Cobbler.

cobbler.utils.**get_supported_system_boot_loaders**() → List[str]

Return the list of currently supported bootloaders.

Returns
The list of currently supported bootloaders.

cobbler.utils.**is_ip**(*strdata: str*) → bool

Return whether the argument is an IP address.

Parameters
strdata – The IP in a string format. This get's passed to the IP object of Python.

cobbler.utils.**is_selinux_enabled**() → bool

This check is achieved via a subprocess call to `selinuxenabled`. Default return is false.

Returns
Whether selinux is enabled or not.

cobbler.utils.**is_str_float**(*value: str*) → bool

Checks if the string value could be converted into a float. This is necessary since the CLI only works with strings but many methods and checks expects a float.

Parameters
value – The value to check

Returns
True if conversion is successful

cobbler.utils.**is_str_int**(*value: str*) → bool

Checks if the string value could be converted into an integer. This is necessary since the CLI only works with strings but many methods and checks expects an integer.

Parameters
value – The value to check

Returns
True if conversion is successful

cobbler.utils.**kopts_overwrite**(*kopts: Dict[Any, Any]*, *cobbler_server_hostname: str = ''*, *distro_breed: str = ''*, *system_name: str = ''*) → None

SUSE is not using 'text'. Instead 'textmode' is used as kernel option.

Parameters

- **kopts** – The kopts of the system.
- **cobbler_server_hostname** – The server setting from our Settings.
- **distro_breed** – The distro for the system to change to kopts for.
- **system_name** – The system to overwrite the kopts for.

cobbler.utils.**local_get_cobbler_api_url**() → str

Get the URL of the Cobbler HTTP API from the Cobbler settings file.

Returns
The api entry point. This does not respect modifications from Loadbalancers or API-Gateways.

cobbler.utils.**local_get_cobbler_xmlrpc_url**() → str

Get the URL of the Cobbler XMLRPC API from the Cobbler settings file.

Returns
The api entry point.

cobbler.utils.**lod_sort_by_key**(*list_to_sort: List[Any]*, *indexkey: Hashable*) → List[Any]

Sorts a list of dictionaries by a given key in the dictionaries.

Note: This is a destructive operation and does not sort the dictionaries.

Parameters

- **list_to_sort** – The list of dictionaries to sort.
- **indexkey** – The key to index to dicts in the list.

Returns
The sorted list.

cobbler.utils.**lod_to_dod**(*_list: List[Any]*, *indexkey: Hashable*) → Dict[Any, Any]

Things like `get_distros()` returns a list of a dictionaries. Convert this to a dict of dicts keyed off of an arbitrary field.

Example: `[ { "a" : 2 }, { "a" : 3 } ] -> { "2" : { "a" : 2 }, "3" : { "a" : "3" } }`

Parameters

- **_list** – The list of dictionaries to use for the conversion.
- **indexkey** – The position to use as dictionary keys.

Returns
The converted dictionary. It is not guaranteed that the same key is not used multiple times.

cobbler.utils.**log_exc**() → None

Log an exception.

cobbler.utils.**named_service_name**() → str

Determine the named service which is normally different on various distros.

Returns
This will return for debian/ubuntu bind9 and on other distros named-chroot or named.

cobbler.utils.**namedconf_location**() → str

This returns the location of the named.conf file.

Returns
If the distro is Debian/Ubuntu then this returns "/etc/bind/named.conf". Otherwise "/etc/named.conf"

cobbler.utils.**os_release**() → Tuple[str, float]

Get the os version of the linux distro. If the get_family() method succeeds then the result is normalized.

Returns
The os-name and os version.

cobbler.utils.**pretty_hex**(*ip_address: IPAddress*, *length: int = 8*) → str

Pads an IP object with leading zeroes so that the result is _length_ hex digits. Also do an upper().

Parameters

- **ip_address** – The IP address to pretty print.
- **length** – The length of the resulting hexstring. If the number is smaller than the resulting hex-string then no front-padding is done.

cobbler.utils.**read_file_contents**(*file_location: str*, *fetch_if_remote: bool = False*) → Optional[str]

Reads the contents of a file, which could be referenced locally or as a URI.

Parameters

- **file_location** – The location of the file to read.
- **fetch_if_remote** – If True a remote file will be tried to read, otherwise remote files are skipped and None is returned.

Returns
Returns None if file is remote and templating of remote files is disabled.

Raises
FileNotFoundError – if the file does not exist at the specified location.

cobbler.utils.**remote_file_exists**(*file_url: str*) → bool

Return True if the remote file exists.

Parameters
file_url – The URL to check.

Returns
True if Cobbler can reach the specified URL, otherwise false.

cobbler.utils.**remove_yum_olddata**(*path: Union[str, PathLike[str]]*) → None

Delete .olddata folders that might be present from a failed run of createrepo.

Parameters
path – The path to check for .olddata files.

cobbler.utils.**revert_strip_none**(*data: Union[str, int, float, bool, List[Any], Dict[Any, Any]]*) → Optional[Union[str, int, float, bool, List[Any], Dict[Any, Any]]]

Does the opposite to strip_none. If a value which represents None is detected, it replaces it with None.

Parameters
data – The data to check.

Returns
The data without None.

cobbler.utils.**rsync_files**(*src: str*, *dst: str*, *args: str*, *quiet: bool = True*) → bool

Sync files from src to dst. The extra arguments specified by args are appended to the command.

Parameters

- **src** – The source for the copy process.
- **dst** – The destination for the copy process.
- **args** – The extra arguments are appended to our standard arguments.
- **quiet** – If `True` no progress is reported. If `False` then progress will be reported by rsync.

Returns
`True` on success, otherwise `False`.

cobbler.utils.**run_triggers**(*api: CobblerAPI*, *ref: Optional[Item] = None*, *globber: str = ''*, *additional: Optional[List[Any]] = None*) → None

Runs all the trigger scripts in a given directory. Example: `/var/lib/cobbler/triggers/blah/*`

As of Cobbler 1.5.X, this also runs Cobbler modules that match the globbing paths.

Python triggers are always run before shell triggers.

Parameters

- **api** – The api object to use for resolving the actions.
- **ref** – Can be a Cobbler object, if not None, the name will be passed to the script. If ref is None, the script will be called with no arguments.
- **globber** – is a wildcard expression indicating which triggers to run.
- **additional** – Additional arguments to run the triggers with.

Raises
CX – Raised in case the trigger failed.

cobbler.utils.**strip_none**(*data: Optional[Union[List[Any], Dict[Any, Any], int, str, float]]*, *omit_none: bool = False*) → Union[List[Any], Dict[Any, Any], int, str, float]

Remove "None" entries from datastructures. Used prior to communicating with XMLRPC.

Parameters

- **data** – The data to strip None away.
- **omit_none** – If the datastructure is not a single item then None items will be skipped instead of replaced if set to "True".

Returns
The modified data structure without any occurrence of None.

cobbler.utils.**subprocess_call**(*cmd: Union[str, List[str]]*, *shell: bool = False*, *process_input: Any = None*) → int

A simple subprocess call with no output capturing.

Parameters

- **cmd** – The command to execute.
- **shell** – Whether to use a shell or not for the execution of the command.
- **process_input** – If there is any process_input needed for that command to stdin.

Returns
The return code of the process

cobbler.utils.**subprocess_get**(*cmd: Union[str, List[str]]*, *shell: bool = True*, *process_input: Any = None*) → str

A simple subprocess call with no return code capturing.

Parameters

- **cmd** – The command to execute.
- **shell** – Whether to use a shell or not for the execution of the command.
- **process_input** – If there is any process_input needed for that command to stdin.

Returns
The data which the subprocess returns.

cobbler.utils.**subprocess_sp**(*cmd: Union[str, List[str]]*, *shell: bool = True*, *process_input: Any = None*) → Tuple[str, int]

Call a shell process and redirect the output for internal usage.

Parameters

- **cmd** – The command to execute in a subprocess call.
- **shell** – Whether to use a shell or not for the execution of the command.
- **process_input** – If there is any input needed for that command to stdin.

Returns
A tuple of the output and the return code.

cobbler.utils.**uniquify**(*seq: List[Any]*) → List[Any]

Remove duplicates from the sequence handed over in the args.

Parameters
seq – The sequence to check for duplicates.

Returns
The list without duplicates.

8.2 Submodules

8.3 cobbler.api module

This module represents the Cobbler Python API. It is used by the XML-RPC API and can be used by external consumers.

Changelog:

Schema: From -> To

Current Schema: Please refer to the documentation visible of the individual methods.

V3.4.0 (unreleased)

- **Added:**
  - clean_items_cache
  - new_item
  - deserialize_item
  - input_string_or_list_no_inherit
  - input_string_or_list
  - input_string_or_dict
  - input_string_or_dict_no_inherit
  - input_boolean
  - input_int
- **Changed:**
  - new_*: Accepts kwargs as a last argument now (so a dict) that makes it possible to seed an object

V3.3.4 (unreleased)

- No changes

V3.3.3

- **Added:**
  - get_item_resolved_value
  - set_item_resolved_value
- **Changed:**
  - dump_vars: Added boolean parameter remove_dicts as a new last argument

V3.3.2

- No changes

V3.3.1

- **Changes:**
  - add_system: Parameter check_for_duplicate_netinfo was removed
  - build_iso: Replaced default None arguments with typed arguments
  - create_grub_images: Renamed to mkloaders

V3.3.0

- **Added:**
  - menus
  - copy_menu
  - remove_menu
  - rename_menu
  - new_menu
  - add_menu
  - find_menu
  - get_menus_since

  - sync_systems
  - sync_dns
  - get_valid_obj_boot_loaders
  - create_grub_images
- **Changed:**
  - Constructor: Added settingsfile_location and execute_settings_automigration as parameters
  - find_items: Accept an empty str for what if the argument name is given.
  - dump_vars: Parameter format was renamed to formatted_output
  - generate_gpxe: Renamed to generate_ipxe; The second parameter is now image and accepts the name of one.
  - sync: Accepts a new parameter called what which is a List[str] that signals what should be synced. An empty list signals a full sync.
  - sync_dhcp: Parameter verbose was removed
- **Removed:**
  - The logger arugment was removed from all methods
  - dlcontent

V3.2.2

- No changes

V3.2.1

- Added primitive type annotations for all parameters of all methods

V3.2.0

- No changes

V3.1.2

- No changes

V3.1.1

- No changes

V3.1.0

- No changes

V3.0.1

- No changes

V3.0.0

- **Added:**
  - power_system: Replaces power_on, power_off, reboot, power_status
- **Changed:**
  - import_tree: kickstart_file is now called autoinstall_file
- **Removed:**
  - update
  - clear
  - deserialize_raw

- `deserialize_item_raw`
- `power_on` - Replaced by power_system
- `power_off` - Replaced by power_system
- `reboot` - Replaced by power_system
- `power_status` - Replaced by power_system

V2.8.5

- Inital tracking of changes.

class cobbler.api.**CobblerAPI**(*is_cobblerd: bool = False*, *settingsfile_location: str = '/etc/cobbler/settings.yaml'*, *execute_settings_automigration: bool = False*)

Bases: `object`

Python API module for Cobbler. See source for cobbler.py, or pydoc, for example usage. Cli apps and daemons should import api.py, and no other Cobbler code.

acl_config(*adduser: Optional[str] = None*, *addgroup: Optional[str] = None*, *removeuser: Optional[str] = None*, *removegroup: Optional[str] = None*) → None

Configures users/groups to run the Cobbler CLI as non-root. Pass in only one option at a time. Powers `cobbler aclconfig`.

Parameters

- **adduser** –
- **addgroup** –
- **removeuser** –
- **removegroup** –

add_distro(*ref: Distro*, *check_for_duplicate_names: bool = False*, *save: bool = True*, *with_triggers: bool = True*) → None

Add a distribution to Cobbler.

Parameters

- **ref** – The identifier for the object to add to a collection.
- **check_for_duplicate_names** – If the name should be unique or can be present multiple times.
- **save** – If the item should be persisted.
- **with_triggers** – If triggers should be run when the object is added.

add_image(*ref: Image*, *check_for_duplicate_names: bool = False*, *save: bool = True*, *with_triggers: bool = True*) → None

Add an image to Cobbler.

Parameters

- **ref** – The identifier for the object to add to a collection.
- **check_for_duplicate_names** – If the name should be unique or can be present multiple times.
- **save** – If the item should be persisted.
- **with_triggers** – If triggers should be run when the object is added.

add_item(*what: str*, *ref: ITEM_UNION*, *check_for_duplicate_names: bool = False*, *save: bool = True*, *with_triggers: bool = True*) → None

Add an abstract item to a collection of its specific items. This is not meant for external use. Please reefer to one of the specific methods `add_<type>`.

Parameters

- **what** – The item type.
- **ref** – The identifier for the object to add to a collection.
- **check_for_duplicate_names** – If the name should be unique or can be present multiple times.
- **save** – If the item should be persisted.
- **with_triggers** – If triggers should be run when the object is added.

add_menu(*ref: Menu*, *check_for_duplicate_names: bool = False*, *save: bool = True*, *with_triggers: bool = True*) → None

Add a submenu to Cobbler.

Parameters

- **ref** – The identifier for the object to add to a collection.
- **check_for_duplicate_names** – If the name should be unique or can be present multiple times.
- **save** – If the item should be persisted.
- **with_triggers** – If triggers should be run when the object is added.

add_profile(*ref: Profile*, *check_for_duplicate_names: bool = False*, *save: bool = True*, *with_triggers: bool = True*) → None

Add a profile to Cobbler.

Parameters

- **ref** – The identifier for the object to add to a collection.
- **check_for_duplicate_names** – If the name should be unique or can be present multiple times.
- **save** – If the item should be persisted.
- **with_triggers** – If triggers should be run when the object is added.

add_repo(*ref: Repo*, *check_for_duplicate_names: bool = False*, *save: bool = True*, *with_triggers: bool = True*) → None

Add a repository to Cobbler.

Parameters

- **ref** – The identifier for the object to add to a collection.
- **check_for_duplicate_names** – If the name should be unique or can be present multiple times.
- **save** – If the item should be persisted.
- **with_triggers** – If triggers should be run when the object is added.

add_system(*ref: System*, *check_for_duplicate_names: bool = False*, *save: bool = True*, *with_triggers: bool = True*) → None

Add a system to Cobbler.

Parameters

- **ref** – The identifier for the object to add to a collection.

- **check_for_duplicate_names** – If the name should be unique or can be present multiple times.
- **save** – If the item should be persisted.
- **with_triggers** – If triggers should be run when the object is added.

authenticate(*user: str*, *password: str*) → bool

(Remote) access control. This depends on the chosen authentication module. Cobbler internal use only.

Parameters

- **user** – The username to check for authentication.
- **password** – The password to check for authentication.

Returns

Whether the action succeeded or not.

authorize(*user: str*, *resource: str*, *arg1: Optional[str] = None*, *arg2: Any = None*) → int

(Remote) access control. This depends on the chosen authorization module. Cobbler internal use only.

Parameters

- **user** – The username to check for authorization.
- **resource** – The type of resource which should be checked for access from the supplied user.
- **arg1** – The actual resource to check for authorization.
- **arg2** – Not known what this parameter does exactly.

Returns

The return code of the action.

auto_add_repos() → None

Import any repos this server knows about and mirror them. Run `cobbler reposync` to apply the changes. Credit: Seth Vidal.

:raises ImportError

build_iso(*iso: str = 'autoinst.iso'*, *profiles: Optional[List[str]] = None*, *systems: Optional[List[str]] = None*, *buildisodir: str = ''*, *distro_name: str = ''*, *standalone: bool = False*, *airgapped: bool = False*, *source: str = ''*, *exclude_dns: bool = False*, *xorrisofs_opts: str = ''*) → None

Build an iso image which may be network bootable or not.

Parameters

- **iso** – The name of the ISO. Defaults to `autoinst.iso`.
- **profiles** – Use these profiles only
- **systems** – Use these systems only
- **buildisodir** – This overwrites the directory from the settings in which the iso is built in.
- **distro_name** – Used with `--standalone` and `--airgapped` to create a distro-based ISO including all associated.
- **standalone** – This means that no network connection is needed to install the generated iso.
- **airgapped** – This option implies `standalone=True`.
- **source** – If the iso should be offline available this is the path to the sources of the image.

- **exclude_dns** – Whether the repositories have to be locally available or the internet is reachable.
- **xorrisofs_opts** – `xorrisofs` options to include additionally.

check() → List[str]

See if all preqs for network booting are valid. This returns a list of strings containing instructions on things to correct. An empty list means there is nothing to correct, but that still doesn't mean there are configuration errors. This is mainly useful for human admins, who may, for instance, forget to properly set up their TFTP servers for PXE, etc.

Returns
A list of things to address.

clean_items_cache(*obj: Union[*Settings, *Dict[str, Any]]*)

Items cache invalidation in case of settings or singatures changes. Cobbler internal use only.

clear_logs(*system:* System) → None

Clears console and anamon logs for system

Parameters
system – The system to clear logs of.

copy_distro(*ref:* Distro, *newname: str*) → None

This method copies a distro which is just different in the name of the object.

Parameters

- **ref** – The object itself which gets copied.
- **newname** – The new name of the newly created object.

copy_image(*ref:* Image, *newname: str*) → None

This method copies an image which is just different in the name of the object.

Parameters

- **ref** – The object itself which gets copied.
- **newname** – The new name of the newly created object.

copy_item(*what: str*, *ref: ITEM_UNION*, *newname: str*) → None

General copy method which is called by the specific methods.

Parameters

- **what** – The collection type which gets copied.
- **ref** – The object itself which gets copied.
- **newname** – The new name of the newly created object.

copy_menu(*ref:* Menu, *newname: str*) → None

This method copies a file which is just different in the name of the object.

Parameters

- **ref** – The object itself which gets copied.
- **newname** – The new name of the newly created object.

copy_profile(*ref:* Profile, *newname: str*) → None

This method copies a profile which is just different in the name of the object.

Parameters

- **ref** – The object itself which gets copied.
- **newname** – The new name of the newly created object.

copy_repo(*ref:* Repo, *newname: str*) → None

This method copies a repository which is just different in the name of the object.

Parameters

- **ref** – The object itself which gets copied.
- **newname** – The new name of the newly created object.

copy_system(*ref:* System, *newname: str*) → None

This method copies a system which is just different in the name of the object.

Parameters

- **ref** – The object itself which gets copied.
- **newname** – The new name of the newly created object.

deserialize() → None

Load cobbler_collections from disk. Cobbler internal use only.

deserialize_item(*obj:* Item) → Dict[str, Any]

Load cobbler item from disk. Cobbler internal use only.

distros() → *Distros*

Return the current list of distributions

dump_vars(*obj:* Item, *formatted_output: bool = False*, *remove_dicts: bool = False*) → Union[Dict[str, Any], str]

Dump all known variables related to that object.

Parameters

- **obj** – The object for which the variables should be dumped.
- **formatted_output** – If True the values will align in one column and be pretty printed for cli example.
- **remove_dicts** – If True the dictionaries will be put into str form.

Returns

A dictionary with all the information which could be collected.

find_distro(*name: str = ''*, *return_list: bool = False*, *no_errors: bool = False*, ***kargs: FIND_KWARGS*) → Optional[Union[List[*distro.Distro*], *distro.Distro*]]

Find a distribution via a name or keys specified in the **kargs.

Parameters

- **name** – The name to search for.
- **return_list** – If only the first result or all results should be returned.
- **no_errors** – Silence some errors which would raise if this turned to False.
- **kargs** – Additional key-value pairs which may help in finding the desired objects.

Returns

A single object or a list of all search results.

find_image(*name: str = ''*, *return_list: bool = False*, *no_errors: bool = False*, ***kargs: FIND_KWARGS*) → Optional[Union[List[image_module.Image], image_module.Image]]

Find an image via a name or keys specified in the **kargs.

Parameters

- **name** – The name to search for.
- **return_list** – If only the first result or all results should be returned.

- **no_errors** – Silence some errors which would raise if this turned to False.
- **kargs** – Additional key-value pairs which may help in finding the desired objects.

Returns
A single object or a list of all search results.

find_items(*what: str = ''*, *criteria: Optional[Dict[Any, Any]] = None*, *name: str = ''*, *return_list: bool = True*, *no_errors: bool = False*) → Optional[Union[ITEM_UNION, List[ITEM_UNION]]]

This is the abstract base method for finding object int the api. It should not be used by external resources. Please reefer to the specific implementations of this method called `find_<object type>`.

Parameters

- **what** – The object type of the item to search for.
- **criteria** – The dictionary with the key-value pairs to find objects with.
- **name** – The name of the object.
- **return_list** – If only the first result or all results should be returned.
- **no_errors** – Silence some errors which would raise if this turned to False.

Returns
The list of items witch match the search criteria.

find_menu(*name: str = ''*, *return_list: bool = False*, *no_errors: bool = False*, ***kargs: FIND_KWARGS*) → Optional[Union[List[*menu.Menu*], *menu.Menu*]]

Find a menu via a name or keys specified in the `**kargs`.

Parameters

- **name** – The name to search for.
- **return_list** – If only the first result or all results should be returned.
- **no_errors** – Silence some errors which would raise if this turned to False.
- **kargs** – Additional key-value pairs which may help in finding the desired objects.

Returns
A single object or a list of all search results.

find_profile(*name: str = ''*, *return_list: bool = False*, *no_errors: bool = False*, ***kargs: FIND_KWARGS*) → Optional[Union[List[profile_module.Profile], profile_module.Profile]]

Find a profile via a name or keys specified in the `**kargs`.

Parameters

- **name** – The name to search for.
- **return_list** – If only the first result or all results should be returned.
- **no_errors** – Silence some errors which would raise if this turned to False.
- **kargs** – Additional key-value pairs which may help in finding the desired objects.

Returns
A single object or a list of all search results.

find_repo(*name: str = ''*, *return_list: bool = False*, *no_errors: bool = False*, ***kargs: FIND_KWARGS*) → Optional[Union[List[*repo.Repo*], *repo.Repo*]]

Find a repository via a name or keys specified in the `**kargs`.

Parameters

- **name** – The name to search for.
- **return_list** – If only the first result or all results should be returned.

- **no_errors** – Silence some errors which would raise if this turned to False.
- **kargs** – Additional key-value pairs which may help in finding the desired objects.

Returns
A single object or a list of all search results.

find_system(*name: str = ''*, *return_list: bool = False*, *no_errors: bool = False*, ***kargs: FIND_KWARGS*) → Optional[Union[List[system_module.System], system_module.System]]

Find a system via a name or keys specified in the **kargs.

Parameters

- **name** – The name to search for.
- **return_list** – If only the first result or all results should be returned.
- **no_errors** – Silence some errors which would raise if this turned to False.
- **kargs** – Additional key-value pairs which may help in finding the desired objects.

Returns
A single object or a list of all search results.

generate_bootcfg(*profile: str = ''*, *system: str = ''*) → str

Generate a boot configuration. The system wins over the profile.

Parameters

- **profile** – The profile to return the configuration for.
- **system** – The system to return the configuration for.

Returns
The generated configuration file.

generate_ipxe(*profile: str*, *image: str*, *system: str*) → str

Generate the ipxe configuration files. The system wins over the profile. Profile and System win over Image.

Parameters

- **profile** – The profile to return the configuration for.
- **image** – The image to return the configuration for.
- **system** – The system to return the configuration for.

Returns
The generated configuration file.

generate_script(*profile: Optional[str]*, *system: Optional[str]*, *name: str*) → str

Generate an autoinstall script for the specified profile or system. The system wins over the profile.

Parameters

- **profile** – The profile name to generate the script for.
- **system** – The system name to generate the script for.
- **name** – The name of the script which should be generated. Must only contain alphanumeric characters, dots and underscores.

Returns
The generated script or an error message.

get_distros_since(*mtime: float*, *collapse: bool = False*) → List[*Distro*]

Returns distros modified since a certain time (in seconds since Epoch)

Parameters

- **mtime** – The timestamp which marks the gate if an object is included or not.
- **collapse** – collapse=True specifies returning a dict instead of objects.

Returns

The list of distros which are newer then the given timestamp.

get_images_since(*mtime: float*, *collapse: bool = False*) → List[*Image*]

Return images modified since a certain time (in seconds since Epoch)

Parameters

- **mtime** – The timestamp which marks the gate if an object is included or not.
- **collapse** – If True then this specifies that a list of dicts should be returned instead of a list of objects.

Returns

The list of images which are newer then the given timestamp.

get_item(*what: str*, *name: str*) → Optional[ITEM_UNION]

Get a general item.

Parameters

- **what** – The item type to retrieve from the internal database.
- **name** – The name of the item to retrieve.

Returns

An item of the desired type.

get_item_resolved_value(*item_uuid: str*, *attribute: str*) → Any

This method helps non Python API consumers to retrieve the final data of a field with inheritance.

This does not help with network interfaces because they don't have a UUID at the moment and thus can't be queried via their UUID.

Parameters

- **item_uuid** – The UUID of the item that should be retrieved.
- **attribute** – The attribute that should be retrieved.

Raises

- **ValueError** – In case a value given was either malformed or the desired item did not exist.
- **TypeError** – In case the type of the method arguments do have the wrong type.
- **AttributeError** – In case the attribute specified is not available on the given item (type).

Returns

The attribute value. Since this might be of type NetworkInterface we cannot yet set this explicitly.

get_items(*what: str*) → COLLECTION_UNION

Get all items of a collection.

Parameters

what – The collection to query.

Returns
The items which were queried. May return no items.

`get_menus_since`(*mtime: float*, *collapse: bool = False*) → List[*Menu*]

Return files modified since a certain time (in seconds since Epoch)

Parameters

- **mtime** – The timestamp which marks the gate if an object is included or not.
- **collapse** – If True then this specifies that a list of dicts should be returned instead of a list of objects.

Returns
The list of files which are newer then the given timestamp.

`get_module_by_name`(*module_name: str*) → Optional[module]

Returns a loaded Cobbler module named 'name', if one exists, else None. Cobbler internal use only.

Parameters
module_name –

Returns

`get_module_from_file`(*section: str*, *name: str*, *fallback: Optional[str] = None*) → module

Looks in `/etc/cobbler/settings.yaml` for a section called 'section' and a key called 'name', and then returns the module that corresponds to the value of that key. Cobbler internal use only.

Parameters

- **section** – The section to look at.
- **name** – The name of the module to retrieve
- **fallback** – The default module in case the requested one is not found.

Returns
The requested Python Module.

`get_module_name_from_file`(*section: str*, *name: str*, *fallback: Optional[str] = None*) → str

Looks up a module the same as `get_module_from_file` but returns the module name rather than the module itself.

Parameters

- **section** –
- **name** –
- **fallback** –

Returns

`get_modules_in_category`(*category: str*) → List[module]

Returns all modules in a given category, for instance "serializer", or "cli". Cobbler internal use only.

Parameters
category – The category to check.

Returns
The list of modules.

`get_profiles_since`(*mtime: float*, *collapse: bool = False*) → List[*Profile*]

Returns profiles modified since a certain time (in seconds since Epoch)

Parameters

- **mtime** – The timestamp which marks the gate if an object is included or not.

- **collapse** – If True then this specifies that a list of dicts should be returned instead of a list of objects.

Returns
The list of profiles which are newer then the given timestamp.

get_repo_config_for_profile(*obj:* Item) → str

Get the repository configuration for the specified profile

Parameters
obj – The profile to return the configuration for.

Returns
The repository configuration as a string.

get_repo_config_for_system(*obj:* Item) → str

Get the repository configuration for the specified system.

Parameters
obj – The system to return the configuration for.

Returns
The repository configuration as a string.

get_repos_since(*mtime: float*, *collapse: bool = False*) → List[*Repo*]

Return repositories modified since a certain time (in seconds since Epoch)

Parameters

- **mtime** – The timestamp which marks the gate if an object is included or not.
- **collapse** – If True then this specifies that a list of dicts should be returned instead of a list of objects.

Returns
The list of repositories which are newer then the given timestamp.

static get_signatures() → Dict[str, Any]

This returns the local signature cache.

Returns
The dict containing all signatures.

get_sync(*verbose: bool = False*) → *CobblerSync*

Get a Cobbler Sync object which may be executed through the call of `obj.run()`.

Parameters
verbose – If the action should be just logged as needed or (if True) as much verbose as possible.

Returns
An instance of the CobblerSync class to execute the sync with.

get_systems_since(*mtime: float*, *collapse: bool = False*) → List[*System*]

Return systems modified since a certain time (in seconds since Epoch)

Parameters

- **mtime** – The timestamp which marks the gate if an object is included or not.
- **collapse** – If True then this specifies that a list of dicts should be returned instead of a list of objects.

Returns
The list of systems which are newer then the given timestamp.

get_template_file_for_profile(*obj: ITEM_UNION*, *path: str*) → str

Get the template for the specified profile.

Parameters

- **obj** – The object which is related to that template.
- **path** – The path to the template.

Returns
The template as in its string representation.

get_template_file_for_system(*obj: ITEM_UNION*, *path: str*) → str

Get the template for the specified system.

Parameters

- **obj** – The object which is related to that template.
- **path** – The path to the template.

Returns
The template as in its string representation.

get_tftp_file(*path: str*, *offset: int*, *size: int*) → Tuple[bytes, int]

Generate and return a file for a TFTP client.

Parameters

- **path** – Path to file
- **offset** – Offset of the requested chunk in the file
- **size** – Size of the requested chunk in the file

Returns
The requested chunk and the length of the whole file

get_valid_obj_boot_loaders(*obj: Union[Distro, Image]*) → List[str]

Return the list of valid boot loaders for the object

Parameters
obj – The object for which the boot loaders should be looked up.

Returns
Get a list of all valid boot loaders.

hardlink() → int

Hardlink all files where this is possible to improve performance.

Returns
The return code of the subprocess call which actually hardlinks the files.

images() → *Images*

Return the current list of images

import_tree(*mirror_url: str*, *mirror_name: str*, *network_root: Optional[str] = None*, *autoinstall_file: Optional[str] = None*, *rsync_flags: Optional[str] = None*, *arch: Optional[str] = None*, *breed: Optional[str] = None*, *os_version: Optional[str] = None*) → bool

Automatically import a directory tree full of distribution files.

Parameters

- **mirror_url** – Can be a string that represents a path, a user@host syntax for SSH, or an rsync:// address. If mirror_url is a filesystem path and mirroring is not desired, set network_root to something like "nfs://path/to/mirror_url/root"
- **mirror_name** – The name of the mirror.

- **network_root** – the remote path (nfs/http/ftp) for the distro files
- **autoinstall_file** – user-specified response file, which will override the default
- **rsync_flags** – Additional flags that will be passed to the rsync call that will sync everything to the Cobbler webroot.
- **arch** – user-specified architecture
- **breed** – user-specified breed
- **os_version** – user-specified OS version

input_boolean(*value: Union[str, bool, int]*) → bool

See also:

input_boolean()

input_int(*value: Union[str, int, float]*) → int

See also:

input_int()

input_string_or_dict(*options: Union[str, List[Any], Dict[Any, Any]]*, *allow_multiples: bool = True*) → Union[str, Dict[Any, Any]]

See also:

input_string_or_dict()

input_string_or_dict_no_inherit(*options: Union[str, List[Any], Dict[Any, Any]]*, *allow_multiples: bool = True*) → Dict[Any, Any]

See also:

input_string_or_dict_no_inherit()

input_string_or_list(*options: Optional[Union[str, List[Any]]]*) → Union[List[Any], str]

See also:

input_string_or_list()

input_string_or_list_no_inherit(*options: Optional[Union[str, List[Any]]]*) → List[Any]

See also:

input_string_or_list_no_inherit()

is_selinux_enabled() → bool

Returns whether selinux is enabled on the Cobbler server. We check this just once at Cobbler API init time, because a restart is required to change this; this does /not/ check enforce/permissive, nor does it need to.

is_selinux_supported() → bool

Returns whether or not the OS is sufficient enough to run with SELinux enabled (currently EL 5 or later).

Returns

False per default. If Distro is Redhat and Version >= 5 then it returns true.

last_modified_time() → float

Returns the time of the last modification to Cobbler, made by any API instance, regardless of the serializer type.

Returns

0 if there is no file where the information required for this method is saved.

log(*msg: str*, *args: Optional[Union[str, List[Optional[str]], Dict[str, Any]]] = None*, *debug: bool = False*) → None

Logs a message with the already initiated logger of this object.

Parameters

- **msg** – The message to log.
- **args** – Optional message which gets appended to the main msg with a ';'.
- **debug** – Weather the logged message is a debug message (true) or info (false).

Deprecated since version 3.3.0: We should use the standard logger.

menus() → *Menus*

Return the current list of menus

mkloaders() → None

Create the GRUB installer images via this API call. It utilizes `grub2-mkimage` behind the curtain.

new_distro(*is_subobject: bool = False*, ***kwargs: Any*) → *Distro*

Returns a new empty distro object. This distro is not automatically persisted. Persistance is achieved via `save()`.

Parameters
is_subobject – If the object is a subobject of an already existing object or not.

Returns
An empty Distro object.

new_image(*is_subobject: bool = False*, ***kwargs: Any*) → *Image*

Returns a new empty image object. This image is not automatically persisted. Persistence is achieved via `save()`.

Parameters
is_subobject – If the object created is a subobject or not.

Returns
An empty image object.

new_item(*what: str = ''*, *is_subobject: bool = False*, ***kwargs: Any*) → ITEM_UNION

Creates a new (unconfigured) object. The object is not persisted.

Parameters

- **what** – Specifies the type of object. Valid item types can be seen at *ItemTypes()*.
- **is_subobject** – If the object is a subobject of an already existing object or not.

Returns
The newly created object.

new_menu(*is_subobject: bool = False*, ***kwargs: Any*) → *Menu*

Returns a new empty menu object. This file is not automatically persisted. Persistence is achieved via `save()`.

Parameters
is_subobject – If the object created is a subobject or not.

Returns
An empty Menu object.

new_profile(*is_subobject: bool = False*, ***kwargs: Any*) → *Profile*

Returns a new empty profile object. This profile is not automatically persisted. Persistence is achieved via `save()`.

Parameters
is_subobject – If the object created is a subobject or not.

Returns
An empty Profile object.

new_repo(*is_subobject: bool = False*, ***kwargs: Any*) → *Repo*

Returns a new empty repo object. This repository is not automatically persisted. Persistence is achieved via `save()`.

Parameters
is_subobject – If the object created is a subobject or not.

Returns
An empty repo object.

new_system(*is_subobject: bool = False*, ***kwargs: Any*) → *System*

Returns a new empty system object. This system is not automatically persisted. Persistence is achieved via `save()`.

Parameters
is_subobject – If the object created is a subobject or not.

Returns
An empty System object.

power_system(*system:* System, *power_operation: str*, *user: Optional[str] = None*, *password: Optional[str] = None*) → Optional[bool]

Power on / power off / get power status /reboot a system.

Parameters

- **system** – Cobbler system
- **power_operation** – power operation. Valid values: on, off, reboot, status
- **user** – power management user
- **password** – power management password

Returns
bool if operation was successful

profiles() → *Profiles*

Return the current list of profiles

remove_distro(*ref: Union[*Distro*, str]*, *recursive: bool = False*, *delete: bool = True*, *with_triggers: bool = True*) → None

Remove a distribution from Cobbler.

Parameters

- **ref** – The internal unique handle for the item.
- **recursive** – If the item should recursively should delete dependencies on itself.
- **delete** – Not known what this parameter does exactly.
- **with_triggers** – Whether you would like to have the removal triggers executed or not.

remove_image(*ref: Union[*Image*, str]*, *recursive: bool = False*, *delete: bool = True*, *with_triggers: bool = True*) → None

Remove a image from Cobbler.

Parameters

- **ref** – The internal unique handle for the item.
- **recursive** – If the item should recursively should delete dependencies on itself.
- **delete** – Not known what this parameter does exactly.

- **with_triggers** – Whether you would like to have the removal triggers executed or not.

remove_item(*what: str*, *ref: Union[ITEM_UNION, str]*, *recursive: bool = False*, *delete: bool = True*, *with_triggers: bool = True*) → None

Remove a general item. This method should not be used by an external api. Please use the specific remove_<itemtype> methods.

Parameters

- **what** – The type of the item.
- **ref** – The internal unique handle for the item.
- **recursive** – If the item should recursively should delete dependencies on itself.
- **delete** – Not known what this parameter does exactly.
- **with_triggers** – Whether you would like to have the removal triggers executed or not.

remove_menu(*ref: Union[Menu, str]*, *recursive: bool = False*, *delete: bool = True*, *with_triggers: bool = True*) → None

Remove a menu from Cobbler.

Parameters

- **ref** – The internal unique handle for the item.
- **recursive** – If the item should recursively should delete dependencies on itself.
- **delete** – Not known what this parameter does exactly.
- **with_triggers** – Whether you would like to have the removal triggers executed or not.

remove_profile(*ref: Union[Profile, str]*, *recursive: bool = False*, *delete: bool = True*, *with_triggers: bool = True*) → None

Remove a profile from Cobbler.

Parameters

- **ref** – The internal unique handle for the item.
- **recursive** – If the item should recursively should delete dependencies on itself.
- **delete** – Not known what this parameter does exactly.
- **with_triggers** – Whether you would like to have the removal triggers executed or not.

remove_repo(*ref: Union[Repo, str]*, *recursive: bool = False*, *delete: bool = True*, *with_triggers: bool = True*) → None

Remove a repository from Cobbler.

Parameters

- **ref** – The internal unique handle for the item.
- **recursive** – If the item should recursively should delete dependencies on itself.
- **delete** – Not known what this parameter does exactly.
- **with_triggers** – Whether you would like to have the removal triggers executed or not.

remove_system(*ref: Union[System, str]*, *recursive: bool = False*, *delete: bool = True*, *with_triggers: bool = True*) → None

Remove a system from Cobbler.

Parameters

- **ref** – The internal unique handle for the item.
- **recursive** – If the item should recursively should delete dependencies on itself.
- **delete** – Not known what this parameter does exactly.
- **with_triggers** – Whether you would like to have the removal triggers executed or not.

rename_distro(*ref:* Distro, *newname: str*) → None

Rename a distro to a new name.

Parameters

- **ref** – The internal unique handle for the item.
- **newname** – The new name for the item.

rename_image(*ref:* Image, *newname: str*) → None

Rename an image to a new name.

Parameters

- **ref** – The internal unique handle for the item.
- **newname** – The new name for the item.

rename_item(*what: str*, *ref: ITEM_UNION*, *newname: str*) → None

Remove a general item. This method should not be used by an external api. Please use the specific rename_<itemtype> methods.

Parameters

- **what** – The type of object which should be renamed.
- **ref** – The internal unique handle for the item.
- **newname** – The new name for the item.

rename_menu(*ref:* Menu, *newname: str*) → None

Rename a menu to a new name.

Parameters

- **ref** – The internal unique handle for the item.
- **newname** – The new name for the item.

rename_profile(*ref:* Profile, *newname: str*) → None

Rename a profile to a new name.

Parameters

- **ref** – The internal unique handle for the item.
- **newname** – The new name for the item.

rename_repo(*ref:* Repo, *newname: str*) → None

Rename a repository to a new name.

Parameters

- **ref** – The internal unique handle for the item.
- **newname** – The new name for the item.

rename_system(*ref:* System, *newname: str*) → None

Rename a system to a new name.

Parameters

- **ref** – The internal unique handle for the item.
- **newname** – The new name for the item.

replicate(*cobbler_master: Optional[str] = None*, *port: str = '80'*, *distro_patterns: str = ''*, *profile_patterns: str = ''*, *system_patterns: str = ''*, *repo_patterns: str = ''*, *image_patterns: str = ''*, *prune: bool = False*, *omit_data: bool = False*, *sync_all: bool = False*, *use_ssl: bool = False*) → None

Pull down data/configs from a remote Cobbler server that is a master to this server.

Parameters

- **cobbler_master** – The hostname/URL of the other Cobbler server
- **port** – The port to use for the replication task.
- **distro_patterns** – The pattern of distros which should be synced.
- **profile_patterns** – The pattern of profiles which should be synced.
- **system_patterns** – The pattern of systems which should be synced.
- **repo_patterns** – The pattern of repositories which should be synced.
- **image_patterns** – The pattern of images which should be synced.
- **prune** – Whether the object not on the master should be removed or not.
- **omit_data** – If the data downloaded by the current Cobbler server should be rsynced to the destination server.
- **sync_all** – This parameter behaves similarly to a dry run argument. If True then everything will executed, if False then only some things are synced.
- **use_ssl** – Whether SSL should be used (True) or not (False).

report(*report_what: str = ''*, *report_name: str = ''*, *report_type: str = ''*, *report_fields: str = ''*, *report_noheaders: bool = False*) → None

Report functionality for Cobbler.

Parameters

- **report_what** – The object type that should be reported.
- **report_name** – The name of the object which should be possibly reported.
- **report_type** – May be either “text”, “csv”, “mediawiki”, “trac” or “doku”.
- **report_fields** – Specify “all” or the fields you want to be reported.
- **report_noheaders** – If the column headers should be included in the output or not.

repos() → *Repos*

Return the current list of repos

reposync(*name: Optional[str] = None*, *tries: int = 1*, *nofail: bool = False*) → None

Take the contents of `/var/lib/cobbler/repos` and update them – or create the initial copy if no contents exist yet.

Parameters

- **name** – The name of the repository to run reposync for.
- **tries** – How many tries should be executed before the action fails.

- **nofail** – If True then the action will fail, otherwise the action will just be skipped. This respects the `tries` parameter.

serialize() → None

Save the cobbler_collections to disk. Cobbler internal use only.

set_item_resolved_value(*item_uuid: str*, *attribute: str*, *value: Any*) → None

This method helps non Python API consumers to use the Python property setters without having access to the raw data of the object. In case you pass a dictionary the method tries to deduplicate it.

This does not help with network interfaces because they don't have a UUID at the moment and thus can't be queried via their UUID.

Warning: This function may throw any exception that is thrown by a setter of a Python property defined in Cobbler.

Parameters

- **item_uuid** – The UUID of the item that should be retrieved.
- **attribute** – The attribute that should be retrieved.
- **value** – The new value to set.

Raises

- **ValueError** – In case a value given was either malformed or the desired item did not exist.
- **TypeError** – In case the type of the method arguments do have the wrong type.
- **AttributeError** – In case the attribute specified is not available on the given item (type).

settings() → *Settings*

Return the application configuration

signature_update() → None

Update all signatures from the URL specified in the settings.

status(*mode: str*) → Union[Dict[Any, Any], str]

Get the status of the current Cobbler instance.

Parameters

mode – "text" or anything else. Meaning whether the output is thought for the terminal or not.

Returns

The current status of Cobbler.

sync(*verbose: bool = False*, *what: Optional[List[str]] = None*) → None

Take the values currently written to the configuration files in /etc, and /var, and build out the information tree found in /tftpboot. Any operations done in the API that have not been saved with serialize() will NOT be synchronized with this command.

Parameters

- **verbose** – If the action should be just logged as needed or (if True) as much verbose as possible.
- **what** – List of strings what services to sync (e.g. dhcp and/or dns). Empty list for full sync.

sync_dhcp() → None

Only build out the DHCP configuration.

sync_dns() → None

Only build out the DNS configuration.

sync_systems(*systems: List[str]*, *verbose: bool = False*) → None

Take the values currently written to the configuration files in /etc, and /var, and build out the information tree found in /tftpboot. Any operations done in the API that have not been saved with serialize() will NOT be synchronized with this command.

Parameters

- **systems** – List of specified systems that needs to be synced
- **verbose** – If the action should be just logged as needed or (if True) as much verbose as possible.

systems() → *Systems*

Return the current list of systems

validate_autoinstall_files() → None

Validate if any of the autoinstallation files are invalid and if yes report this.

version(*extended: bool = False*) → Union[float, Dict[str, Union[str, List[Any]]]]

What version is Cobbler?

If extended == False, returns a float for backwards compatibility If extended == True, returns a dict:

gitstamp – the last git commit hash gitdate – the last git commit date on the builder machine builddate – the time of the build version – something like "1.3.2" version_tuple – something like [1, 3, 2]

Parameters

extended – False returns a float, True a Dictionary.

8.4 cobbler.autoinstall_manager module

This module contains code in order to create the automatic installation files. For example kickstarts, autoyast files or preseed files.

class cobbler.autoinstall_manager.**AutoInstallationManager**(*api: CobblerAPI*)

Bases: object

Manage automatic installation templates, snippets and final files

generate_autoinstall(*profile: Optional[str] = None*, *system: Optional[str] = None*) → str

Generates the autoinstallation for a system or a profile. You may only specifify one parameter. If you specify both, the system is generated and the profile argument is ignored.

Parameters

- **profile** – The Cobbler profile you want an autoinstallation generated for.
- **system** – The Cobbler system you want an autoinstallation generated for.

Returns

The rendered template for the system or profile.

get_autoinstall_snippets() → List[str]

Get a list of all autoinstallation snippets.

Returns

The list of snippets

`get_autoinstall_templates`() → List[str]

Get automatic OS installation templates

Returns
A list of automatic installation templates

`is_autoinstall_in_use`(*name: str*) → bool

Reports the status if a given system is currently being provisioned.

Parameters
name – The name of the system.

Returns
Whether the system is in install mode or not.

`log_autoinstall_validation_errors`(*errors_type: int*, *errors: List[Any]*)

Log automatic installation file errors

Parameters

- **errors_type** – validation errors type
- **errors** – A list with all the errors which occurred.

`read_autoinstall_snippet`(*file_path: str*) → str

Reads a autoinstall snippet from underneath the configured snippet base dir.

Parameters
file_path – The relative file path under the configured snippets base dir.

Returns
The read snippet.

`read_autoinstall_template`(*file_path: str*) → str

Read an automatic OS installation template

Parameters
file_path – automatic installation template relative file path

Returns
automatic installation template content

`remove_autoinstall_snippet`(*file_path: str*) → bool

Remove the autoinstall snippet with the given path.

Parameters
file_path – The path relative to the configured snippet root.

Returns
A boolean indicating the success of the task.

`remove_autoinstall_template`(*file_path: str*)

Remove an automatic OS installation template

Parameters
file_path – automatic installation template relative file path

`validate_autoinstall_file`(*obj: Item*, *is_profile: bool*) → List[Any]

Validate automatic installation file used by a system/profile.

Parameters

- **obj** – system/profile
- **is_profile** – if obj is a profile

Returns
[bool, int, list] list with validation result, errors type and list of errors

validate_autoinstall_files() → bool

Determine if Cobbler automatic OS installation files will be accepted by corresponding Linux distribution installers. The presence of an error does not imply that the automatic installation file is bad, only that the possibility exists. Automatic installation file validators are not available for all automatic installation file types and on all operating systems in which Cobbler may be installed.

Returns
True if all automatic installation files are valid, otherwise false.

validate_autoinstall_snippet_file_path(*snippet: str*, *new_snippet: bool = False*) → str

Validate the snippet's relative file path.

Parameters

- **snippet** – automatic installation snippet relative file path
- **new_snippet** – when set to true new filenames are allowed

Returns
Snippet if successful otherwise raises an exception.

Raises

- **TypeError** – Raised in case `snippet` is not a string.
- **ValueError** – Raised in case snippet file is invalid.
- **OSError** – Raised in case snippet file location is not found.

validate_autoinstall_template_file_path(*autoinstall: str*, *for_item: bool = True*, *new_autoinstall: bool = False*) → str

Validate the automatic installation template's relative file path.

Parameters

- **autoinstall** – automatic installation template relative file path
- **for_item** – enable/disable special handling for Item objects
- **new_autoinstall** – when set to true new filenames are allowed

Returns
automatic installation template relative file path

Raises

- **TypeError** – Raised in case `autoinstall` is not a string.
- **OSError** – Raised in case template file not found.
- **ValueError** – Raised in case template file is invalid.

write_autoinstall_snippet(*file_path: str*, *data: str*)

Writes a snippet with the given content to the relative path under the snippet root directory.

Parameters

- **file_path** – The relative path under the configured snippet base dir.
- **data** – The snippet code.

write_autoinstall_template(*file_path: str*, *data: str*) → bool

Write an automatic OS installation template

Parameters

- **file_path** – automatic installation template relative file path
- **data** – automatic installation template content

8.5 cobbler.autoinstallgen module

Builds out filesystem trees/data based on the object tree. This is the code behind 'cobbler sync'.

class cobbler.autoinstallgen.**AutoInstallationGen**(*api:* CobblerAPI)

Bases: object

Handles conversion of internal state to the tftpboot tree layout

add_autoyast_script(*document: Document*, *script_type: str*, *source: str*)

Add scripts to an existing AutoYaST XML.

Parameters

- **document** – The existing AutoYaST XML object.
- **script_type** – The type of the script which should be added.
- **source** – The source of the script. This should be ideally a string.

create_autoyast_script(*document: Document*, *script: str*, *name: str*) → Element

This method attaches a script with a given name to an existing AutoYaST XML file.

Parameters

- **document** – The existing AutoYaST XML file.
- **script** – The script to attach.
- **name** – The name of the script.

Returns

The AutoYaST file with the attached script.

generate_autoinstall(*profile: Optional[Profile] = None*, *system: Optional[System] = None*) → str

This is an internal method for generating an autoinstall config/script. Please use the `generate_autoinstall_for_*` methods. If you insist on using this mehtod please only supply a profile or a system, not both.

Parameters

- **profile** – The profile to use for generating the autoinstall config/script.
- **system** – The system to use for generating the autoinstall config/script. If both arguments are given, this wins.

Returns

The autoinstall script or configuration file as a string.

generate_autoinstall_for_profile(*profile: str*) → str

Generate an autoinstall config or script for a profile.

Parameters

profile – The Profile to generate the script/config for.

Returns

The generated output or an error message with a human readable description.

Raises

CX – Raised in case the profile references a missing distro.

generate_autoinstall_for_system(*sys_name: str*) → str

Generate an autoinstall config or script for a system.

Parameters

sys_name – The system name to generate an autoinstall script for.

Returns

The generated output or an error message with a human readable description.

Raises
CX – Raised in case the system references a missing profile.

generate_autoyast(*profile: Optional[Profile] = None*, *system: Optional[System] = None*, *raw_data: Optional[str] = None*) → str

Generate auto installation information for SUSE distribution (AutoYaST XML file) for a specific system or general profile. Only a system OR profile can be supplied, NOT both.

Parameters

- **profile** – The profile to generate the AutoYaST file for.
- **system** – The system to generate the AutoYaST file for.
- **raw_data** – The raw data which should be included in the profile.

Returns
The generated AutoYaST XML file.

generate_config_stanza(*obj: Union[Profile, System]*, *is_profile: bool = True*) → str

Add in automatic to configure /etc/yum.repos.d on the remote system if the automatic installation file (template file) contains the magic $yum_config_stanza.

Parameters

- **obj** – The profile or system to generate a generate a config stanza for.
- **is_profile** – If the object is a profile. If False it is assumed that the object is a system.

Returns
The curl command to execute to get the configuration for a system or profile.

generate_repo_stanza(*obj: Union[Profile, System]*, *is_profile: bool = True*) → str

Automatically attaches yum repos to profiles/systems in automatic installation files (template files) that contain the magic $yum_repo_stanza variable. This includes repo objects as well as the yum repos that are part of split tree installs, whose data is stored with the distro (example: RHEL5 imports)

Parameters

- **obj** – The profile or system to generate the repo stanza for.
- **is_profile** – If True then obj is a profile, otherwise obj has to be a system. Otherwise this method will silently fail.

Returns
The string with the attached yum repos.

get_last_errors() → List[Any]

Returns the list of errors generated by the last template render action.

Returns
The list of error messages which are available. This may not only contain error messages related to generating autoinstallation configuration and scripts.

8.6 cobbler.cexceptions module

Custom exceptions for Cobbler

exception cobbler.cexceptions.**CX**(*value: Any*, **args: Iterable[str]*)

Bases: *CobblerException*

This is a general exception which gets thrown often inside Cobbler.

exception cobbler.cexceptions.**CobblerException**(*value: Any*, **args: Iterable[str]*)

Bases: Exception

This is the default Cobbler exception where all other exceptions are inheriting from.

8.7 cobbler.cli module

Command line interface for Cobbler.

class cobbler.cli.**CobblerCLI**(*cliargs*)

Bases: object

Main CLI Class which contains the logic to communicate with the Cobbler Server.

check_setup() → int

Detect permissions and service accessibility problems and provide nicer error messages for them.

cleanup_fault_string(*fault_str: str*) → str

Make a remote exception nicely readable by humans so it's not evident that is a remote fault. Users should not have to understand tracebacks.

Parameters
fault_str – The stacktrace to niceify.

Returns
A nicer error messsage.

direct_command(*action_name: str*)

Process non-object based commands like "sync" and "hardlink".

Parameters
action_name – The action to execute.

Returns
Depending on the action.

follow_task(*task_id*)

Parse out this task's specific messages from the global log

Parameters
task_id – The id of the task to follow.

get_direct_action(*object_type*, *args*) → Optional[str]

If this is a general command, e.g. "cobbler hardlink", return the action, like "hardlink"

Parameters

- **object_type** – Must be None or None is returned.
- **args** – The arg from the CLI.

Returns
The action key, "version" or None.

get_fields(*object_type: str*) → list

For a given name of an object type, return the FIELDS data structure.

Parameters
object_type – The object to return the fields of.

Returns
The fields or None

get_object_action(*object_type*, *args*) → Optional[str]

If this is a CLI command about an object type, e.g. "cobbler distro add", return the action, like "add"

Parameters

- **object_type** – The object type.
- **args** – The args from the CLI.

Returns

The action or None.

get_object_type(*args*) → Optional[str]

If this is a CLI command about an object type, e.g. "cobbler distro add", return the type, like "distro"

Parameters

args – The args from the CLI.

Returns

The object type or None

object_command(*object_type: str*, *object_action: str*) → int

Process object-based commands such as "distro add" or "profile rename"

Parameters

- **object_type** – The object type to execute an action for.
- **object_action** – The action to execute.

Returns

Depending on the object and action.

Raises

- **NotImplementedError** –
- **RuntimeError** –

print_help() → int

Prints general-top level help, e.g. "cobbler –help" or "cobbler" or "cobbler command-does-not-exist"

print_object_help(*object_type*) → int

Prints the subcommands for a given object, e.g. "cobbler distro –help"

Parameters

object_type – The object type to print the help for.

print_task(*task_id*)

Pretty print a task executed on the server. This prints to stdout.

Parameters

task_id – The id of the task to be pretty printed.

run(*args*) → int

Process the command line and do what the user asks.

Parameters

args – The args of the CLI

start_task(*name: str*, *options: dict*) → str

Start an asynchronous task in the background.

Parameters

- **name** – "background_" % name function must exist in remote.py. This function will be called in a subthread.
- **options** – Dictionary of options passed to the newly started thread

Returns
Id of the newly started task

cobbler.cli.**add_options_from_fields**(*object_type*, *parser*, *fields*, *network_interface_fields*, *settings*, *object_action*)

Add options to the command line from the fields queried from the Cobbler server.

Parameters

- **object_type** – The object type to add options for.
- **parser** – The optparse instance to add options to.
- **fields** – The list of fields to add options for.
- **network_interface_fields** – The list of network interface fields if the object type is a system.
- **settings** – Global cobbler settings as returned from `CollectionManager.settings()`
- **object_action** – The object action to add options for. May be "add", "edit", "find", "copy", "rename", "remove". If none of these options is given then this method does nothing.

cobbler.cli.**get_comma_separated_args**(*option: Option*, *opt_str*, *value: str*, *parser: OptionParser*)

Simple callback function to achieve option split with comma.

Reference for the method signature can be found at:
https://docs.python.org/3/library/optparse.html#defining-a-callback-option

Parameters

- **option** – The option the callback is executed for
- **opt_str** – Unused for this callback function. Would be the extended option if the user used the short version.
- **value** – The value which should be split by comma.
- **parser** – The optparse instance which the callback should be added to.

cobbler.cli.**list_items**(*remote*, *otype*)

List all items of a given object type and print it to stdout.

Parameters

- **remote** – The remote to use as the query-source.
- **otype** – The object type to query.

cobbler.cli.**main**() → int

CLI entry point

cobbler.cli.**n2s**(*data*)

Return spaces for None

Parameters
data – The data to check for.

Returns
The data itself or an empty string.

cobbler.cli.**opt**(*options*, *k*, *defval=''*)

Returns an option from an Optparse values instance

Parameters

- **options** – The options object to search in.

- **k** – The key which is in the optparse values instance.
- **defval** – The default value to return.

Returns
The value for the specified key.

cobbler.cli.**report_item**(*remote*, *otype: str*, *item=None*, *name=None*)

Return a single item in a given collection. Either this is an item object or this method searches for a name.

Parameters

- **remote** – The remote to use as the query-source.
- **otype** – The object type to query.
- **item** – The item to display
- **name** – The name to search for and display.

cobbler.cli.**report_items**(*remote*, *otype: str*)

Return all items for a given collection.

Parameters

- **remote** – The remote to use as the query-source. The remote to use as the query-source.
- **otype** – The object type to query.

cobbler.cli.**report_single_breed**(*name: str*, *items: dict*) → int

Helper function which prints a single signature breed list to the terminal.

cobbler.cli.**to_string_from_fields**(*item_dict*, *fields*, *interface_fields=None*) → str

item_dict is a dictionary, fields is something like item_distro.FIELDS :param item_dict: The dictionary representation of a Cobbler item. :param fields: This is the list of fields a Cobbler item has. :param interface_fields: This is the list of fields from a network interface of a system. This is optional. :return: The string representation of a Cobbler item with all its values.

8.8 cobbler.cobblerd module

8.9 cobbler.configgen module

configgen.py: Generate configuration data.

module for generating configuration manifest using autoinstall_meta data and templates for a given system (hostname)

class cobbler.configgen.**ConfigGen**(*cobbler_api: CobblerAPI*, *hostname: str*)

Bases: object

Generate configuration data for Cobbler's management resource "repos". Mainly used by Koan to configure systems.

gen_config_data() → Dict[Any, Any]

Generate configuration data for repos.

Returns
A dict which has all config data in it.

gen_config_data_for_koan() → str

Encode configuration data. Return json object for Koan.

Returns
A json string for koan.

get_cobbler_resource(*resource_key: str*) → Union[List[Any], str, Dict[Any, Any]]

Wrapper around Cobbler blender method

Parameters
resource_key – Not known what this actually is doing.

Returns
The blendered data. In some cases this is a str, in others it is a list or it might be a dict. In case the key is not found it will return an empty string.

resolve_resource_var(*string_data: Union[ResourceAction, str]*) → str

Substitute variables in strings with data from the `autoinstall_meta` dictionary of the system.

Parameters
string_data – The template which will then be substituted by the variables in this class.

Returns
A str with the substituted data. If the host_vars are not of type dict then this will return an empty str.

Raises
KeyError – When the autoinstall_meta variable does not contain the required Keys in the dict.

8.10 cobbler.decorator module

This module provides decorators that are required for Cobbler to work as expected.

class cobbler.decorator.**InheritableDictProperty**(*fget=None*, *fset=None*, *fdel=None*, *doc=None*)

Bases: *InheritableProperty*

This property is supposed to provide a way to identify properties in code that can be set to inherit.

class cobbler.decorator.**InheritableProperty**(*fget=None*, *fset=None*, *fdel=None*, *doc=None*)

Bases: *LazyProperty*

This property is supposed to provide a way to identify properties in code that can be set to inherit.

inheritable = True

class cobbler.decorator.**LazyProperty**(*fget=None*, *fset=None*, *fdel=None*, *doc=None*)

Bases: property

This property is supposed to provide a way to override the lazy-read value getter.

8.11 cobbler.download_manager module

Cobbler DownloadManager

class cobbler.download_manager.**DownloadManager**

Bases: object

TODO

urlread(*url: str*, *proxies: Any = None*, *cert: Optional[Union[str, Tuple[str, str]]] = None*) → Response

Read the content of a given URL and pass the requests. Response object to the caller.

Parameters

- **url** – The URL the request.
- **proxies** – Override the default Cobbler proxies.

- **cert** – Override the default Cobbler certs.

Returns
The Python `requests.Response` object.

8.12 cobbler.enums module

This module is responsible for containing all enums we use in Cobbler. It should not be dependent upon any other module except the Python standard library.

class cobbler.enums.**Archs**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: *ConvertableEnum*

This enum describes all system architectures which Cobbler is able to provision.

AARCH64 = 'aarch64'

ARM = 'arm'

I386 = 'i386'

IA64 = 'ia64'

PPC = 'ppc'

PPC64 = 'ppc64'

PPC64EL = 'ppc64el'

PPC64LE = 'ppc64le'

S390 = 's390'

S390X = 's390x'

X86_64 = 'x86_64'

class cobbler.enums.**BaudRates**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

This enum describes all baud rates which are commonly used.

B0 = 0

B110 = 110

B115200 = 115200

B1200 = 1200

B128000 = 128000

B14400 = 14400

B19200 = 19200

B2400 = 2400

B256000 = 256000

B300 = 300

B38400 = 38400

B4800 = 4800

B57600 = 57600

B600 = 600

B9600 = 9600

DISABLED = -1

class cobbler.enums.**ConvertableEnum**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

Abstract class to convert the enum via our convert method.

classmethod to_enum(*value: Union[str, CONVERTABLEENUM]*) → CONVERTABLEENUM

This method converts the chosen str to the corresponding enum type.

Parameters
value – str which contains the to be converted value.

Returns
The enum value.

Raises

- **TypeError** – In case value was not of type str.
- **ValueError** – In case value was not in the range of valid values.

class cobbler.enums.**DHCP**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

TODO

V4 = 4

V6 = 6

class cobbler.enums.**EventStatus**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: *ConvertableEnum*

This enums describes the status an event can have. The cycle is the following:

"Running" –> "Complete" or "Failed"

COMPLETE = 'complete'

Shows that an event did complete as desired

FAILED = 'failed'

Shows that an event did not complete as expected

INFO = 'notification'

Default Event status

RUNNING = 'running'

Shows that an event is currently being processed by the server

class cobbler.enums.**ImageTypes**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: *ConvertableEnum*

This enum represents all image types which Cobbler can manage.

DIRECT = 'direct'

ISO = 'iso'

MEMDISK = 'memdisk'

VIRT_CLONE = 'virt-clone'

class cobbler.enums.**ItemTypes**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: *ConvertableEnum*

This enum represents all valid item types in Cobbler. If a new item type is created it must be added into this enum. Abstract base item types don't have to be added here.

DISTRO = 'distro'

See *Distro()*

IMAGE = 'image'

See *Image()*

MENU = 'menu'

See *Menu()*

PROFILE = 'profile'

See *Profile()*

REPO = 'repo'

See *Repo()*

SYSTEM = 'system'

See *System()*

class cobbler.enums.**MirrorType**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: *ConvertableEnum*

This enum represents all mirror types which Cobbler can manage.

BASEURL = 'baseurl'

METALINK = 'metalink'

MIRRORLIST = 'mirrorlist'

NONE = 'none'

class cobbler.enums.**NetworkInterfaceType**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

This enum represents all interface types Cobbler is able to set up on a target host.

BMC = 6

BOND = 1

BONDED_BRIDGE_SLAVE = 5

BOND_SLAVE = 2

BRIDGE = 3

BRIDGE_SLAVE = 4

INFINIBAND = 7

NA = 0

class cobbler.enums.**RepoArchs**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: *ConvertableEnum*

This enum describes all repository architectures Cobbler is able to serve in case the content of the repository is serving the same architecture.

AARCH64 = 'aarch64'

ARM = 'arm'

I386 = 'i386'

IA64 = 'ia64'

NOARCH = 'noarch'

NONE = 'none'

PPC = 'ppc'

PPC64 = 'ppc64'

PPC64EL = 'ppc64el'

PPC64LE = 'ppc64le'

S390 = 's390'

SRC = 'src'

X86_64 = 'x86_64'

class cobbler.enums.**RepoBreeds**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: *ConvertableEnum*

This enum describes all repository breeds Cobbler is able to manage.

APT = 'apt'

NONE = 'none'

RHN = 'rhn'

RSYNC = 'rsync'

WGET = 'wget'

YUM = 'yum'

class cobbler.enums.**ResourceAction**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: *ConvertableEnum*

This enum represents all actions a resource may execute.

CREATE = 'create'

REMOVE = 'remove'

class cobbler.enums.**TlsRequireCert**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: *ConvertableEnum*

This enum represents all TLS validation server cert types which Cobbler can manage.

ALLOW = 'allow'

DEMAND = 'demand'

HARD = 'hard'

NEVER = 'never'

class cobbler.enums.**VirtDiskDrivers**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: *ConvertableEnum*

This enum represents all virtual disk driver Cobbler can handle.

INHERITED = '<<inherit>>'

QCOW2 = 'qcow2'

QED = 'qed'

RAW = 'raw'

VDI = 'vdi'

VDMK = 'vdmk'

class cobbler.enums.**VirtType**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: *ConvertableEnum*

This enum represents all known types of virtualization Cobbler is able to handle via Koan.

AUTO = 'auto'

INHERITED = '<<inherit>>'

KVM = 'kvm'

OPENVZ = 'openvz'

QEMU = 'qemu'

VMWARE = 'vmware'

VMWAREW = 'vmwarew'

XENFV = 'xenfv'

XENPV = 'xenpv'

8.13 cobbler.grub module

Module that contains GRUB related helper functionality.

cobbler.grub.**parse_grub_remote_file**(*file_location: str*) → Optional[str]

Parses a URI which grub would try to load from the network.

Parameters
file_location – The location which grub would try to load from the network.

Returns
In case the URL could be parsed it is returned in the converted format. Otherwise None is returned.

Raises

- **TypeError** – In case file_location is not of type `str`.
- **ValueError** – In case the file location does not contain a valid IPv4 or IPv6 address

8.14 cobbler.module_loader module

Module loader, adapted for Cobbler usage

class cobbler.module_loader.**ModuleLoader**(*api: CobblerAPI*, *module_path: str = ''*)

Bases: object

Class for dynamically loading Cobbler Plugins on startup

get_module_by_name(*name: str*) → Optional[module]

Get a module by its name. The category of the module is not needed.

Parameters
name – The name of the module.

Returns
The module asked by the function parameter.

get_module_from_file(*category: str*, *field: str*, *fallback_module_name: Optional[str] = None*) → module

Get Python module, based on name defined in configuration file

Parameters

- **category** – field category in configuration file
- **field** – field in configuration file
- **fallback_module_name** – default value used if category/field is not found in configuration file

Raises
CX – If unable to load Python module

Returns
A Python module.

get_module_name(*category: str*, *field: str*, *fallback_module_name: Optional[str] = None*) → str

Get module name from the settings.

Parameters

- **category** – Field category in configuration file.
- **field** – Field in configuration file

- **fallback_module_name** – Default value used if category/field is not found in configuration file

Raises

- **FileNotFoundError** – If unable to find configuration file.
- **ValueError** – If the category does not exist or the field is empty.
- **CX** – If the field could not be read and no fallback_module_name was given.

Returns
The name of the module.

get_modules_in_category(*category: str*) → List[module]

Return all modules of a module category.

Parameters
category – The module category.

Returns
A list of all modules of that category. Returns an empty list if the Category does not exist.

load_modules() → Tuple[Dict[str, module], Dict[str, Dict[str, module]]]

Load the modules from the path handed to the function into Cobbler.

Returns
Two dictionary's with the dynamically loaded modules.

8.15 cobbler.power_manager module

Power management library. Encapsulate the logic to run power management commands so that the Cobbler user does not have to remember different power management tools syntaxes. This makes rebooting a system for OS installation much easier.

class cobbler.power_manager.**PowerManager**(*api: CobblerAPI*)

Bases: object

Handles power management in systems

get_power_status(*system: System*, *user: Optional[str] = None*, *password: Optional[str] = None*) → Optional[bool]

Get power status for a system that has power management configured.

Parameters

- **system** (System) – Cobbler system
- **user** – power management user
- **password** – power management password

Returns
if system is powered on

power_off(*system: System*, *user: Optional[str] = None*, *password: Optional[str] = None*) → None

Powers down a system that has power management configured.

Parameters

- **system** (System) – Cobbler system
- **user** – power management user
- **password** – power management password

power_on(*system:* System, *user:* *Optional[str]* = *None*, *password:* *Optional[str]* = *None*) → None

Powers up a system that has power management configured.

Parameters

- **system** (System) – Cobbler system
- **user** – power management user
- **password** – power management password

reboot(*system:* System, *user:* *Optional[str]* = *None*, *password:* *Optional[str]* = *None*) → None

Reboot a system that has power management configured.

Parameters

- **system** (System) – Cobbler system
- **user** – power management user
- **password** – power management password

cobbler.power_manager.**get_power_command**(*power_type: str*) → Optional[str]

Get power management command path

Parameters
power_type – power management type

Returns
power management command path

cobbler.power_manager.**get_power_types**() → List[str]

Get possible power management types.

Returns
Possible power management types

cobbler.power_manager.**validate_power_type**(*power_type: str*) → None

Check if a power management type is valid.

Parameters
power_type – Power management type.

Raises
CX – if power management type is invalid

8.16 cobbler.remote module

This module contains all code related to the Cobbler XML-RPC API.

Changelog:

Schema: From -> To

Current Schema: Please refer to the documentation visible of the individual methods.

V3.4.0 (unreleased)

- **Added:**
  - set_item_resolved_value
  - input_string_or_list_no_inherit
  - input_string_or_list
  - input_string_or_dict
  - input_string_or_dict_no_inherit

        - input_boolean
        - input_int
    - **Changed:**
        - `get_random_mac: Change default *virt_type* ` to kvm

V3.3.4 (unreleased)

- No changes

V3.3.3

- **Added:**
    - get_item_resolved_value
    - dump_vars

V3.3.2

- No changes

V3.3.1

- **Changed:**
    - background_mkgrub: Renamed to background_mkloaders

V3.3.0

- **Added:**
    - background_syncsystems
    - background_mkgrub
    - get_menu
    - find_menu
    - get_menu_handle
    - remove_menu
    - copy_menu
    - rename_menu
    - new_menu
    - modify_menu
    - save_menu
    - get_valid_distro_boot_loaders
    - get_valid_image_boot_loaders
    - get_valid_profile_boot_loaders
    - get_valid_system_boot_loaders
    - get_menus_since
    - get_menu_as_rendered
- **Changed:**
    - generate_gpxe: Renamed to generate_ipxe
- **Removed:**
    - background_dlcontent
    - get_distro_for_koan

  - get_profile_for_koan
  - get_system_for_koan
  - get_repo_for_koan
  - get_image_for_koan
  - get_mgmtclass_for_koan
  - get_package_for_koan
  - get_file_for_koan
  - get_file_for_koan

V3.2.2

- No changes

V3.2.1

- **Added:**
  - auto_add_repos

V3.2.0

- No changes

V3.1.2

- No changes

V3.1.1

- No changes

V3.1.0

- No changes

V3.0.1

- No changes

V3.0.0

- **Added:**
  - generate_profile_autoinstall
  - generate_system_autoinstall
  - get_valid_archs
  - read_autoinstall_template
  - write_autoinstall_template
  - remove_autoinstall_template
  - read_autoinstall_snippet
  - write_autoinstall_snippet
  - remove_autoinstall_snippet
- **Changed:**
  - get_kickstart_templates: Renamed to get_autoinstall_templates
  - get_snippets: Renamed to get_autoinstall_snippets
  - is_kickstart_in_use: Renamed to is_autoinstall_in_use
  - generate_kickstart: Renamed to generate_autoinstall

- **Removed:**
  - update
  - read_or_write_kickstart_template
  - read_or_write_snippet

V2.8.5

- Inital tracking of changes.

class cobbler.remote.**CobblerXMLRPCInterface**(*api:* CobblerAPI)

Bases: object

This is the interface used for all XMLRPC methods, for instance, as used by koan or CobblerWeb.

Most read-write operations require a token returned from "login". Read operations do not.

auto_add_repos(*token: str*)

Parameters
token – The API-token obtained via the login() method.

background_aclsetup(*options: Dict[str, Any]*, *token: str*) → str

Get the acl configuration from the config and set the acls in the backgroud.

Parameters
- **options** – Not known what this parameter does.
- **token** – The API-token obtained via the login() method. The API-token obtained via the login() method.

Returns
The id of the task which was started.

background_buildiso(*options: Dict[str, Any]*, *token: str*) → str

Generates an ISO in /var/www/cobbler/pub that can be used to install profiles without using PXE.

Parameters
- **options** – This parameter does contain the options passed from the CLI or remote API who called this.
- **token** – The API-token obtained via the login() method. The API-token obtained via the login() method.

Returns
The id of the task which was started.

background_hardlink(*options: Dict[str, Any]*, *token: str*) → str

Hardlink all files as a background task.

Parameters
- **options** – Not known what this parameter does.
- **token** – The API-token obtained via the login() method. The API-token obtained via the login() method.

Returns
The id of the task which was started.

background_import(*options: Dict[str, Any]*, *token: str*) → str

Import an ISO image in the background.

Parameters
- **options** – Not known what this parameter does.

- **token** – The API-token obtained via the login() method. The API-token obtained via the login() method.

Returns
The id of the task which was started.

background_load_items() → str

Loading items

background_mkloaders(*options: Dict[str, Any]*, *token: str*) → str

TODO

Parameters

- **options** – TODO
- **token** – TODO

Returns
TODO

background_power_system(*options: Dict[str, Any]*, *token: str*) → str

Power a system asynchronously in the background.

Parameters

- **options** – Not known what this parameter does.
- **token** – The API-token obtained via the login() method. The API-token obtained via the login() method.

Returns
The id of the task which was started.

background_replicate(*options: Dict[str, Any]*, *token: str*) → str

Replicate Cobbler in the background to another Cobbler instance.

Parameters

- **options** – Not known what this parameter does.
- **token** – The API-token obtained via the login() method. The API-token obtained via the login() method.

Returns
The id of the task which was started.

background_reposync(*options: Dict[str, Any]*, *token: str*) → str

Run a reposync in the background.

Parameters

- **options** – Not known what this parameter does.
- **token** – The API-token obtained via the login() method. The API-token obtained via the login() method.

Returns
The id of the task which was started.

background_signature_update(*options: Dict[str, Any]*, *token: str*) → str

Run a signature update in the background.

Parameters

- **options** – Not known what this parameter does.
- **token** – The API-token obtained via the login() method. The API-token obtained via the login() method.

Returns
The id of the task which was started.

background_sync(*options: Dict[str, Any]*, *token: str*) → str

Run a full Cobbler sync in the background.

Parameters

- **options** – Possible options: verbose, dhcp, dns
- **token** – The API-token obtained via the login() method. The API-token obtained via the login() method.

Returns
The id of the task which was started.

background_syncsystems(*options: Dict[str, Any]*, *token: str*) → str

Run a lite Cobbler sync in the background with only systems specified.

Parameters

- **options** – Unknown what this parameter does.
- **token** – The API-token obtained via the login() method.

Returns
The id of the task that was started.

background_validate_autoinstall_files(*options: Dict[str, Any]*, *token: str*) → str

Validate all autoinstall files in the background.

Parameters

- **options** – Not known what this parameter does.
- **token** – The API-token obtained via the login() method. The API-token obtained via the login() method.

Returns
The id of the task which was started.

check(*token: str*) → List[str]

Returns a list of all the messages/warnings that are things that admin may want to correct about the configuration of the Cobbler server. This has nothing to do with "check_access" which is an auth/authz function in the XMLRPC API.

Parameters
token – The API-token obtained via the login() method. The API-token obtained via the login() method.

Returns
A list of things to address.

check_access(*token: Optional[str]*, *resource: str*, *arg1: Optional[str] = None*, *arg2: Any = None*) → int

Check if the token which was provided has access.

Parameters

- **token** – The token to check access for.
- **resource** – The resource for which access shall be checked.
- **arg1** – Arguments to hand to the authorization provider.
- **arg2** – Arguments to hand to the authorization provider.

Returns
If the operation was successful return `1`. If unsuccessful then return `0`. Other codes may be returned if specified by the currently configured authorization module.

`check_access_no_fail`(*token: str*, *resource: str*, *arg1: Optional[str] = None*, *arg2: Any = None*) → int

This is called by the WUI to decide whether an element is editable or not. It differs form check_access in that it is supposed to /not/ log the access checks (TBA) and does not raise exceptions.

Parameters

- **token** – The token to check access for.
- **resource** – The resource for which access shall be checked.
- **arg1** – Arguments to hand to the authorization provider.
- **arg2** – Arguments to hand to the authorization provider.

Returns
1 if the object is editable or 0 otherwise.

`clear_system_logs`(*object_id: str*, *token: str*) → bool

clears console logs of a system

Parameters

- **object_id** – The object id of the system to clear the logs of.
- **token** – The API-token obtained via the login() method.

Returns
True if the operation succeeds.

`copy_distro`(*object_id: str*, *newname: str*, *token: Optional[str] = None*)

Copies a distribution and renames it afterwards.

Parameters

- **object_id** – The object id of the item in question.
- **newname** – The new name for the copied object.
- **token** – The API-token obtained via the login() method.

Returns
True if the action succeeded.

`copy_image`(*object_id: str*, *newname: str*, *token: Optional[str] = None*)

Copies an image and renames it afterwards.

Parameters

- **object_id** – The object id of the item in question.
- **newname** – The new name for the copied object.
- **token** – The API-token obtained via the login() method.

Returns
True if the action succeeded.

`copy_item`(*what: str*, *object_id: str*, *newname: str*, *token: Optional[str] = None*)

Creates a new object that matches an existing object, as specified by an id.

Parameters

- **what** – The item type which should be copied.
- **object_id** – The object id of the item in question.

- **newname** – The new name for the copied object.
- **token** – The API-token obtained via the login() method.

Returns
True if the action succeeded.

copy_menu(*object_id: str*, *newname: str*, *token: Optional[str] = None*)

Copies a menu and rename it afterwards.

Parameters

- **object_id** – The object id of the item in question.
- **newname** – The new name for the copied object.
- **token** – The API-token obtained via the login() method.

Returns
True if the action succeeded.

copy_profile(*object_id: str*, *newname: str*, *token: Optional[str] = None*)

Copies a profile and renames it afterwards.

Parameters

- **object_id** – The object id of the item in question.
- **newname** – The new name for the copied object.
- **token** – The API-token obtained via the login() method.

Returns
True if the action succeeded.

copy_repo(*object_id: str*, *newname: str*, *token: Optional[str] = None*)

Copies a repository and renames it afterwards.

Parameters

- **object_id** – The object id of the item in question.
- **newname** – The new name for the copied object.
- **token** – The API-token obtained via the login() method.

Returns
True if the action succeeded.

copy_system(*object_id: str*, *newname: str*, *token: Optional[str] = None*)

Copies a system and renames it afterwards.

Parameters

- **object_id** – The object id of the item in question.
- **newname** – The new name for the copied object.
- **token** – The API-token obtained via the login() method.

Returns
True if the action succeeded.

disable_netboot(*name: str*, *token: Optional[str] = None*, ***rest: Any*) → bool

This is a feature used by the `pxe_just_once` support, see manpage. Sets system named "name" to no-longer PXE. Disabled by default as this requires public API access and is technically a read-write operation.

Parameters

- **name** – The name of the system to disable netboot for.

- **token** – The API-token obtained via the login() method.
- **rest** – This parameter is unused.

Returns
A boolean indicated the success of the action.

dump_vars(*item_uuid: str*, *formatted_output: bool = False*, *remove_dicts: bool = True*)

This function dumps all variables related to an object. The difference to the above mentioned function is that it accepts the item uid instead of the Python object itself.

See also:

Logically identical to *dump_vars()*

extended_version(*token: Optional[str] = None*, ***rest: Any*) → Dict[str, Union[str, List[str]]]

Returns the full dictionary of version information. See api.py for documentation.

Parameters

- **token** – The API-token obtained via the login() method.
- **rest** – This is dropped in this method since it is not needed here.

Returns
The extended version of Cobbler

find_distro(*criteria: Optional[Dict[str, Any]] = None*, *expand: bool = False*, *resolved: bool = False*, *token: Optional[str] = None*, ***rest: Any*) → List[Any]

Find a distro matching certain criteria.

Parameters

- **criteria** – The criteria a distribution needs to match.
- **expand** – Not only get the names but also the complete object in form of a dict.
- **resolved** – This only has an effect when `expand = True`. It returns the resolved representation of the object instead of the raw data.
- **token** – The API-token obtained via the login() method.
- **rest** – This parameter is not used currently.

Returns
All distributions which have matched the criteria.

find_image(*criteria: Optional[Dict[str, Any]] = None*, *expand: bool = False*, *resolved: bool = False*, *token: Optional[str] = None*, ***rest: Any*) → List[Any]

Find an image matching certain criteria.

Parameters

- **criteria** – The criteria a distribution needs to match.
- **expand** – Not only get the names but also the complete object in form of a dict.
- **resolved** – This only has an effect when `expand = True`. It returns the resolved representation of the object instead of the raw data.
- **token** – The API-token obtained via the login() method.
- **rest** – This parameter is not used currently.

Returns
All images which have matched the criteria.

find_items(*what: str*, *criteria: Optional[Dict[str, Any]] = None*, *sort_field: Optional[str] = None*, *expand: bool = False*, *resolved: bool = False*, *token: Optional[str] = None*, ***rest: Any*) → List[Any]

Works like get_items but also accepts criteria as a dict to search on.

Example: `{ "name" : "*.example.org" }`

Wildcards work as described by 'pydoc fnmatch'.

Parameters

- **what** – The object type to find.
- **criteria** – The criteria an item needs to match.
- **sort_field** – The field to sort the results after.
- **expand** – Not only get the names but also the complete object in form of a dict.
- **resolved** – This only has an effect when `expand = True`. It returns the resolved representation of the object instead of the raw data.

Returns

A list of dicts.

find_items_paged(*what: str*, *criteria: Optional[Dict[str, Any]] = None*, *sort_field: Optional[str] = None*, *page: int = 1*, *items_per_page: int = 25*, *resolved: bool = False*, *token: Optional[str] = None*)

Returns a list of dicts as with find_items but additionally supports returning just a portion of the total list, for instance in supporting a web app that wants to show a limited amount of items per page.

Parameters

- **what** – The object type to find.
- **criteria** – The criteria a distribution needs to match.
- **sort_field** – The field to sort the results after.
- **page** – The page to return
- **items_per_page** – The number of items per page.
- **resolved** – This only has an effect when `expand = True`. It returns the resolved representation of the object instead of the raw data.
- **token** – The API-token obtained via the login() method.

Returns

The found items.

find_menu(*criteria: Optional[Dict[str, Any]] = None*, *expand: bool = False*, *resolved: bool = False*, *token: Optional[str] = None*, ***rest: Any*) → List[Any]

Find a menu matching certain criteria.

Parameters

- **criteria** – The criteria a distribution needs to match.
- **expand** – Not only get the names but also the complete object in form of a dict.
- **resolved** – This only has an effect when `expand = True`. It returns the resolved representation of the object instead of the raw data.
- **token** – The API-token obtained via the login() method.
- **rest** – This parameter is not used currently.

Returns

All files which have matched the criteria.

find_profile(*criteria: Optional[Dict[str, Any]] = None*, *expand: bool = False*, *resolved: bool = False*, *token: Optional[str] = None*, ***rest: Any*) → List[Any]

Find a profile matching certain criteria.

Parameters

- **criteria** – The criteria a distribution needs to match.
- **expand** – Not only get the names but also the complete object in form of a dict.
- **resolved** – This only has an effect when `expand = True`. It returns the resolved representation of the object instead of the raw data.
- **token** – The API-token obtained via the login() method.
- **rest** – This parameter is not used currently.

Returns

All profiles which have matched the criteria.

find_repo(*criteria: Optional[Dict[str, Any]] = None*, *expand: bool = False*, *resolved: bool = False*, *token: Optional[str] = None*, ***rest: Any*) → List[Any]

Find a repository matching certain criteria.

Parameters

- **criteria** – The criteria a distribution needs to match.
- **expand** – Not only get the names but also the complete object in form of a dict.
- **resolved** – This only has an effect when `expand = True`. It returns the resolved representation of the object instead of the raw data.
- **token** – The API-token obtained via the login() method.
- **rest** – This parameter is not used currently.

Returns

All repositories which have matched the criteria.

find_system(*criteria: Optional[Dict[str, Any]] = None*, *expand: bool = False*, *resolved: bool = False*, *token: Optional[str] = None*, ***rest: Any*) → List[Any]

Find a system matching certain criteria.

Parameters

- **criteria** – The criteria a distribution needs to match.
- **expand** – Not only get the names but also the complete object in form of a dict.
- **resolved** – This only has an effect when `expand = True`. It returns the resolved representation of the object instead of the raw data.
- **token** – The API-token obtained via the login() method.
- **rest** – This parameter is not used currently.

Returns

All systems which have matched the criteria.

find_system_by_dns_name(*dns_name: str*) → Dict[str, Any]

This is used by the puppet external nodes feature.

Parameters

dns_name – The dns name of the system. This should be the fqdn and not only the hostname.

Returns

All system information or an empty dict.

generate_autoinstall(*profile: Optional[str] = None*, *system: Optional[str] = None*, *REMOTE_ADDR: Optional[Any] = None*, *REMOTE_MAC: Optional[Any] = None*, ***rest: Any*) → str

Generate the autoinstallation file and return it.

Parameters

- **profile** – The profile to generate the file for.
- **system** – The system to generate the file for.
- **REMOTE_ADDR** – This is dropped in this method since it is not needed here.
- **REMOTE_MAC** – This is dropped in this method since it is not needed here.
- **rest** – This is dropped in this method since it is not needed here.

Returns

The str representation of the file.

generate_bootcfg(*profile: Optional[str] = None*, *system: Optional[str] = None*, ***rest: Any*) → str

This generates the bootcfg for a system which is related to a certain profile.

Parameters

- **profile** – The profile which is associated to the system.
- **system** – The system which the bootcfg should be generated for.
- **rest** – This is dropped in this method since it is not needed here.

Returns

The generated bootcfg.

generate_ipxe(*profile: Optional[str] = None*, *image: Optional[str] = None*, *system: Optional[str] = None*, ***rest: Any*) → str

Generate the ipxe configuration.

Parameters

- **profile** – The profile to generate iPXE config for.
- **image** – The image to generate iPXE config for.
- **system** – The system to generate iPXE config for.
- **rest** – This is dropped in this method since it is not needed here.

Returns

The configuration as a str representation.

generate_profile_autoinstall(*profile: str*)

Generate a profile autoinstallation.

Parameters

profile – The profile to generate the file for.

Returns

The str representation of the file.

generate_script(*profile: Optional[str] = None*, *system: Optional[str] = None*, *name: str = ''*) → str

This generates the autoinstall script for a system or profile. Profile and System cannot be both given, if they are, Profile wins.

Parameters

- **profile** – The profile name to generate the script for.
- **system** – The system name to generate the script for.

- **name** – Name of the generated script. Must only contain alphanumeric characters, dots and underscores.

Returns
Some generated script.

`generate_system_autoinstall`(*system: str*)

Generate a system autoinstallation.

Parameters
system – The system to generate the file for.

Returns
The str representation of the file.

`get_authn_module_name`(*token: str*) → str

Get the name of the currently used authentication module.

Parameters
token – The API-token obtained via the login() method. Cobbler token, obtained form login()

Returns
The name of the module.

`get_autoinstall_snippets`(*token: Optional[str] = None*, ***rest: Any*)

Returns all the automatic OS installation templates' snippets.

Parameters

- **token** – The API-token obtained via the login() method.
- **rest** – This is dropped in this method since it is not needed here.

Returns
A list with all snippets.

`get_autoinstall_templates`(*token: Optional[str] = None*, ***rest: Any*)

Returns all of the automatic OS installation templates that are in use by the system.

Parameters

- **token** – The API-token obtained via the login() method.
- **rest** – This is dropped in this method since it is not needed here.

Returns
A list with all templates.

`get_blended_data`(*profile: Optional[str] = None*, *system: Optional[str] = None*)

Combine all data which is available from a profile and system together and return it.

Deprecated since version 3.4.0: Please make use of the dump_vars endpoint.

Parameters

- **profile** – The profile of the system.
- **system** – The system for which the data should be rendered.

Returns
All values which could be blended together through the inheritance chain.

`get_config_data`(*hostname: str*) → str

Generate configuration data for the system specified by hostname.

Parameters
hostname – The hostname for what to get the config data of.

Returns
The config data as a json for Koan.

`get_distro`(*name: str*, *flatten: bool = False*, *resolved: bool = False*, *token: Optional[str] = None*, ***rest: Any*)

Get a distribution.

Parameters

- **name** – The name of the distribution to get.
- **flatten** – If the item should be flattened.
- **resolved** – If this is True, Cobbler will resolve the values to its final form, rather than give you the objects raw value.
- **token** – The API-token obtained via the login() method. The API-token obtained via the login() method.
- **rest** – Not used with this method currently.

Returns
The item or None.

`get_distro_as_rendered`(*name: str*, *token: Optional[str] = None*, ***rest: Any*) → Union[List[Any], Dict[Any, Any], int, str, float]

Get distribution after passing through Cobbler's inheritance engine.

Parameters

- **name** – distro name
- **token** – authentication token
- **rest** – This is dropped in this method since it is not needed here.

Returns
Get a template rendered as a distribution.

`get_distro_handle`(*name: str*)

Get a handle for a distribution which allows you to use the functions `modify_*` or `save_*` to manipulate it.

Parameters
name – The name of the item.

Returns
The handle of the desired object.

`get_distros`(*page: Any = None*, *results_per_page: Any = None*, *token: Optional[str] = None*, ***rest: Any*) → List[Dict[str, Any]]

This returns all distributions.

Parameters

- **page** – This parameter is not used currently.
- **results_per_page** – This parameter is not used currently.
- **token** – The API-token obtained via the login() method. The API-token obtained via the login() method.
- **rest** – This parameter is not used currently.

Returns
The list with all distros.

get_distros_since(*mtime: float*) → Union[List[Any], Dict[Any, Any], int, str, float]

Return all of the distro objects that have been modified after mtime.

Parameters
mtime – The time after which all items should be included. Everything before this will be excluded.

Returns
The list of items which were modified after `mtime`.

get_event_log(*event_id: str*) → str

Returns the contents of a task log. Events that are not task-based do not have logs.

Parameters
event_id – The event-id generated by Cobbler.

Returns
The event log or a ?.

get_events(*for_user: str = ''*) → Dict[str, List[Union[str, float]]]

Returns a dict(key=event id) = [statetime, name, state, [read_by_who]]

Parameters
for_user – (Optional) Filter events the user has not seen yet. If left unset, it will return all events.

Returns
A dictionary with all the events (or all filtered events).

get_image(*name: str*, *flatten: bool = False*, *resolved: bool = False*, *token: Optional[str] = None*, ***rest: Any*)

Get an image.

Parameters

- **name** – The name of the image to get.
- **flatten** – If the item should be flattened.
- **resolved** – If this is True, Cobbler will resolve the values to its final form, rather than give you the objects raw value.
- **token** – The API-token obtained via the login() method. The API-token obtained via the login() method.
- **rest** – Not used with this method currently.

Returns
The item or None.

get_image_as_rendered(*name: str*, *token: Optional[str] = None*, ***rest: Any*) → Union[List[Any], Dict[Any, Any], int, str, float]

Get repository after passing through Cobbler's inheritance engine.

Parameters

- **name** – image name
- **token** – authentication token
- **rest** – This is dropped in this method since it is not needed here.

Returns
Get a template rendered as an image.

get_image_handle(*name: str*)

Get a handle for an image which allows you to use the functions `modify_*` or `save_*` to manipulate it.

Parameters
name – The name of the item.

Returns
The handle of the desired object.

get_images(*page: Any = None*, *results_per_page: Any = None*, *token: Optional[str] = None*, ***rest: Any*) → List[Dict[str, Any]]

This returns all images.

Parameters

- **page** – This parameter is not used currently.
- **results_per_page** – This parameter is not used currently.
- **token** – The API-token obtained via the login() method. The API-token obtained via the login() method.
- **rest** – This parameter is not used currently.

Returns
The list of all images.

get_images_since(*mtime: float*) → Union[List[Any], Dict[Any, Any], int, str, float]

See documentation for get_distros_since

Parameters
mtime – The time after which all items should be included. Everything before this will be excluded.

Returns
The list of items which were modified after `mtime`.

get_item(*what: str*, *name: str*, *flatten: bool = False*, *resolved: bool = False*)

Returns a dict describing a given object.

Parameters

- **what** – "distro", "profile", "system", "image", "repo", etc
- **name** – the object name to retrieve
- **flatten** – reduce dicts to string representations (True/False)
- **resolved** – If this is True, Cobbler will resolve the values to its final form, rather than give you the objects raw value.

Returns
The item or None.

get_item_handle(*what: str*, *name: str*) → str

Given the name of an object (or other search parameters), return a reference (object id) that can be used with `modify_*` functions or `save_*` functions to manipulate that object.

Parameters

- **what** – The collection where the item is living in.
- **name** – The name of the item.

Returns
The handle of the desired object.

get_item_names(*what: str*) → List[str]

This is just like get_items, but transmits less data.

Parameters
what – is the name of a Cobbler object type, as described for get_item.

Returns
Returns a list of object names (keys) for the given object type.

`get_item_resolved_value`(*item_uuid: str*, *attribute: str*) → Union[str, int, float, List[Any], Dict[Any, Any]]

See also:

Logically identical to *`get_item_resolved_value()`*

`get_items`(*what: str*) → List[Dict[str, Any]]

Individual list elements are the same for get_item.

Parameters
what – is the name of a Cobbler object type, as described for get_item.

Returns
This returns a list of dicts.

`get_menu`(*name: str*, *flatten: bool = False*, *resolved: bool = False*, *token: Optional[str] = None*, ***rest: Any*)

Get a menu.

Parameters

- **name** – The name of the file to get.
- **flatten** – If the item should be flattened.
- **resolved** – If this is True, Cobbler will resolve the values to its final form, rather than give you the objects raw value.
- **token** – The API-token obtained via the login() method. The API-token obtained via the login() method.
- **rest** – Not used with this method currently.

Returns
The item or None.

`get_menu_as_rendered`(*name: str*, *token: Optional[str] = None*, ***rest: Any*) → Union[List[Any], Dict[Any, Any], int, str, float]

Get menu after passing through Cobbler's inheritance engine

Parameters

- **name** – Menu name
- **token** – Authentication token
- **rest** – This is dropped in this method since it is not needed here.

Returns
Get a template rendered as a file.

`get_menu_handle`(*name: str*)

Get a handle for a menu which allows you to use the functions `modify_*` or `save_*` to manipulate it.

Parameters
name – The name of the item.

Returns
The handle of the desired object.

`get_menus`(*page: Any = None*, *results_per_page: Any = None*, *token: Optional[str] = None*, ***rest: Any*) → List[Dict[str, Any]]

This returns all menus.

Parameters

- **page** – This parameter is not used currently.
- **results_per_page** – This parameter is not used currently.
- **token** – The API-token obtained via the login() method.
- **rest** – This parameter is not used currently.

Returns
The list of all files.

get_menus_since(*mtime: float*) → Union[List[Any], Dict[Any, Any], int, str, float]

See documentation for get_distros_since

Parameters
mtime – The time after which all items should be included. Everything before this will be excluded.

Returns
The list of items which were modified after `mtime`.

get_profile(*name: str*, *flatten: bool = False*, *resolved: bool = False*, *token: Optional[str] = None*, ***rest: Any*)

Get a profile.

Parameters

- **name** – The name of the profile to get.
- **flatten** – If the item should be flattened.
- **resolved** – If this is True, Cobbler will resolve the values to its final form, rather than give you the objects raw value.
- **token** – The API-token obtained via the login() method. The API-token obtained via the login() method.
- **rest** – Not used with this method currently.

Returns
The item or None.

get_profile_as_rendered(*name: str*, *token: Optional[str] = None*, ***rest: Any*) → Union[List[Any], Dict[Any, Any], int, str, float]

Get profile after passing through Cobbler's inheritance engine.

Parameters

- **name** – profile name
- **token** – authentication token
- **rest** – This is dropped in this method since it is not needed here.

Returns
Get a template rendered as a profile.

get_profile_handle(*name: str*)

Get a handle for a profile which allows you to use the functions `modify_*` or `save_*` to manipulate it.

Parameters
name – The name of the item.

Returns
The handle of the desired object.

get_profiles(*page: Any = None*, *results_per_page: Any = None*, *token: Optional[str] = None*, ***rest: Any*) → List[Dict[str, Any]]

This returns all profiles.

Parameters

- **page** – This parameter is not used currently.
- **results_per_page** – This parameter is not used currently.
- **token** – The API-token obtained via the login() method. The API-token obtained via the login() method.
- **rest** – This parameter is not used currently.

Returns
The list with all profiles.

get_profiles_since(*mtime: float*) → Union[List[Any], Dict[Any, Any], int, str, float]

See documentation for get_distros_since

Parameters
mtime – The time after which all items should be included. Everything before this will be excluded.

Returns
The list of items which were modified after `mtime`.

get_random_mac(*virt_type: str = 'kvm'*, *token: Optional[str] = None*, ***rest: Any*) → str

Wrapper for `utils.get_random_mac()`. Used in the webui.

Parameters

- **virt_type** – The type of the virtual machine.
- **token** – The API-token obtained via the login() method. Auth token to authenticate against the api.
- **rest** – This is dropped in this method since it is not needed here.

Returns
The random mac address which shall be used somewhere else.

get_repo(*name: str*, *flatten: bool = False*, *resolved: bool = False*, *token: Optional[str] = None*, ***rest: Any*)

Get a repository.

Parameters

- **name** – The name of the repository to get.
- **flatten** – If the item should be flattened.
- **resolved** – If this is True, Cobbler will resolve the values to its final form, rather than give you the objects raw value.
- **token** – The API-token obtained via the login() method. The API-token obtained via the login() method.
- **rest** – Not used with this method currently.

Returns
The item or None.

get_repo_as_rendered(*name: str*, *token: Optional[str] = None*, ***rest: Any*) → Union[List[Any], Dict[Any, Any], int, str, float]

Get repository after passing through Cobbler's inheritance engine.

Parameters

- **name** – repository name
- **token** – authentication token

- **rest** – This is dropped in this method since it is not needed here.

Returns
Get a template rendered as a repository.

get_repo_config_for_profile(*profile_name: str*, ***rest: Any*)

Return the yum configuration a given profile should use to obtain all of it's Cobbler associated repos.

Parameters

- **profile_name** – The name of the profile for which the repository config should be looked up.
- **rest** – This is dropped in this method since it is not needed here.

Returns
The repository configuration for the profile.

get_repo_config_for_system(*system_name: str*, ***rest: Any*)

Return the yum configuration a given profile should use to obtain all of it's Cobbler associated repos.

Parameters

- **system_name** – The name of the system for which the repository config should be looked up.
- **rest** – This is dropped in this method since it is not needed here.

Returns
The repository configuration for the system.

get_repo_handle(*name: str*)

Get a handle for a repository which allows you to use the functions `modify_*` or `save_*` to manipulate it.

Parameters
name – The name of the item.

Returns
The handle of the desired object.

get_repos(*page: Any = None*, *results_per_page: Any = None*, *token: Optional[str] = None*, ***rest: Any*) → List[Dict[str, Any]]

This returns all repositories.

Parameters

- **page** – This parameter is not used currently.
- **results_per_page** – This parameter is not used currently.
- **token** – The API-token obtained via the login() method. The API-token obtained via the login() method.
- **rest** – This parameter is not used currently.

Returns
The list of all repositories.

get_repos_compatible_with_profile(*profile: str*, *token: Optional[str] = None*, ***rest: Any*) → List[Dict[Any, Any]]

Get repos that can be used with a given profile name.

Parameters

- **profile** – The profile to check for compatibility.
- **token** – The API-token obtained via the login() method.
- **rest** – This is dropped in this method since it is not needed here.

Returns
The list of compatible repositories.

`get_repos_since`(*mtime: float*) → Union[List[Any], Dict[Any, Any], int, str, float]

See documentation for get_distros_since

Parameters
mtime – The time after which all items should be included. Everything before this will be excluded.

Returns
The list of items which were modified after `mtime`.

`get_settings`(*token: Optional[str] = None*, ***rest: Any*) → Dict[str, Any]

Return the contents of our settings file, which is a dict.

Parameters
- **token** – The API-token obtained via the login() method.
- **rest** – Unused parameter.

Returns
Get the settings which are currently in Cobbler present.

`get_signatures`(*token: Optional[str] = None*, ***rest: Any*) → Dict[Any, Any]

Return the contents of the API signatures

Parameters
- **token** – The API-token obtained via the login() method.
- **rest** – This is dropped in this method since it is not needed here.

Returns
Get the content of the currently loaded signatures file.

`get_status`(*mode: str = 'normal'*, *token: Optional[str] = None*, ***rest: Any*) → Union[Dict[Any, Any], str]

Returns the same information as *cobbler status* While a read-only operation, this requires a token because it's potentially a fair amount of I/O

Parameters
- **mode** – How the status should be presented.
- **token** – The API-token obtained via the login() method. Auth token to authenticate against the api.
- **rest** – This parameter is currently unused for this method.

Returns
The human or machine readable status of the status of Cobbler.

`get_system`(*name: str*, *flatten: bool = False*, *resolved: bool = False*, *token: Optional[str] = None*, ***rest: Any*)

Get a system.

Parameters
- **name** – The name of the system to get.
- **flatten** – If the item should be flattened.
- **resolved** – If this is True, Cobbler will resolve the values to its final form, rather than give you the objects raw value.
- **token** – The API-token obtained via the login() method. The API-token obtained via the login() method.

- **rest** – Not used with this method currently.

Returns
The item or None.

get_system_as_rendered(*name: str*, *token: Optional[str] = None*, ***rest: Any*) → Union[List[Any], Dict[Any, Any], int, str, float]

Get profile after passing through Cobbler's inheritance engine.

Parameters

- **name** – system name
- **token** – authentication token
- **rest** – This is dropped in this method since it is not needed here.

Returns
Get a template rendered as a system.

get_system_handle(*name: str*)

Get a handle for a system which allows you to use the functions `modify_*` or `save_*` to manipulate it.

Parameters
name – The name of the item.

Returns
The handle of the desired object.

get_systems(*page: Any = None*, *results_per_page: Any = None*, *token: Optional[str] = None*, ***rest: Any*) → List[Dict[str, Any]]

This returns all Systems.

Parameters

- **page** – This parameter is not used currently.
- **results_per_page** – This parameter is not used currently.
- **token** – The API-token obtained via the login() method. The API-token obtained via the login() method.
- **rest** – This parameter is not used currently.

Returns
The list of all systems.

get_systems_since(*mtime: float*) → Union[List[Any], Dict[Any, Any], int, str, float]

See documentation for get_distros_since

Parameters
mtime – The time after which all items should be included. Everything before this will be excluded.

Returns
The list of items which were modified after `mtime`.

get_task_status(*event_id: str*) → List[Union[str, float, List[str]]]

Get the current status of the task.

Parameters
event_id – The unique id of the task.

Returns
The event status.

get_template_file_for_profile(*profile_name: str*, *path: str*, ***rest: Any*)

Return the templated file requested for this profile

Parameters

- **profile_name** – The name of the profile to get the template file for.
- **path** – The path to the template which is requested.
- **rest** – This is dropped in this method since it is not needed here.

Returns

The template file as a str representation.

get_template_file_for_system(*system_name: str*, *path: str*, ***rest: Any*)

Return the templated file requested for this system

Parameters

- **system_name** – The name of the system to get the template file for.
- **path** – The path to the template which is requested.
- **rest** – This is dropped in this method since it is not needed here.

Returns

The template file as a str representation.

get_tftp_file(*path: str*, *offset: int*, *size: int*, *token: str*) → Tuple[bytes, int]

Generate and return a file for a TFTP client.

Parameters

- **path** – Path to file
- **token** – The API-token obtained via the login() method
- **offset** – Offset of the requested chunk in the file
- **size** – Size of the requested chunk in the file

Returns

The requested chunk and the length of the whole file

get_user_from_token(*token: Optional[str]*) → str

Given a token returned from login, return the username that logged in with it.

Parameters

token – The API-token obtained via the login() method. The API-token obtained via the login() method.

Returns

The username if the token was valid.

Raises

- ***CX*** – If the token supplied to the function is invalid.
- **ValueError** – In case "token" did not fulfil the requirements to be a token.

get_valid_archs(*token: Optional[str] = None*) → List[str]

Return the list of valid architectures as read in from the distro signatures data

Parameters

token – The API-token obtained via the login() method.

Returns

Get a list of all valid architectures.

get_valid_breeds(*token: Optional[str] = None*, ***rest: Any*) → List[str]

Return the list of valid breeds as read in from the distro signatures data

Parameters

- **token** – The API-token obtained via the login() method.
- **rest** – This is dropped in this method since it is not needed here.

Returns

All valid OS-Breeds which are present in Cobbler.

get_valid_distro_boot_loaders(*distro_name: Optional[str]*, *token: Optional[str] = None*)

Return the list of valid boot loaders for the distro

Parameters

- **token** – The API-token obtained via the login() method.
- **distro_name** – The name of the distro for which the boot loaders should be looked up.

Returns

Get a list of all valid boot loaders.

get_valid_image_boot_loaders(*image_name: Optional[str]*, *token: Optional[str] = None*)

Return the list of valid boot loaders for the image

Parameters

- **token** – The API-token obtained via the login() method.
- **image_name** – The name of the image for which the boot loaders should be looked up.

Returns

Get a list of all valid boot loaders.

get_valid_os_versions(*token: Optional[str] = None*, ***rest: Any*) → List[str]

Return the list of valid os_versions as read in from the distro signatures data

Parameters

- **token** – The API-token obtained via the login() method.
- **rest** – This is dropped in this method since it is not needed here.

Returns

Get all valid OS-Versions

get_valid_os_versions_for_breed(*breed: str*, *token: Optional[str] = None*, ***rest: Any*) → List[str]

Return the list of valid os_versions for the given breed

Parameters

- **breed** – The OS-Breed which is requested.
- **token** – The API-token obtained via the login() method.
- **rest** – This is dropped in this method since it is not needed here.

Returns

All valid OS-versions for a certain breed.

get_valid_profile_boot_loaders(*profile_name: Optional[str]*, *token: Optional[str] = None*)

Return the list of valid boot loaders for the profile

Parameters

- **token** – The API-token obtained via the login() method.

- **profile_name** – The name of the profile for which the boot loaders should be looked up.

Returns
Get a list of all valid boot loaders.

get_valid_system_boot_loaders(*system_name: Optional[str]*, *token: Optional[str] = None*) → List[str]

Return the list of valid boot loaders for the system

Parameters

- **token** – The API-token obtained via the login() method.
- **system_name** – The name of the system for which the boot loaders should be looked up.

Returns
Get a list of all valid boot loaders.get_valid_archs

has_item(*what: str*, *name: str*, *token: Optional[str] = None*)

Returns True if a given collection has an item with a given name, otherwise returns False.

Parameters

- **what** – The collection to search through.
- **name** – The name of the item.
- **token** – The API-token obtained via the login() method.

Returns
True if item was found, otherwise False.

input_boolean(*value: Union[str, bool, int]*) → bool

See also:

input_boolean()

input_int(*value: Union[str, int, float]*) → int

See also:

input_int()

input_string_or_dict(*options: Union[str, List[Any], Dict[Any, Any]]*, *allow_multiples: bool = True*) → Union[str, Dict[Any, Any]]

See also:

input_string_or_dict()

input_string_or_dict_no_inherit(*options: Union[str, List[Any], Dict[Any, Any]]*, *allow_multiples: bool = True*) → Dict[Any, Any]

See also:

input_string_or_dict_no_inherit()

input_string_or_list(*options: Optional[Union[str, List[Any]]]*) → Union[List[Any], str]

See also:

input_string_or_list()

input_string_or_list_no_inherit(*options: Optional[Union[str, List[Any]]]*) → List[Any]

See also:

input_string_or_list_no_inherit()

is_autoinstall_in_use(*ai: str*, *token: Optional[str] = None*, ***rest: Any*)

Check if the autoinstall for a system is in use.

Parameters

- **ai** – The name of the system which could potentially be in autoinstall mode.
- **token** – The API-token obtained via the login() method.
- **rest** – This is dropped in this method since it is not needed here.

Returns

True if this is the case, otherwise False.

last_modified_time(*token: Optional[str] = None*) → float

Return the time of the last modification to any object. Used to verify from a calling application that no Cobbler objects have changed since last check. This method is implemented in the module api under the same name.

Parameters

token – The API-token obtained via the login() method. The API-token obtained via the login() method.

Returns

0 if there is no file where the information required for this method is saved.

login(*login_user: str*, *login_password: str*) → str

Takes a username and password, validates it, and if successful returns a random login token which must be used on subsequent method calls. The token will time out after a set interval if not used. Re-logging in permitted.

Parameters

- **login_user** – The username which is used to authenticate at Cobbler.
- **login_password** – The password which is used to authenticate at Cobbler.

Returns

The token which can be used further on.

logout(*token: str*) → bool

Retires a token ahead of the timeout.

Parameters

token – The API-token obtained via the login() method. Cobbler token, obtained form login()

Returns

if operation was successful or not

modify_distro(*object_id: str*, *attribute: str*, *arg: Any*, *token: str*)

Modify a single attribute of a distribution.

Parameters

- **object_id** – The id of the object which shall be modified.
- **attribute** – The attribute name which shall be edited.
- **arg** – The new value for the argument.
- **token** – The API-token obtained via the login() method.

Returns

True if the action was successful. Otherwise False.

modify_image(*object_id: str*, *attribute: str*, *arg: Any*, *token: str*)

Modify a single attribute of an image.

Parameters

- **object_id** – The id of the object which shall be modified.
- **attribute** – The attribute name which shall be edited.
- **arg** – The new value for the argument.
- **token** – The API-token obtained via the login() method.

Returns

True if the action was successful. Otherwise False.

modify_item(*what: str*, *object_id: str*, *attribute: str*, *arg: Union[str, int, float, List[str], Dict[str, Any]]*, *token: str*) → bool

Adjusts the value of a given field, specified by 'what' on a given object id. Allows modification of certain attributes on newly created or existing distro object handle.

Parameters

- **what** – The type of object to modify.
- **object_id** – The id of the object which shall be modified.
- **attribute** – The attribute name which shall be edited.
- **arg** – The new value for the argument.
- **token** – The API-token obtained via the login() method.

Returns

True if the action was successful. Otherwise False.

modify_menu(*object_id: str*, *attribute: str*, *arg: Any*, *token: str*)

Modify a single attribute of a menu.

Parameters

- **object_id** – The id of the object which shall be modified.
- **attribute** – The attribute name which shall be edited.
- **arg** – The new value for the argument.
- **token** – The API-token obtained via the login() method.

Returns

True if the action was successful. Otherwise False.

modify_profile(*object_id: str*, *attribute: str*, *arg: Any*, *token: str*)

Modify a single attribute of a profile.

Parameters

- **object_id** – The id of the object which shall be modified.
- **attribute** – The attribute name which shall be edited.
- **arg** – The new value for the argument.
- **token** – The API-token obtained via the login() method.

Returns

True if the action was successful. Otherwise False.

modify_repo(*object_id: str*, *attribute: str*, *arg: Any*, *token: str*)

Modify a single attribute of a repository.

Parameters

- **object_id** – The id of the object which shall be modified.
- **attribute** – The attribute name which shall be edited.
- **arg** – The new value for the argument.
- **token** – The API-token obtained via the login() method.

Returns

True if the action was successful. Otherwise False.

modify_setting(*setting_name: str*, *value: Union[str, bool, float, int, Dict[Any, Any], List[Any]]*, *token: str*) → int

Modify a single attribute of a setting.

Parameters

- **setting_name** – The name of the setting which shall be adjusted.
- **value** – The new value for the setting.
- **token** – The API-token obtained via the login() method.

Returns

0 on success, 1 on error.

modify_system(*object_id: str*, *attribute: str*, *arg: Any*, *token: str*)

Modify a single attribute of a system.

Parameters

- **object_id** – The id of the object which shall be modified.
- **attribute** – The attribute name which shall be edited.
- **arg** – The new value for the argument.
- **token** – The API-token obtained via the login() method.

Returns

True if the action was successful. Otherwise False.

new_distro(*token: str*)

See `new_item()`.

Parameters

token – The API-token obtained via the login() method.

Returns

The object id for the newly created object.

new_image(*token: str*)

See `new_item()`.

Parameters

token – The API-token obtained via the login() method.

Returns

The object id for the newly created object.

new_item(*what: str*, *token: str*, *is_subobject: bool = False*, ***kwargs: Any*) → str

Creates a new (unconfigured) object, returning an object handle that can be used.

Creates a new (unconfigured) object, returning an object handle that can be used with `modify_*` methods and then finally `save_*` methods. The handle only exists in memory until saved.

Parameters

- **what** – specifies the type of object: `distro`, `profile`, `system`, `repo`, `image` or `menu`.
- **token** – The API-token obtained via the login() method.
- **is_subobject** – If the object is a subobject of an already existing object or not.

Returns
The object id for the newly created object.

new_menu(*token: str*)

See `new_item()`.

Parameters
token – The API-token obtained via the login() method.

Returns
The object id for the newly created object.

new_profile(*token: str*)

See `new_item()`.

Parameters
token – The API-token obtained via the login() method.

Returns
The object id for the newly created object.

new_repo(*token: str*)

See `new_item()`.

Parameters
token – The API-token obtained via the login() method.

Returns
The object id for the newly created object.

new_subprofile(*token: str*)

See `new_item()`.

Parameters
token – The API-token obtained via the login() method.

Returns
The object id for the newly created object.

new_system(*token: str*)

See `new_item()`.

Parameters
token – The API-token obtained via the login() method.

Returns
The object id for the newly created object.

ping() → bool

Deprecated method. Now does nothing.

Returns
Always True

power_system(*system_id: str*, *power: str*, *token: str*) → bool

Execute power task synchronously.

Returns true if the operation succeeded or if the system is powered on (in case of status). False otherwise.

Parameters

- **token** – The API-token obtained via the login() method. The API-token obtained via the login() method. All tasks require tokens.
- **system_id** – system handle
- **power** – power operation (on/off/status/reboot)

read_autoinstall_snippet(*file_path: str*, *token: str*) → str

Read an automatic OS installation snippet file

Parameters

- **file_path** – automatic OS installation snippet file path
- **token** – The API-token obtained via the login() method. Cobbler token, obtained form login()

Returns
file content

read_autoinstall_template(*file_path: str*, *token: str*) → str

Read an automatic OS installation template file

Parameters

- **file_path** – automatic OS installation template file path
- **token** – The API-token obtained via the login() method. Cobbler token, obtained form login()

Returns
file content

register_new_system(*info: Dict[str, Any]*, *token: Optional[str] = None*, ***rest: Any*) → int

If register_new_installs is enabled in settings, this allows /usr/bin/cobbler-register (part of the koan package) to add new system records remotely if they don't already exist. There is a cobbler_register snippet that helps with doing this automatically for new installs but it can also be used for existing installs.

See "AutoRegistration" on the Wiki.

Parameters

- **info** – The system information which is provided by the system.
- **token** – The API-token obtained via the login() method.
- **rest** – This is dropped in this method since it is not needed here.

Returns
Return 0 if everything succeeded.

remove_autoinstall_snippet(*file_path: str*, *token: str*) → bool

Remove an automated OS installation snippet file

Parameters

- **file_path** – automated OS installation snippet file path
- **token** – Cobbler token, obtained form login()

Returns
bool if operation was successful

remove_autoinstall_template(*file_path: str*, *token: str*) → bool

Remove an automatic OS installation template file

Parameters

- **file_path** – automatic OS installation template file path
- **token** – The API-token obtained via the login() method. Cobbler token, obtained form login()

Returns
bool if operation was successful

remove_distro(*name: str*, *token: str*, *recursive: bool = True*)

Deletes a distribution from Cobbler.

Parameters

- **name** – The name of the item to remove.
- **token** – The API-token obtained via the login() method.
- **recursive** – If items which are depending on this one should be erased too.

Returns
True if the action was successful.

remove_image(*name: str*, *token: str*, *recursive: bool = True*)

Deletes an image from Cobbler.

Parameters

- **name** – The name of the item to remove.
- **token** – The API-token obtained via the login() method.
- **recursive** – If items which are depending on this one should be erased too.

Returns
True if the action was successful.

remove_item(*what: str*, *name: str*, *token: str*, *recursive: bool = True*) → bool

Deletes an item from a collection. Note that this requires the name of the distro, not an item handle.

Parameters

- **what** – The item type of the item to remove.
- **name** – The name of the item to remove.
- **token** – The API-token obtained via the login() method.
- **recursive** – If items which are depending on this one should be erased too.

Returns
True if the action was successful.

remove_menu(*name: str*, *token: str*, *recursive: bool = True*)

Deletes a menu from Cobbler.

Parameters

- **name** – The name of the item to remove.
- **token** – The API-token obtained via the login() method.
- **recursive** – If items which are depending on this one should be erased too.

Returns
True if the action was successful.

remove_profile(*name: str*, *token: str*, *recursive: bool = True*)

Deletes a profile from Cobbler.

Parameters

- **name** – The name of the item to remove.

- **token** – The API-token obtained via the login() method.
- **recursive** – If items which are depending on this one should be erased too.

Returns
True if the action was successful.

remove_repo(*name: str*, *token: str*, *recursive: bool = True*)
Deletes a repository from Cobbler.

Parameters

- **name** – The name of the item to remove.
- **token** – The API-token obtained via the login() method.
- **recursive** – If items which are depending on this one should be erased too.

Returns
True if the action was successful.

remove_system(*name: str*, *token: str*, *recursive: bool = True*)
Deletes a system from Cobbler.

Parameters

- **name** – The name of the item to remove.
- **token** – The API-token obtained via the login() method.
- **recursive** – If items which are depending on this one should be erased too.

Returns
True if the action was successful.

rename_distro(*object_id: str*, *newname: str*, *token: Optional[str] = None*) → bool
Renames a distribution specified by object_id to a new name.

Parameters

- **object_id** – The id which refers to the object.
- **newname** – The new name for the object.
- **token** – The API-token obtained via the login() method.

Returns
True if the action succeeded.

rename_image(*object_id: str*, *newname: str*, *token: Optional[str] = None*) → bool
Renames an image specified by object_id to a new name.

Parameters

- **object_id** – The id which refers to the object.
- **newname** – The new name for the object.
- **token** – The API-token obtained via the login() method.

Returns
True if the action succeeded.

rename_item(*what: str*, *object_id: str*, *newname: str*, *token: Optional[str] = None*) → bool
Renames an object specified by object_id to a new name.

Parameters

- **what** – The type of object which shall be renamed to a new name.
- **object_id** – The id which refers to the object.

- **newname** – The new name for the object.
- **token** – The API-token obtained via the login() method.

Returns
True if the action succeeded.

rename_menu(*object_id: str*, *newname: str*, *token: Optional[str] = None*) → bool

Renames a menu specified by object_id to a new name.

Parameters

- **object_id** – The id which refers to the object.
- **newname** – The new name for the object.
- **token** – The API-token obtained via the login() method.

Returns
True if the action succeeded.

rename_profile(*object_id: str*, *newname: str*, *token: Optional[str] = None*) → bool

Renames a profile specified by object_id to a new name.

Parameters

- **object_id** – The id which refers to the object.
- **newname** – The new name for the object.
- **token** – The API-token obtained via the login() method.

Returns
True if the action succeeded.

rename_repo(*object_id: str*, *newname: str*, *token: Optional[str] = None*) → bool

Renames a repository specified by object_id to a new name.

Parameters

- **object_id** – The id which refers to the object.
- **newname** – The new name for the object.
- **token** – The API-token obtained via the login() method.

Returns
True if the action succeeded.

rename_system(*object_id: str*, *newname: str*, *token: Optional[str] = None*) → bool

Renames a system specified by object_id to a new name.

Parameters

- **object_id** – The id which refers to the object.
- **newname** – The new name for the object.
- **token** – The API-token obtained via the login() method.

Returns
True if the action succeeded.

run_install_triggers(*mode: str*, *objtype: str*, *name: str*, *ip: str*, *token: Optional[str] = None*, ***rest: Any*)

This is a feature used to run the pre/post install triggers. See CobblerTriggers on Wiki for details

Parameters

- **mode** – The mode of the triggers. May be “pre”, “post” or “firstboot”.
- **objtype** – The type of object. This should correspond to the collection type.

- **name** – The name of the object.
- **ip** – The ip of the objet.
- **token** – The API-token obtained via the login() method.
- **rest** – This is dropped in this method since it is not needed here.

Returns
True if everything worked correctly.

save_distro(*object_id: str*, *token: str*, *editmode: str = 'bypass'*)

Saves a newly created or modified object to disk. Calling save is required for any changes to persist.

Parameters

- **object_id** – The id of the object to save.
- **token** – The API-token obtained via the login() method.
- **editmode** – The mode which shall be used to persist the changes. Currently "new" and "bypass" are supported.

Returns
True if the action succeeded.

save_image(*object_id: str*, *token: str*, *editmode: str = 'bypass'*)

Saves a newly created or modified object to disk. Calling save is required for any changes to persist.

Parameters

- **object_id** – The id of the object to save.
- **token** – The API-token obtained via the login() method.
- **editmode** – The mode which shall be used to persist the changes. Currently "new" and "bypass" are supported.

Returns
True if the action succeeded.

save_item(*what: str*, *object_id: str*, *token: str*, *editmode: str = 'bypass'*)

Saves a newly created or modified object to disk. Calling save is required for any changes to persist.

Parameters

- **what** – The type of object which shall be saved. This corresponds to the collections.
- **object_id** – The id of the object to save.
- **token** – The API-token obtained via the login() method.
- **editmode** – The mode which shall be used to persist the changes. Currently "new" and "bypass" are supported.

Returns
True if the action succeeded.

save_menu(*object_id: str*, *token: str*, *editmode: str = 'bypass'*)

Saves a newly created or modified object to disk. Calling save is required for any changes to persist.

Parameters

- **object_id** – The id of the object to save.
- **token** – The API-token obtained via the login() method.
- **editmode** – The mode which shall be used to persist the changes. Currently "new" and "bypass" are supported.

Returns
True if the action succeeded.

save_profile(*object_id: str*, *token: str*, *editmode: str = 'bypass'*)

Saves a newly created or modified object to disk. Calling save is required for any changes to persist.

Parameters

- **object_id** – The id of the object to save.
- **token** – The API-token obtained via the login() method.
- **editmode** – The mode which shall be used to persist the changes. Currently "new" and "bypass" are supported.

Returns

True if the action succeeded.

save_repo(*object_id: str*, *token: str*, *editmode: str = 'bypass'*)

Saves a newly created or modified object to disk. Calling save is required for any changes to persist.

Parameters

- **object_id** – The id of the object to save.
- **token** – The API-token obtained via the login() method.
- **editmode** – The mode which shall be used to persist the changes. Currently "new" and "bypass" are supported.

Returns

True if the action succeeded.

save_system(*object_id: str*, *token: str*, *editmode: str = 'bypass'*)

Saves a newly created or modified object to disk. Calling save is required for any changes to persist.

Parameters

- **object_id** – The id of the object to save.
- **token** – The API-token obtained via the login() method.
- **editmode** – The mode which shall be used to persist the changes. Currently "new" and "bypass" are supported.

Returns

True if the action succeeded.

set_item_resolved_value(*item_uuid: str*, *attribute: str*, *value: Any*, *token: Optional[str] = None*)

See also:

Logically identical to *set_item_resolved_value()*

sync(*token: str*) → bool

Run sync code, which should complete before XMLRPC timeout. We can't do reposync this way. Would be nice to send output over AJAX/other later.

Parameters

token – The API-token obtained via the login() method. Cobbler token, obtained form login()

Returns

bool if operation was successful

sync_dhcp(*token: str*) → bool

Run sync code, which should complete before XMLRPC timeout. We can't do reposync this way. Would be nice to send output over AJAX/other later.

Parameters

token – The API-token obtained via the login() method. Cobbler token, obtained form login()

Returns
bool if operation was successful

`token_check`(*token: str*) → bool

Checks to make sure a token is valid or not.

Parameters
token – The API-token obtained via the login() method. Cobbler token, obtained form login()

Returns
if operation was successful or not

`upload_log_data`(*sys_name: str*, *file: str*, *size: int*, *offset: int*, *data: Binary*, *token: Optional[str] = None*) → bool

This is a logger function used by the "anamon" logging system to upload all sorts of misc data from Anaconda. As it's a bit of a potential log-flooder, it's off by default and needs to be enabled in our settings.

Parameters
- **sys_name** – The name of the system for which to upload log data.
- **file** – The file where the log data should be put.
- **size** – The size of the data which will be received.
- **offset** – The offset in the file where the data will be written to.
- **data** – The data that should be logged.
- **token** – The API-token obtained via the login() method.

Returns
True if everything succeeded.

`version`(*token: Optional[str] = None*, ***rest: Any*)

Return the Cobbler version for compatibility testing with remote applications. See api.py for documentation.

Parameters
- **token** – The API-token obtained via the login() method.
- **rest** – This is dropped in this method since it is not needed here.

Returns
The short version of Cobbler.

`write_autoinstall_snippet`(*file_path: str*, *data: str*, *token: str*) → bool

Write an automatic OS installation snippet file

Parameters
- **file_path** – automatic OS installation snippet file path
- **data** – new file content
- **token** – Cobbler token, obtained form login()

Returns
if operation was successful

`write_autoinstall_template`(*file_path: str*, *data: str*, *token: str*) → bool

Write an automatic OS installation template file

Parameters
- **file_path** – automatic OS installation template file path

- **data** – new file content
- **token** – The API-token obtained via the login() method. Cobbler token, obtained form login()

Returns
bool if operation was successful

xapi_object_edit(*object_type: str*, *object_name: str*, *edit_type: str*, *attributes: Dict[str, Union[str, int, float, List[str]]]*, *token: str*)

Extended API: New style object manipulations, 2.0 and later.

Extended API: New style object manipulations, 2.0 and later preferred over using `new_*`, `modify_*`, `save_*` directly. Though we must preserve the old ways for backwards compatibility these cause much less XMLRPC traffic.

Ex: xapi_object_edit("distro","el5","add",{"kernel":"/tmp/foo","initrd":"/tmp/foo"},token)

Parameters

- **object_type** – The object type which corresponds to the collection type the object is in.
- **object_name** – The name of the object under question.
- **edit_type** – One of 'add', 'rename', 'copy', 'remove'
- **attributes** – The attributes which shall be edited. This should be JSON-style string.
- **token** – The API-token obtained via the login() method.

Returns
True if the action succeeded.

xmlrpc_hacks(*data: Optional[Union[List[Any], Dict[Any, Any], int, str, float]]*) → Union[List[Any], Dict[Any, Any], int, str, float]

Convert None in XMLRPC to just '~' to make extra sure a client that can't allow_none can deal with this.

ALSO: a weird hack ensuring that when dicts with integer keys (or other types) are transmitted with string keys.

Parameters
data – The data to prepare for the XMLRPC response.

Returns
The converted data.

class cobbler.remote.**CobblerXMLRPCServer**(*args: Any*)

Bases: ThreadingMixIn, SimpleXMLRPCServer

This is the class for the main Cobbler XMLRPC Server. This class does not directly contain all XMLRPC methods. It just starts the server.

class cobbler.remote.**ProxiedXMLRPCInterface**(*api:* CobblerAPI, *proxy_class: Type[Any]*)

Bases: object

TODO

class cobbler.remote.**RequestHandler**(*request*, *client_address*, *server*)

Bases: SimpleXMLRPCRequestHandler

TODO

do_OPTIONS() → None
TODO

end_headers() → None
TODO

8.17 cobbler.serializer module

Serializer code for Cobbler

class cobbler.serializer.**Serializer**(*api:* CobblerAPI)

Bases: object

Serializer interface that is used to access data in Cobbler independent of the actual data source.

deserialize(*collection:* Collection*[ITEM]*, *topological:* *bool* = *True*) → None

Load a collection from disk.

Parameters

- **collection** – The Cobbler collection to know the type of the item.
- **topological** – Sort collection based on each items' depth attribute in the list of collection items. This ensures properly ordered object loading from disk with objects having parent/child relationships, i.e. profiles/subprofiles. See cobbler/items/item.py

deserialize_item(*collection_type:* *str*, *item_name:* *str*) → Dict[str, Any]

Load a collection item from disk.

Parameters

- **collection_type** – The collection type to deserialize.
- **item_name** – The collection item name to deserialize.

serialize(*collection:* Collection*[ITEM]*) → None

Save a collection to disk

Parameters
collection – The collection to serialize.

serialize_delete(*collection:* Collection*[ITEM]*, *item:* *ITEM*) → None

Delete a collection item from disk

Parameters

- **collection** – The Cobbler collection to know the type of the item.
- **item** – The collection item to delete.

serialize_item(*collection:* Collection*[ITEM]*, *item:* *ITEM*) → None

Save a collection item to disk

Parameters

- **collection** – The Cobbler collection to know the type of the item.
- **item** – The collection item to serialize.

8.18 cobbler.services module

Mod Python service functions for Cobbler's public interface (aka cool stuff that works with wget/curl)

Changelog:

Schema: From -> To

Current Schema: Please refer to the documentation visible of the individual methods.

V3.4.0 (unreleased)

- No changes

V3.3.4 (unreleased)

- No changes

V3.3.3

- **Removed:**
  - `look`

V3.3.2

- No changes

V3.3.1

- No changes

V3.3.0

- **Added:**
  - `settings`
- **Changed:**
  - `gpxe`: Renamed to `ipxe`

V3.2.2

- No changes

V3.2.1

- No changes

V3.2.0

- No changes

V3.1.2

- No changes

V3.1.1

- No changes

V3.1.0

- No changes

V3.0.1

- No changes

V3.0.0

- **Added:**
  - `autoinstall`
  - `find_autoinstall`

V2.8.5

- Inital tracking of changes.

`class cobbler.services.CobblerSvc(server: str = '')`

Bases: `object`

Interesting mod python functions are all keyed off the parameter mode, which defaults to index. All options are passed as parameters into the function.

autodetect(***rest: Union[str, int, List[str]]*) → str

This tries to autodect the system with the given information. If more than one candidate is found an error message is returned.

Parameters
rest – The keys "REMOTE_MACS", "REMOTE_ADDR" or "interfaces".

Returns
The name of the possible object or an error message.

autoinstall(*profile: Optional[str] = None*, *system: Optional[str] = None*, *REMOTE_ADDR: Optional[str] = None*, *REMOTE_MAC: Optional[str] = None*, ***rest: Any*) → str

Generate automatic installation files.

Parameters

- **profile** –
- **system** –
- **REMOTE_ADDR** –
- **REMOTE_MAC** –
- **rest** – This parameter is unused.

Returns

bootcfg(*profile: Optional[str] = None*, *system: Optional[str] = None*, ***rest: Any*) → str

Generate a boot.cfg config file. Used primarily for VMware ESXi.

Parameters

- **profile** –
- **system** –
- **rest** – This parameter is unused.

Returns

events(*user: str = ''*, ***rest: Any*) → str

If no user is given then all events are returned. Otherwise only event associated to a user are returned.

Parameters

- **user** – Filter the events for a given user.
- **rest** – This parameter is unused.

Returns
A JSON object which contains all events.

find_autoinstall(*system: Optional[str] = None*, *profile: Optional[str] = None*, ***rest: Union[str, int]*) → str

Find an autoinstallation for a system or a profile. If this is not known different parameters can be passed to rest to find it automatically. See "autodetect".

Parameters

- **system** – The system to find the autoinstallation for,
- **profile** – The profile to find the autoinstallation for.
- **rest** – The metadata to find the autoinstallation automatically.

Returns
The autoinstall script or error message.

findks(*system: Optional[str] = None*, *profile: Optional[str] = None*, ***rest: Union[str, int]*) → str

This is a legacy function which enabled Cobbler partly to be backward compatible to 2.6.6 releases.

It should be only be used if you must. Please use find_autoinstall if possible! :param system: If you wish to find a system please set this parameter to not null. Hand over the name of it. :param profile: If you wish to find a system please set this parameter to not null. Hand over the name of it. :param rest: If you wish you can try to let Cobbler autodetect the system with the MAC address. :return: Returns the autoinstall/kickstart profile.

index(***args: Any*) → str

Just a placeholder method as an entry point.

Parameters
args – This parameter is unused.

Returns
"no mode specified"

ipxe(*profile: Optional[str] = None*, *image: Optional[str] = None*, *system: Optional[str] = None*, *mac: Optional[str] = None*, ***rest: Any*) → str

Generates an iPXE configuration.

Parameters

- **profile** – A profile.
- **image** – An image.
- **system** – A system.
- **mac** – A MAC address.
- **rest** – This parameter is unused.

ks(*profile: Optional[str] = None*, *system: Optional[str] = None*, *REMOTE_ADDR: Optional[str] = None*, *REMOTE_MAC: Optional[str] = None*, ***rest: Any*) → str

Generate automatic installation files. This is a legacy function for part backward compatibility to 2.6.6 releases.

Parameters

- **profile** –
- **system** –
- **REMOTE_ADDR** –
- **REMOTE_MAC** –
- **rest** – This parameter is unused.

Returns

list(*what: str = 'systems'*, ***rest: Any*) → str

Return a list of objects of a desired category. Defaults to "systems".

Parameters

- **what** – May be "systems", "profiles", "distros", "images", "repos" or "menus"
- **rest** – This parameter is unused.

Returns
The list of object names.

nopxe(*system: Optional[str] = None*, ***rest: Any*) → str

Disables the network boot for the given system.

Parameters

- **system** – The system to disable netboot for.
- **rest** – This parameter is unused.

Returns
A boolean status if the action succeed or not.

property remote: ServerProxy

Sets up the connection to the Cobbler XMLRPC server. This is the version that does not require a login.

script(*profile: Optional[str] = None*, *system: Optional[str] = None*, ***rest: Any*) → str

Generate a script based on snippets. Useful for post or late-action scripts where it's difficult to embed the script in the response file.

Parameters

- **profile** – The profile to generate the script for.
- **system** – The system to generate the script for.
- **rest** – This may contain a parameter with the key "query_string" which has a key "script" which may be an array. The element from position zero is taken.

Returns
The generated script.

settings(***kwargs: Any*) → str

Get the application configuration.

Returns
Settings object.

template(*profile: Optional[str] = None*, *system: Optional[str] = None*, *path: Optional[str] = None*, ***rest: Any*) → str

Generate a templated file for the system. Either specify a profile OR a system.

Parameters

- **profile** – The profile to provide for the generation of the template.
- **system** – The system to provide for the generation of the template.
- **path** – The path to the template.
- **rest** – This parameter is unused.

Returns
The rendered template.

trig(*mode: str = '?'*, *profile: Optional[str] = None*, *system: Optional[str] = None*, *REMOTE_ADDR: Optional[str] = None*, ***rest: Any*) → str

Hook to call install triggers. Only valid for a profile OR a system.

Parameters

- **mode** – Can be "pre", "post" or "firstboot". Everything else is invalid.
- **profile** – The profile object to run triggers for.
- **system** – The system object to run triggers for.
- **REMOTE_ADDR** – The ip if the remote system/profile.
- **rest** – This parameter is unused.

Returns
The return code of the action.

yum(*profile: Optional[str] = None*, *system: Optional[str] = None*, ***rest: Any*) → str

Generate a repo config. Either specify a profile OR a system.

Parameters

- **profile** – The profile to provide for the generation of the template.
- **system** – The system to provide for the generation of the template.
- **rest** – This parameter is unused.

Returns

The generated repository config.

cobbler.services.**application**(*environ: Dict[str, Any]*, *start_response: Callable[[str, List[Any]], None]*) → List[bytes]

UWSGI entrypoint for Gunicorn

Parameters

- **environ** –
- **start_response** –

Returns

8.19 cobbler.templar module

Cobbler uses Cheetah templates for lots of stuff, but there's some additional magic around that to deal with snippets/etc. (And it's not spelled wrong!)

class cobbler.templar.**Templar**(*api: CobblerAPI*)

Bases: object

Wrapper to encapsulate all logic of Cheetah vs. Jinja2. This also enables us to remove and add templating as desired via our self-defined API in this class.

check_for_invalid_imports(*data: str*) → None

Ensure that Cheetah code is not importing Python modules that may allow for advanced privileges by ensuring we whitelist the imports that we allow.

Parameters

data – The Cheetah code to check.

Raises

CX – Raised in case there could be a pontentially insecure import in the template.

render(*data_input: Union[TextIO, str]*, *search_table: Dict[Any, Any]*, *out_path: Optional[str]*, *template_type: str = 'default'*) → str

Render data_input back into a file.

Parameters

- **data_input** – is either a str or a TextIO object.
- **search_table** – is a dict of metadata keys and values.
- **out_path** – Optional parameter which (if present), represents the target path to write the result into.
- **template_type** – May currently be "cheetah" or "jinja2". "default" looks in the settings.

Returns

The rendered template.

render_cheetah(*raw_data: str*, *search_table: Dict[Any, Any]*) → str

Render data_input back into a file.

Parameters

- **raw_data** – Is the template code which is not rendered into the result.
- **search_table** – is a dict of metadata keys and values (though results are always returned)

Returns
The rendered Cheetah Template.

Raises

- **SyntaxError** – Raised in case the NFS paths has an invalid syntax.
- ***CX*** – Raised in case there was an error when templating.

render_jinja2(*raw_data: str*, *search_table: Dict[Any, Any]*) → str

Render data_input back into a file.

Parameters

- **raw_data** – Is the template code which is not rendered into the result.
- **search_table** – is a dict of metadata keys and values

Returns
The rendered Jinja2 Template.

8.20 cobbler.template_api module

Cobbler provides builtin methods for use in Cheetah templates. $SNIPPET is one such function and is now used to implement Cobbler's SNIPPET:: syntax.

class cobbler.template_api.**CobblerTemplate**(***kwargs: Any*)

Bases: DynamicallyCompiledCheetahTemplate

This class will allow us to include any pure python builtin functions. It derives from the cheetah-compiled class above. This way, we can include both types (cheetah and pure python) of builtins in the same base template.

SNIPPET(*file: str*) → Any

Include the contents of the named snippet here. This is equivalent to the #include directive in Cheetah, except that it searches for system and profile specific snippets, and it includes the snippet's namespace.

This may be a little frobby, but it's really cool. This is a pure python portion of SNIPPET that appends the snippet's searchList to the caller's searchList. This makes any #defs within a given snippet available to the template that included the snippet.

Parameters
file – The snippet file to read and include in the template.

Returns
The updated template.

classmethod compile(**args: Any*, ***kwargs: Any*) → bytes

Compile a cheetah template with Cobbler modifications. Modifications include SNIPPET:: syntax replacement and inclusion of Cobbler builtin methods. Please be aware that you cannot use the baseclass attribute of Cheetah anymore due to the fact that we are using it in our implementation to enable the Cheetah Macros.

Parameters

- **args** – These just get passed right to Cheetah.

- **kwargs** – We just execute our own preprocessors and remove them and let afterwards handle Cheetah the rest.

Returns
The compiled template.

read_snippet(*file: str*) → Optional[str]

Locate the appropriate snippet for the current system and profile and read its contents.

This file could be located in a remote location.

This will first check for a per-system snippet, a per-profile snippet, a distro snippet, and a general snippet.

Parameters
file – The name of the file to read. Depending on the context this gets expanded automatically.

Returns
None (if the snippet file was not found) or the string with the read snippet.

Raises

- **AttributeError** – Raised in case `autoinstall_snippets_dir` is missing.
- **FileNotFoundError** – Raised in case some files are not found.

sedesc(*value: str*) → str

Escape a string for use in sed.

This function is used by several cheetah methods in cheetah_macros. It can be used by the end user as well.

Example: Replace all instances of `/etc/banner` with a value stored in `$new_banner`

..code:

```
sed 's/$sedesc("/etc/banner")/$sedesc($new_banner)/'
```

Parameters
value – The phrase to escape.

Returns
The escaped phrase.

cobbler.template_api.**generate_cheetah_macros**() → Template

TODO

Returns
TODO

cobbler.template_api.**read_macro_file**(*location: str = '/etc/cobbler/cheetah_macros'*) → str

TODO

Parameters
location – TODO

Returns
TODO

8.21 cobbler.tftpgen module

Generate files provided by TFTP server based on Cobbler object tree. This is the code behind 'cobbler sync'.

class cobbler.tftpgen.**TFTPGen**(*api:* CobblerAPI)

Bases: object

Generate files provided by TFTP server

build_kernel(*metadata: Dict[str, Any]*, *system:* System, *profile:* Profile, *distro:* Distro, *image:* *Optional[*Image*]* = *None*, *boot_loader: str* = *'pxe'*)

Generates kernel and initrd metadata.

Parameters

- **metadata** – Pass additional parameters to the ones being collected during the method.
- **system** – The system to generate the pxe-file for.
- **profile** – The profile to generate the pxe-file for.
- **distro** – If you don't ship an image, this is needed. Otherwise this just supplies information needed for the templates.
- **image** – If you want to be able to deploy an image, supply this parameter.
- **boot_loader** – Can be any of those returned by utils.get_supported_system_boot_loaders().

build_kernel_options(*system: Optional[*System*]*, *profile: Optional[*Profile*]*, *distro: Optional[*Distro*]*, *image: Optional[*Image*]*, *arch:* Archs, *autoinstall_path: str*) → str

Builds the full kernel options line.

Parameters

- **system** – The system to generate the kernel options for.
- **profile** – Although the system contains the profile please specify it explicitly here.
- **distro** – Although the profile contains the distribution please specify it explicitly here.
- **image** – The image to generate the kernel options for.
- **arch** – The processor architecture to generate the kernel options for.
- **autoinstall_path** – The autoinstallation path. Normally this will be a URL because you want to pass a link to an autoyast, preseed or kickstart file.

Returns

The generated kernel line options.

copy_bootloaders(*dest: str*) → None

Copy bootloaders to the configured tftpboot directory NOTE: we support different arch's if defined in our settings file.

copy_images() → None

Like copy_distros except for images.

copy_single_distro_file(*d_file: str*, *distro_dir: str*, *symlink_ok: bool*) → None

Copy a single file (kernel/initrd) to distro's images directory

Parameters

- **d_file** – distro's kernel/initrd absolut or remote file path value
- **distro_dir** – directory (typically in {www,tftp}/images) where to copy the file
- **symlink_ok** – whethere it is ok to symlink the file. Typically false in case the file is used by daemons run in chroot environments (tftpd,..)

Raises
FileNotFoundError – Raised in case no kernel was found.

copy_single_distro_files(*distro: Distro*, *dirtree: str*, *symlink_ok: bool*)

Copy the files needed for a single distro.

Parameters

- **distro** – The distro to copy.
- **dirtree** – This is the root where the images are located. The folder "images" gets automatically appended.
- **symlink_ok** – If it is okay to use a symlink to link the destination to the source.

copy_single_image_files(*img: Image*)

Copies an image to the images directory of Cobbler.

Parameters
img – The image to copy.

generate_bootcfg(*what: str*, *name: str*) → str

Generate a bootcfg for a system of profile.

Parameters

- **what** – The type for what the bootcfg is generated for. Must be "profile" or "system".
- **name** – The name of the item which the bootcfg should be generated for.

Returns
The fully rendered bootcfg as a string.

generate_ipxe(*what: str*, *name: str*) → str

Generate the ipxe files.

Parameters

- **what** – Either "profile" or "system". All other item types not valid.
- **name** – The name of the profile or system.

Returns
The rendered template.

generate_pxe_menu(*path: Path*, *metadata: Dict[str, Union[str, Dict[str, str]]]*) → Optional[str]

Generate the requested menu file.

Parameters

- **path** – Path to the menu file.
- **metadata** – Menu items and other metadata for the generator.

generate_script(*what: str*, *objname: str*, *script_name: str*) → str

Generate a script from a autoinstall script template for a given profile or system.

Parameters

- **what** – The type for what the bootcfg is generated for. Must be "profile" or "system".
- **objname** – The name of the item which the bootcfg should be generated for.
- **script_name** – The name of the template which should be rendered for the system or profile.

Returns
The fully rendered script as a string.

generate_system_file(*system: System*, *path: Path*, *metadata: Dict[str, Union[str, Dict[str, str]]]*) → Optional[str]

Generate a single file for a system if the file is related to the system.

Parameters

- **system** – The system to generate the file for.
- **path** – The path to the file.
- **metadata** – Menu items and other metadata for the generator.

Returns

The contents of the file or None if the system does not provide this file.

generate_tftp_file(*path: Path*, *offset: int*, *size: int*) → Tuple[bytes, int]

Generate and return a file for a TFTP client.

Parameters

- **path** – Normalized absolute path to the file
- **offset** – Offset of the requested chunk in the file
- **size** – Size of the requested chunk in the file

Returns

The requested chunk and the length of the whole file

get_images_menu(*menu: Optional[Menu]*, *metadata: Dict[str, Any]*, *arch: Optional[Archs]*) → None

Generates profiles metadata for pxe, ipxe and grub.

Parameters

- **menu** – The menu for which boot files are generated. (Optional)
- **metadata** – Pass additional parameters to the ones being collected during the method.
- **arch** – The processor architecture to generate the menu items for. (Optional)

get_menu_items(*arch: Optional[Archs] = None*) → Dict[str, Union[str, Dict[str, str]]]

Generates menu items for pxe, ipxe and grub. Grub menu items are grouped into submenus by profile.

Parameters

arch – The processor architecture to generate the menu items for. (Optional)

Returns

A dictionary with the pxe, ipxe and grub menu items. It has the keys from utils.get_supported_system_boot_loaders().

get_menu_level(*menu: Optional[Menu] = None*, *arch: Optional[Archs] = None*) → Dict[str, Union[str, Dict[str, str]]]

Generates menu items for submenus, pxe, ipxe and grub.

Parameters

- **menu** – The menu for which boot files are generated. (Optional)
- **arch** – The processor architecture to generate the menu items for. (Optional)

Returns

A dictionary with the pxe and grub menu items. It has the keys from utils.get_supported_system_boot_loaders().

get_profiles_menu(*menu: Optional[Menu]*, *metadata: Dict[str, Any]*, *arch: Optional[Archs]*)

Generates profiles metadata for pxe, ipxe and grub.

Parameters

- **menu** – The menu for which boot files are generated. (Optional)

- **metadata** – Pass additional parameters to the ones being collected during the method.
- **arch** – The processor architecture to generate the menu items for. (Optional)

get_submenus(*menu: Optional[Menu]*, *metadata: Dict[Any, Any]*, *arch: Optional[Archs]*)

Generates submenus metatdata for pxe, ipxe and grub.

Parameters

- **menu** – The menu for which boot files are generated. (Optional)
- **metadata** – Pass additional parameters to the ones being collected during the method.
- **arch** – The processor architecture to generate the menu items for. (Optional)

make_pxe_menu() → Dict[str, Union[str, Dict[str, str]]]

Generates pxe, ipxe and grub boot menus.

write_all_system_files(*system: System*, *menu_items: Dict[str, Union[str, Dict[str, str]]]*) → None

Writes all files for tftp for a given system with the menu items handed to this method. The system must have a profile attached. Otherwise this method throws an error.

Directory structure:

```
TFTP Directory/
    pxelinux.cfg/
        01-aa-bb-cc-dd-ee-ff
    grub/
        system/
            aa:bb:cc:dd:ee:ff
        system_link/
            <system_name>
```

Parameters

- **system** – The system to generate files for.
- **menu_items** – The list of labels that are used for displaying the menu entry.

write_pxe_file(*filename: Optional[str]*, *system: Optional[System]*, *profile: Optional[Profile]*, *distro: Optional[Distro]*, *arch: Optional[Archs]*, *image: Optional[Image] = None*, *metadata: Optional[Dict[str, Union[str, Dict[str, str]]]] = None*, *bootloader_format: str = 'pxe'*) → str

Write a configuration file for the boot loader(s).

More system-specific configuration may come in later, if so that would appear inside the system object in api.py Can be used for different formats, “pxe” (default) and “grub”.

Parameters

- **filename** – If present this writes the output into the giving filename. If not present this method just returns the generated configuration.
- **system** – If you supply a system there are other templates used then when using only a profile/image/ distro.
- **profile** – The profile to generate the pxe-file for.
- **distro** – If you don’t ship an image, this is needed. Otherwise this just supplies information needed for the templates.
- **arch** – The processor architecture to generate the pxefile for.
- **image** – If you want to be able to deploy an image, supply this parameter.
- **metadata** – Pass additional parameters to the ones being collected during the method.

- **bootloader_format** – Can be any of those returned by utils.get_supported_system_boot_loaders().

Returns
The generated filecontent for the required item.

write_templates(*obj: ITEM_UNION*, *write_file: bool = False*, *path: Optional[str] = None*) → Dict[str, str]

A semi-generic function that will take an object with a template_files dict {source:destiation}, and generate a rendered file. The write_file option allows for generating of the rendered output without actually creating any files.

Parameters

- **obj** – The object to write the template files for.
- **write_file** – If the generated template should be written to the disk.
- **path** – TODO: A useless parameter?

Returns
A dict of the destination file names (after variable substitution is done) and the data in the file.

8.22 cobbler.validate module

Cobbler module that is related to validating data for other internal Cobbler modules.

cobbler.validate.**hostname**(*dnsname: str*) → str

Validate the DNS name.

Parameters
dnsname – Hostname or FQDN

Returns
Hostname or FQDN

Raises
TypeError – If the Hostname/FQDN is not a string or in an invalid format.

cobbler.validate.**ipv4_address**(*addr: str*) → str

Validate an IPv4 address.

Parameters
addr – IPv4 address

Returns
IPv4 address

Raises

- **TypeError** – Raised if `addr` is not a string.
- **AddressValueError** – Raised in case `addr` is not a valid IPv4 address.
- **NetmaskValueError** – Raised in case `addr` is not a valid IPv4 netmask.

cobbler.validate.**ipv4_netmask**(*addr: str*) → str

Validate an IPv4 netmask.

Parameters
addr – IPv4 netmask

Returns
IPv4 netmask

Raises

- **TypeError** – Raised if `addr` is not a string.
- **AddressValueError** – Raised in case `addr` is not a valid IPv4 address.
- **NetmaskValueError** – Raised in case `addr` is not a valid IPv4 netmask.

`cobbler.validate.ipv6_address`(*addr: str*) → str

Validate an IPv6 address.

Parameters
addr – IPv6 address

Returns
The IPv6 address.

Raises

- **TypeError** – Raised if `addr` is not a string.
- **AddressValueError** – Raised in case `addr` is not a valid IPv6 address.

`cobbler.validate.mac_address`(*mac: str*, *for_item: bool = True*) → str

Validate as an Ethernet MAC address.

Parameters

- **mac** – MAC address
- **for_item** – If the check should be performed for an item or not.

Returns
MAC address

Raises

- **ValueError** – Raised in case `mac` has an invalid format.
- **TypeError** – Raised in case `mac` is not a string.

`cobbler.validate.name_servers`(*nameservers: Union[str, List[str]]*, *for_item: bool = True*) → Union[str, List[str]]

Validate nameservers IP addresses, works for IPv4 and IPv6

Parameters

- **nameservers** – string or list of nameserver addresses
- **for_item** – enable/disable special handling for Item objects

Returns
The list of valid nameservers.

Raises

- **TypeError** – Raised if `nameservers` is not a string or list.
- **AddressValueError** – Raised in case `nameservers` is not a valid address.

`cobbler.validate.name_servers_search`(*search: Union[str, List[str]]*, *for_item: bool = True*) → Union[str, List[str]]

Validate nameservers search domains.

Parameters

- **search** – One or more search domains to validate.
- **for_item** – enable/disable special handling for Item objects

Returns
The list of valid nameservers.

Raises
`TypeError` – Raised if `search` is not a string or list.

`cobbler.validate.validate_autoinstall_script_name(name: str) → bool`

This validates if the name given for the script is valid in the context of the API call made. It will be handed to tftpgen.py#generate_script in the end.

Parameters
name – The name of the script. Will end up being a filename. May have an extension but should never be a path.

Returns
If this is a valid script name or not.

`cobbler.validate.validate_boot_remote_file(value: str) → bool`

This validates if the passed value is a valid value for `remote_boot_{kernel,initrd}`.

Parameters
value – Must be a valid URI starting with http or tftp. ftp is not supported and thus invalid.

Returns
False in any case. If value is valid, `True` is returned.

`cobbler.validate.validate_breed(breed: str) → str`

This is a setter for the operating system breed.

Parameters
breed – The os-breed which shall be set.

Raises
- `TypeError` – If breed is not a str.
- `ValueError` – If breed is not a supported breed.

`cobbler.validate.validate_grub_remote_file(value: str) → bool`

This validates if the passed value is a valid value for `remote_grub_{kernel,initrd}`.

Parameters
value – Must be a valid grub formatted URI starting with http or tftp. ftp is not supported and thus invalid.

Returns
False in any case. If value is valid, `True` is returned.

`cobbler.validate.validate_obj_name(object_name: str) → bool`

This validates the name of an object against the Cobbler specific object name schema.

Parameters
object_name – The object name candidate.

Returns
True in case it matches the RE_OBJECT_NAME regex, False in all other cases.

`cobbler.validate.validate_obj_type(object_type: str) → bool`

This validates the given object type against the available object types in Cobbler.

Parameters
object_type – The str with the object type to validate.

Returns
True in case it is one, False in all other cases.

`cobbler.validate.validate_os_version(os_version: str, breed: str) → str`

This is a setter for the operating system version of an object.

Parameters

- **os_version** – The version which shall be set.
- **breed** – The breed to validate the os_version for.

cobbler.validate.**validate_repos**(*repos: Union[List[str], str]*, *api:* CobblerAPI, *bypass_check: bool = False*) → Union[List[str], str]

This is a setter for the repository.

Parameters

- **repos** – The repositories to set for the object.
- **api** – The api to find the repos.
- **bypass_check** – If the newly set repos should be checked for existence.

cobbler.validate.**validate_serial_baud_rate**(*baud_rate: Union[int, str,* BaudRates*]*) → *BaudRates*

The baud rate is very import that the communication between the two devices can be established correctly. This is the setter for this parameter. This effectively is the speed of the connection.

Parameters
baud_rate – The baud rate to set.

Returns
The validated baud rate.

cobbler.validate.**validate_serial_device**(*value: Union[str, int]*) → int

Set the serial device for an object.

Parameters
value – The number of the serial device.

Returns
The validated device number

cobbler.validate.**validate_uuid**(*possible_uuid: str*) → bool

Validate if the handed string is a valid UUIDv4 hex representation.

Parameters
possible_uuid – The str with the UUID.

Returns
True in case it is one, False otherwise.

cobbler.validate.**validate_virt_auto_boot**(*value: Union[str, bool, int]*) → Union[bool, str]

For Virt only. Specifies whether the VM should automatically boot upon host reboot 0 tells Koan not to auto_boot virtuals.

Parameters
value – May be True or False.

cobbler.validate.**validate_virt_bridge**(*vbridge: str*) → str

The default bridge for all virtual interfaces under this profile.

Parameters
vbridge – The bridgename to set for the object.

Raises
TypeError – In case vbridge was not of type str.

cobbler.validate.**validate_virt_cpus**(*num: Union[str, int]*) → int

For Virt only. Set the number of virtual CPUs to give to the virtual machine. This is fed to virtinst RAW, so Cobbler will not yelp if you try to feed it 9999 CPUs. No formatting like 9,999 please :)

Zero means that the number of cores is inherited. Negative numbers are forbidden

Parameters
num – The number of cpu cores. If you pass the magic inherit string it will be converted to 0.

cobbler.validate.**validate_virt_file_size**(*num: Union[str, int, float]*) → Union[str, float]

For Virt only: Specifies the size of the virt image in gigabytes. Older versions of koan (x<0.6.3) interpret 0 as "don't care". Newer versions (x>=0.6.4) interpret 0 as "no disks"

Parameters
num – is a non-negative integer (0 means default). Can also be a comma seperated list – for usage with multiple disks (not working at the moment)

cobbler.validate.**validate_virt_path**(*path: str*, *for_system: bool = False*) → str

Virtual storage location suggestion, can be overriden by koan.

Parameters

- **path** – The path to the storage.
- **for_system** – If this is set to True then the value is inherited from a profile.

cobbler.validate.**validate_virt_pxe_boot**(*value: bool*) → bool

For Virt only. Specifies whether the VM should use PXE for booting 0 tells Koan not to PXE boot virtuals

Parameters
value – May be True or False.

Returns
True or False

cobbler.validate.**validate_virt_ram**(*value: Union[int, str]*) → Union[str, int]

For Virt only. Specifies the size of the Virt RAM in MB.

Parameters
value – 0 tells Koan to just choose a reasonable default.

Returns
An integer in all cases, except when `value` is the magic inherit string.

8.23 cobbler.yumgen module

Builds out filesystem trees/data based on the object tree. This is the code behind 'cobbler sync'.

class cobbler.yumgen.**YumGen**(*api: CobblerAPI*)

Bases: `object`

TODO

get_yum_config(*obj: Item*, *is_profile: bool*) → str

Return one large yum repo config blob suitable for use by any target system that requests it.

Parameters

- **obj** – The object to generate the yumconfig for.
- **is_profile** – If the requested object is a profile. (Parameter not used currently)

Returns
The generated yumconfig or the errors.

8.24 Module contents

This is the main Cobbler module. It contains all code related to the Cobbler server and the CLI. External applications should only make use of the `cobbler.api` module.

CHAPTER

NINE

RELEASE NOTES FOR COBBLER

The release notes can be found on GitHub.

CHAPTER

TEN

LIMITATIONS AND SURPRISES

10.1 Templating

Before templates are passed to Jinja or Cheetah there is a pre-processing of templates happening. During pre-processing Cobbler replaces variables like `@@my_key@@` in the template. Those keys are currently limited by the regex of `\S`, which translates to `[^ \t\n\r\f\v]`.

10.2 Restarting the daemon

Once you have a Cobbler distro imported or manually added you have to make sure the source for the Kernel & initrd is available all the time. Thus I highly recommend you to add the ISOs to your `/etc/fstab` to make them persistent across reboots. If you forget to remount them the Cobbler daemon won't start!

10.3 Kernel options

The user (so you) is responsible for generating the correct quoting of the Kernel Command Line. We manipulate the arguments you give us in a way that we add wrapping double quotes around them when the value contains a space.

The Linux Kernel describes its quoting at: The kernel's command-line parameters

Consult the documentation of your operating system for how it deals with this if it is not Linux.

10.4 Special Case: Uyuni/SUSE Manager

Note: SUSE Manager is a flavor of Uyuni. The term Uyuni refers to both pieces of software in this context.

Uyuni uses Cobbler for driving auto-installations. When using Cobbler in the context of Uyuni, you need to know that Cobbler is not seen as the source of truth by Uyuni. This means, in case you don't have any auto-installation configured in Uyuni, the content visible in Cobbler is deleted.

Because of the same reason, during the runtime of Cobbler you may see systems popping on and off as the content of Cobbler is managed by Uyuni (in particular, the taskomatic task `kickstart_cleanup` executes cleanup on the Cobbler content)

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y