Cobbler Documentation

Release 3.3.0

Enno Gotthold

Sep 06, 2021

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 1

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
cleaning link caches
running: find /var/lib/tftpboot/images/.link_cache -maxdepth 1 -type f -links 1 -
↪exec rm -f '{}' ';'
```

(continues on next page)

(continued from previous page)

```
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
```

(continues on next page)

(continued from previous page)

```
$ 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 2

Install Guide

Setting up and running *cobblerd* is not a easy task. Knowledge in Apache2 configuration (setting up SSL, virtual hosts, apache module and wsgi) 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 34 - `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 ancilary 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:

- createrepo_c
- httpd / apache2
- xorriso
- mod_wsgi / libapache2-mod-wsgi
- mod_ssl / libapache2-mod-ssl
- python-cheetah
- python-netaddr
- python-librepo
- python-schema
- PyYAML / python-yaml
- rsync
- syslinux
- tftp-server / atftpd
- dnf-plugins-core

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

- python3-ldap3 (or via PyPi: ldap3)

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-Cheetah3
- python3-Sphinx
- python3-coverage
- openssl
- apache2-devel (and thus apache2)

- A TFTP server

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 34
- CentOS 8
- Debian 10 Buster

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 34: `./docker/rpms/build-and-install-rpms.sh fc34 docker/rpms/Fedora_34/Fedora34.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`

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 master branch is the development series, and always uses an odd number for the minor version (for example, 3.1.0).

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`) and adjust `ExecStart` from `/usr/bin/cobblerd` to `/usr/local/bin/cobblerd`.
2. Install `apache2-mod_wsgi-python3` or the package responsible for your distro. (On Debian: `libapache2-mod-wsgi-py3`)
3. Enable the proxy module of Apache2 (`a2enmod proxy` or something similar) if not enabled.
4. Restart Apache and `cobblerd`.

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 3

Scripts

Warning: All execution examples are not meant to be copy&pasted! Cobbler instances are very custom and each command needs to be adjusted to your environment.

3.1 migrate-data-v2-to-v3.py

3.1.1 Description

This script tries to convert your old Cobbler 2.x.x data to Cobbler 3.x.x data. It won't make backups and can't rollback the changes it did.

3.1.2 Execution examples

```
python3 migrate-data-v2-to-v3.py
```

3.1.3 Author

Orion Poplawski

3.2 mkgrub.sh

3.2.1 Description

This script will try to generate UEFI bootable bootloaders for your Cobbler installation. The script was written for Bash but should be POSIX compliant. Be advised that only executing this script won't make your Cobbler installation UEFI ready.

3.2.2 Execution examples

```
export SYSLINUX_DIR=/usr/share/...;./mkgrub.sh
export $GRUB2_MOD_DIR=/usr/share/...;./mkgrub.sh
./mkgrub.sh
```

3.2.3 Author

Thomas Renninger

3.3 settings-migration-v1-to-v2.sh

3.3.1 Description

This script will try to replace your old `modules.conf` file (< 3.0.1) to a new one (>= 3.0.1).

3.3.2 Execution examples

```
./settings-migration-v1-to-v2.sh -h
./settings-migration-v1-to-v2.sh -r -f /etc/cobbler/modules.conf
./settings-migration-v1-to-v2.sh -n -f /etc/cobbler/modules.conf
./settings-migration-v1-to-v2.sh -s -f /etc/cobbler/modules.conf
```

3.3.3 Author

Enno Gotthold

CHAPTER 4

Cobbler CLI

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

4.1 General Principles

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

4.1.1 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.

4.1.2 Images

4.1.3 Repositories

4.1.4 Management Classes

4.1.5 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|mgmtclass|package|file|menu remove --
↪name=string
```

4.1.6 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|mgmtclass|package|file|menu edit --
↪name=string [parameterlist]
```

4.1.7 Copying

Objects can also be copied:

```
cobbler distro|profile|system|repo|image|mgmtclass|package|file|menu copy --
↪name=oldname --newname=newname
```

4.1.8 Renaming

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

```
cobbler distro|profile|system|repo|image|mgmtclass|package|file|menu rename --
↪name=oldname --newname=newname
```

4.2 CLI-Commands

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

Long Usage:

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

4.2.1 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 `authz_ownership` module which must be enabled in `/etc/cobbler/modules.conf`. 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).

4.2.2 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.

Continued 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 `authz_ownership` module which must be enabled in `/etc/cobbler/modules.conf`. 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.

Continued 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 `xenbr0`. If using KVM, 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.

Continued 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`). 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.

4.2.3 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

Continued 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

Continued 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 https://github.com/cobbler/cobbler/wiki/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
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.

Continued on next page

Table 2 – continued from previous page

Name	Description
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`
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.

Continued on next page

Table 2 – continued from previous page

Name	Description
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.
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.

Continued on next page

Table 2 – continued from previous page

Name	Description
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 `authz_ownership` module which must be enabled in `/etc/cobbler/modules.conf`. 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?).
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 `xenbr0`. If using KVM, 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.

Continued on next page

Table 2 – continued from previous page

Name	Description
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`). 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.

4.2.4 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 `authz_ownership` module which must be enabled in `/etc/cobbler/modules.conf`. The value for `--owners` is a space separated list of users and groups as specified in `/etc/cobbler/users.conf`.	

```
$ 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.

4.2.5 Cobbler image

Example:

```
$ cobbler image
```

4.2.6 cobbler mgmtclass

Management classes allows Cobbler to function as an configuration management system. Cobbler currently supports the following resource types:

1. Packages
2. Files

Resources are executed in the order listed above.

```
$ cobbler mgmtclass add --name=string --comment=string [--packages=list] [--
↪files=list]
```

Name	Description
classname	Class Name (Actual Class Name (leave blank to use the name field)).
comment	A comment that describes the functions of the management class.
files	Specifies a list of file resources required by the management class.
name	The name of the mgmtclass. Use this name when adding a management class to a system, profile, or distro. To add a mgmtclass to an existing system use something like (`cobbler system edit --name="madhatter" --mgmt-classes="http mysql"`).
packages	Specifies a list of package resources required by the management class.

4.2.7 Cobbler package

Package resources are managed using `cobbler package add`

Actions:

Name	Description
install	Install the package. [Default]
uninstall	Uninstall the package.

Attributes:

Name	Description
installer	Which package manager to use, valid options [rpm|yum].
name	Cobbler object name.
version	Which version of the package to install.

Example:

```
$ cobbler package add --name=string --comment=string [--action=install|uninstall] --
↪installer=string [--version=string]
```

4.2.8 Cobbler file

Actions:

Name	Description
create	Create the file. [Default]
remove	Remove the file.

Attributes:

Name	Description
group	The group owner of the file.
mode	Permission mode (as in chmod).
name	Name of the cobbler file object
path	The path for the file.
template	The template for the file.
user	The user for the file.

Example:

```
$ cobbler file add --name=string --comment=string [--action=string] --mode=string --
↪group=string --owner=string --path=string [--template=string]
```

4.2.9 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.

4.2.10 cobbler aclsetup

Example:

```
$ cobbler aclsetup
```

4.2.11 Cobbler buildiso

Example:

```
$ cobbler buildiso
```

4.2.12 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
```

4.2.13 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, mgmtclass, package, file)

```
$ cobbler list
```

4.2.14 Cobbler replicate

Cobbler can replicate configurations from a master Cobbler server. Each Cobbler server is still expected to have a locally relevant `/etc/cobbler/cobbler.conf` and `modules.conf`, as these files are 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]
```

4.2.15 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, mgmtclass, package, file, menu).

```
$ cobbler report --name=[object-name]
```

–name=[object-name]

Optional parameter which filters for object with the given name.

4.2.16 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 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.

4.2.17 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]
```

4.2.18 Cobbler validate-autoinstalls

Example:

```
$ cobbler validate-autoinstalls
```

4.2.19 Cobbler version

Example:

```
$ cobbler version
```

4.2.20 Cobbler signature

Example:

```
$ cobbler signature
```

4.2.21 Cobbler hardlink

Example:

```
$ cobbler hardlink
```

4.3 EXIT_STATUS

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

4.4 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 5

Cobblerd

Cobbler - a provisioning and update server

5.1 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.

5.2 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.

5.3 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.

5.4 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.

5.5 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 6

Cobbler Configuration

There are two main settings files which are located per default at `/etc/cobbler/`:

- The file `settings.yaml` is following YAML specification.
- The file `modules.conf` is following INI specification.

Note: Since we are cleaning a lot of tech-debt this may change over time. We are trying to find the balance which format is the best for us to handle in the code and the best for admins to handle in the config files.

Warning: If you are using `allow_dynamic_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)

There are additional configuration file locations which need to follow the YAML Syntax. These are loaded from the `include` directory in the `settings.yaml` file. Any key specified in one of these files overwrites values from the main file.

Warning: When using `allow_dynamic_settings` the values are only persisted in the file `settings.yaml`. This may lead to a non expected behaviour after `cobblerd` restarts. This is a known issue.

6.1 Updates to the yaml-settings-file

6.1.1 Starting with 3.3.0

- The setting `enable_gpxe` was replaced with `enable_ipxe`.
- The `settings.d` directory (`/etc/cobbler/settings.d/`) was depreaced 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.

6.1.2 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.

6.2 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
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
master	–	–	–	–	–	–	–	–	–	–

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

Note: Downgrades are not supported!

6.3 `settings.yaml`

6.3.1 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`

6.3.2 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`

6.3.3 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`

6.3.4 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`

6.3.5 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`

6.3.6 always_write_dhcp_entries

Always write DHCP entries, regardless if netboot is enabled.

default: `False`

6.3.7 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`

6.3.8 auth_token_expiration

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

default: `3600`

6.3.9 authn_pam_service

If using authn_pam in the `modules.conf`, this can be configured to change the PAM service authentication will be tested against.

default: `"login"`

6.3.10 autoinstall

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

default: `default.ks`

6.3.11 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`

6.3.12 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`

6.3.13 bind_chroot_path

Set to path of bind chroot to create bind-chroot compatible bind configuration files. This should be automatically detected.

default: `""`

6.3.14 bind_master

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

default: `127.0.0.1`

6.3.15 bind_zonefile_path

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

default: `"@@bind_zonefiles@@"`

6.3.16 boot_loader_conf_template_dir

Location of templates used for boot loader config generation.

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

6.3.17 grubconfig_dir

The location where Cobbler searches for GRUB configuration files.

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

6.3.18 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: TODO
- 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: [ "" ]
```

6.3.19 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`

6.3.20 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`

6.3.21 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`

6.3.22 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`

6.3.23 cobbler_master

Used for replicating the Cobbler instance.

default: `""`

6.3.24 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`

6.3.25 createrepo_flags

Default `createrepo_flags` to use for new repositories.

default: `"-c cache -s sha"`

6.3.26 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: []
```

6.3.27 default_ownership

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

default:

- `admin`

6.3.28 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."`

6.3.29 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"`

6.3.30 default_virt_bridge

For libvirt based installs in Koan, if no virt-bridge is specified, which bridge do we try? For EL 4/5 hosts this should be `xenbr0`, for all versions of Fedora, try `virbr0`. 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: `xenbr0`

6.3.31 default_virt_disk_driver

The on-disk format for the virtualization disk.

default: `raw`

6.3.32 default_virt_file_size

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

default: `5`

6.3.33 default_virt_ram

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

default: `512`

6.3.34 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:

- `xenpv`
- `xenfv`
- `qemu`
- `vmware`

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

default: `xenpv`

6.3.35 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`

6.3.36 enable_menu

Controls whether Cobbler will add each new profile entry to the default PXE boot menu. This can be overridden 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`

6.3.37 http_port

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

default: `80`

6.3.38 include

Include other configuration snippets with this regular expression. This is a list of folders.

default: `[ "/etc/cobbler/settings.d/*.settings" ]`

Note: Will be deprecated in future releases.

6.3.39 iso_template_dir

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

default: `/etc/cobbler/iso`

6.3.40 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`

6.3.41 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: `{}`

6.3.42 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: ''
ldap_search_passwd: ''
ldap_search_prefix: 'uid='
ldap_tls_cacertfile: ''
ldap_tls_keyfile: ''
ldap_tls_certfile: ''
```

6.3.43 bind_manage_ipmi

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

default: `False`

6.3.44 manage_dhcp

Set to `True` to enable Cobbler's DHCP management features. The choice of DHCP management engine is in `/etc/cobbler/modules.conf`.

default: `True`

6.3.45 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 `/etc/cobbler/modules.conf` whether this isc module is chosen for dhcp generation.

default: `False`

6.3.46 manage_dhcp_v6

Set to `true` to enable DHCP IPv4 address configuration generation. This currently only works with manager.isc DHCP module (isc dhcpd6 daemon). See `/etc/cobbler/modules.conf` whether this isc module is chosen for dhcp generation.

default: `False`

6.3.47 manage_dns

Set to `True` to enable Cobbler's DNS management features. The choice of DNS management engine is in `/etc/cobbler/modules.conf`.

default: `False`

6.3.48 manage_*_zones

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

defaults:

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

6.3.49 manage_genders

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

default: `False`

6.3.50 manage_rsync

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

default: `False`

6.3.51 manage_tftpd

Set to `True` to enable Cobbler's TFTP management features. The choice of TFTP management engine is in `/etc/cobbler/modules.conf`.

default: `True`

6.3.52 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
```

6.3.53 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`

6.3.54 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`

6.3.55 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`

6.3.56 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`

6.3.57 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`

6.3.58 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: `[]`

6.3.59 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`

6.3.60 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
```

6.3.61 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: `""`

6.3.62 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`

6.3.63 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`

6.3.64 puppet_server

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

default: `'puppet'`

6.3.65 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`

6.3.66 puppetca_path

Location of the puppet executable, used for revoking certificates.

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

6.3.67 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`

6.3.68 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`

6.3.69 redhat_management_permissive

If using `authn_spacewalk` in `modules.conf` 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`

6.3.70 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"`

6.3.71 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: `""`

6.3.72 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`

6.3.73 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`

6.3.74 replicate_repo_rsync_options

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

default: `"-avzH"`

6.3.75 replicate_rsync_options

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

default: `"-avzH"`

6.3.76 reposync_flags

Flags to use for yum's reposync. If your version of yum reposync does not support `-l`, you may need to remove that option.

default: `"-l -n -d"`

6.3.77 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"`

6.3.78 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
```

6.3.79 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`

6.3.80 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"
```

6.3.81 serializer_pretty_json

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

default: `False`

6.3.82 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`

6.3.83 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`

6.3.84 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`

6.3.85 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`

6.3.86 tftpboot_location

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

Default: `/srv/tftpboot`

6.3.87 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`

6.3.88 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`

6.3.89 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
```

6.3.90 windows_enabled

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

default: `False`

For more information see *Automatic Windows installation with Cobbler*.

6.3.91 windows_template_dir

Location of templates used for Windows.

default: `/etc/cobbler/windows`

For more information see *Automatic Windows installation with Cobbler*.

6.3.92 samba_distro_share

Samba share name for distros

default: `DISTRO`

For more information see *Automatic Windows installation with Cobbler*.

6.3.93 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`

6.3.94 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`

6.3.95 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`

6.3.96 yumdownloader_flags

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

default: `"--resolve"`

6.4 modules.conf

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

6.4.1 authentication

What users can log into the WebUI and Read-Write XML-RPC?

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
- authentication.testing – Username/password is always testing/testing (debug)
- (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 documention, yet.

default: `authn_configfile`

Hash algorithms:

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`

6.4.2 authorization

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 documention, yet.

default: `authz_allowall`

6.4.3 dns

Chooses the DNS management engine if `manage_dns` is enabled in `/etc/cobbler/settings.yaml`, 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 configuration management*.

default: `managers.bind`

6.4.4 dhcp

Chooses the DHCP management engine if `manage_dhcp` is enabled in `/etc/cobbler/settings.yaml`, 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`

6.4.5 tftpd

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`

CHAPTER 7

User Guide

7.1 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.

7.1.1 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.

7.1.2 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.

7.1.3 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.

7.1.4 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

7.1.5 Ansible

Official integration:

- https://github.com/cobbler/ansible

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

7.1.6 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

7.1.7 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

7.1.8 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.

```
node_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.

7.1.9 cfengine support

Documentation to be added

7.1.10 bcfg2 support

Documentation to be added

7.1.11 Chef

Documentation to be added.

There is some integration information on bootstrapping chef clients with cobbler in [this blog article](http://blog.milford.io/2012/03/getting-a-basic-cobbler-server-going-on-centos/)

7.1.12 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.

7.1.13 Attachments

- [puppet_node.py](/cobbler/attachment/wiki/UsingCobblerWithConfigManagementSystem/puppet_node.py) (2.5 kB) -Alternate External Nodes Script, added by shenson on 12/09/10 20:33:36.

7.2 Automatic Windows installation with Cobbler

One of the challenges for creating your own Windows network installation scenario with Cobbler is preparing the necessary files in a Linux environment. However, generating the necessary binaries can be greatly simplified by using the cobbler post trigger on the sync command. Below is an example of such a trigger, which prepares the necessary files for legacy BIOS mode boot. Boot to UEFI Mode with iPXE is simpler and can be implemented by replacing the first 2 steps and several others with creating an iPXE boot menu.

Trigger `sync_post_wingen.py`:

- some of the files are created from standard ones (`pxeboot.n12`, `bootmgr.exe`) by directly replacing one string with another directly in the binary
- in the process of changing the `bootmgr.exe` file, the checksum of the PE file will change and it needs to be recalculated. The trigger does this with `python-pefile`
- `python3-hivex` is used to modify Windows boot configuration data (BCD). For pxelinux distro boot_loader in BCD, paths to `winpe.wim` and `boot.sdi` are generated as `/images/<distro_name>`, and for iPXE with wimboot - `\Boot`.
- uses `wimlib-tools` to replace `startnet.cmd startup` script in WIM image

Windows answer files (`autounattended.xml`) are generated using Cobbler templates, with all of its conditional code generation capabilities, depending on the Windows version, architecture (32 or 64 bit), installation profile, etc.

startup scripts for WIM images (startnet.cmd) and a script that is launched after OS installation (`post_install.cmd`) are also generated from templates

Post-installation actions such as installing additional software, etc., are performed using the Automatic Installation Template (`win.ks`).

A logically automatic network installation of Windows 7 and newer can be represented as follows:

PXE + Legacy BIOS Boot

```
Original files: pxeboot.n12 → bootmgr.exe → BCD → winpe.wim → startnet.cmd →␣
↪autounattended.xml
Cobbler profile 1: pxeboot.001 → boot001.exe → 001 → wi001.wim → startnet.cmd →␣
↪autounatten001.xml → post_install.cmd profile_name
...
```

iPXE + UEFI Boot

```
Original files: ipxe-x86_64.efi → wimboot → bootmgr.exe → BCD → winpe.wim →␣
↪startnet.cmd → autounattended.xml
Cobbler profile 1: ipxe-x86_64.efi → wimboot → bootmgr.exe → 001 → wi001.wim →␣
↪startnet.cmd → autounatten001.xml → post_install.cmd profile_name
...
```

For older versions (Windows XP, 2003) + RIS:

```
Original files: pxeboot.n12 → setupldr.exe → winnt.sif → post_install.cmd␣
↪profile_name
Cobbler profile <xxx>: pxeboot.<xxx> → setup<xxx>.exe → wi<xxx>.sif → post_
↪install.cmd profile_name
```

7.2.1 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/WAIK. - 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/WAIK - 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/WAIK. - 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/WAIK. 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.

The following metadata does not specify boot file names and is an example of using metadata to generate files from Cobbler templates.

- clean_disk

  The presence of this key in the metadata (regardless of its value) leads to the preliminary deletion of all data and the disk partition table before installing the OS. Used in the answerfile.template and also in startnet.template in Windows XP and Windows 2003 Server installations using WinPE.

7.2.2 Preparing for an unattended network installation of Windows

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

```
windows_enabled: true
```

- import the Windows distributions to `/var/www/cobbler/distro_mirror`:

```
cobbler import --name=Win10_EN-x64 --path=/mnt
```

This command will determine the version and architecture of the Windows distribution, will extract the necessary boot files from the distribution and create a distro and profile named `Win10_EN-x64`.

- For customization winpe.win you need - ADK for Windows 10 / 8.1

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

or

- WAIK for Windows 7

```
Start -> All Programs -> Microsoft Windows AIK -> Deployment Tools Command Prompt
```

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

After executing the command, the WinPE image will be located in `.\winpe.wim` for WAIK and in `media\sources\boo`

- If necessary, add drivers to the image

Example:

```
dism /mount-wim /wimfile:media\sources\boot.wim /index:1 /mountdir:mount
dism /image:mount /add-driver /driver:D:\NetKVM\w10\amd64
dism /image:mount /add-driver /driver:D:\viostor\w10\amd64
dism /unmount-wim /mountdir:mount /commit
```

- Copy the resulting WiNPE image from Windows to the `boot` directory of the distro
- Share `` `/var/www/cobbler/distro_mirror` `` via Samba:

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

- You can use `tftpd.rules` to indicate the actual locations of the `bootmgr.exe` and `BCD` files generated by the trigger.

```
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
```

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

```
vi /etc/tftpd.rules
rg  \\                                  / # Convert backslashes to slashes
r   (wine.\.sif)                        /WinXp_EN-i386/\1
r   (xple.)                             /WinXp_EN-i386/\1

r   (wi2k.\.sif)                        /Win2k3-Server_EN-x64/\1
r   (w2k3.)                             /Win2k3-Server_EN-x64/\1

r   (boot7e.\.exe)                      /images/Win7_EN-x64/\1
r   (/Boot/)(7E.)                       /images/Win7_EN-x64/\2

r   (boot28.\.exe)                      /images/Win2k8-Server_EN-x64/\1
r   (/Boot/)(28.)                       /images/Win2k8-Server_EN-x64/\2

r   (boot9r.\.exe)                      /images/Win2019-Server_EN-x64/\1
r   (/Boot/)(9r.)                       /images/Win2019-Server_EN-x64/\2

r   (boot6e.\.exe)                      /images/Win2016-Server_EN-x64/\1
r   (/Boot/)(6e.)                       /images/Win2016-Server_EN-x64/\2

r   (boot2e.\.exe)                      /images/Win2012-Server_EN-x64/\1
r   (/Boot/)(2e.)                       /images/Win2012-Server_EN-x64/\2

r   (boot81.\.exe)                      /images/Win8_EN-x64/\1
r   (/Boot/)(B8.)                       /images/Win8_EN-x64/\2

r   (boot1e.\.exe)                      /images/Win10_EN-x64/\1
r   (/Boot/)(1E.)                       /images/Win10_EN-x64/\2

r   (.*)(/WinXp...-i386/)(.*)           /images\2\L\3
r   (.*)(/Win2k3-Server_EN-x64/)(.*)    /images\2\L\3

r   (.*)(bootxea.exe)                   /images/WinXp_EN-i386/\2
r   (.*)(XEa)                           /images/WinXp_EN-i386/\2

r   (.*)(boot3ea.exe)                   /images/Win2k3-Server_EN-x64/\2
r   (.*)(3Ea)                           /images/Win2k3-Server_EN-x64/\2
```

7.2.3 Final steps

- Restart the services:

```
systemctl daemon-reload
systemctl restart tftp
systemctl restart smb
systemctl restart nmb
```

- add additional distros for PXE boot:

```
cobbler distro add --name=Win10_EN-x64 \
--kernel=/var/www/cobbler/distro_mirror/Win10_EN-x64/boot/pxeboot.n12 \
--initrd=/var/www/cobbler/distro_mirror/Win10_EN-x64/boot/boot.sdi \
--arch=x86_64 --breed=windows --os-version=10
```

or for iPXE:

```
cobbler distro add --name=Win10_EN-x64 \
--kernel=/var/lib/tftpboot/wimboot \
--initrd=/var/www/cobbler/distro_mirror/Win10_EN-x64/boot/boot.sdi \
--remote-boot-kernel=http://@@http_server@@/cobbler/images/@@distro_name@@/wimboot␣
↪\
```

(continues on next page)

(continued from previous page)

```
--remote-boot-initrd=http://@@http_server@@/cobbler/images/@@distro_name@@/boot.
↪sdi \
--arch=x86_64 --breed=windows --os-version=10 \
--boot-loaders=ipxe
```

- and additional profiles for PXE boot:

```
cobbler profile add --name=Win10_EN-x64 --distro=Win10_EN-x64 --autoinstall=win.ks␣
↪\
--autoinstall-meta='kernel=win10a.0 bootmgr=boot1ea.exe bcd=1Ea winpe=winpe.wim␣
↪answerfile=autounattended.xml'

cobbler profile add --name=Win10-profile1 --parent=Win10_EN-x64 \
--autoinstall-meta='kernel=win10b.0 bootmgr=boot1eb.exe bcd=1Eb winpe=winp1.wim␣
↪answerfile=autounattende1.xml'

cobbler profile add --name=Win10-profile2 --parent=Win10_EN-x64 \
--autoinstall-meta='kernel=win10c.0 bootmgr=boot1ec.exe bcd=1Ec winpe=winp2.wim␣
↪answerfile=autounattende2.xml'
```

The boot menu will look like this:

```
LABEL Win10_EN-x64
        MENU LABEL Win10_EN-x64
        kernel /images/Win10_EN-x64/win10a.0
LABEL Win10_EN-x64-profile1
        MENU LABEL Win10_EN-x64-profile1
        kernel /images/Win10_EN-x64/win10b.0
LABEL Win10_EN-x64-profile1
        MENU LABEL Win10_EN-x64-profile2
        kernel /images/Win10_EN-x64/win10c.0
```

or for iPXE:

```
cobbler profile add --name=Win10_EN-x64 --distro=Win10_EN-x64 --autoinstall=win.ks␣
↪\
--autoinstall-meta='bootmgr=boot1ea.exe bcd=1Ea winpe=winpe.wim␣
↪answerfile=autounattended.xml' \
--boot-loaders=ipxe

cobbler profile add --name=Win10-profile1 --parent=Win10_EN-x64 \
--autoinstall-meta='bootmgr=boot1eb.exe bcd=1Eb winpe=winp1.wim␣
↪answerfile=autounattende1.xml' \
--boot-loaders=ipxe

cobbler profile add --name=Win10-profile2 --parent=Win10_EN-x64 \
--autoinstall-meta='bootmgr=boot1ec.exe bcd=1Ec winpe=winp2.wim␣
↪answerfile=autounattende2.xml' \
--boot-loaders=ipxe
```

The boot menu will look like this:

```
:Win10_EN-x64
kernel http://<http_server>/cobbler/images/Win10_EN-x64/wimboot
initrd --name boot.sdi http://<http_server>/cobbler/images/Win10_EN-x64/boot.sdi␣
↪boot.sdi
initrd --name bootmgr.exe http://<http_server>/cobbler/images/Win10_EN-x64/boot1ea.
↪exe bootmgr.exe
initrd --name bcd http://<http_server>/cobbler/images/Win10_EN-x64/1Ea bcd
initrd --name winpe.wim http://<http_server>/cobbler/images/Win10_EN-x64/winpe.wim␣
↪winpe.wim
```

(continues on next page)

(continued from previous page)

```
boot

:Win10_EN-x64-profile1
kernel http://<http_server>/cobbler/images/Win10_EN-x64/wimboot
initrd --name boot.sdi http://<http_server>/cobbler/images/Win10_EN-x64/boot.sdi␣
↪boot.sdi
initrd --name bootmgr.exe http://<http_server>/cobbler/images/Win10_EN-x64/boot1eb.
↪exe bootmgr.exe
initrd --name bcd http://<http_server>/cobbler/images/Win10_EN-x64/1Eb bcd
initrd --name winpe.wim http://<http_server>/cobbler/images/Win10_EN-x64/winp1.wim␣
↪winpe.wim
boot

:Win10_EN-x64-profile2
kernel http://<http_server>/cobbler/images/Win10_EN-x64/wimboot
initrd --name boot.sdi http://<http_server>/cobbler/images/Win10_EN-x64/boot.sdi␣
↪boot.sdi
initrd --name bootmgr.exe http://<http_server>/cobbler/images/Win10_EN-x64/boot1ec.
↪exe bootmgr.exe
initrd --name bcd http://<http_server>/cobbler/images/Win10_EN-x64/1Ec bcd
initrd --name winpe.wim http://<http_server>/cobbler/images/Win10_EN-x64/winp2.wim␣
↪winpe.wim
boot
```

- cobbler sync
  - kernel from `autoinstall-meta` of profile or from `kernel` of distro property will be copied to `/var/lib/tftpboot/<distro_name>`
  - if the kernel is `pxeboot.n12`, then the `bootmgr.exe` substring is replaced in the copied copy of kernel with the value passed via `bootmgr` of the `autoinstall-meta` profile propery
- Install Windows

7.2.4 Legacy Windows XP and Windows 2003 Server

- WinPE 3.0 and winboot 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 \
--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'
```

(continues on next page)

(continued from previous page)

```
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 `winboot` 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
```

(continues on next page)

(continued from previous page)

```
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
```

7.2.5 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 \
--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'
```

(continues on next page)

(continued from previous page)

```
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'
```

7.3 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.

7.3.1 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])
```

(continues on next page)

(continued from previous page)

```
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

7.3.2 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 modules.conf

To create a module, add a Python file in `/usr/lib/python$version/site-packages/cobbler/modules`. Then, in the appropriate part of `/etc/cobbler/modules.conf`, 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:

```
[serializers]
settings = serializer.file
```

The format of `/etc/cobbler/modules.conf` is that of Python's ConfigParser module.

A setup file consists of sections, lead by a "[section]" header, and followed by "name: value" entries with continuations and such in the style of RFC 822.

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.

7.3.3 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.

7.4 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

7.4.1 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.

7.4.2 Configure Cobbler

Configure Cobbler to have **caching disabled**.

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

7.4.3 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
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.

7.4.4 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
├── 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"
```

(continues on next page)

(continued from previous page)

```
}

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.
```

(continues on next page)

(continued from previous page)

```
../../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.

7.5 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

7.6 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.

7.7 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.

7.8 Power Management

Cobbler contains a power management feature that allows the user to associate system records in Cobbler with the power management configuration attached to them. This can ease installation by making it easy to reassign systems to new operating systems and then reboot those systems.

7.9 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
```

7.10 Repository Management

7.10.1 REPO MANAGEMENT

This has already been covered a good bit in the command reference section.

Yum repository management is an optional feature, and is not required to provision through Cobbler. However, if Cobbler is configured to mirror certain repositories, it can then be used to associate profiles with those repositories. Systems installed under those profiles will then 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 even within Anaconda. This can be useful if (A) you have a large install base, (B) you want fast installation and upgrades for your systems, or (C) 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 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 yum 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=False`).

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 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.

7.10.2 Importing trees

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`.

Example 1: `cobbler import --path=rsync://mirrorserver.example.com/path/ --name=fedora --arch=x86`

Example 2: `cobbler import --path=root@192.168.1.10:/stuff --name=bar`

Example 3: `cobbler import --path=/mnt/dvd --name=baz --arch=x86_64`

Example 4: `cobbler import --path=/path/to/stuff --name=glorp`

Example 5: `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`. –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, you can feed the option `--autoinstall` to import, to bypass the built-in automatic installation file auto-detection.

7.10.3 Repository mirroring workflow

The following example shows how to set up a repo mirror for all enabled Cobbler host repositories and two additional repositories, and 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"
```

7.10.4 Import Workflow

Import is a very useful command that makes starting out with Cobbler very quick and easy.

This example shows how to create a provisioning infrastructure from a distribution mirror or DVD ISO. Then a default PXE configuration is created, 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, though to make sure things are named correctly, it's always a good idea to specify `--arch`. For instance, if you import a distribution named "fedora8" from an ISO, and it's 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, as Cobbler will set the names for things based on the paths it finds.

```
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
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
```

7.11 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.

7.12 Autoinstallation

7.12.1 Automatic installation templating

The `--autoinstall_meta` options above 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

7.12.2 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.

7.12.3 Kickstart validation

To check for potential errors in kickstarts, prior to installation, use `cobbler validateks`. This function will check all profile and system kickstarts for detectable errors. Since pykickstart is not future-Anaconda-version aware, there may be some false positives. It should be noted that `cobbler validateks` runs on the rendered kickstart output, not kickstart templates themselves.

7.13 Network Topics

7.13.1 PXE Menus

Cobbler will automatically generate PXE menus for all profiles it has defined. 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`.

7.13.2 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
```

7.13.3 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
```

7.13.4 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.

7.14 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.

7.14.1 DHCP Management

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

Choose either `management = isc_and_bind` in `/etc/cobbler/dhcp.template` or `management = "dnsmasq"` in `/etc/cobbler/modules.conf`. Then set `manage_dhcp=1` in `/etc/cobbler/settings.yaml`.

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.

Depending on your choice, Cobbler will use `/etc/cobbler/dhcpd.template` or `/etc/cobbler/dnsmasq.template` as a starting point. This file must be user edited for the user's particular networking environment. Read the file and understand how the particular app (ISC dhcpd or dnsmasq) work before proceeding.

If you already have DHCP configuration data that you would like to preserve (say DHCP 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, the DHCP configuration file 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.

If omapi_enabled is set to 1 in `/etc/cobbler/settings.yaml`, the need to sync when adding new system records can be eliminated. However, the OMAPI feature is experimental and is not recommended for most users.

7.14.2 DNS configuration management

Cobbler can optionally manage DNS configuration using BIND and dnsmasq.

Choose either `management = isc_and_bind` or `management = dnsmasq` in `/etc/cobbler/modules.conf` and then enable `manage_dns` in `/etc/cobbler/settings.yaml`.

This feature is off by default. If using BIND, you must define the zones to be managed with the options `manage_forward_zones` and `manage_reverse_zones`. (See the Wiki for more information on this).

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.

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

All managed files (whether zone files and `named.conf` for BIND, or `dnsmasq.conf` for dnsmasq) 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.

7.15 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

7.16 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 8

Developer Guide

Our project lives on GitHub! Please visit our wiki there to get familiar with developer specific instructions: GitHub Cobbler Wiki

CHAPTER 9

cobbler package

9.1 Subpackages

9.1.1 cobbler.actions package

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**(*collection_mgr*)

Bases: object

modacl(*isadd: bool*, *isuser: bool*, *who*)

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=None*, *addgroup=None*, *removeuser=None*, *removegroup=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.buildiso module

Builds bootable CD images that have PXE-equivalent behavior for all Cobbler distros/profiles/systems currently in memory.

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.actions.buildiso.**BuildIso**(*collection_mgr*, *verbose: bool = False*)
Bases: object

Handles conversion of internal state to the isolinux tree layout

add_remaining_kopts(*koptdict: dict*) → str
Add remaining kernel_options to append_line

Parameters **koptdict** – The kernel options which are not present in append_line.

Returns A single line with all kernel options

copy_boot_files(*distro*, *destdir*, *prefix: Optional[str] = None*)
Copy kernel/initrd to destdir with (optional) newfile prefix

Parameters

- **distro** – Distro object to return the boot files for.
- **destdir** – The destionation direcotry.
- **prefix** – The file prefix.

filter_systems_or_profiles(*selected_items*, *list_type: str*) → list
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

- **selected_items** – The filter to match certain objects with. The filter will be applied to the object name.
- **list_type** – Must be “profile” or “system”.

Returns A list of valid profiles OR systems.

generate_netboot_iso(*imagesdir*, *isolinuxdir*, *profiles=None*, *systems=None*, *exclude_dns: Optional[bool] = None*)
Create bootable CD image to be used for network installations

Parameters

- **imagesdir** – Currently unused parameter.
- **isolinuxdir** – The parent directory where the isolinux.cfg is located.
- **profiles** – The filter to generate a netboot iso for. You may specify multiple profiles on the CLI space separated.
- **systems** – The filter to generate a netboot iso for. You may specify multiple systems on the CLI space separated.
- **exclude_dns** – If this is True then the dns server is skipped. None or False will set it.

generate_standalone_iso(*imagesdir*, *isolinuxdir*, *distname*, *filesource*, *airgapped: bool*, *profiles*)
Create bootable CD image to be used for handsoff CD installtions

Parameters

- **imagesdir** – Unused Parameter.
- **isolinuxdir** – The parent directory where the file isolinux.cfg is located at.
- **distname** – The name of the Cobbler distribution.
- **filesource** – Not clear what this exactly does
- **airgapped** – Whether the repositories have to be locally available or the internet is reachable.
- **profiles** – The list of profiles to include.

make_shorter(*distname: str*) → str
Return A short distro identifier.

Parameters **distname** – The distro name to return a identifier for.

Returns A short distro identifier

run(*iso=None*, *buildisodir=None*, *profiles=None*, *systems=None*, *distro=None*, *standalone: Optional[bool] = None*, *airgapped: Optional[bool] = None*, *source=None*, *exclude_dns: Optional[bool] = None*, *xorrisofs_opts: Optional[str] = None*)
A

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** –
- **systems** – Don't use that when building standalone isos.
- **distro** – (For standalone only)
- **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.

cobbler.actions.check module

class cobbler.actions.check.**CobblerCheck**(*collection_mgr*)
Bases: object

Validates whether the system is reasonably well configured for serving up content. This is the code behind 'cobbler check'.

check_bind_bin(*status*)
Check if bind is installed.

Parameters **status** – The status list with possible problems.

check_bootloaders(*status*)
Check if network bootloaders are installed

Parameters **status** – The status list with possible problems.

check_ctftpd_dir(*status*)
Check if cobbler.conf's tftpboot directory exists.

Parameters **status** – The status list with possible problems.

check_debmirror(*status*)
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.

check_dhcpd_bin(*status*)
Check if dhcpd is installed.

Parameters **status** – The status list with possible problems.

check_dhcpd_conf(*status*)
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.

check_dnsmasq_bin(*status*)
Check if dnsmasq is installed.

Parameters **status** – The status list with possible problems.

check_for_cman(*status*)
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*)
Check if the default password of Cobbler was changed.

Parameters **status** – The status list with possible problems.

check_for_ksvalidator(*status*)
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*)
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*)
Check if there are unsynchronized repositories which need an update.

Parameters **status** – The status list with possible problems.

check_for_wget_curl(*status*)
Check to make sure wget or curl is installed

Parameters **status** – The status list with possible problems.

check_iptables(*status*)
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*)
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*)
Check that rsync is enabled to autostart.

Parameters **status** – The status list with possible problems.

check_selinux(*status*)
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*, *which*, *notes=''*)
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*)
Check if cobbler.conf's tftpboot directory exists

Parameters **status** – The status list with possible problems.

check_yum(*status*)
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()
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=None*)
Bases: object

run()
Simply hardlinks directories that are Cobbler managed.

cobbler.actions.log module

Copyright 2009, Red Hat, Inc and Others Bill Peck <bpeck@redhat.com>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.actions.log.**LogTool**(*collection_mgr*, *system*, *api*)
Bases: object

Helpers for dealing with System logs, anamon, etc..

clear()
Clears the system logs

cobbler.actions.replicate module

Replicate from a Cobbler master.

Copyright 2007-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail> Scott Henson <shenson@redhat.com>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.actions.replicate.**Replicate**(*collection_mgr*)
Bases: object

This class contains the magic to replicate a Cobbler instance to another Cobbler instance.

add_objects_not_on_local(*obj_type*)
Add objects locally which are not present on the slave but on the master.

Parameters **obj_type** –

generate_include_map()
Not known what this exactly does.

link_distros()
Link a distro from its location into the web directory to make it available for usage.

remove_objects_not_on_master(*obj_type*)
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*)
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()
Replicate the local and remote data to each another.

rsync_it(*from_path*, *to_path*, *type: Optional[str] = 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.
- **type** – If set to "repo" this will take the repo rsync options instead of the global ones.

run(*cobbler_master=None*, *port: str = '80'*, *distro_patterns=None*, *profile_patterns=None*, *system_patterns=None*, *repo_patterns=None*, *image_patterns=None*, *mgmtclass_patterns=None*, *package_patterns=None*, *file_patterns=None*, *prune: bool = False*, *omit_data=False*, *sync_all: bool = False*, *use_ssl: bool = False*)
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.
- **mgmtclass_patterns** – The pattern of management classes to sync.
- **package_patterns** – The pattern of packages to sync.
- **file_patterns** – The pattern of files 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

Copyright 2007-2009, Red Hat, Inc and Others Anderson Silva <ansilva@redhat.com> Michael DeHaan <michael.dehaan AT gmail>

This software may be freely redistributed under the terms of the GNU general public license.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

class cobbler.actions.report.**Report**(*collection_mgr*)
Bases: object

fielder(*structure*, *fields_list*)
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*, *order*, *report_type*, *noheaders: bool*)
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*, *order*, *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*, *order*, *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.

reporting_list_names2(*collection*, *name*)
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*, *order*, *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*, *report_name*, *report_type*, *report_noheaders: bool*) → str
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.

reporting_print_sorted(*collection*)
Prints all objects in a collection sorted by name

Parameters **collection** – The collection to print.

reporting_print_x_fields(*collection*, *report_name*, *report_type*, *report_fields*, *report_noheaders: bool*)
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*, *order*, *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=None*, *report_name=None*, *report_type: Optional[str] = None*, *report_fields=None*, *report_noheaders: Optional[bool] = 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.

Copyright 2006-2007, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.actions.reposync.**RepoSync**(*collection_mgr*, *tries: int = 1*, *nofail: bool = False*)

Bases: object

Handles conversion of internal state to the tftpboot tree layout.

apt_sync(*repo*)

Handle copying of http:// and ftp:// debian repos.

Parameters **repo** – The apt repository to sync.

create_local_file(*dest_path: str*, *repo*, *output: bool = True*)

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*, *dirname: str*, *fnames*)

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*) → Union[str, 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*)

reposync_cmd() → str
Determine reposync command

Returns The path to the reposync command. If dnf exists it is used instead of reposync.

rhn_sync(*repo*)
Handle mirroring of RHN repos.

Parameters **repo** – The repo object to synchronize.

rsync_sync(*repo*)
Handle copying of rsync:// and rsync-over-ssh repos.

Parameters **repo** – The repo to sync via rsync.

run(*name=None*, *verbose: bool = True*)
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*)
Conditionally sync a repo, based on type.

Parameters **repo** – The repo to sync.

update_permissions(*repo_path*)
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*)
Handle mirroring of directories using wget

Parameters **repo** – The repo object to sync via wget.

yum_sync(*repo*)
Handle copying of http:// and ftp:// yum repos.

Parameters **repo** – The yum reporitory to sync.

cobbler.actions.reposync.**repo_walker**(*top*, *func*, *arg*)
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**(*collection_mgr*, *mode*)

Bases: object

catalog(*profile_or_system: str*, *name: str*, *ip*, *start_or_stop: str*, *ts: float*)

Add a system to `cobbler status`.

Parameters

- **profile_or_system** – This can be `system` or `profile`.
- **name** – The name of the object.
- **ip** – The ip of the system to watch.
- **start_or_stop** – This parameter may be `start` or `stop`
- **ts** – Don't know what this does.

get_printable_results()

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()

Look through all systems which were collected and update the status.

Returns Return `ip_data` of the object.

run()

Calculate and print a automatic installation status report.

scan_logfiles()

Scan the install log-files - starting with the oldest file.

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**(*collection_mgr*, *verbose: bool = True*, *dhcp=None*, *dns=None*, *tftpd=None*)

Bases: object

Handles conversion of internal state to the tftpboot tree layout

add_single_distro(*name*)
Sync adding a single distro.

Parameters **name** – The name of the distribution.

add_single_image(*name*)
Sync adding a single image.

Parameters **name** – The name of the image.

add_single_profile(*name: str*, *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(*name: str*)
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(*name*)
Sync removing a single distro.

Parameters **name** – The name of the distribution.

remove_single_image(*name*)
Sync removing a single image.

Parameters **name** – The name of the image.

remove_single_menu(*rebuild_menu: bool = True*)
Sync removing a single menu. :param rebuild_menu: Whether to rebuild the grub/... menu or not.

remove_single_profile(*name: str*, *rebuild_menu: bool = True*)
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(*name: str*)
Sync removing a single system.

Parameters **name** – The name of the system.

rsync_gen()
Generate rsync modules of all repositories and distributions

Raises **OSError** –

run()
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*)
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.

9.1.2 cobbler.cobbler_collections package

Submodules

cobbler.cobbler_collections.collection module

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.cobbler_collections.collection.**Collection**(*collection_mgr*)
Bases: object

Base class for any serializable list of things.

SEARCH_REKEY = {'boot_loader': 'boot_loaders', 'dhcp-tag': 'dhcp_tag', 'enable_gp

add(*ref*, *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*, *check_for_duplicate_netinfo: bool = False*)
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 renamed.
- **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.
- **check_for_duplicate_netinfo** – This checks for duplicate network information. This only has an effect on systems.

Raises

- **TypError** – Raised in case `ref` is None.
- **ValueError** – Raised in case the name of `ref` is empty.

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)

copy(*ref*, *newname*)
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.

factory_produce(*api*, *seed_data*)
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=False*, ***kargs*) → Union[List[cobbler.items.item.Item], cobbler.items.item.Item, None]
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*)
Create all collection object items from `_list`.

Parameters **_list** – The list with all item dictionaries.

get(*name*)
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.

remove(*name: str*, *with_delete: bool = True*, *with_sync: bool = True*, *with_triggers: bool = True*, *recursive: bool = False*)
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: cobbler.items.item.Item*, *newname*, *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
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

class cobbler.cobbler_collections.distros.**Distros**(*collection_mgr*)
Bases: *cobbler.cobbler_collections.collection.Collection*

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*, *item_dict*)
Return a Distro forged from item_dict

remove(*name*, *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 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.files module

Copyright 2010, Kelsey Hightower Kelsey Hightower <kelsey.hightower@gmail.com>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.cobbler_collections.files.**Files**(*collection_mgr*)
Bases: *cobbler.cobbler_collections.collection.Collection*

Files provide a container for file resources.

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*, *item_dict*)
Return a File forged from item_dict

remove(*name*, *with_delete: bool = True*, *with_sync: bool = True*, *with_triggers: bool = True*, *recursive: bool = False*)
Remove element named 'name' from the collection

:raises CX

cobbler.cobbler_collections.images module

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This software may be freely redistributed under the terms of the GNU general public license.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

class cobbler.cobbler_collections.images.**Images**(*collection_mgr*)
Bases: *cobbler.cobbler_collections.collection.Collection*

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*, *item_dict*)
Return a Distro forged from item_dict

remove(*name*, *with_delete: bool = True*, *with_sync: bool = True*, *with_triggers: bool = True*, *recursive: bool = True*)
Remove element named 'name' from the collection

:raises CX

cobbler.cobbler_collections.manager module

Repository of the Cobbler object model

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.cobbler_collections.manager.**CollectionManager**(*api*)
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()
Load all cobbler_collections from disk

Raises ***CX*** – if there is an error in deserialization

distros()
Return the definitive copy of the Distros collection

files() → cobbler.cobbler_collections.files.Files
Return the definitive copy of the Files collection

get_items(*collection_type: str*) → Union[cobbler.cobbler_collections.distros.Distros, cobbler.cobbler_collections.profiles.Profiles, cobbler.cobbler_collections.systems.Systems, cobbler.cobbler_collections.repos.Repos, cobbler.cobbler_collections.images.Images, cobbler.cobbler_collections.mgmtclasses.Mgmtclasses, cobbler.cobbler_collections.packages.Packages, cobbler.cobbler_collections.files.Files, cobbler.cobbler_collections.menus.Menus]
Get a full collection of a single type.

Valid Values vor `collection_type` are: "distro", "profile", "repo", "image", "mgmtclass", "package", "file" 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() → cobbler.cobbler_collections.images.Images
Return the definitive copy of the Images collection

menus()
Return the definitive copy of the Menus collection

mgmtclasses() → cobbler.cobbler_collections.mgmtclasses.Mgmtclasses
Return the definitive copy of the Mgmtclasses collection

packages() → cobbler.cobbler_collections.packages.Packages
Return the definitive copy of the Packages collection

profiles() → cobbler.cobbler_collections.profiles.Profiles
Return the definitive copy of the Profiles collection

repos() → cobbler.cobbler_collections.repos.Repos
Return the definitive copy of the Repos collection

serialize()
Save all cobbler_collections to disk

serialize_delete(*collection*, *item*)
Delete a collection item from disk

Parameters

- **collection** – collection
- **item** – collection item

serialize_item(*collection*, *item*)
Save a collection item to disk

Parameters

- **collection** – Collection
- **item** – collection item

settings()
Return the definitive copy of the application settings

systems() → cobbler.cobbler_collections.systems.Systems
Return the definitive copy of the Systems collection

cobbler.cobbler_collections.menus module

class cobbler.cobbler_collections.menus.**Menus**(*collection_mgr*)
Bases: *cobbler.cobbler_collections.collection.Collection*

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*, *item_dict*)
Return a Menu forged from item_dict

Parameters

- **api** – The cobblerd API.
- **item_dict** – The seed data.

Returns A new menu instance.

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

Parameters

- **name** – The name of the menu
- **with_delete** – TODO
- **with_sync** – TODO
- **with_triggers** – TODO
- **recursive** – TODO

Raises ***CX*** – Raised in case you want to delete a none existing menu.

cobbler.cobbler_collections.mgmtclasses module

class cobbler.cobbler_collections.mgmtclasses.**Mgmtclasses**(*collection_mgr*)
Bases: *cobbler.cobbler_collections.collection.Collection*

A mgmtclass provides a container for management resources.

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*, *item_dict*)
Return a mgmtclass forged from item_dict

Parameters

- **api** – TODO
- **item_dict** – TODO

Returns TODO

remove(*name*, *with_delete: bool = True*, *with_sync: bool = True*, *with_triggers: bool = True*, *recursive: bool = False*)
Remove element named 'name' from the collection

:raises CX

cobbler.cobbler_collections.packages module

Copyright 2010, Kelsey Hightower Kelsey Hightower <kelsey.hightower@gmail.com>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.cobbler_collections.packages.**Packages**(*collection_mgr*)
Bases: *cobbler.cobbler_collections.collection.Collection*

A package provides a container for package resources.

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*, *item_dict*)
Return a Package forged from item_dict

remove(*name*, *with_delete: bool = True*, *with_sync: bool = True*, *with_triggers: bool = True*, *recursive: bool = False*)
Remove element named 'name' from the collection

:raises CX

cobbler.cobbler_collections.profiles module

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.cobbler_collections.profiles.**Profiles**(*collection_mgr*)
Bases: *cobbler.cobbler_collections.collection.Collection*

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*, *item_dict*)
Return a Distro forged from item_dict

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

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.cobbler_collections.repos.**Repos**(*collection_mgr*)
Bases: *cobbler.cobbler_collections.collection.Collection*

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*, *item_dict*)
Return a Distro forged from item_dict

remove(*name*, *with_delete: bool = True*, *with_sync: bool = True*, *with_triggers: bool = True*, *recursive: bool = False*)
Remove element named 'name' from the collection

:raises CX

cobbler.cobbler_collections.systems module

Copyright 2008-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.cobbler_collections.systems.**Systems**(*collection_mgr*)
Bases: *cobbler.cobbler_collections.collection.Collection*

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*, *item_dict*)
Return a Distro forged from item_dict

Parameters

- **api** – TODO
- **item_dict** – TODO

Returns TODO

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.

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.

9.1.3 cobbler.items package

Submodules

cobbler.items.distro module

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.items.distro.**Distro**(*api*, **args*, ***kwargs*)
Bases: *cobbler.items.item.Item*

A Cobbler distribution object

COLLECTION_TYPE = 'distro'

arch
Return the architecture of the distribution

Returns Return the current architecture.

boot_loaders
TODO

Returns The bootloaders.

breed
TODO

Returns

check_if_valid()
Check if a distro object is valid. If invalid an exception is raised.

children
TODO

Returns

from_dict(*dictionary: dict*)
Initializes the object with attributes from the dictionary.

Parameters **dictionary** – The dictionary with values.

initrd
TODO

Returns

kernel
TODO

Returns

make_clone()
Clone a distro object.

Returns The cloned object. Not persisted on the disk or in a database.

os_version
TODO

Returns

parent
Distros don't have parent objects.

redhat_management_key
Get the redhat management key. This is probably only needed if you have spacewalk, uyuni or SUSE Manager running.

Returns The key as a string.

remote_boot_initrd
TODO

Returns

remote_boot_kernel
TODO

Returns

remote_grub_initrd
TODO

Returns

remote_grub_kernel
TODO

Returns

source_repos
TODO

Returns

supported_boot_loaders
Some distributions, particularly on powerpc, can only be netbooted using specific bootloaders.

Returns The bootloaders which are available for being set.

tree_build_time
TODO

Returns

cobbler.items.file module

Copyright 2006-2009, MadHatter Kelsey Hightower <kelsey.hightower@gmail.com>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.items.file.**File**(*api*, **args*, ***kwargs*)
Bases: *cobbler.items.resource.Resource*

A Cobbler file object.

COLLECTION_TYPE = 'file'

TYPE_NAME = 'file'

check_if_valid()
Insure name, path, owner, group, and mode are set. Templates are only required for files, is_dir = False

:raises CX

from_dict(*dictionary: dict*)
Initializes the object with attributes from the dictionary.

Parameters **dictionary** – The dictionary with values.

is_dir
TODO

Returns

make_clone()
Clone this file object. Please manually adjust all values yourself to make the cloned object unique.

Returns The cloned instance of this object.

cobbler.items.image module

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.items.image.**Image**(*api*, **args*, ***kwargs*)
Bases: *cobbler.items.item.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'

arch
TODO

Returns

autoinstall
TODO

Returns

boot_loaders

Returns The bootloaders.

breed
TODO

Returns

file
TODO

Returns

from_dict(*dictionary: dict*)
Initializes the object with attributes from the dictionary.

Parameters **dictionary** – The dictionary with values.

image_type
TODO

Returns

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.

menu
TODO

Returns

network_count
TODO

Returns

os_version
TODO

Returns

supported_boot_loaders

Returns The bootloaders which are available for being set.

virt_auto_boot
TODO

Returns

virt_bridge
TODO

Returns

virt_cpus
TODO

Returns

virt_disk_driver
TODO

Returns

virt_file_size
TODO

Returns

virt_path
TODO

Returns

virt_ram
TODO

Returns

virt_type
TODO

Returns

cobbler.items.item module

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This software may be freely redistributed under the terms of the GNU general public license.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

class cobbler.items.item.**Item**(*api*, *is_subobject: bool = False*)
Bases: object

An Item is a serializable thing that can appear in a Collection

COLLECTION_TYPE = 'generic'

TYPE_NAME = 'generic'

autoinstall_meta
Automatic Installation Template Metadata

Returns The metadata or an empty dict.

boot_files
Files copied into tftpboot beyond the kernel/initrd

Returns

check_if_valid()
Raise exceptions if the object state is inconsistent

:raises CX

children
TODO

Returns An empty list.

comment
For every object you are able to set a unique comment which will be persisted on the object.

Returns The comment or an emtpy string.

ctime
TODO

Returns

depth
TODO

Returns

descendants
Get objects that depend on this object, i.e. those that would be affected by a cascading delete, etc.

Returns This is a list of all descendants. May be empty if none exist.

deserialize(*item_dict: dict*)
This is currently a proxy for `from_dict()`.

Parameters **item_dict** – The dictionary with the data to deserialize.

dump_vars(*formatted_output: bool = True*)
Dump all variables.

Parameters **formatted_output** – Whether to format the output or not.

Returns The raw or formatted data.

fetchable_files
A comma seperated list of `virt_name=path_to_template` that should be fetchable via tftp or a webserver

Returns

find_match(*kwargs*, *no_errors=False*)
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*, *key*, *value*, *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*)
Modify this object to take on values in `dictionary`.

Parameters **dictionary** – This should contain all values which should be updated.

get_children(*sort_list: bool = False*) → List[str]
TODO

Returns

get_conceptual_parent()
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.

is_subobject
TODO

Returns True in case the object is a subobject, False otherwise.

kernel_options
TODO

Returns

kernel_options_post
TODO

Returns

make_clone()
Must be defined in any subclass

mgmt_classes
For external config management

Returns An empty list or the list of mgmt_classes.

mgmt_parameters
Parameters which will be handed to your management application (Must be a valid YAML dictionary)

Returns The mgmt_parameters or an empty dict.

mtime
Represents the last modification time of the object via the API.

Returns The float which can be fed into a Python time object.

name
The objects name.

Returns The name of the object

owners
TODO

Returns

parent
TODO

Returns

serialize() → dict
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 = None*)
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.

template_files
File mappings for built-in configuration management

Returns

to_dict() → dict
This converts everything in this object to a dictionary.

Returns A dictionary with all values present in this object.

uid
The uid is the internal unique representation of a Cobbler object. It should never be used twice, even after an object was deleted.

Returns

cobbler.items.menu module

class cobbler.items.menu.**Menu**(*api*, **args*, ***kwargs*)
Bases: *cobbler.items.item.Item*

A Cobbler menu object.

COLLECTION_TYPE = 'menu'

check_if_valid()
Check if the profile is valid. This checks for an existing name and a distro as a conceptual parent.

children
TODO

Returns

display_name
TODO

Returns

from_dict(*dictionary: dict*)
Initializes the object with attributes from the dictionary.

Parameters **dictionary** – The dictionary with values.

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.

parent
Parent Menu of a menu instance.

Returns The menu object or None

cobbler.items.mgmtclass module

Copyright 2010, Kelsey Hightower Kelsey Hightower <kelsey.hightower@gmail.com>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.items.mgmtclass.**Mgmtclass**(*api*, **args*, ***kwargs*)

Bases: *cobbler.items.item.Item*

TODO Explain purpose of the class

COLLECTION_TYPE = 'mgmtclass'

TYPE_NAME = 'mgmtclass'

check_if_valid()

Check if this object is in a valid state. This currently checks only if the name is present.

Raises ***CX*** – Raised in case no name is given.

class_name

TODO

Returns

files

TODO

Returns

from_dict(*dictionary: dict*)

Initializes the object with attributes from the dictionary.

Parameters **dictionary** – The dictionary with values.

is_definition

TODO

Returns

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.

packages

TODO

Returns

params

TODO

Returns

cobbler.items.package module

Copyright 2006-2009, MadHatter Kelsey Hightower <kelsey.hightower@gmail.com>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.items.package.**Package**(*api*, **args*, ***kwargs*)

Bases: *cobbler.items.resource.Resource*

TODO Explanation of this class TODO Type Checks in method bodys

COLLECTION_TYPE = 'package'

TYPE_NAME = 'package'

check_if_valid()

Checks if the object is in a valid state. This only checks currently if the name is present.

:raises CX

from_dict(*dictionary: dict*)

Initializes the object with attributes from the dictionary.

Parameters **dictionary** – The dictionary with values.

installer

TODO

Returns

make_clone()

Clone this package object. Please manually adjust all value yourself to make the cloned object unique.

Returns The cloned instance of this object.

version

TODO

Returns

cobbler.items.profile module

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.items.profile.**Profile**(*api*, **args*, ***kwargs*)

Bases: *cobbler.items.item.Item*

A Cobbler profile object.

COLLECTION_TYPE = 'profile'

TYPE_NAME = 'profile'

arch
TODO

Returns

autoinstall
TODO

Returns

boot_loaders

Returns The bootloaders.

check_if_valid()
Check if the profile is valid. This checks for an existing name and a distro as a conceptual parent.

:raises CX

children
TODO

Returns

dhcp_tag
TODO

Returns

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.

enable_ipxe
TODO

Returns

enable_menu
TODO

Returns

filename
TODO

Returns

from_dict(*dictionary: dict*)
Initializes the object with attributes from the dictionary.

Parameters **dictionary** – The dictionary with values.

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.

menu
TODO

Returns

name_servers
TODO

Returns

name_servers_search
TODO

Returns

next_server_v4
TODO

Returns

next_server_v6
TODO

Returns

parent
Return object next highest up the tree. If this property is not set it falls back to the value of the `distro`. In case neither distro nor parent is set, it returns None (which would make the profile invalid).

Returns

proxy
TODO

Returns

redhat_management_key
Getter of the redhat management key of the profile or it's parent.

Returns Returns the redhat_management_key of the profile.

repos
TODO

Returns

server
TODO

Returns

virt_auto_boot
TODO

Returns

virt_bridge
TODO

Returns

virt_cpus
TODO

Returns

virt_disk_driver
TODO

Returns

virt_file_size
TODO

Returns

virt_path
TODO

Returns

virt_ram
TODO

Returns

virt_type
TODO

Returns

cobbler.items.repo module

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.items.repo.**Repo**(*api*, **args*, ***kwargs*)
Bases: *cobbler.items.item.Item*

A Cobbler repo object.

COLLECTION_TYPE = 'repo'

TYPE_NAME = 'repo'

apt_components
TODO

Returns

apt_dists
TODO

Returns

arch
TODO

Returns

breed
TODO

Returns

check_if_valid()
Checks if the object is valid. Currently checks for name and mirror to be present.

:raises CX

createrepo_flags
TODO

Returns

environment
TODO

Returns

from_dict(*dictionary: dict*)
Initializes the object with attributes from the dictionary.

Parameters **dictionary** – The dictionary with values.

keep_updated
TODO

Returns

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.

mirror
TODO

Returns

mirror_locally
TODO

Returns

mirror_type
TODO

Returns

os_version
TODO

Returns

priority
TODO

Returns

proxy
TODO

Returns

rpm_list
TODO

Returns

rsyncopts
TODO

Returns

yumopts
TODO

Returns

cobbler.items.resource module

An Resource is a serializable thing that can appear in a Collection

Copyright 2006-2009, Red Hat, Inc and Others Kelsey Hightower <khightower@gmail.com>

This software may be freely redistributed under the terms of the GNU general public license.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

class cobbler.items.resource.**Resource**(*api*, **args*, ***kwargs*)
Bases: *cobbler.items.item.Item*

Base Class for management resources.

TODO: Type declarations in the method signatures and type checks in the bodys.

action
TODO

Returns

from_dict(*dictionary: dict*)
Initializes the object with attributes from the dictionary.

Parameters **dictionary** – The dictionary with values.

group
TODO

Returns

make_clone()
Clone this file object. Please manually adjust all values yourself to make the cloned object unique.

Returns The cloned instance of this object.

mode
TODO

Returns

owner
TODO

Returns

path
TODO

Returns

template
TODO

Returns

cobbler.items.system module

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.items.system.**NetworkInterface**(*api*)
Bases: object

A subobject of a Cobbler System which represents the network interfaces

bonding_opts
TODO

Returns

bridge_opts
TODO

Returns

cnames
TODO

Returns

connected_mode
TODO

Returns

deserialize(*interface_dict: dict*)
TODO

Returns

dhcp_tag
TODO

Returns

dns_name
TODO

Returns

from_dict(*dictionary: dict*)
TODO

Parameters **dictionary** –

if_gateway
TODO

Returns

interface_master
TODO

Returns

interface_type
TODO

Returns

ip_address
TODO

Returns

ipv6_address
TODO

Returns

ipv6_default_gateway
TODO

Returns

ipv6_mtu
TODO

Returns

ipv6_prefix
TODO

Returns

ipv6_secondaries
TODO

Returns

ipv6_static_routes
TODO

Returns

mac_address
TODO

Returns

management
TODO

Returns

modify_interface(*_dict: dict*)
TODO

mtu
TODO

Returns

netmask
TODO

Returns

serialize()
TODO

Returns

static
TODO

Returns

static_routes
TODO

Returns

to_dict() → dict
TODO

Returns

virt_bridge
TODO

Returns

class cobbler.items.system.**System**(*api*, **args*, ***kwargs*)
Bases: *cobbler.items.item.Item*

A Cobbler system object.

COLLECTION_TYPE = 'system'

autoinstall
TODO

Returns

boot_loaders
TODO

Returns

check_if_valid()
:raises CX

children
TODO

Returns

delete_interface(*name: str*)
Used to remove an interface.

Raises **TypeError** – If the name of the interface is not of type str

enable_ipxe
TODO

Returns

filename
TODO

Returns

from_dict(*dictionary: dict*)
Initializes the object with attributes from the dictionary.

Parameters **dictionary** – The dictionary with values.

gateway
TODO

Returns

get_config_filename(*interface: str*, *loader: Optional[str] = None*)
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*)
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.

hostname
TODO

Returns

image
TODO

Returns

interfaces
TODO

Returns

ipv6_autoconfiguration
TODO

Returns

ipv6_default_device
TODO

Returns

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** –

make_clone()
Must be defined in any subclass

name_servers
TODO FIXME: Differentiate between IPv4/6

Returns

name_servers_search
TODO

Returns

netboot_enabled
TODO

Returns

next_server_v4
TODO

Returns

next_server_v6
TODO

Returns

parent
Return object next highest up the tree. This may be a profile or an image.

Returns None when there is no parent or the corresponding Item.

power_address
TODO

Returns

power_id
TODO

Returns

power_identity_file
TODO

Returns

power_options
TODO

Returns

power_pass
TODO

Returns

power_type
TODO

Returns

power_user
TODO

Returns

profile
TODO

Returns

proxy
TODO

Returns

redhat_management_key
TODO

Returns

rename_interface(*old_name: str*, *new_name: str*)
Used to rename an interface.

:raises CX

repos_enabled
TODO

Returns

serial_baud_rate
TODO

Returns

serial_device
TODO

Returns

server
TODO

Returns

status
TODO

Returns

virt_auto_boot
TODO

Returns

virt_cpus
TODO

Returns

virt_disk_driver
TODO

Returns

virt_file_size
TODO

Returns

virt_path
TODO

Returns

virt_pxe_boot
TODO

Returns

virt_ram
TODO

Returns

virt_type
TODO

Returns

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.

9.1.4 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

Copyright 2007-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

cobbler.modules.authentication.configfile.**authenticate**(*api_handle*, *username: str*, *password: str*) → bool

Validate a username/password combo.

Thanks to http://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**(*text: str*) → str

Converts a str object to a hash which was configured in modules.conf of the Cobbler settings.

Parameters **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.

Copyright 2007-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

cobbler.modules.authentication.denyall.**authenticate**(*api_handle*, *username*, *password*) → bool

Validate a username/password combo, returning True/False

Thanks to http://trac.edgewall.org/ticket/845 for supplying the algorithm info.

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

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

cobbler.modules.authentication.ldap.**authenticate**(*api_handle*, *username*, *password*) → 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

Copyright 2007-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

PAM python code based on the pam_python code created by Chris AtLee: http://atlee.ca/software/pam/

#———————————————— pam_python (c) 2007 Chris AtLee <chris@atlee.ca> Licensed under the MIT license: http://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: _ctypes.Structure

wrapper class for pam_conv structure

appdata_ptr
Structure/Union member

conv
Structure/Union member

class cobbler.modules.authentication.pam.**PamHandle**
Bases: _ctypes.Structure

wrapper class for pam_handle_t

handle
Structure/Union member

class cobbler.modules.authentication.pam.**PamMessage**
Bases: _ctypes.Structure

wrapper class for pam_message structure

msg
Structure/Union member

msg_style
Structure/Union member

class cobbler.modules.authentication.pam.**PamResponse**
Bases: _ctypes.Structure

wrapper class for pam_response structure

resp
Structure/Union member

resp_retcode
Structure/Union member

cobbler.modules.authentication.pam.**authenticate**(*api_handle*, *username: str*, *password: str*) → bool
Validate PAM authentication, returning whether the authentication was successful or not.

Parameters

- **api_handle** – Used for resolving the 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.

Copyright 2008-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This software may be freely redistributed under the terms of the GNU general public license.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

cobbler.modules.authentication.passthru.**authenticate**(*api_handle*, *username*, *password*) → 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.

Copyright 2007-2008, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

`cobbler.modules.authentication.spacewalk.`**`authenticate`**(*api_handle*, *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/master/java/code/src/com/redhat/rhn/frontend/xmlrpc/auth/AuthHandler.java#L133

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.

cobbler.modules.authentication.testing module

Authentication module that denies everything. Unsafe demo. Allows anyone in with testing/testing.

Copyright 2007-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

`cobbler.modules.authentication.testing.`**`authenticate`**(*api_handle*, *username: str*, *password: str*) → bool

Validate a username/password combo, returning True/False

Thanks to http://trac.edgewall.org/ticket/845 for supplying the algorithm info.

Parameters

- **`api_handle`** – This parameter is not used currently.

- **username** – The username which should be checked.
- **password** – The password which should be checked.

Returns True if username is "testing" and password is "testing". Otherwise False.

cobbler.modules.authentication.testing.**register**() → str
The mandatory Cobbler module registration hook.

Returns Always "authn"

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/`.

cobbler.modules.authorization package

Submodules

cobbler.modules.authorization.allowall module

Authorization module that allows everything, which is the default for new Cobbler installs.

Copyright 2007-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

cobbler.modules.authorization.allowall.**authorize**(*api_handle*, *user*, *resource*, *arg1=None*, *arg2=None*) → bool
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 True

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.

This software may be freely redistributed under the terms of the GNU general public license.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

cobbler.modules.authorization.configfile.**authorize**(*api_handle*, *user*, *resource*, *arg1=None*, *arg2=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.

Copyright 2008-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

cobbler.modules.authorization.ownership.**authorize**(*api_handle*, *user*, *resource*, *arg1=None*, *arg2=None*) → Union[bool, 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 `True` or `1` if okay, otherwise `False`.

`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/`.

cobbler.modules.installation package

Submodules

cobbler.modules.installation.post_log module

(C) 2008-2009, Red Hat Inc. Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

`cobbler.modules.installation.post_log.`**register**`()` → str
The mandatory Cobbler module registration hook.

`cobbler.modules.installation.post_log.`**run**(*api*, *args*) → int

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

class `cobbler.modules.installation.post_power.`**reboot**(*api*, *target*)
Bases: `threading.Thread`

run()
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*, *args*) → int
Obligatory trigger hook.

Parameters

- **api** – The api to resolve information with.
- **args** – This is an array containing two objects. 0: String with the content "target" or "profile". 1: The name of target or profile

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: http://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*, *args*) → 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 may be system or profile and the second is the name of this system or profile.

Returns 0 or nothing.

cobbler.modules.installation.post_report module

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*, *args*) → 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: "target" 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

(C) 2008-2009, Red Hat Inc. James Laska <jlaska@redhat.com> Bill Peck <bpeck@redhat.com>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

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*, *args*) → 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

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*, *args: list*) → 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: http://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*, *args*) → 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'.

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail> John Eckersberg <jeckersb@redhat.com>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

cobbler.modules.managers.bind.**get_manager**(*collection_mgr*)
This returns the object to manage a BIND server located locally on the Cobbler server.

Parameters **collection_mgr** – The collection manager 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'.

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail> John Eckersberg <jeckersb@redhat.com>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

`cobbler.modules.managers.dnsmasq.`**`get_manager`**(*collection_mgr*)
Creates a manager object to manage a dnsmasq server.

Parameters **`collection_mgr`** – The collection manager 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.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*, *args*) → 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*, *profiles_genders*, *distros_genders*, *mgmtcls_genders*)
Genders file is over-written when `manage_genders` is set in our settings.

Parameters

- **`config`** – The config file to template with the data.
- **`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

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail> John Eckersberg <jeckersb@redhat.com>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

`cobbler.modules.managers.import_signatures.`**`get_import_manager`**(*collection_mgr*)
Get an instance of the import manager which enables you to import various things.

Parameters **`collection_mgr`** – The collection Manager instance of Cobbler

Returns The object to import data with.

`cobbler.modules.managers.import_signatures.`**`import_walker`**(*top: str*, *func: Callable*, *arg: Any*)
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.

Parameters

- **`top`** – The most top directory for which func should be run.
- **`func`** – A function which is called as described in the above description.
- **`arg`** – Passing `None` for this is common.

`cobbler.modules.managers.import_signatures.`**`register`**() → str
The mandatory Cobbler module registration hook.

cobbler.modules.managers.in_tftpd module

This is some of the code behind 'cobbler sync'.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

`cobbler.modules.managers.in_tftpd.`**`get_manager`**(*collection_mgr*)
Creates a manager object to manage an in_tftp server.

Parameters **`collection_mgr`** – The collection manager 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'.

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail> John Eckersberg <jeckersb@redhat.com>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

cobbler.modules.managers.isc.**get_manager**(*collection_mgr*)
Creates a manager object to manage an isc dhcp server.

Parameters **collection_mgr** – The collection manager 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'.

Copyright 2014, Mittwald CM Service GmbH & Co. KG Martin Helmich <m.helmich@mittwald.de> Daniel Krämer <d.kraemer@mittwald.de>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

cobbler.modules.managers.ndjbdns.**get_manager**(*collection_mgr*)
Creates a manager object to manage an isc dhcp server.

Parameters **collection_mgr** – The collection manager 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.

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

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

`cobbler.modules.serializers.file.`**`deserialize`**(*collection*, *topological: bool = True*)
Load a collection from file system.

Parameters

- **`collection`** – The collection to deserialize.
- **`topological`** – If the collection list should be sorted by the collection dict key 'depth' value or not.

`cobbler.modules.serializers.file.`**`deserialize_raw`**(*collection_types: str*)
Loads a collection from the disk.

Parameters **`collection_types`** – The type of collection to load.

Returns The loaded dictionary.

`cobbler.modules.serializers.file.`**`register`**() → str
The mandatory Cobbler module registration hook.

`cobbler.modules.serializers.file.`**`serialize`**(*collection*)
Save a collection to file system

Parameters **`collection`** – collection

`cobbler.modules.serializers.file.`**`serialize_delete`**(*collection*, *item*)
Delete a collection item from file system.

Parameters

- **`collection`** – collection
- **`item`** – collection item

`cobbler.modules.serializers.file.`**`serialize_item`**(*collection*, *item*)
Save a collection item to file system

Parameters

- **collection** – collection
- **item** – collection item

cobbler.modules.serializers.file.**what**() → str
Module identification function

cobbler.modules.serializers.mongodb module

Cobbler's Mongo database based object serializer. Experimental version.

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail> James Cammarata <jimi@sngx.net>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

cobbler.modules.serializers.mongodb.**deserialize**(*collection*, *topological: bool = True*)
Load a collection from the database.

Parameters

- **collection** – The collection to deserialize.
- **topological** – If the collection list should be sorted by the collection dict depth value or not.

cobbler.modules.serializers.mongodb.**deserialize_raw**(*collection_type: str*)
Get a collection from mongodb and parse it into an object.

Parameters **collection_type** – The collection type to fetch.

Returns The first element of the collection requested.

cobbler.modules.serializers.mongodb.**register**() → str
The mandatory Cobbler module registration hook.

cobbler.modules.serializers.mongodb.**serialize**(*collection*)
Save a collection to database

Parameters **collection** – collection

cobbler.modules.serializers.mongodb.**serialize_delete**(*collection*, *item*)
Delete a collection item from database.

Parameters

- **collection** – collection
- **item** – collection item

cobbler.modules.serializers.mongodb.**serialize_item**(*collection*, *item*)
Save a collection item to database.

Parameters

- **collection** – collection
- **item** – collection item

cobbler.modules.serializers.mongodb.**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.

Submodules

cobbler.modules.nsupdate_add_system_post module

cobbler.modules.nsupdate_add_system_post.**nslog**(*msg*)
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*, *args*)
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

cobbler.modules.nsupdate_delete_system_pre.**nslog**(*msg*)
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*, *args*)
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

(C) 2009, Red Hat Inc. Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

`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*, *args*`)`
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*, *args*`)` → 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

`cobbler.modules.sync_post_wingen.`**`bcdedit`**`(`*orig_bcd*, *new_bcd*, *wim*, *sdi*, *startoptions=None*`)`

`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*, *args*`)`

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.

9.1.5 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*) → dict
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*) → dict
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*) → 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*) → dict
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*) → dict
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*) → 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*) → dict
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*) → dict
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*) → 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*) → dict
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*) → dict
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*) → 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*) → dict
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*) → dict
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*) → 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*) → dict
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*) → dict
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*) → 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*) → dict
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*) → dict
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*) → 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*) → dict
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*) → dict
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*) → 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*) → dict
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.**normalize**(*settings: dict*) → dict
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*) → 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.helper module

Helper module which contains shared logic for adjusting the settings.

class cobbler.settings.migrations.helper.**Setting**(*location: Union[str, list], value*)
Bases: object

Specifies a setting object

key_name
Returns the location.

split_str_location(*location: str*) → List[str]
Split the given location at "." Necessary for nesting in our setttings file

Example: manage.dhcp_v4 restart.dhcp_v4

Parameters **location** –

cobbler.settings.migrations.helper.**key_add**(*new: cobbler.settings.migrations.helper.Setting*, *settings: dict*)
Add a new settings key.

Parameters

- **new** – The new setting to add.
- **settings** – [description]

cobbler.settings.migrations.helper.**key_delete**(*delete: str*, *settings: dict*)
Deletes a given setting

Parameters

- **delete** – The name of the setting to be deleted.
- **setting** – The settings dict where the the key should be deleted.

cobbler.settings.migrations.helper.**key_drop_if_default**(*settings: dict*, *defaults: dict*) → dict
Drop all keys which values are identical to the dafault 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*) → cobbler.settings.migrations.helper.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: cobbler.settings.migrations.helper.Setting*, *new_location: List[str]*, *settings: dict*)
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** –

cobbler.settings.migrations.helper.**key_rename**(*old_name: cobbler.settings.migrations.helper.Setting*, *new_name: str*, *settings: dict*)
Wrapper for key_move()

Parameters

- **old_name** – The old name
- **new_name** – The new name
- **settings** –

cobbler.settings.migrations.helper.**key_set_value**(*new: cobbler.settings.migrations.helper.Setting*, *settings: dict*)
Change the value of a setting.

Parameters

- **new** – A Settings object with the new information.
- **setting** – 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*, *settings_path: pathlib.Path*) → dict

Auto migration to the most recent version.

Parameters

- **yaml_dict** – The settings dict to migrate.
- **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/cobble*

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.**get_installed_version**(*filepath: Union[str, pathlib.Path] = '/etc/cobbler/version'*) → cobbler.settings.migrations.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: cobbler.settings.migrations.CobblerVersion*) → schema.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*) → cobbler.settings.migrations.CobblerVersion

Return the correspondig version of the given settings dict.

Parameters **yaml_dict** – The settings dict to get the version from.

Returns The discovered Cobbler Version or EMPTY_VERSION

cobbler.settings.migrations.**migrate**(*yaml_dict: dict*, *settings_path: pathlib.Path*, *old: cobbler.settings.migrations.CobblerVersion = CobblerVersion(major=0, minor=0, patch=0)*, *new: cobbler.settings.migrations.CobblerVersion = CobblerVersion(major=0, minor=0, patch=0)*) → dict

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.

Raises **ValueError** – Raised if attempting to downgraade.

Returns The migrated dict.

cobbler.settings.migrations.**normalize**(*settings: dict*) → dict

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.**validate**(*settings: dict*, *settings_path: pathlib.Path = ''*) → 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

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*)

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='/etc/cobbler/settings.yaml'*)

Saves the settings to the disk.

to_dict() → dict
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

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.**autodetect_bind_chroot**()
Autodetect bind chroot configuration

cobbler.settings.**migrate**(*yaml_dict: dict*, *settings_path: pathlib.Path*) → dict
Migrates the current settings

Parameters

- **yaml_dict** – The settings dict
- **settings_path** – The settings path

Returns The migrated settings

cobbler.settings.**parse_bind_config**(*configpath: str*)
Parse the Bind9 configuration file and adjust the Cobbler default settings according to the readings.

Parameters **configpath** – The path in the filesystem where the file can be read.

cobbler.settings.**read_settings_file**(*filepath='/etc/cobbler/settings.yaml'*) → Dict[Hashable, 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.

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='/ect/cobbler/settings.yaml'*) → Dict[Hashable, Any]
Reads settings files from `filepath` and all paths in *include* (which is read from the settings file) and saves the content in a dictionary. Any key may be overwritten in a later loaded settings file. The last loaded file wins.

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*, *filepath='/etc/cobbler/settings.yaml'*) → 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.

Returns True if the action succeeded. Otherwise return False.

cobbler.settings.**validate_settings**(*settings_content: dict*) → dict

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. :raises SchemaError: In case the data given is invalid. :return: The Settings of Cobbler which can be safely used inside this instance.

9.2 Submodules

9.3 cobbler.api module

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.api.**CobblerAPI**(*is_cobblerd: bool = False*, *settingsfile_location: str = '/etc/cobbler/settings.yaml'*, *execute_settings_automigration: bool = True*)

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=None*, *addgroup=None*, *removeuser=None*, *removegroup=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*, *check_for_duplicate_names: bool = False*, *save: bool = True*)

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.

add_file(*ref*, *check_for_duplicate_names: bool = False*, *save: bool = True*)

Add a file 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.

add_image(*ref*, *check_for_duplicate_names: bool = False*, *save: bool = True*)
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.

add_item(*what*, *ref*, *check_for_duplicate_names: bool = False*, *save: bool = True*)
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.

add_menu(*ref*, *check_for_duplicate_names=False*, *save=True*)
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.

add_mgmtclass(*ref*, *check_for_duplicate_names: bool = False*, *save: bool = True*)
Add a management class 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.

add_package(*ref*, *check_for_duplicate_names: bool = False*, *save: bool = True*)
Add a package 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.

add_profile(*ref*, *check_for_duplicate_names: bool = False*, *save: bool = True*)
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.

add_repo(*ref*, *check_for_duplicate_names: bool = False*, *save: bool = True*)
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.

add_system(*ref*, *check_for_duplicate_names: bool = False*, *check_for_duplicate_netinfo=False*, *save: bool = True*)
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.
- **check_for_duplicate_netinfo** – If the name of the network interface should be unique or can be present multiple times.
- **save** – If the item should be persisted.

authenticate(*user*, *password*)
(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*, *resource*, *arg1=None*, *arg2=None*)
(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()
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=None*, *profiles=None*, *systems=None*, *buildisodir=None*, *distro=None*, *standalone=None*, *airgapped=None*, *source=None*, *exclude_dns=None*, *xorrisofs_opts=None*)
Build an iso image which may be network bootable or not.

Parameters

- **iso** –
- **profiles** –
- **systems** –
- **buildisodir** –
- **distro** –
- **standalone** –
- **airgapped** –
- **source** –
- **exclude_dns** –
- **xorrisofs_opts** –

check() → Union[None, list]
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 None or a list of things to address.

clear_logs(*system*)
Clears console and anamon logs for system

Parameters **system** – The system to clear logs of.

copy_distro(*ref*, *newname*)
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_file(*ref*, *newname*)
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_image(*ref*, *newname*)
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*, *ref*, *newname*)
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*, *newname*)
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_mgmtclass(*ref*, *newname*)
This method copies a management class 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_package(*ref*, *newname*)
This method copies a package 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*, *newname*)
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*, *newname*)
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*, *newname*)
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()
Load cobbler_collections from disk. Cobbler internal use only.

distros()
Return the current list of distributions

dump_vars(*obj*, *formatted_output: bool = False*)
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.

Returns A dictionary with all the information which could be collected.

files()
Return the current list of files

find_distro(*name=None*, *return_list=False*, *no_errors=False*, ***kargs*)
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_file(*name=None*, *return_list=False*, *no_errors=False*, ***kargs*)
Find a file 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=None*, *return_list=False*, *no_errors=False*, ***kargs*)
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: dict = None*, *name: Optional[str] = None*, *return_list: bool = True*, *no_errors: bool = False*)
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=None*, *return_list=False*, *no_errors=False*, ***kargs*)
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_mgmtclass(*name=None*, *return_list=False*, *no_errors=False*, ***kargs*)
Find a management class 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_package(*name=None*, *return_list=False*, *no_errors=False*, ***kargs*)
Find a package 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=None*, *return_list=False*, *no_errors=False*, ***kargs*)
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=None*, *return_list=False*, *no_errors=False*, ***kargs*)
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=None*, *return_list=False*, *no_errors=False*, ***kargs*)
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*, *system*)
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*, *image*, *system*)
Generate the ipxe configuration files. The system wins over the profile.

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*, *system*, *name*)
Generate an autoinstall script for the specified profile or system. The system wins over the profile.

Parameters

- **profile** – The profile to generate the script for.
- **system** – The system to generate the script for.
- **name** – The name of the script which should be generated.

Returns The generated script or an error message.

get_distros_since(*mtime: float*, *collapse: bool = False*)
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_files_since(*mtime: float*, *collapse: bool = False*) → list
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_images_since(*mtime: float*, *collapse: bool = False*) → list
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*)
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_items(*what: str*)
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=False*) → list
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_mgmtclasses_since(*mtime: float*, *collapse: bool = False*) → list
Return management classes 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 management classes which are newer then the given timestamp.

get_module_by_name(*module_name*)
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*, *name*, *fallback=None*)
Looks in `/etc/cobbler/modules.conf` 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** –
- **name** –
- **fallback** –

Returns

get_module_name_from_file(*section*, *name*, *fallback=None*)
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*)
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_packages_since(*mtime: float*, *collapse: bool = False*) → list
Return packages 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 packages which are newer then the given timestamp.

get_profiles_since(*mtime: float*, *collapse: bool = False*) → list
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*) → 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*) → 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
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.

get_signatures() → dict
This returns the local signature cache.

Returns The dict containing all signatures.

get_sync(*verbose: bool = False*)
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
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*, *path*) → 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*, *path*)
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_valid_obj_boot_loaders(*obj*)
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()
Hardlink all files where this is possible to improve performance.

Returns The return code of the subprocess call which actually hardlinks the files.

images()
Return the current list of images

import_tree(*mirror_url: str*, *mirror_name: str*, *network_root=None*, *autoinstall_file=None*, *rsync_flags=None*, *arch=None*, *breed=None*, *os_version=None*) → Optional[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** –
- **autoinstall_file** –
- **rsync_flags** –
- **arch** –

- **breed** –
- **os_version** –

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=None*, *debug: bool = False*)
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()
Return the current list of menus

mgmtclasses()
Return the current list of mgmtclasses

new_distro(*is_subobject: bool = False*)
Returns a new empty distro object. This distro is not automatically persited. Persistance is achieved via `save()`.

Parameters **is_subobject** – If the object created is a subobject or not.

Returns An empty Distro object.

new_file(*is_subobject: bool = False*)
Returns a new empty file 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 File object.

new_image(*is_subobject: bool = False*)
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_menu(*is_subobject: bool = False*)
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 File object.

new_mgmtclass(*is_subobject: bool = False*)
Returns a new empty mgmtclass object. This mgmtclass is not automatically persisted. Persistence is achieved via `save()`.

Parameters **is_subobject** – If the object created is a subobject or not.

Returns An empty mgmtclass object.

new_package(*is_subobject: bool = False*)
Returns a new empty package object. This package is not automatically persisted. Persistence is achieved via `save()`.

Parameters **is_subobject** – If the object created is a subobject or not.

Returns An empty Package object.

new_profile(*is_subobject: bool = False*)
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*)
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*)
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.

packages()
Return the current list of packages

power_system(*system: str*, *power_operation: str*, *user: Optional[str] = None*, *password: Optional[str] = None*)
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()
Return the current list of profiles

remove_distro(*ref: str*, *recursive: bool = False*, *delete: bool = True*, *with_triggers: bool = True*)
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_file(*ref: str*, *recursive: bool = False*, *delete: bool = True*, *with_triggers: bool = True*)
Remove a file 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: str*, *recursive: bool = False*, *delete: bool = True*, *with_triggers: bool = True*)
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: str*, *recursive: bool = False*, *delete: bool = True*, *with_triggers: bool = True*)
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*, *recursive=False*, *delete=True*, *with_triggers=True*)
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_mgmtclass(*ref: str*, *recursive: bool = False*, *delete: bool = True*, *with_triggers: bool = True*)
Remove a management class 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_package(*ref: str*, *recursive: bool = False*, *delete: bool = True*, *with_triggers: bool = True*)
Remove a package 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: str*, *recursive: bool = False*, *delete: bool = True*, *with_triggers: bool = True*)
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: str*, *recursive: bool = False*, *delete: bool = True*, *with_triggers: bool = True*)
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: str*, *recursive: bool = False*, *delete: bool = True*, *with_triggers: bool = True*)
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*, *newname*)
Rename a distro to a new name.

Parameters

- **ref** – The internal unique handle for the item.
- **newname** – The new name for the item.

rename_file(*ref*, *newname*)
Rename a file to a new name.

Parameters

- **ref** – The internal unique handle for the item.
- **newname** – The new name for the item.

rename_image(*ref*, *newname: str*)
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*, *ref*, *newname*)
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*, *newname*)
Rename a menu to a new name.

Parameters

- **ref** – The internal unique handle for the item.
- **newname** – The new name for the item.

rename_mgmtclass(*ref*, *newname*)
Rename a management class to a new name.

Parameters

- **ref** – The internal unique handle for the item.
- **newname** – The new name for the item.

rename_package(*ref*, *newname*)
Rename a package to a new name.

Parameters

- **ref** – The internal unique handle for the item.
- **newname** – The new name for the item.

rename_profile(*ref*, *newname*)
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*, *newname*)
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*, *newname*)
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 = ''*, *mgmtclass_patterns=None*, *package_patterns=None*, *file_patterns: bool = False*, *prune: bool = False*, *omit_data: bool = False*, *sync_all: bool = False*, *use_ssl: bool = False*)

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.
- **mgmtclass_patterns** – The pattern of management classes which should be synced.
- **package_patterns** – The pattern of packages which should be synced.
- **file_patterns** – The pattern of files 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=None*, *report_name=None*, *report_type=None*, *report_fields=None*, *report_noheaders=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()

Return the current list of repos

reposync(*name=None*, *tries: int = 1*, *nofail: bool = False*)

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()
Save the cobbler_collections to disk. Cobbler internal use only.

settings()
Return the application configuration

signature_update()
Update all signatures from the URL specified in the settings.

status(*mode*)
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: list = []*)
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()
Only build out the DHCP configuration.

sync_dns()
Only build out the DNS configuration.

sync_systems(*systems: List[str], verbose: bool = False*)
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()
Return the current list of systems

validate_autoinstall_files()
Validate if any of the autoinstallation files are invalid and if yes report this.

version(*extended: bool = False*)
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.

9.4 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**(*collection_mgr*)
Bases: object

Manage automatic installation templates, snippets and final files

generate_autoinstall(*profile=None*, *system=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
Get a list of all autoinstallation snippets.

Returns The list of snippets

get_autoinstall_templates() → list
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*)
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*, *is_profile: bool*) → list
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

9.5 cobbler.autoinstallgen module

Builds out filesystem trees/data based on the object tree. This is the code behind 'cobbler sync'.

class cobbler.autoinstallgen.**AutoInstallationGen**(*collection_mgr*)

Bases: object

Handles conversion of internal state to the tftpboot tree layout

addAutoYaSTScript(*document*, *type*, *source*)

Add scripts to an existing AutoYaST XML.

Parameters

- **document** – The existing AutoYaST XML object.
- **type** – The type of the script which should be added.
- **source** – The source of the script. This should be ideally a string.

createAutoYaSTScript(*document*, *script*, *name*)

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=None*, *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(*g*) → str

Generate an autoinstall config or script for a profile.

Parameters **g** – 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
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=None*, *system=None*, *raw_data=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*, *is_profile: bool = True*)
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*, *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
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.

9.6 cobbler.cexceptions module

Custom exceptions for Cobbler

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

exception cobbler.cexceptions.**CX**(*value*, **args*)
Bases: *cobbler.cexceptions.CobblerException*

This is a general exception which gets thrown often inside Cobbler.

exception cobbler.cexceptions.**CobblerException**(*value*, **args*)
Bases: Exception

This is the default Cobbler exception where all other exceptions are inheriting from.

9.7 cobbler.cli module

Command line interface for Cobbler.

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.cli.**CobblerCLI**(*cliargs*)
Bases: object

Main CLI Class which contains the logic to communicate with the Cobbler Server.

check_setup()
Detect permissions and service accessibility problems and provide nicer error messages for them.

cleanup_fault_string(*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 **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*) → Optional[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*)
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*)
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: optparse.Option*, *opt_str*, *value: str*, *parser: optparse.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**()

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.**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.

9.8 cobbler.cobblerd module

Cobbler daemon for logging remote syslog traffic during automatic installation

Copyright 2007-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

cobbler.cobblerd.**core**(*cobbler_api: cobbler.api.CobblerAPI*)
Starts Cobbler.

Parameters **cobbler_api** – The cobbler_api instance which is used for this method.

cobbler.cobblerd.**do_xmlrpc_rw**(*cobbler_api: cobbler.api.CobblerAPI*, *port*)
This trys to bring up the Cobbler xmlrpc_api and restart it if it fails.

Parameters

- **cobbler_api** – The cobbler_api instance which is used for this method.
- **port** – The port where the xmlrpc api should run on.

cobbler.cobblerd.**regen_ss_file**()
This is only used for Kerberos auth at the moment. It identifies XMLRPC requests from Apache that have already been cleared by Kerberos.

9.9 cobbler.configgen module

configgen.py: Generate configuration data.

Copyright 2010 Kelsey Hightower Kelsey Hightower <kelsey.hightower@gmail.com>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

module for generating configuration manifest using autoinstall_meta data, mgmtclasses, resources, and templates for a given system (hostname)

class cobbler.configgen.**ConfigGen**(*cobbler_api*, *hostname: str*)
Bases: object

Generate configuration data for Cobbler's management resources: repos, files and packages. Mainly used by Koan to configure systems.

gen_config_data() → dict
Generate configuration data for repos, files and packages.

Returns A dict which has all config data in it.

Raises ***CX*** – In case the package or file resource is not defined.

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, str, dict]
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: 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.

9.10 cobbler.download_manager module

Cobbler DownloadManager

class cobbler.download_manager.**DownloadManager**(*collection_mgr*)

Bases: object

download_file(*url*, *dst*, *proxies=None*, *cert=None*)

Donwload a file from a URL and save it to any disc location.

Parameters

- **url** – The URL the request.
- **dst** – The destination file path.
- **proxies** – Override the default Cobbler proxies.
- **cert** – Override the default Cobbler certs.

urlread(*url*, *proxies=None*, *cert=None*)

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.

9.11 cobbler.enums module

TODO

class cobbler.enums.**Archs**

Bases: enum.Enum

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**
Bases: enum.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

class cobbler.enums.**ImageTypes**
Bases: enum.Enum

This enum represents all image types which Cobbler can manage.

DIRECT = 'direct'

ISO = 'iso'

MEMDISK = 'memdisk'

VIRT_CLONE = 'virt-clone'

class cobbler.enums.**MirrorType**
Bases: enum.Enum

This enum represents all mirror types which Cobbler can manage.

BASEURL = 'baseurl'

METALINK = 'metalink'

MIRRORLIST = 'mirrorlist'

class cobbler.enums.**NetworkInterfaceType**
Bases: enum.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**
Bases: enum.Enum

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**
Bases: enum.Enum

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**
Bases: enum.Enum

This enum represents all actions a resource may execute.

CREATE = 'create'

REMOVE = 'remove'

class cobbler.enums.**VirtDiskDrivers**
Bases: enum.Enum

This enum represents all virtual disk driver Cobbler can handle.

INHERTIED = '<<inherit>>'

QCOW2 = 'qcow2'

QED = 'qed'

RAW = 'raw'

VDI = 'vdi'

VDMK = 'vdmk'

class cobbler.enums.**VirtType**
Bases: enum.Enum

This enum represents all known types of virtualization Cobbler is able to handle via Koan.

AUTO = 'auto'

INHERTIED = '<<inherit>>'

KVM = 'kvm'

OPENVZ = 'openvz'

QEMU = 'qemu'

VMWARE = 'vmware'

VMWAREW = 'vmwarew'

XENFV = 'xenfv'

XENPV = 'xenpv'

9.12 cobbler.grub module

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

9.13 cobbler.manager module

Base class for modules.managers.* classes

Copyright 2021 SUSE LLC Thomas Renninger <trenn@suse.de>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.manager.**ManagerModule**(*collection_mgr*)
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 like dhcp can be managed via isc.py or dnsmasq.py (compare with `/etc/cobbler/modules.py`).

regen_ethers()
ISC/BIND doesn't use this. It is there for compability reasons with other managers.

restart_service()
Write module specific config files. E.g. dhcp manager would write `/etc/dhcpd.conf` here

sync()
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()
Static method to identify the manager module. Must be overwritten by the inheriting class

write_configs()
Write module specific config files. E.g. dhcp manager would write `/etc/dhcpd.conf` here

9.14 cobbler.module_loader module

Module loader, adapted for Cobbler usage

Copyright 2006-2009, Red Hat, Inc and Others Adrian Likins <alikins@redhat.com> Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

cobbler.module_loader.**get_module_by_name**(*name: str*)
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.

cobbler.module_loader.**get_module_from_file**(*category: str*, *field: str*, *fallback_module_name: Optional[str] = None*)
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.

cobbler.module_loader.**get_module_name**(*category: str*, *field: str*, *fallback_module_name: Optional[str] = None*) → str

Get module name from configuration file (currently hardcoded `/etc/cobbler/modules.conf`).

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.

cobbler.module_loader.**get_modules_in_category**(*category: str*) → list

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.

cobbler.module_loader.**load_modules**(*module_path: str = '/home/docs/checkouts/readthedocs.org/user_builds/cobbler/checkouts/latest/c*

Load the modules from the path handed to the function into Cobbler.

Parameters **module_path** – The path which should be considered as the root module path.

Returns Two dictionary's with the dynamically loaded modules.

9.15 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*, *collection_mgr*)

Bases: `object`

Handles power management in systems

get_power_status(*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*, *user: Optional[str] = None*, *password: Optional[str] = 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*, *user: Optional[str] = None*, *password: Optional[str] = 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*, *user: Optional[str] = None*, *password: Optional[str] = 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
Get possible power management types.

Returns Possible power management types

cobbler.power_manager.**validate_power_type**(*power_type: str*)
Check if a power management type is valid.

Parameters **power_type** – Power management type.

Raises ***CX*** – if power management type is invalid

9.16 cobbler.remote module

Copyright 2007-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.remote.**CobblerThread**(*event_id*, *remote*, *options: dict*, *task_name: str*, *api*)
Bases: threading.Thread

Code for Cobbler's XMLRPC API.

on_done()
This stub is needed to satisfy the Python inheritance chain.

run()
Run the thread.

Returns The return code of the action. This may possibly a boolean or a Linux return code.

class cobbler.remote.**CobblerXMLRPCInterface**(*api*)
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*)

Parameters **token** – The API-token obtained via the login() method.

background_aclsetup(*options*, *token*) → 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*, *token*) → str
Generates an ISO in /var/www/cobbler/pub that can be used to install profiles without using PXE.

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_hardlink(*options*, *token*) → 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*, *token*) → 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_power_system(*options*, *token*) → 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*, *token*) → 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*, *token*) → 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*, *token*) → 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*, *token*) → 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*, *token*) → 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*, *token*) → 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*) → Union[None, list]
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 None or a list of things to address.

check_access(*token*, *resource*, *arg1=None*, *arg2=None*)
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 Whether the authentication was successful or not.

check_access_no_fail(*token*, *resource*, *arg1=None*, *arg2=None*) → bool
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 True if the object is editable or False otherwise.

clear_system_logs(*object_id*, *token=None*)
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*, *newname*, *token=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_file(*object_id*, *newname*, *token=None*)
Copies a file 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_image(*object_id*, *newname*, *token=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*, *object_id*, *newname*, *token=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*, *newname*, *token=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_mgmtclass(*object_id*, *newname*, *token=None*)
Copies a management class 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_package(*object_id*, *newname*, *token=None*)
Copies a package 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*, *newname*, *token=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*, *newname*, *token=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*, *newname*, *token=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*, *token=None*, ***rest*) → 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.

extended_version(*token=None*, ***rest*)
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=None*, *expand=False*, *token=None*, ***rest*)
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.
- **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_file(*criteria=None*, *expand=False*, *token=None*, ***rest*)
Find a file 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.
- **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_image(*criteria=None*, *expand=False*, *token=None*, ***rest*)
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.
- **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*, *criteria: dict = None*, *sort_field=None*, *expand: bool = True*) → list
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.

Returns A list of dicts.

find_items_paged(*what*, *criteria=None*, *sort_field=None*, *page=None*, *items_per_page: int = None*, *token=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.
- **token** – The API-token obtained via the login() method.

Returns The found items.

find_menu(*criteria=None*, *expand=False*, *token=None*, ***rest*)
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.
- **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_mgmtclass(*criteria=None*, *expand=False*, *token=None*, ***rest*)
Find a management class 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.
- **token** – The API-token obtained via the login() method.
- **rest** – This parameter is not used currently.

Returns All management classes which have matched the criteria.

find_package(*criteria=None*, *expand=False*, *token=None*, ***rest*)
Find a package 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.
- **token** – The API-token obtained via the login() method.
- **rest** – This parameter is not used currently.

Returns All packages which have matched the criteria.

find_profile(*criteria=None*, *expand=False*, *token=None*, ***rest*)
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.
- **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=None*, *expand=False*, *token=None*, ***rest*)
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.
- **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=None*, *expand=False*, *token=None*, ***rest*)
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.
- **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*)
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=None*, *system=None*, *REMOTE_ADDR=None*, *REMOTE_MAC=None*, ***rest*)
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=None*, *system=None*, ***rest*)
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=None*, *image=None*, *system=None*, ***rest*)
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*)
Generate a profile autoinstallation.

Parameters **profile** – The profile to generate the file for.

Returns The str representation of the file.

generate_script(*profile=None*, *system=None*, *name=None*, ***rest*)
Not known what this does exactly.

Parameters

- **profile** – Not known for what the profile is needed.
- **system** – Not known for what the system is needed.
- **name** – Name of the generated script.
- **rest** – This is dropped in this method since it is not needed here.

Returns Some generated script.

generate_system_autoinstall(*system*)
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*)
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=None*, ***rest*)
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=None*, ***rest*)
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=None*, *system=None*)
Combine all data which is available from a profile and system together and return it.

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
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=False*, *token=None*, ***rest*)
Get a distribution.

Parameters

- **name** – The name of the distribution to get.
- **flatten** – If the item should be flattened.
- **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: str = None*, ***rest*)
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*, *token*)
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.
- **token** – The API-token obtained via the login() method.

Returns The handle of the desired object.

get_distros(*page=None*, *results_per_page=None*, *token=None*, ***rest*)
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*)
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
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
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_file(*name: str*, *flatten=False*, *token=None*, ***rest*)
Get a file.

Parameters

- **name** – The name of the file to get.
- **flatten** – If the item should be flattened.
- **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_file_as_rendered(*name: str*, *token: str = None*, ***rest*)
Get file after passing through Cobbler's inheritance engine

Parameters

- **name** – file 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_file_handle(*name*, *token*)
Get a handle for a file which allows you to use the functions modify_* or save_* to manipulate it.

Parameters

- **name** – The name of the item.
- **token** – The API-token obtained via the login() method.

Returns The handle of the desired object.

get_files(*page=None*, *results_per_page=None*, *token=None*, ***rest*)
This returns all files.

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_files_since(*mtime: 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_image(*name: str*, *flatten=False*, *token=None*, ***rest*)
Get an image.

Parameters

- **name** – The name of the image to get.
- **flatten** – If the item should be flattened.
- **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: str = None*, ***rest*)
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*, *token*)
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.
- **token** – The API-token obtained via the login() method.

Returns The handle of the desired object.

get_images(*page=None*, *results_per_page=None*, *token=None*, ***rest*)
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*)
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=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)

Returns The item or None.

get_item_handle(*what*, *name*, *token=None*)
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.
- **token** – The API-token obtained via the login() method.

Returns The handle of the desired object.

get_item_names(*what: 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_items(*what: str*)
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*, *token=None*, ***rest*)
Get a menu.

Parameters

- **name** – The name of the file to get.
- **flatten** – If the item should be flattened.
- **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*)
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*, *token*)
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.
- **token** – The API-token obtained via the login() method.

Returns The handle of the desired object.

get_menus(*page=None*, *results_per_page=None*, *token=None*, ***rest*)
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*)
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_mgmtclass(*name: str*, *flatten=False*, *token=None*, ***rest*)
Get a management class.

Parameters

- **name** – The name of the management class to get.
- **flatten** – If the item should be flattened.
- **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_mgmtclass_as_rendered(*name: str*, *token: str = None*, ***rest*)
Get management class after passing through Cobbler's inheritance engine

Parameters

- **name** – management class name
- **token** – authentication token
- **rest** – This is dropped in this method since it is not needed here.

Returns Get a template rendered as a management class.

get_mgmtclass_handle(*name*, *token*)
Get a handle for a management class which allows you to use the functions `modify_*` or `save_*` to manipulate it.

Parameters

- **name** – The name of the item.
- **token** – The API-token obtained via the login() method.

Returns The handle of the desired object.

get_mgmtclasses(*page=None*, *results_per_page=None*, *token=None*, ***rest*)
This returns all managementclasses.

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 managementclasses.

get_mgmtclasses_since(*mtime: 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_package(*name: str*, *flatten=False*, *token=None*, ***rest*)
Get a package.

Parameters

- **name** – The name of the package to get.
- **flatten** – If the item should be flattened.
- **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_package_as_rendered(*name: str*, *token: str = None*, ***rest*)
Get package after passing through Cobbler's inheritance engine

Parameters

- **name** – package name
- **token** – authentication token
- **rest** – This is dropped in this method since it is not needed here.

Returns Get a template rendered as a package.

get_package_handle(*name*, *token*)
Get a handle for a package which allows you to use the functions `modify_*` or `save_*` to manipulate it.

Parameters

- **name** – The name of the item.
- **token** – The API-token obtained via the login() method.

Returns The handle of the desired object.

get_packages(*page=None*, *results_per_page=None*, *token=None*, ***rest*)
This returns all packages.

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 packages tracked in Cobbler.

get_packages_since(*mtime: 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=False*, *token=None*, ***rest*)
Get a profile.

Parameters

- **name** – The name of the profile to get.
- **flatten** – If the item should be flattened.
- **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: str = None*, ***rest*)
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*, *token*)
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.
- **token** – The API-token obtained via the login() method.

Returns The handle of the desired object.

get_profiles(*page=None*, *results_per_page=None*, *token=None*, ***rest*)
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*)
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='xenpv'*, *token=None*, ***rest*)
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=False*, *token=None*, ***rest*)
Get a repository.

Parameters

- **name** – The name of the repository to get.
- **flatten** – If the item should be flattened.
- **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: str = None*, ***rest*)
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*, ***rest*)
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*, ***rest*)
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*, *token*)
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.
- **token** – The API-token obtained via the login() method.

Returns The handle of the desired object.

get_repos(*page=None*, *results_per_page=None*, *token=None*, ***rest*)
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=None*, *token=None*, ***rest*) → list
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*)
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=None*, ***rest*)
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=None*, ***rest*)
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='normal'*, *token=None*, ***rest*)
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=False*, *token=None*, ***rest*)
Get a system.

Parameters

- **name** – The name of the system to get.
- **flatten** – If the item should be flattened.
- **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: str = None*, ***rest*)
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*, *token*)
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.
- **token** – The API-token obtained via the login() method.

Returns The handle of the desired object.

get_systems(*page=None*, *results_per_page=None*, *token=None*, ***rest*)
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*)
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*)
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*, *path*, ***rest*)
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*, *path*, ***rest*)
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_user_from_token(*token*)
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.

get_valid_archs(*token=None*)
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=None*, ***rest*)
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*, *token=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*, *token=None*)
Return the list of valid boot loaders for the image

Parameters

- **token** – The API-token obtained via the login() method.
- **distro_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=None*, ***rest*)
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*, *token=None*, ***rest*)
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*, *token=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*, *token=None*)
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.

has_item(*what*, *name*, *token=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.

is_autoinstall_in_use(*ai*, *token=None*, ***rest*)
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=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*, *login_password*)
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*) → 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*, *attribute*, *arg*, *token*)
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_file(*object_id*, *attribute*, *arg*, *token*)
Modify a single attribute of a file.

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*, *attribute*, *arg*, *token*)
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*, *object_id*, *attribute*, *arg*, *token*) → 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.1
- **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*, *attribute*, *arg*, *token*)
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_mgmtclass(*object_id*, *attribute*, *arg*, *token*)
Modify a single attribute of a managementclass.

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_package(*object_id*, *attribute*, *arg*, *token*)
Modify a single attribute of a package.

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*, *attribute*, *arg*, *token*)
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*, *attribute*, *arg*, *token*)
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*, *token*) → 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*, *attribute*, *arg*, *token*)
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*)
See `new_item()`.

Parameters **token** – The API-token obtained via the login() method.

Returns The object id for the newly created object.

new_file(*token*)
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*)
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*, *token*, *is_subobject: bool = False*)
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` `mgmtclass`, `package`, `file` 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*)
See `new_item()`.

Parameters **token** – The API-token obtained via the login() method.

Returns The object id for the newly created object.

new_mgmtclass(*token*)
See `new_item()`.

Parameters **token** – The API-token obtained via the login() method.

Returns The object id for the newly created object.

new_package(*token*)
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*)
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*)
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*)
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*)
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*, *power*, *token*) → 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
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
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*, *token=None*, ***rest*)
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*)
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*)
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*, *token*, *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_file(*name*, *token*, *recursive: bool = True*)
Deletes a file 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*, *token*, *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*, *name*, *token*, *recursive: bool = True*)
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*, *token*, *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_mgmtclass(*name*, *token*, *recursive: bool = True*)
Deletes a managementclass 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_package(*name*, *token*, *recursive: bool = True*)
Deletes a package 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*, *token*, *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*, *token*, *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*, *token*, *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*, *newname*, *token=None*)
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_file(*object_id*, *newname*, *token=None*)
Renames a file 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*, *newname*, *token=None*)
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*, *object_id*, *newname*, *token=None*)
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*, *newname*, *token=None*)
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_mgmtclass(*object_id*, *newname*, *token=None*)
Renames a managementclass 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_package(*object_id*, *newname*, *token=None*)
Renames a package 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*, *newname*, *token=None*)
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*, *newname*, *token=None*)
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*, *newname*, *token=None*)
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*, *objtype*, *name*, *ip*, *token=None*, ***rest*)
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*, *token*, *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_file(*object_id*, *token*, *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*, *token*, *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*, *object_id*, *token*, *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*, *token*, *editmode='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_mgmtclass(*object_id*, *token*, *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_package(*object_id*, *token*, *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*, *token*, *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*, *token*, *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*, *token*, *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.

sync(*token*)
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*)
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*) → 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*, *file*, *size*, *offset*, *data*, *token=None*, ***rest*)
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 recieved.
- **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.
- **rest** – This is dropped in this method since it is not needed here.

Returns True if everything succeeded.

version (*token=None*, ***rest*)
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*) → 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*)
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*, *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*)
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*)
Bases: socketserver.ThreadingMixIn, xmlrpc.server.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*, *proxy_class*)
Bases: object

class cobbler.remote.**RequestHandler**(*request*, *client_address*, *server*)
Bases: xmlrpc.server.SimpleXMLRPCRequestHandler

do_OPTIONS()

end_headers()
Send the blank line ending the MIME headers.

9.17 cobbler.serializer module

Serializer code for Cobbler Now adapted to support different storage backends

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

cobbler.serializer.**deserialize**(*collection*, *topological: bool = True*)
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

cobbler.serializer.**handler**(*num*, *frame*)

cobbler.serializer.**serialize**(*collection*)
Save a collection to disk

Parameters **collection** – The collection to serialize.

cobbler.serializer.**serialize_delete**(*collection*, *item*)
Delete a collection item from disk

Parameters

- **collection** – The Cobbler collection to know the type of the item.
- **item** – The collection item to delete.

cobbler.serializer.**serialize_item**(*collection*, *item*)
Save a collection item to disk

Parameters

- **collection** – The Cobbler collection to know the type of the item.
- **item** – The collection item to serialize.

9.18 cobbler.services module

Mod Python service functions for Cobbler's public interface (aka cool stuff that works with wget/curl)

based on code copyright 2007 Albert P. Tobey <tobert@gmail.com> additions: 2007-2009 Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.services.**CobblerSvc**(*server=None*, *req=None*)
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*) → 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=None*, *system=None*, *REMOTE_ADDR=None*, *REMOTE_MAC=None*, ***rest*)
Generate automatic installation files.

Parameters

- **profile** –

- **system** –
- **REMOTE_ADDR** –
- **REMOTE_MAC** –
- **rest** – This parameter is unused.

Returns

bootcfg (*profile=None*, *system=None*, ***rest*)
Generate a boot.cfg config file. Used primarily for VMware ESXi.

Parameters

- **profile** –
- **system** –
- **rest** – This parameter is unused.

Returns

debug (*profile=None*, ***rest*)

events (*user=''*, ***rest*) → 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=None*, *profile=None*, ***rest*)
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=None*, *profile=None*, ***rest*)
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*) → str
Just a placeholder method as an entry point.

Parameters **args** – This parameter is unused.

Returns "no mode specified"

ipxe (*profile=None*, *image=None*, *system=None*, *mac=None*, ***rest*)
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=None*, *system=None*, *REMOTE_ADDR=None*, *REMOTE_MAC=None*, ***rest*)
Generate automatic installation files. This is a legacy function for part backward compability to 2.6.6 releases.

Parameters

- **profile** –
- **system** –
- **REMOTE_ADDR** –
- **REMOTE_MAC** –
- **rest** – This parameter is unused.

Returns

list(*what='systems'*, ***rest*) → str
Return a list of objects of a desired category. Defaults to "systems".

Parameters

- **what** – May be "systems", "profiles", "distros", "images", "repos", "mgmtclasses", "packages", "files" or "menus"
- **rest** – This parameter is unused.

Returns The list of object names.

look(***rest*) → str
Debug only: Show the handed dict via repr to the requester. :param rest: The dict to represent. :return: The dict reformated with repr()

nopxe(*system=None*, ***rest*) → 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.

puppet(*hostname=None*, ***rest*) → str
Dump the puppet data which is available for Cobbler.

Parameters

- **hostname** – The hostname for the system which should the puppet data be dumped for.
- **rest** – This parameter is unused.

Returns The yaml for the host.

script(*profile=None*, *system=None*, ***rest*) → 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() → cobbler.settings.Settings
Get the application configuration.

Returns Settings object.

template(*profile=None*, *system=None*, *path=None*, ***rest*) → 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=None*, *system=None*, *REMOTE_ADDR=None*, ***rest*) → 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=None*, *system=None*, ***rest*) → 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.

9.19 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!)

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.templar.**Templar**(*collection_mgr*)

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*)

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, out_path: Optional[str], template_type='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*, *search_table: dict*) → 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*) → 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.

9.20 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.

Written by Daniel Guernsey <danpg102@gmail.com> Contributions by Michael DeHaan <michael.dehaan AT gmail> US Government work; No explicit copyright attached to this file.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.template_api.**CobblerTemplate**(***kwargs*)

Bases: DynamicallyCompiledCheetahTemplate.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. We don't need to override __init__

SNIPPET(*file: str*)

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*, ***kwargs*) → 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**()

cobbler.template_api.**read_macro_file**(*location='/etc/cobbler/cheetah_macros'*)

9.21 cobbler.tftpgen module

Generate files provided by TFTP server based on Cobbler object tree. This is the code behind 'cobbler sync'.

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.tftpgen.**TFTPGen**(*collection_mgr*)

Bases: object

Generate files provided by TFTP server

build_kernel(*metadata*, *system*, *profile*, *distro*, *image=None*, *boot_loader: str = 'pxe'*)

Generates kernel and initrd metadata.

Parameters

- **metadata** – Pass additional parameters to the ones being collected during the method.
- **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*, *profile*, *distro*, *image*, *arch: str*, *autoinstall_path*) → 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*)
Copy bootloaders to the configured tftpboot directory NOTE: we support different arch's if defined in our settings file.

copy_images()
Like copy_distros except for images.

copy_single_distro_file(*d_file: str*, *distro_dir: str*, *symlink_ok: bool*)
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(*d*, *dirtree*, *symlink_ok: bool*)
Copy the files needed for a single distro.

Parameters

- **d** – 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*)
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_script(*what: str*, *objname: str*, *script_name*) → 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.

get_images_menu(*menu*, *metadata*, *arch: str*)
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[str] = None*) → dict
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=None*, *arch: str = None*) → dict
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*, *metadata*, *arch: str*)
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*, *metadata: dict*, *arch: str*)
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, str]
Generates pxe, ipxe and grub boot menus.

write_all_system_files(*system*, *menu_items*)
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.

Parameters

- **system** – The system to generate files for.
- **menu_items** – TODO

write_pxe_file(*filename*, *system*, *profile*, *distro*, *arch: cobbler.enums.Archs*, *image=None*, *metadata=None*, *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.
- **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*, *write_file: bool = False*, *path=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.

9.22 cobbler.utils module

Misc heavy lifting functions for Cobbler

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.utils.**DHCP**
Bases: enum.Enum

An enumeration.

V4 = (4,)

V6 = 6

class cobbler.utils.**MntEntObj**(*input: str = None*)
Bases: object

mnt_dir = None

mnt_freq = 0

mnt_fsname = None

mnt_opts = None

mnt_passno = 0

mnt_type = None

cobbler.utils.**blender**(*api_handle*, *remove_dicts: bool*, *root_obj*)
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.**cachefile**(*src: str*, *dst: str*)
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.**cheetah_exc**(*exc*) → 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.**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.**copyfile**(*src: str*, *dst: str*)
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.

Raises **OSError** – Raised in case `src` could not be read.

cobbler.utils.**copyfile_pattern**(*pattern*, *dst*, *require_match: bool = True*, *symlink_ok: bool = False*, *cache: bool = True*, *api=None*)
Copy 1 or more files with a pattern into a destination.

Parameters

- **pattern** – The pattern for finding the required files.
- **dst** – The destination for the file(s) found.
- **require_match** – If the glob pattern does not find files should an error message be thrown or not.
- **symlink_ok** – If it is okay to just use a symlink to link the file to the destination.
- **cache** – If it is okay to use a file from the cache (which could be possibly newer) or not.
- **api** – Passed to `linkfile()`.

Raises ***CX*** – Raised in case files not found according to `pattern`.

cobbler.utils.**copyremotefile**(*src: str*, *dst1: str*, *api=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.**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: cobbler.utils.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*)
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*, *subkey: str*)
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*) → Union[str, dict]
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*)
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*) → 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.**find_distro_path**(*settings*, *distro*)
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.

Parameters

- **settings** – The settings to resolve user configurable actions with.
- **distro** – The distribution to find the path of.

Returns The path to the distribution files.

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
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*) → Optional[dict]
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*, *full: bool = True*)
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_file_device_path**(*fname*)

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.**get_host_ip**(*ip*, *shorten=True*) → str
Return the IP encoding needed for the TFTP boot tree.

Parameters

- **ip** – 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_mtab**(*mtab='/etc/mtab'*, *vfstype: bool = False*) → list
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.**get_random_mac**(*api_handle*, *virt_type='xenpv'*) → 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_distro_boot_loaders**(*distro*, *api_handle=None*)
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.**get_supported_system_boot_loaders**() → List[str]
Return the list of currently supported bootloaders.

Returns The list of currently supported bootloaders.

cobbler.utils.**get_valid_archs**()
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.**get_valid_breeds**() → list
Return a list of valid breeds found in the import signatures

cobbler.utils.**get_valid_os_versions**() → list
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.**get_valid_os_versions_for_breed**(*breed*) → list
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.**grab_tree**(*api_handle*, *item*) → list
Climb the tree and get every node.

Parameters

- **api_handle** – The api to use for checking the tree.
- **item** – The item to check for parents

Returns The list of items with all parents from that object upwards the tree. Contains at least the item itself.

cobbler.utils.**hashfile**(*fn*, *lcache=None*)
Returns the sha1sum of the file

Parameters

- **fn** – 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.**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_string_or_dict**(*options: Union[str, list, dict], allow_multiples=True*)
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 tuple of True and a dict.

Raises **TypeError** – Raised in case the input type is wrong.

cobbler.utils.**input_string_or_list**(*options: Union[str, list]*) → Union[list, 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** – Raised in case the input type is wrong.

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_remote_file**(*file*) → 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.**is_safe_to_hardlink**(*src: str*, *dst: str*, *api*) → 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.**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.**is_systemd**() → bool
Return whether or not this system uses systemd.

This method currently checks if the path `/usr/lib/systemd/systemd` exists.

cobbler.utils.**kopts_overwrite**(*kopts: dict*, *cobbler_server_hostname: str = ''*, *distro_breed: str = ''*, *system_name: str = ''*)
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.**link_distro**(*settings*, *distro*)
Link a Cobbler distro from its source into the web directory to make it reachable from the outside.

Parameters

- **settings** – The settings to resolve user configurable actions with.
- **distro** – The distro to link into the Cobbler web directory.

cobbler.utils.**linkfile**(*src: str*, *dst: str*, *symlink_ok: bool = False*, *cache: bool = True*, *api=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

- **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.
- **api** – This parameter is needed to check if a file can be hardlinked. This method fails if this parameter is not present.

Raises ***CX*** – Raised in case the API is not given.

cobbler.utils.**load_signatures**(*filename*, *cache: bool = True*)
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.**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*, *indexkey*) → list
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*, *indexkey*) → dict
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**()
Log an exception.

cobbler.utils.**mkdir**(*path*, *mode=493*)
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 error code 17.

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**()
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.**path_tail**(*apath*, *bpath*) → 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.**pretty_hex**(*ip*, *length=8*) → str
Pads an IP object with leading zeroes so that the result is _length_ hex digits. Also do an upper().

Parameters

- **ip** – 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*, *fetch_if_remote=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*) → 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*)
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*)
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.**rmfile**(*path: str*)
Delete a single file.

Parameters **path** – The file to delete.

cobbler.utils.**rmglob_files**(*path: str*, *glob_pattern: str*)
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.**rmtree**(*path: str*) → Optional[bool]
Delete a complete directory or just a single file.

Parameters **path** – The directory or folder to delete.

Returns May possibly return true on success or may return None on success.

Raises ***CX*** – Raised in case path does not exist.

cobbler.utils.**rmtree_contents**(*path: str*)
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.**rsync_files**(*src: str*, *dst: str*, *args: str*, *quiet: bool = True*)
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_this**(*cmd: str*, *args: str*)
A simple wrapper around subprocess calls.

Parameters

- **cmd** – The command to run in a shell process.
- **args** – The arguments to attach to the command.

cobbler.utils.**run_triggers**(*api*, *ref*, *globber*, *additional: list = 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 argumenets.
- **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.**safe_filter**(*var*)
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.`**`strip_none`**(*data*, *omit_none: bool = False*)
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*, *shell: bool = True*, *input=None*)
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.
- **`input`** – If there is any input needed for that command to stdin.

Returns The return code of the process

`cobbler.utils.`**`subprocess_get`**(*cmd*, *shell: bool = True*, *input=None*)
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.
- **`input`** – If there is any input needed for that command to stdin.

Returns The data which the subprocess returns.

`cobbler.utils.`**`subprocess_sp`**(*cmd*, *shell: bool = True*, *input=None*)
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.
- **`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*) → list
Remove duplicates from the sequence handed over in the args.

Parameters **`seq`** – The sequence to check for duplicates.

Returns The list without duplicates.

9.23 cobbler.validate module

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

`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=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], for_item: bool = True*) → Union[str, list]
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], for_item: bool = True*) → Union[str, list]
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_arch**(*arch: Union[str, cobbler.enums.Archs]*) → cobbler.enums.Archs
This is a validator for system architectures. If the arch is not valid then an exception is raised.

Parameters **arch** – The desired architecture to set for the object.

Raises

- **TypeError** – In case the any type other then str or enums.Archs was supplied.
- **ValueError** – In case the supplied str could not be converted.

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_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: list*, *api*, *bypass_check: bool = False*)
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, cobbler.enums.BaudRates]*) → cobbler.enums.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_virt_auto_boot**(*value: bool*) → bool
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_disk_driver**(*driver: Union[cobbler.enums.VirtDiskDrivers, str]*)
For Virt only. Specifies the on-disk format for the virtualized disk

Parameters **driver** – The virt driver to set.

cobbler.validate.**validate_virt_file_size**(*num: Union[str, int, 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*)
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.

cobbler.validate.**validate_virt_type**(*vtype: Union[cobbler.enums.VirtType, str]*)
Virtualization preference, can be overridden by koan.

Parameters **vtype** – May be one of "qemu", "kvm", "xenpv", "xenfv", "vmware", "vmwarew", "openvz" or "auto"

9.24 cobbler.yumgen module

Builds out filesystem trees/data based on the object tree. This is the code behind 'cobbler sync'.

Copyright 2006-2009, Red Hat, Inc and Others Michael DeHaan <michael.dehaan AT gmail>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

class cobbler.yumgen.**YumGen**(*collection_mgr*)
Bases: object

get_yum_config(*obj*, *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.

9.25 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 10

Release Notes for Cobbler

The release notes can be found on GitHub.

CHAPTER 11

Limitations and Surprises

11.1 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]`.

CHAPTER 12

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