

# Unify

## A chroot package building environment for Pkgcore

Luis Francisco Araujo

[<araujo@gentoo.org>](mailto:araujo@gentoo.org)

08 - 21 - 2008

Unify is a python application that builds a specific package-dependent format chroot environment to be used as a host system for building packages inside it using the Gentoo Pkgcore manager.

Unify is originally intended to work primarily with RPM and DPKG packaging formats; but it is designed in such a scalable way that can easily incorporate support for many more packaging formats.

Each packaging format to be included into the Unify framework requires:

- A native package manager available for such a format.
- A well defined way of bootstrapping an entire system using said format.

### How Does Unify work?

Unify has been designed using python, so, many of the implementation decisions have been influenced by such a language (and I hope that is it for the better).

The Unify approach is based on a module organization with the following hierarchy:

`unify.chroot.Root()` : This class is the entry point for all the Unify operations. It is the root class for Unify and implements the most general and basic steps for the creation of any chroot environment.

This class contains an important method called `unify.chroot.Root.init()` that will run the execution thread for the initial Unify phase.

This class will return a 'Root' object containing all the necessary information for further Unify phases. For example, this object contains the path for the Unify chroot building location at `/var/lib/unify` and so on. Therefore, this will **always** be the first method to be executed before any other Unify phase.

In other words, this object will contain and take care of properly initializing all the necessary and common information shared by the rest of all the phases. This object will also ensure the type of API that will follow future implemented packaging formats.

Next in the hierarchy, we have the packaging format dependent classes and objects located inside the `unify.pm` module. We currently have implemented:

**`unify.pm.dpkg.Dpkg()`**: This is the `dpkg` class implementing all the necessary steps for the proper initialization and building of a `Dpkg`-based environment.

**`unify.pm.rpm.Rpm()`**: This is the `rpm` class implementing all the necessary steps for the initialization and building of a `Rpm`-based environment.

These classes have to receive as a first parameter a `unify.chroot.Root` object ; so, any further implemented packaging format classes and objects need to follow this standard. This way we ensure a safe and modular scalability between different packaging formats. All these package dependent modules

also need to implement a Pkgcore class. The one currently implemented is located at:

**unify.pm.dpkg.Pkgcore()**: This class will create a pkgcore sub-environment specially and functionally suitable for a dpkg system.

This can be considered like the lower class in the hierarchy, but also one of the most important for Unify, since this class will require to setup and tweak the final environment for pkgcore, including the installation of it inside the chroot.

This class setups all the necessary files and tools for installing and getting working pkgcore inside the given chroot. This class is also responsible of implementing the instance method to build the final package; in this case, the .deb package for the unify.pm.dpkg module.

Any further implemented packaging format inside Unify needs to follow this standard, and adds its own suitable Pkgcore() class routine too, since setting up this environment can pretty much vary from one chroot system to another.

### The Unify Pkgcore Spec File Translator

Located inside the unify.specft module ; this is such an important part of Unify that requires its own section.

This module contains all the classes responsible of translating the package information from the Gentoo <category/package> atom format to the respective specification file (control file for dpkg, and spec file for rpm).

This module takes care of two things mainly:

- User friendly interface to retrieve information from the portage tree using the pkgcore API.
- Translating this information from the pkgcore API to the respective package format specification file.

This module also has incorporated a stand-alone interface; so it can serve as a whole tool to get translated information from pkgcore into either (currently implemented) rpm spec files or dpkg control files.

For example:

```
$ python /usr/lib/python/site-packages/unify/specft.py -r app-arch/arc > arc.spec
```

This would get a suitable rpm spec file for the package 'arc' from the portage tree using pkgcore into the file 'arc.spec'.

The unify.specft module is composed of:

The main **unify.specft.EbuildSpec()** class: Most of the methods implemented inside this class maps directly to a pkgcore API method, so we can see it as a higher level interface suitable for invoking pkgcore capabilities inside Unify.

This class also offers a common framework for accessing pkgcore since EbuildSpec() is intended to be the super-class for any format translator class.

Any new packaging format translator needs to be implemented on its own class inside this unify.specft module and as sub-class of EbuildSpec().

We currently have implemented the following format translator classes inside this module:

**unify.specft.RpmSpec()**: Implements the methods for translating the package information from Pkgcore to the Rpm format. It is a sub-class of EbuildSpec().

**Unify.specft.DebSpec()**: Implements the methods for translating the package information from Pkgcore to the Deb format. It is a sub-class of EbuildSpec().

## Unify Current State

Unify is an application still under development with really ambitious goals, which currently support the primary features in a very early stage of functioning.

Unify follows an incremental approach for its interface; and the current implemented functionalities by the time of this report includes:

### **Init**

Usage mode: *-i , --init*

What is it?

Generic chroot initialization.

What does it do?

This phase involves the initialization of the most basic variables for the further correct functioning of all the rest of the phases. Variables like the location of the Unify chroot, the general directories needed inside any of the rest of the phases, and the directory where the packages built will be placed are all initialized through this option.

During this phase, also a basic chroot environment will be created. Directories like */proc*, and files like */etc/resolv.conf* are created too.

Status: *init*

### **Init-Dpkg**

Usage mode: *-d, --init-dpkg=[cfg]*

What is it?

Dpkg chroot environment initialization.

What does it does?

This option builds a dpkg based system using the debootstrap tool. It reads all the necessary information required for the creation of this environment from the *cfg* file.

After this phase, a self-contained Dpkg system should already be successfully setup to be used for building packages. This phase also involves mounting the necessary virtual file-systems like *proc*, and *sys*; and also the creation of the unify user inside of said chroot.

Status: *init-dpkg*

Note: In order to be executed, this option requires a previous execution of the 'init' phase.

## **Init-Rpm**

Usage mode: *-r, --init-rpm=[cfg]*

What is it?

Rpm chroot environment initialization.

What does it do?

This option builds a rpm based system using the yum package manager. It also reads all the required information for building this environment from the *cfg* file.

This phase does exactly the same than the 'Init-Dpkg' option, but for a Rpm system. After this phase is run, a self-contained Rpm chroot system should be successfully setup and ready for building rpm packages inside it. During this phase, all the necessary virtual file-systems are mounted and the unify user is created.

Status: *init-rpm*

Note: In order to be executed, this option requires a previous execution of the 'init' phase.

## **Build-Deb**

Usage mode: *-k, --build-deb=[cfg] <category/package>*

What is it?

This option builds a .deb package from the *<category/package>* atom using pkgcore inside the initialized Dpkg environment.

What does it do?

This option will execute the 'init' and 'init-dpkg' phases and will build a pkgcore sub-environment inside the Dpkg chroot.

Unify builds the package identified with the *<category/package>* atom from the portage tree using pkgcore inside the Dpkg chroot and will package it as .deb using dpkg.

This phase outputs a perfectly usable .deb package inside the */builddir* directory of the chroot.

Status: *pkgcore-dpkg*

## **Clean**

Usage mode: *-c, --clean*

What is it?

It cleans the current chroot location.

What does it do?

It removes all the files and unmount all the necessary file-systems from the default Unify chroot location at */var/lib/unify/root*. It preserves the 'status.log' file.

Status: *clean*

## **Show-Status**

Usage mode: *--show-status*

What is it?

It shows the current status for the Unify chroot.

What does it do?

Since Unify follows an incremental approach for all its operations, we need a way to verify what stage we are at a given moment. This is the option for checking that.

Unify saves the stage information inside */var/lib/unify/status.log* as a default location, and such a file will always be preserved even when we clean out the chroot.

## **Help**

Usage mode: *-h*

What is it?

It shows the main Unify help menu listing all the available options.

## Not-Implemented-Yet

- The build-rpm option , which should work similar to build-deb but for rpm is not yet fully implemented.
- The interpretation of dependencies between different packaging formats is not yet implemented.

---

This project has been developed for the Oregon State University Open Source Laboratory during the Google Summer of Code 2008.