NAME

Dpkg::Deps − parse and manipulate dependencies of Debian packages

DESCRIPTION

The Dpkg::Deps module provides classes implementing various types of dependencies.

The most important function is **deps_parse()**, it turns a dependency line in a set of Dpkg::Deps::{Simple,AND,OR,Union} objects depending on the case.

FUNCTIONS

All the deps_* functions are exported by default.

deps_eval_implication($rel_p, `$v_p`, `$rel_q`, `$v_q`)

($rel_p, `$v_p`) and ($rel_q, `$v_q`) express two dependencies as (relation, version). The relation variable can have the following values that are exported by Dpkg::Version: REL_EQ, REL_LT, REL_LE, REL_GT, REL_GT.

This functions returns 1 if the ‘‘p’’ dependency implies the ‘‘q’’ dependency. It returns 0 if the ‘‘p’’ dependency implies that ‘‘q’’ is not satisfied. It returns undef when there’s no implication.

The `$v_p` and `$v_q` parameter should be Dpkg::Version objects.

`$dep` = deps_concat(@dep_list)

This function concatenates multiple dependency lines into a single line, joining them with ‘‘, ’’ if appropriate, and always returning a valid string.

`$dep` = deps_parse($line, `%options`)

This function parses the dependency line and returns an object, either a Dpkg::Deps::AND or a Dpkg::Deps::Union. Various options can alter the behaviour of that function.

use_arch (defaults to 1)

Take into account the architecture restriction part of the dependencies. Set to 0 to completely ignore that information.

host_arch (defaults to the current architecture)

Define the host architecture. By default it uses **Dpkg::Arch::get_host_arch()** to identify the proper architecture.

build_arch (defaults to the current architecture)

Define the build architecture. By default it uses **Dpkg::Arch::get_build_arch()** to identify the proper architecture.

reduce_arch (defaults to 0)

If set to 1, ignore dependencies that do not concern the current host architecture. This implicitly strips off the architecture restriction list so that the resulting dependencies are directly applicable to the current architecture.

use_profiles (defaults to 1)

Take into account the profile restriction part of the dependencies. Set to 0 to completely ignore that information.

build_profiles (defaults to no profile)

Define the active build profiles. By default no profile is defined.

reduce_profiles (defaults to 0)

If set to 1, ignore dependencies that do not concern the current build profile. This implicitly strips off the profile restriction formula so that the resulting dependencies are directly applicable to the current profiles.

reduce_restrictions (defaults to 0)

If set to 1, ignore dependencies that do not concern the current set of restrictions. This implicitly strips off any architecture restriction list or restriction formula so that the resulting dependencies are directly applicable to the current restriction. This currently implies `reduce_arch` and `reduce_profiles`, and overrides them if set.

union (defaults to 0)
: If set to 1, returns a Dpkg::Deps::Union instead of a Dpkg::Deps::AND. Use this when parsing non-dependency fields like Conflicts.

virtual (defaults to 0)
: If set to 1, allow only virtual package version relations, that is none, or "=". This should be set whenever working with Provides fields.

build_dep (defaults to 0)
: If set to 1, allow build-dep only arch qualifiers, that is ":native". This should be set whenever working with build-deps.

tests_dep (defaults to 0)
: If set to 1, allow tests-specific package names in dependencies, that is "@" and "@builddeps@" (since dpkg 1.18.7). This should be set whenever working with dependency fields from *debian/tests/control*.

`$bool` = deps_iterate($deps, `$callback_func`)

This function visits all elements of the dependency object, calling the callback function for each element.

The callback function is expected to return true when everything is fine, or false if something went wrong, in which case the iteration will stop.

Return the same value as the callback function.

deps_compare($a, `$b`)

Implements a comparison operator between two dependency objects. This function is mainly used to implement the **sort()** method.

CLASSES − Dpkg::Deps::*

There are several kind of dependencies. A Dpkg::Deps::Simple dependency represents a single dependency statement (it relates to one package only). Dpkg::Deps::Multiple dependencies are built on top of this class and combine several dependencies in different manners. Dpkg::Deps::AND represents the logical "AND" between dependencies while Dpkg::Deps::OR represents the logical "OR". Dpkg::Deps::Multiple objects can contain Dpkg::Deps::Simple object as well as other Dpkg::Deps::Multiple objects.

In practice, the code is only meant to handle the realistic cases which, given Debian's dependencies structure, imply those restrictions: AND can contain Simple or OR objects, OR can only contain Simple objects.

Dpkg::Deps::KnownFacts is a special class that is used while evaluating dependencies and while trying to simplify them. It represents a set of installed packages along with the virtual packages that they might provide.

CHANGES

Version 1.07 (dpkg 1.20.0)

New option: Add virtual option to **Dpkg::Deps::deps_parse()**.

Version 1.06 (dpkg 1.18.7; module version bumped on dpkg 1.18.24)

New option: Add tests_dep option to **Dpkg::Deps::deps_parse()**.

Version 1.05 (dpkg 1.17.14)

New function: **Dpkg::Deps::deps_iterate()**.

Version 1.04 (dpkg 1.17.10)

New options: Add use_profiles, build_profiles, reduce_profiles and reduce_restrictions to **Dpkg::Deps::deps_parse()**.

Version 1.03 (dpkg 1.17.0)

New option: Add build_arch option to **Dpkg::Deps::deps_parse()**.

Version 1.02 (dpkg 1.17.0)
New function: **Dpkg::Deps::deps_concat()**

Version 1.01 (dpkg 1.16.1)
<Used to document changes to Dpkg::Deps::* modules before they were split.>

Version 1.00 (dpkg 1.15.6)
Mark the module as public.