NAME

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

DESCRIPTION

The Dpkg::Deps module provides objects 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.

my `$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.

my `$dep` = deps_parse($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 implicitely 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.

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.

my `$bool` = deps_iterate($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.

OBJECTS – 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 object and combine several dependencies in a 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 object 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.

COMMON FUNCTIONS

`$dep`->*is_empty()*
: Returns true if the dependency is empty and doesn't contain any useful information. This is true when a Dpkg::Deps::Simple object has not yet been initialized or when a (descendant of) Dpkg::Deps::Multiple contains an empty list of dependencies.

`$dep`->*get_deps()*
: Returns a list of sub-dependencies. For Dpkg::Deps::Simple it returns itself.

`$dep`->output([$fh])
"$dep"
: Returns a string representing the dependency. If `$fh` is set, it prints the string to the filehandle.

`$dep`->implies($other_dep)
: Returns 1 when `$dep` implies `$other_dep`. Returns 0 when `$dep` implies NOT($other_dep). Returns undef when there's no implication. `$dep` and `$other_dep` do not need to be of the same type.

`$dep`->*sort()*
: Sorts alphabetically the internal list of dependencies. It's a no-op for Dpkg::Deps::Simple objects.

`$dep`->arch_is_concerned($arch)
: Returns true if the dependency applies to the indicated architecture. For multiple dependencies, it returns true if at least one of the sub-dependencies apply to this architecture.

`$dep`->reduce_arch($arch)
: Simplifies the dependency to contain only information relevant to the given architecture. A Dpkg::Deps::Simple object can be left empty after this operation. For Dpkg::Deps::Multiple objects, the non-relevant sub-dependencies are simply removed.

  This trims off the architecture restriction list of Dpkg::Deps::Simple objects.

`$dep`->get_evaluation($facts)
: Evaluates the dependency given a list of installed packages and a list of virtual packages provided. Those lists are part of the Dpkg::Deps::KnownFacts object given as parameters.

  Returns 1 when it's true, 0 when it's false, undef when some information is lacking to conclude.

`$dep`->simplify_deps($facts, `@assumed_deps`)
: Simplifies the dependency as much as possible given the list of facts (see object Dpkg::Deps::KnownFacts) and a list of other dependencies that are known to be true.

`$dep`->*has_arch_restriction()*
: For a simple dependency, returns the package name if the dependency applies only to a subset of architectures. For multiple dependencies, it returns the list of package names that have such a restriction.

`$dep`->*reset()*
: Clears any dependency information stored in `$dep` so that `$dep`->*is_empty()* returns true.

Dpkg::Deps::Simple

Such an object has four interesting properties:

package
: The package name (can be undef if the dependency has not been initialized or if the simplification of the dependency lead to its removal).

relation
: The relational operator: ‘‘=’’, ‘‘<<’’, ‘‘<=’’, ‘‘>=’’ or ‘‘>>’’. It can be undefined if the dependency had no version restriction. In that case the following field is also undefined.

version
: The version.

arches
: The list of architectures where this dependency is applicable. It's undefined when there's no restriction, otherwise it's an array ref. It can contain an exclusion list, in that case each architecture is prefixed with an exclamation mark.

archqual
: The arch qualifier of the dependency (can be undef if there's none). In the dependency ‘‘python:any (>= 2.6)’’, the arch qualifier is ‘‘any’’.

METHODS

`$simple_dep`->parse_string('dpkg-dev (>= 1.14.8) [!hurd-i386]')
: Parses the dependency and modifies internal properties to match the parsed dependency.

`$simple_dep`->merge_union($other_dep)
: Returns true if `$simple_dep` could be modified to represent the union of both dependencies. Otherwise returns false.

Dpkg::Deps::Multiple
This is the base class for Dpkg::Deps::{AND,OR,Union}. It implements the following methods:

`$mul`->add($dep)
Adds a new dependency object at the end of the list.

Dpkg::Deps::AND
This object represents a list of dependencies who must be met at the same time.

`$and`->output([$fh])
The output method uses ", " to join the list of sub-dependencies.

Dpkg::Deps::OR
This object represents a list of dependencies of which only one must be met for the dependency to be true.

`$or`->output([$fh])
The output method uses " | " to join the list of sub-dependencies.

Dpkg::Deps::Union
This object represents a list of relationships.

`$union`->output([$fh])
The output method uses ", " to join the list of relationships.

`$union`->implies($other_dep)
`$union`->get_evaluation($other_dep)
Those methods are not meaningful for this object and always return undef.

`$union`->simplify_deps($facts)
The simplication is done to generate an union of all the relationships. It uses `$simple_dep`->merge_union($other_dep) to get its job done.

Dpkg::Deps::KnownFacts
This object represents a list of installed packages and a list of virtual packages provided (by the set of installed packages).

my `$facts` = Dpkg::Deps::KnownFacts->*new()*;
Creates a new object.

`$facts`->add_installed_package($package, `$version`, `$arch`, `$multiarch`)
Records that the given version of the package is installed. If `$version`/$arch is undefined we know that the package is installed but we don't know which version/architecture it is. `$multiarch` is the Multi-Arch field of the package. If `$multiarch` is undef, it will be equivalent to "Multi-Arch: no".

Note that `$multiarch` is only used if `$arch` is provided.

`$facts`->add_provided_package($virtual, `$relation`, `$version`, `$by`)
Records that the "$by" package provides the `$virtual` package. `$relation` and `$version` correspond to the associated relation given in the Provides field (if present).

my ($check, `$param`) = `$facts`->check_package($package)
`$check` is one when the package is found. For a real package, `$param` contains the version. For a virtual package, `$param` contains an array reference containing the list of packages that provide it (each package is listed as [`$provider`, `$relation`, `$version`]).

This function is obsolete and should not be used. Dpkg::Deps::KnownFacts is only meant to be filled with data and then passed to Dpkg::Deps methods where appropriate, but it should not be directly queried.

CHANGES

Version 1.05
New function: *Dpkg::Deps::deps_iterate()*.

Version 1.04
New options: Add use_profiles, build_profiles, reduce_profiles and reduce_restrictions to *Dpkg::Deps::deps_parse()*.

New methods: Add `$dep->`*profile_is_concerned()* and `$dep->`*reduce_profiles()* for all dependency objects.

Version 1.03
New option: Add build_arch option to *Dpkg::Deps::deps_parse()*.

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

Version 1.01
New method: Add `$dep->`*reset()* for all dependency objects.

New property: Dpkg::Deps::Simple now recognizes the arch qualifier ‘‘any’’ and stores it in the ‘‘archqual’’ property when present.

New option: Dpkg::Deps::KnownFacts->*add_installed_package()* now accepts 2 supplementary parameters ($arch and `$multiarch`).

Deprecated method: Dpkg::Deps::KnownFacts->*check_package()* is obsolete, it should not have been part of the public API.

Version 1.00
Mark the module as public.