NAME

Lintian::Collect::Binary – Lintian interface to binary package data collection

SYNOPSIS

```
my ($name, $type, $dir) = ('foobar', 'binary', '/path/to/lab-entry');
my $collect = Lintian::Collect->new ($name, $type, $dir);
if ($collect->native) {
    print "Package is native\n";
}
```

DESCRIPTION

Lintian::Collect::Binary provides an interface to package data for binary packages. It implements data collection methods specific to binary packages.

This module is in its infancy. Most of Lintian still reads all data from files in the laboratory whenever that data is needed and generates that data via collect scripts. The goal is to eventually access all data about binary packages via this module so that the module can cache data where appropriate and possibly retire collect scripts in favor of caching that data in memory.

CLASS METHODS

new (PACKAGE)

Creates a new Lintian::Collect::Binary object. Currently, PACKAGE is ignored. Normally, this method should not be called directly, only via the Lintian::Collect constructor.

INSTANCE METHODS

In addition to the instance methods listed below, all instance methods documented in the Lintian::Collect and the Lintian::Collect::Package modules are also available.

native

Returns true if the binary package is native and false otherwise. Nativeness will be judged by the source version number.

If the version number is absent, this will return false (as native packages are a lot rarer than non-native ones).

Needs-Info requirements for using *native*: Same as field

changelog

Returns the changelog of the binary package as a Parse::DebianChangelog object, or undef if the changelog doesn't exist. The changelog-file collection script must have been run to create the changelog file, which this method expects to find in *changelog*.

Needs-Info requirements for using *changelog*: changelog-file

control ([FILE])

This method is deprecated. Consider using "control_index_resolved_path(PATH)" instead, which returns Lintian::Path objects.

Returns the path to FILE in the control.tar.gz. FILE must be either a Lintian::Path object (>= 2.5.13~) or a string denoting the requested path. In the latter case, the path must be relative to the root of the control.tar.gz member and should be normalized.

It is not permitted for FILE to be `undef`. If the "root" dir is desired either invoke this method without any arguments at all, pass it the correct Lintian::Path or the empty string.

To get a list of entries in the control.tar.gz or the file meta data of the entries (as path objects), see "sorted_control_index" and "control_index (FILE)".

The caveats of unpacked also apply to this method. However, as the control.tar.gz is not known to contain symlinks, a simple file type check is usually enough.

Needs-Info requirements for using *control*: bin-pkg-control

control_index (FILE)

Returns a path object to FILE in the control.tar.gz. FILE must be relative to the root of the control.tar.gz and must be without leading slash (or "./"). If FILE is not in the control.tar.gz, it returns `undef`.

To get a list of entries in the control.tar.gz, see "sorted_control_index". To actually access the underlying file (e.g. the contents), use "control ([FILE])".

Note that the "root directory" (denoted by the empty string) will always be present, even if the underlying tarball omits it.

Needs-Info requirements for using *control_index*: bin-pkg-control

sorted_control_index

Returns a sorted array of file names listed in the control.tar.gz. The names will not have a leading slash (or "./") and can be passed to "control ([FILE])" or "control_index (FILE)" as is.

The array will not contain the entry for the "root" of the control.tar.gz.

Needs-Info requirements for using *sorted_control_index*: Same as control_index

control_index_resolved_path(PATH)

Resolve PATH (relative to the root of the package) and return the entry denoting the resolved path.

The resolution is done using resolve_path.

Needs-Info requirements for using *control_index_resolved_path*: Same as control_index

strings (FILE)

Returns an open handle, which will read the data from coll/strings for FILE. If coll/strings did not collect any strings about FILE, this returns an open read handle with no content.

Caller is responsible for closing the handle either way.

Needs-Info requirements for using *strings*: strings

scripts

Returns a hashref mapping a FILE to its script/interpreter information (if FILE is a script). If FILE is not a script, it is not in the hash (and callers should use exists to test membership to ensure this invariant holds).

The value for a given FILE consists of a table with the following keys (and associated value):

calls_env

Returns a truth value if the script uses env (/usr/bin/env or /bin/env) in the "#!". Otherwise it is `undef`.

interpreter

This is the interpreter used. If calls_env is true, this will be the first argument to env. Otherwise it will be the command listed after the "#!".

NB: Some template files have "#!" lines like "#!@PERL@" or "#!perl". In this case, this value will be `@PERL@` or perl (respectively).

name

Return the file name of the script. This will be identical to key to look up this table.

Needs-Info requirements for using *scripts*: scripts

objdump_info

Returns a hashref mapping a FILE to the data collected by objdump-info or `undef` if no data is available for that FILE. Data is generally only collected for ELF files.

Needs-Info requirements for using *objdump_info*: objdump-info hardening_info

Returns a hashref mapping a FILE to its hardening issues.

NB: This is generally only useful for checks/binaries to emit the hardening–no–* tags.

Needs-Info requirements for using *hardening_info*: hardening-info

java_info

Returns a hashref containing information about JAR files found in binary packages, in the form *file name* –> *info*, where *info* is a hash containing the following keys:

manifest

A hash containing the contents of the JAR file manifest. For instance, to find the classpath of `$file`, you could use:

```
 if (exists $info->java_info->{$file}{'manifest'}) {
     my $cp = $info->java_info->{$file}{'manifest'}{'Class-Path'};
     # ...
 }
```

NB: Not all jar files have a manifest. For those without, this will value will not be available. Use exists (rather than defined) to check for it.

files

A table of the files in the JAR. Each key is a file name and its value is its "Major class version" for Java or "–" if it is not a class file.

error

If it exists, this is an error that occurred during reading of the zip file. If it exists, it is unlikely that the other fields will be present.

Needs-Info requirements for using *java_info*: java-info

relation (FIELD)

Returns a Lintian::Relation object for the specified FIELD, which should be one of the possible relationship fields of a Debian package or one of the following special values:

all The concatenation of Pre-Depends, Depends, Recommends, and Suggests.

strong

The concatenation of Pre-Depends and Depends.

weak

The concatenation of Recommends and Suggests.

If FIELD isn't present in the package, the returned Lintian::Relation object will be empty (always satisfied and implies nothing).

Needs-Info requirements for using *relation*: Same as field

is_pkg_class ([TYPE])

Returns a truth value if the package is the given TYPE of special package. TYPE can be one of "transitional", "debug" or "any-meta". If omitted it defaults to "any-meta". The semantics for these values are:

transitional

The package is (probably) a transitional package (e.g. it is probably empty, just depend on stuff will eventually disappear.)

Guessed from package description.

any-meta

This package is (probably) some kind of meta or task package. A meta package is usually empty and just depend on stuff. It will also return a truth value for "tasks" (i.e. tasksel "tasks").

A transitional package will also match this.

Guessed from package description, section or package name.

debug

The package is (probably) a package containing debug symbols.

Guessed from the package name.

auto-generated

The package is (probably) a package generated automatically (e.g. a dbgsym package)

Guessed from the "Auto-Built-Package" field.

Needs-Info requirements for using *is_pkg_class*: Same as field

is_conffile (FILE)

Returns a truth value if FILE is listed in the conffiles control file. If the control file is not present or FILE is not listed in it, it returns `undef`.

Note that FILE should be the filename relative to the package root (even though the control file uses absolute paths). If the control file does relative paths, they are assumed to be relative to the package root as well (and used without warning).

Needs-Info requirements for using *is_conffile*: Same as control_index_resolved_path

AUTHOR

Originally written by Frank Lichtenheld <djpig@debian.org> for Lintian.

SEE ALSO

lintian (1), Lintian::Collect, Lintian::Relation