RepoLib Documentation

Ian Santopietro

Oct 03, 2022

Developer Documentation

RepoLib is a Python library and CLI tool-set for managing your software system software repositories. It's currently set up to handle APT repositories on Debian-based linux distributions.

RepoLib is intended to operate on DEB822-format sources as well as legacy one-line format sources. It aims to provide feature parity with software-properties for most commonly used functions. It also allows for simple, automated conversion from legacy "one-line" style source to newer DEB822 format sources. These sources will eventually deprecate the older one-line sources and are much easier to parse for both human and machine. For a detailed explanation of the DEB822 Source format, see *Explanation of the DEB822 Source Format*.

RepoLib provides much faster access to a subset of `SoftwareProperties` features than `SoftwareProperties` itself does. Its scope is somewhat more limited because of this, but the performance improvements gained are substantial. `SoftwareProperties` also does not yet feature support for managing DEB822 format sources, and instead only manages one-line sources.

RepoLib is available under the GNU LGPL.

CHAPTER 1

RepoLib

The repolib-module module simplifies working with APT sources, especially those stored in the DEB822 format. It translates these sources into Python Objects for easy reading, parsing, and manipulation of them within Python programs. The program takes a user-defined sources filename and reads the contents, parsing it into a Python object which can then be iterated upon and which uses native data types where practicable. The repolib-module module also allows easy creation of new DEB822-style sources from user-supplied data.

1.1 Source object

The Source object is the base class for all of the sources used within RepoLib. The Source class itself is a subclass of the deb822() class from the Python Debian module, which provides a dict-like interface for setting data as well as several methods for dumping data to files and other helpful functions.

class repolib.Source (file=None) Create a new *Source object*.

The Source object has the following attributes:

- *ident* - The system-identifier to use for this source.
- *name* - The human-readable name of the source. (default: '')
- *enabled* - Whether the source is enabled or not at creation. (default: True)
- *types* - A list of the types that the source should use. (default: [])
- *uris* - A list of URIs from which to fetch software or check for updates. (default: [])
- *suites* - Suites in which to narrow available software. (default: [])
- *components* - Components of the source repository to enable. (default: [])
- signed_by - The path to the signing key for this source
- file - The file_object for this source
- key - The key_object for this source.
- twin_source: - This source should be saved with both binary and source code types enabled.

The following decribe how each of these are used.

1.1.1 ident

The `ident` is the system-identifier for this source. This determines the filename the source will use on-disk as well as the way to specify a source to load.

1.1.2 name

This is a human-readable and nicely-formatted name to help a user recognize what this source is. Any unicode character is allowed in this field. If a source is opened which doesn't have a name field, the filename will be used.

name is a string value, set to `''` by default. If there is no name in a loaded source, it will be set to the same as the filename (minus the extension).

This field maps to the `X-Repolib-Name:` field in the .sources file, which is ignored by Apt and other standards-compliant sources parsers.

1.1.3 enabled

Apt sources can be disbaled without removing them entirely from the system. A disabled source is excluded from updates and new software installs, but it can be easily re-enabled at any time. It defaults to `True`.

This field maps to the `Enabled:` field in the .sources file. This is optional per the DEB822 standard, however if set to anything other than `no`, the source is considered enabled.

1.1.4 types

Debian archives may contain either binary packages or source code packages, and this value specifies which of those Apt should look for in the source. `deb` is used to look for binary packages, while `deb-src` looks for source code packages. RepoLib stores this value as a list of aptsourcetype-enum's, and defaults to ''[AptSourceType.BINARY]'.

This field maps to the `Types:` field in the sources file.

1.1.5 uris

A list of string values describing the URIs from which to fetch package lists and software for updates and installs. The currently recognized URI types are:

- file
- cdrom
- http
- ftp
- copy
- rsh
- ssh

DEB822 sources may directly contain an arbitrary number of URIs. Legacy sources may also have multiple URIs; however, these require writing a new deb line for each URI as the one-line format only allows a single URI per source.

Note: Although these are the currently-recognized official URI types, Apt can be extended with additional URI schemes through extension packages. Thus it is **not** recommended to parse URIs by type and instead rely on user input being correct and to throw exceptions when that isn't the case.

1.1.6 suites

The Suite, also known as the **distribution** specifies a subdirectory of the main archive root in which to look for software. This is typically used to differentiate versions for the same OS, e.g. `disco` or `cosmic` for Ubuntu, or `squeeze` and `unstable` for Debian.

DEB822 Sources allow specifying an arbitrary number of suites. One-line sources also support multiple suites, but require an additional repo line for each as the one-line format only allows a single suite for each source.

This value maps to the `Suites:` field in the sources file.

1.1.7 components

This value is a list of strings describing the enabled distro components to download software from. Common values include `main`, `restricted`, `nonfree`, etc.

1.1.8 signed_by

The path to the keyring containing the signing key used to verify packages downloaded from this repository. This should generally match the key-path attribute for this source's key object.

1.1.9 file

The file_object for the file which contains this source.

1.1.10 key

The key_object for this source.

1.2 Methods

Source.get_description() -> str Returns a `str` containing a UI-compatible description of the source.

1.2.1 reset_values()

Source.reset_values() Initializes the Source's attributes with default data in-order. This is recommended to ensure that the fields in the underlying deb822 Dict are in order and correctly capitalized.

1.2.2 load_from_data()

Source.load_from_data(data: list) -> None Loads configuration information from a list of data, rather than using manual entry. The data can either be a list of strings with DEB822 data, or a single-element list containing a one-line legacy line.

1.2.2.1 data

The data load load into the source. If this contains a legacy-format one-line repository, it must be a single-element list. Otherwise, it should contain a list of strings, each being a line from a DEB822 configuration.

1.2.3 generate_default_ident()

Source.generate_default_ident(prefix: str = '') -> str Generates a suitable default ident, optionally with a prefix, and sets it. The generated ident is also returned for processing convenience.

1.2.3.1 prefix

The prefix to prepend to the ident.

generate_default_name()

Source.generate_default_name() -> Generates a default name for the source and sets it. The generated name is also returned for convenience.

load_key()

Source.load_key(ignore_errors: bool = True) -> None Finds the signing key for this source spefified in signed_by and loads a key_object for it.

1.2.3.2 ignore_errors

If *False*, raise a exc_sourceerror if the key can't be loaded or doesn't exist.

1.2.4 save()

Source.save() Proxy method for the file-save method for this source's file_object.

There are three output properties which contain the current source data for output in a variety of formats.

1.2.5 deb822

Source.deb822 A representation of the source data as a DEB822-formatted string

1.2.6 legacy

Source.legacy A one-line formatted string of the source. It twin_source is `True`, then there will additionally be a `deb-src` line following the primary line.

1.2.7 ui

Source.ui A representation of the source with certain key names translated to better represent their use in a UI for display to a user.

1.3 RepoLib Enums

RepoLib uses a variety of Enums to help ensure that data values are consistent when they need to be set to specific string values for compatibility.

1.3.1 SourceFormat

repolib.SourceFormat() Encodes the two formats of source files, either `.list` for legacy format files or `.sources` for DEB822-formatted files.

- `DEFAULT` - DEB822 formatting (value: `"sources"`)
- `LEGACY` - Legacy, one-line formatting (value: `"list"`)

1.3.2 SourceType

repolib.SourceType() Encodes the type of packages the repository provides (binary or source code).

- `BINARY` - Binary package source type (value: `"deb"`)
- `SOURCECODE` - Source code package type (value : `"deb-src"`)

1.3.3 AptSourceEnabled

repolib.AptSourceEnabled() Used to encode whether the source is enabled or not.

- `TRUE` - The source should be enabled (value: `"yes"`).
- `FALSE` - The source should not be enabled (value: `"no"`).

CHAPTER 2

repolib Module

The repolib Module is the main module for the package. It allows interfacing with the various Classes, Subclasses, and Functions provided by RepoLib.

2.1 Module-level Attributes

There are a couple of module-level attributes and functions provided directly in the module.

2.1.1 VERSION

repolib.VERSION Provides the current version of the library.

2.1.2 LOG_FILE_PATH

repolib.LOG_FILE_PATH Stores the current path to the log file

repolib.LOG_LEVEL Stores the current logging level. Note: Change this level using the module_set_logging_level function.

2.2 Configuration directories

repolib.KEYS_DIR repolib.SOURCES_DIR

Stores the current Pathlib.Path pointing at the signing key and sources directories, respectively. Used for building path names and reading configuration.

2.3 DISTRO_CODENAME

repolib.DISTRO_CODENAME The current CODENAME field from LSB. If LSB is not available, it will default to `linux`.

2.4 CLEAN_CHARS

repolib.CLEAN_CHARS A `dict` which maps invalid characters for the *ident* attributes which cannot be used and their replacements. These limitations are based on invalid characters in unix-compatible filenames.

2.5 sources

repolib.sources A `dict` storing all current sources configured on the system. To save resources, this list is only loaded/parsed when module_load_all_sources is called, since many simple operations don't need the full list of currently-configured sources.

2.6 files

repolib.files A `dict` containing any source file objects present in the configured sources dir (See module_dirs). This list is empty until module_load_all_sources is called, since many simple operations don't need the full list of currently-installed config files.

2.7 keys

repolib.keys A :obj'dict' containing any installed repository signing keys on the system. This list is empty until module_load_all_sources is called, since many simple operations don'tneed the full list of currently-installed keys.

2.7.1 compare_sources()

repolib.compare_sources(source1, source2, excl_keys:list) -> bool Compares two sources based on arbitrary criteria.

This looks at a given list of keys, and if the given keys between the two given sources are identical, returns True.

Returns: `bool` *True* if the sources are identical, otherwise *False*.

2.7.1.1 source1, source2

The two source objects to compare.

2.7.1.2 excl_keys

`list` A list of DEB822key names to ignore when comparing. Even if these items don't match, this function still returns true if all other keys match.

2.7.2 combine_sources()

repolib.combine_sources(source1, source2) -> None Copies all of the data in *source2* and adds it to *source1*.

This avoids duplicating data and ensures that all of both sources' data are present in the unified source

2.7.2.1 source1

The source into which both sources should be merged

2.7.2.2 source2

The source from which to copy to *source1*

2.7.3 url_validator()

repolib.url_validator(url: str) -> bool Validates a given URL and attempts to see if it's a valid Debian respository URL. Returns *True* if the URL appears to be valid and *False* if not.

2.7.3.1 url

:obj:‘str‘The url to validate

2.8 validate_debline()

repolib.util.validate_debline(line: str) -> bool Validate if the provided debline `line` is valid or not.

Returns `True` if `line` is valid, otherwise `False`.

`str` The line to validate

2.8.1 strip_hashes()

repolib.strip_hashes(line: str) -> str Strips leading hash (#) characters from a line and returns the result.

2.8.1.1 line

`str` The line to strip

2.8.2 load_all_sources()

repolib.load_all_sources() -> None

Loads all sources from the current system configuration.

2.8.3 set_testing()

repolib.set_testing(testing: bool = True) -> None Enables or disables testing mode in Repolib

When in testing mode, repolib will operate within a temporary directory rather tan on your live system configuration. This can be used for testing out changes to the program without worry about changes to the system config. It's also used for unit testing.

2.8.3.1 testing

`bool` - Wether testing mode should be enabled (*True*) or not (*False*)

2.9 Example

The following code as a Python program that creates in `example.sources` file in */etc/apt/sources.list.d* with some sample data, then modifies the suites used by the source and prints it to the console, before finally saving the new, modified source to disk:

```
import repolib
source = repolib.Source()
file = repolib.SourceFile(name='example')

source.ident = 'example-source'
source.name = 'Example Source'
source.types = [repolib.SourceType.BINARY]
source.uris = ['http://example.com/software']
source.suites = ['focal']
source.components = ['main']
file.add_source(source)

print(source.ui)

file.save()
```

When run with the appropriate arguments, it prints the contents of the source to console and then saves a new `example.sources` file:

```
$
example-source:
Name: Example Source
Enabled: yes
Types: deb
URIs: http://example.com/software
Suites: focal
Components: main

$ ls -la /etc/apt/sources.list.d/example.sources
-rw-r--r-- 1 root root 159 May  1 15:21 /etc/apt/sources.list.d/example.sources
```

Below is a walkthrough of this example.

2.9.1 Creating Source and File Objects

The first step in using *RepoLib* is creating *Source object* and file_object:

```
source = repolib.Source()
file = repolib.SourceFile(name='example')
```

The *Source object* will hold all of the information for the source to be created. The file_object represents the output file on disk, and allows for advanced usage like multiple sources per file.

2.9.2 Adding and Manipulating Data

The *Source object* contains attributes which describe the connfiguration aspects of the source required to fetch and install software. Generally, these attributes are lists of strings which describe different aspects of the source. They can be set or retrieved like any other attributes:

```
source.uris = ['http://example.com/software']
source.suites = ['focal']
```

This will add a `focal` suite to our source and add a URI from which to download package lists.

Source object also presents a dict-like interface for setting and getting configuration data. Key names are case-insensitive and their order within the object are preserved.

2.9.3 Adding the Source to the File

Before the *Source object* can be saved, it needs to be added to a file_object:

```
file.add_source(source)
```

This will add *source* to the *file*'s lists of sources, as well as setting the *source*'s file attribute to *file*.

2.9.4 Saving Data to Disk

Once a source has the correct data and has been added to a file object, it can be saved into the system configuration using file-save:

```
file.save()
```

When called, this writes the sources stored in the file to disk. This does not destroy the object, so that it may be further manipulated by the program.

Note: While data within the source or file objects can be manipulated after calling file-save, any subsequent changes will not be automatically written to disk as well. The file-save method must be called to commit changes to disk.

CHAPTER 3

Explanation of the DEB822 Source Format

The sources described in `/etc/apt/sources.list.d/` on a Debian-based OS are designed to support any number of different active and inactive sources, as well as a large variety of source media and transport protocols. The files describe one or more sources each and contain multiline stanzas defining one or more sources per stanza, with the most preferred source listed first. Information about available packages and versions from each source is fetched using the `apt update` command, or with an equivalent command from a different frontend.

3.1 sources.list.d

APT source information is stored locally within the `/etc/apt/sources.list.d` directory. In this directory are one or more files describing one or more sources each. For *Deb822-style Format* sources, each file needs to have the .sources extension. The filenames may only contain letters, digits, underscore, hyphen, and period characters. Files with other characters in their filenames will cause APT to print a notice that it has ignore that file (unless the file matches a pattern in the `Dir::Ignore-Files-Silently` configuration list, which will force APT to silently ignore the file.

3.2 One-Line-Style Format

In order to understand some of the decisions behind using the *Deb822-style Format* sources, it is helpful to understand the older *One-Line-Style Format*.

One-Line-Style Format sources occupy one line in a file ending in `.list`. The line begins with a type (i.e. `deb` or `deb-src` followed by options and arguments for this type. Individual entries cannot be continued onto multiple lines (hence the "one-line" portion of this format's name). Empty lines in `.list` files are ignored, and a `#` character anywhere on the line signifies that the remainder of that line is a comment. Consequently an entry can be disabled by commenting out that entire line (prefixing it with a `#`). If options are provided, they are space-separated and all together are enclosed within square brackets (`[]`). Options allowing multiple values should separate each value with a comma (`,`) and each option name is separated from its values by an equals sign (=).

This is the traditional format and is supported by all current APT versions. It has the advantage of being relatively compact for single-sources and relatively easy for humans to parse.

3.2.1 Disadvantages

Problems with the *One-Line-Style Format* begin when parsing entries via machine. Traditional, optionless entries are relatively simple to parse, as each different portion of the entry is separated with a space. With options, however, this is no longer the case. The presence of options causes there to be no, 1, or multiple segments of configuration between the type declaration and the URI. Additionally, APT sources support a variety of URI schemas, with the capability for extensions to add additional schemas on certain configurations. Thus, supporting modern, optioned *One-Line-Style Format* source entries requires use of either regular expressions or multi-level parsing in order to adequately parse the entry. Further compounding this support is the fact that *One-Line-Style Format* entries can have one or more components, preventing parsing of sources backwards from the end towards the front.

Consider the following examples:

```
deb [] http://example.com.ubuntu disco main restricted multiverse universe
deb [ arch=amd64 ] http://example.com/ubuntu disco main nonfree
deb [lang=en_US] http://example.com/ubuntu disco main restricted universe multiverse
deb [ arch=amd64,armel lang=en_US,en_CA ] http://example.com/ubuntu disco main
```

Each of these entries are syntactically valid source entries, each cleanly splits into eight segments when splitting on spaces. Depending on which entry being parsed, the URI portion of the entry may be in index 2, 4, or 5 while options (where present) may be in index 1, 2, or 3. If we want to work backwards, then the URI is in either index -3, -4, or -6. The only segments guaranteed to be present at any given index is the type. The situation is even more complicated when considering that entries may have at a minimum 3 elements, and at a maximum 12 or more elements.

In addition to parsing difficulty, *One-Line-Style Format* entries may only specify a single suite and URI per entry, meaning that having two active mirrors for a given source requires doubling the number of entries configured. You must also create an extra set of duplicated entries for each suite you want to configure. This can make tracking down duplicated entries difficult for users and leads to longer-than-necessary configuration.

3.3 Deb822-style Format

This format addresses the file-length, duplication, and machine-parsability issues present in the *One-Line-Style Format*. Each source is configured in a single stanza of configuration, with lines explicitly describing their function for the source. They also allow for lists of values for most options, meaning that mirrors or multiple suites can be defined within a single source. A # character at the beginning of a line marks the entire line as a comment. Entries can again be disabled by commenting out each line within the stanza; however, as a convenience this format also brings an `Enabled:` field which, when set to `no` disables the entry as well. Removing this field or setting it to `yes` re-enables it. Options have the same format as every other field, thus a stanza may be parsed by checking the beginning of each line for a fixed substring, and if the line doesn't match a known substring, it can be assumed the line is an option and the line can be ignored. Unknown options are ignored by all versions of APT. This has the unintentional side effect of adding extensibility to the source; by selecting a carefully namespaced field name, third-party applications and libraries can add their own fields to sources without causing breakage by APT. This can include comments, version information, and (as is the case with repolib-module, pretty, human-readable names.

From the `sources.list(5)` manual page:

> This is a new format supported by apt itself since version 1.1. Previous versions ignore such files with a notice message as described earlier. It is intended to make this format gradually the default format, deprecating the previously described one-line-style format, as it is easier to create, extend and modify for humans and machines alike especially if a lot of sources and/or options are involved.

CHAPTER 4

DEB822 Source Format Specifications

Following is a description of each field in the deb822 source format.

4.1 Enabled:

Enabled: (value: "yes" or "no", required: No, default: "yes") Tells APT whether the source is enabled or not. Disabled sources are not queried for package lists, effectively removing them from the system sources while still allowing reference or re-enabling at any time.

4.2 Types:

Types: (value: "deb" or "deb-src", required: Yes) Defines which types of packages to look for from a given source; either binary: `deb` or source code: `deb-src`. The `deb` type references a typical two-level Debian archive providing packages containing pre-compiled binary data intended for execution on user machines. The `deb-src` type references a Debian distribution's source code in the same form as the `deb` type. A `deb-src` line is required to fetch source pacakge indices.

4.3 URIs:

URIs: (value: string(s), required: Yes) The URI must specify the base of the Debian distribution archive, from which APT finds the information it needs. There must be a URI component present in order for the source to be valid; multipls URIs can be configured simultaneously by adding a space-separated list of URIs.

A list of the current built-in URI Schemas supported by APT is available at the Debian sources.list manpage.

4.4 Suites:

Suites: (value: strings(s), required: Yes) The Suite can specify an exact path in relation to the URI(s) provided, in which case the *Components:* **must** be omitted and suite **must** end with a slash (`/`). Alternatively, it may take the form of a distribution version (e.g. a version codename like `disco` or `artful`). If the suite does not specify a path, at least one deb822-field-component **must** be present.

4.5 Components:

Components: (value: string(s), required: see *Suites:*) Components specify different sections of one distribution version present in a Suite. If *Suites:* specifies an exact path, then no Components may be specified. Otherwise, a component **must** be present.

4.6 Options

Sources may specify a number of options. These options and their values will generally narrow a set of software to be available from the source or in some other way control what software is downloaded from it. An exhaustive list of options can be found at the Debian sources.list manpage.

4.7 RepoLib-Specific Deb822 Fields

RepoLib presently defines a single internal-use fields which it adds to deb822 sources that it modifies.

4.7.1 X-Repolib-Name:

X-Repolib-Name: (value: string, required: no, default: filename) This field defines a formatted name for the source which is suitable for inclusion within a graphical UI or other interface which presents source information to an end-user. As a repolib-module specific field, this is silently ignored by APT and other tools operating with deb822 sources and is only intended to be utilized by repolib-module itself.

CHAPTER 5

Examples

The following specifies a binary and source-code source fetching from the primary Ubuntu archive with multiple suites for updates as well as several components:

```
Enabled: yes
Types: deb deb-src
URIs: http://archive.ubuntu.com/ubuntu
Suites: disco disco-updates disco-security disco-backports
Components: main universe multiverse restricted
```

This is a source for fetching Google's Chrome browser, which specifies a CPU architecture option and a RepoLib Name:

```
X-Repolib-Name: Google Chrome
Enabled: yes
URIs: http://dl.google.com/linux/chrome/deb
Suites: stable
Components: main
Architectures: amd64
```

This specifies a source for downloading packages for System76's Pop!_OS:

```
X-Repolib-Name: Pop!_OS PPA
Enabled: yes
Types: deb
URIs: http://ppa.launchpad.net/system76/pop/ubuntu
Suites: disco
Components: main
```

Following is a PPA source which has been disabled:

```
X-Repolib-Name: ZNC Stable
Enabled: no
Types: deb
URIs: http://ppa.launchpad.net/teward/znc/ubuntu
```

(continues on next page)

(continued from previous page)

```
Suites: disco
Components: main
```

CHAPTER 6

Apt Manage

`apt-manage` is a command line tool for managing your local software sources using RepoLib. Run `apt-manage` standalone to get a listing of all of the software repositories currently configured:

```
$ apt-manage
Configured sources:
system
pop-os-apps
ppa-system76-pop
```

`apt-manage` operates on both DEB822-formated sources (located in the `/etc/apt/sources.list.d/*.sources` files) as well as using traditional one-line format sources (in `/etc/apt/sources.list.d/*.list` files).

6.1 Adding Sources

The `add` subcommand is used to add new repositories to the software sources. You can specify a deb-line to configure into the system or a `ppa:` shortcut to add the new source directly:

```
$ sudo apt-manage add deb http://apt.pop-os.org/ubuntu disco main
$ sudo apt-manage add ppa:system76/pop
```

If an internet connection is available, `apt-manage` will additionally attempt to install the signing key for any `ppa:` shortcuts added.

6.1.1 Options for adding sources

Various options control adding sources to the system.

6.1.1.1 Source Code, Details, Disabling Sources

To enable source code for the added repository, use the `--source-code` flag:

```
$ apt-manage add --source-code ppa:system76/pop
```

The new source can be disabled upon adding it using the `--disable` flag:

```
$ apt-manage add --disable ppa:system76/pop
```

Details for the PPA are printed for review prior to adding the source by default. This will print the generated configuration for the source as well as any available details fetched for the source (typically only available for PPA sources). To suppress this output, include `--terse`.

6.1.1.2 Source File Format

The format which RepoLib saves the repository on disk in depends on the type of repository being added, but regardless the `--format` flag can be used to force either legacy `.list` format or modern `.sources` format:

```
apt-manage add popdev:master --format=list
apt-manage add ppa:system76/pop --format=sources
```

6.1.1.3 Names and Identifiers

Names for PPA sources are automatically detected from Launchpad if an internet connection is available. Otherwise they are automatically generated based on the source type and details. Optionally, a name can be specified when the source is added:

```
$ apt-manage add ppa:system76/pop --name "PPA for Pop_OS Software"
```

System-identifiers determine how the source is subsequently located within RepoLib and on the system. It matches the filename for the source's configuration file. It is automatically generated based on the source type, or can be specified manually upon creation using the `--identifier` flag:

```
$ apt-manage add ppa:system76/pop --identifier pop-ppa
```

Note: Even though `apt-manage` allows modifcation or management of DEB822-format sources, it does not currently support adding them to the system directly. DEB822 sources can be manually added or added using third-party tools, and `apt-manage` will correctly operate on them subsequently.

6.2 Listing Configuration Details

To get a list of all of the sources configured on the system, use the `list` subcommand:

```
$ apt-manage list
Configured sources:
system - Pop_OS System Sources
pop-os-apps - Pop_OS Applications
ppa-system76-pop - Pop!_OS PPA
```

The sources are listed with the system source (if detected/configured) first, followed by all DEB822-format sources detected first, then by all one-line format sources. The system-identifier (used to identify sources in the system) is listed at the beginning of the line, and the name of the source is listed after.

Details of an individual source can be printed by specifying a source's system-identifier:

```
$ apt-manage list ppa-system76-pop
Details for source ppa-system76-pop:
Name: Pop!_OS PPA
Enabled: yes
Types: deb deb-src
URIs: http://apt.pop-os.org/release
Suites: focal
Components: main
```

6.2.1 Listing all details of all sources at once

Details about all sources can be listed at once using the `--verbose` flag:

```
$ apt-manage list --verbose
Configured sources:
system - Pop_OS System Sources
Name: Pop_OS System Sources
Enabled: yes
Types: deb deb-src
URIs: http://us.archive.ubuntu.com/ubuntu/
Suites: focal focal-security focal-updates focal-backports
Components: main restricted universe multiverse

pop-os-apps - Pop_OS Applications
Name: Pop_OS Applications
Enabled: yes
Types: deb
URIs: http://apt.pop-os.org/proprietary
Suites: focal
Components: main

ppa-system76-pop - Pop!_OS PPA
Name: Pop!_OS PPA
Enabled: yes
Types: deb deb-src
URIs: http://apt.pop-os.org/release
Suites: focal
Components: main
```

6.2.1.1 Note

Passing the `--verbose` flag only applies to listing all sources. It has no effect if a source is specified using the system-identifier; in that case, only the specified source is printed. Additionally, if there are sources files which contain errors, the `--verbose` flag will print details about them, including the contents of the files and the stack trace of the exception which caused the error.

6.2.2 Legacy sources.list entries

The contents of the system `sources.list` file can be appended to the end of the output using the `--legacy` flag.

6.3 Modifying Sources

Modifications can be made to various configured sources using the `modify` subcommand.

6.3.1 Enabling/Disabling Sources: –enable | –disable

Sources can be disabled, which prevents software/updates from being installed from the source but keeps it present in the system configuration for later use or records for later. To disable a source, use `--disable`:

```
$ apt-manage modify ppa-system76-pop --disable
```

To re-enable a source after it's been disabled, use `--enable`:

```
$ apt-manage modify ppa-system76-pop --enable
```

6.3.2 Changing names of sources: –name

RepoLib allows setting up human-readable names for use in GUIs or other user-facing contexts. To set or change a name of a source, use `--name`:

```
$ apt-manage modify ppa-system76-pop --name "Pop_OS PPA"
```

6.3.3 Suites: –add-suite | –remove-suite

Suites for sources can be added or removed from the configuration. In one-line sources, these are added with multiple lines, since each one-line source can have only one suite each. DEB822 sources can have multiple suites.

To add a suite, use `--add-suite`:

```
$ apt-manage modify ppa-system76-pop --add-suite groovy
```

Use `--remove-suite` to remove a suite:

```
$ apt-manage modify ppa-system76-pop --remove-suite focal
```

6.3.4 Components: –add-component | –remove-component

Both types of source format can have multiple components for each source. Note that all components for one-line format sources will share all of a source's components.

Components are managed similarly to suites:

```
$ apt-manage modify system --add-component universe
$ apt-manage modify system --remove-component restricted
```

6.3.5 URIs: –add-uri | –remove-uri

DEB822 sources may contain an arbitrary number of URIs. One-line sources require an additional line for each individual URI added. All suites on a source are all applied to all of the URIs equally.

URIs are managed similarly to both suites and components:

```
$ apt-manage modify system --add-uri http://apt.pop-os.org/ubuntu
$ apt-manage modify system --remove-uri http://us.archive.ubuntu.com/ubuntu
```

6.3.5.1 Notes

Multiple modifications may be applied on a single `apt-manage modify` calls:

```
$ apt-manage modify system --name "Pop_OS 20.10 System Sources" \
    --add-suite groovy \
    --remove-suite focal focal-proposed \
    --add-uri http://apt.pop-os.org/ubuntu \
    --remove-uri http://us.archive.ubuntu.com/ubuntu
```

6.4 Removing Sources

To remove a source from the system configuration, use the `remove` subcommand:

```
$ apt-manage remove ppa-system76-pop
```

6.5 Managing Signing Keys

Signing keys are an important part of repository security and are generally required to be used in repositories for all recent versions of APT. As previous methods of handling Apt keys have been deprecated, Apt Manage provides easy tools to use for managing signing keys for repositories in the `key` subcommand.

Most of the tools in the `key` subcommand are centered around adding a signing key to a repository:

```
apt-manage key repo-id --fingerprint 63C46DF0140D738961429F4E204DD8AEC33A7AFF
```

Apt Manage supports adding keys from a variety of sources:

6.5.1 Existing Keyring Files, –name, –path

`--name` sets the signed_by value of the existing repository to the name of a file within the system key configuration directory:

```
apt-manage key popdev-master --name popdev
```

`--path` sets the signed_by value of the existing repository to the path of a file on disk:

```
apt-manage key popdev-master --path /etc/apt/keyrings/popdev-archive-keyring.gpg
```

6.5.2 Keyring Files Stored on the Internet, –url

`--url` will download a key file from the internet and install it into the system, then set the repository to use that key:

```
apt-manage key popdev-master --url https://example.com/sigining-key.asc
```

6.5.3 Keys Stored on a Public Keyserver

`--fingerprint` will fetch the specified fingerprint from a public keyserver. By default, keys will be fetched from `keyserver.ubuntu.com`, but any SKS keyserver can be specified using the `--keyserver=` argument:

```
apt-manage key ppa-system76-pop \
    --fingerprint=E6AC16572ED1AD6F96C7EBE01E5F8BBC5BEB10AE

apt-manage key popdev-master \
    --fingerprint=63C46DF0140D738961429F4E204DD8AEC33A7AFF \
    --keyserver=https://keyserver.example.com/
```

6.5.4 Adding ASCII-Armored Keys Directly, –ascii

`--ascii` Will take plain ascii data from the command line and add it to a new keyring file, then set the repository to use that key:

```
apt-manage key popdev-master --ascii "$(/tmp/popdev-key.asc)"
```

6.5.5 Removing Keys

Generally, manually removing keys is not necessary because removing a source automatically removes the key (if it is the only source using that key). However, If there is a need to remove a key manually (e.g. the signing key has changed and must be re-added), then removal is supported:

```
apt-manage key popdev-master --remove
```

This will remove the key from the repository configuration and if no other sources are using a particular key, it will also remove the keyring file from disk.

CHAPTER 7

Installation

There are a variety of ways to install RepoLib

7.1 From System Package Manager

If your operating system packages repolib, you can install it by running:

```
sudo apt install python3-repolib
```

7.1.1 Uninstall

To uninstall, simply do:

```
sudo apt remove python3-repolib
```

7.2 From PyPI

Repolib is available on PyPI. You can install it for your current user with:

```
pip3 install repolib
```

Alternatively, you can install it system-wide using:

```
sudo pip3 install repolib
```

7.2.1 Uninstall

To uninstall, simply do:

```
sudo pip3 uninstall repolib
```

7.3 From Git

First, clone the git repository onto your local system:

```
git clone https://github.com/isantop/repolib
cd repolib
```

7.4 Debian

On debian based distributions, you can build a .deb package locally and install it onto your system. You will need the following build-dependencies:

- debhelper (>= 11)
- dh-python
- lsb-release
- python3-all
- python3-dbus
- python3-debian
- python3-setuptools
- python3-distutils
- python3-pytest
- python3-gnupg

You can use this command to install these all in one go:

```
sudo apt install debhelper dh-python python3-all python3-setuptools python3-gnupg
```

Then build and install the package:

```
debuild -us -uc
cd ..
sudo dpkg -i python3-repolib_*.deb
```

7.4.1 Uninstall

To uninstall, simply do:

```
sudo apt remove python3-repolib
```

7.5 setuptools setup.py

You can build and install the package using python3-setuptools. First, install the dependencies:

```
sudo apt install python3-all python3-setuptools
```

Then build and install the package:

```
sudo python3 ./setup.py install
```

7.5.1 Uninstall

You can uninstall RepoLib by removing the following files/directories:

- /usr/local/lib/python3.7/dist-packages/repolib/
- /usr/local/lib/python3.7/dist-packages/repolib-*.egg-info
- /usr/local/bin/apt-manage

This command will remove all of these for you:

```
sudo rm -r /usr/local/lib/python3.7/dist-packages/repolib* /usr/local/bin/apt-manage
```