nvchecker

Release 2.11dev

unknown

Nov 20, 2022

CONTENTS

CHAPTER

ONE

USAGE OF NVCHECKER COMMANDS

nvchecker (short for *new version checker*) is for checking if a new version of some software has been released.

This is the version 2.0 branch. For the old version 1.x, please switch to the `v1.x` branch.

1.1 Dependency

- Python 3.7+
- Python library: structlog, tomli, appdirs
- One of these Python library combinations (ordered by preference):
  - tornado + pycurl
  - aiohttp
  - httpx with http2 support (experimental; only latest version is supported)
  - tornado
- All commands used in your software version configuration files

1.2 Install and Run

To install:

```
pip3 install nvchecker
```

To use the latest code, you can also clone this repository and run:

```
python3 setup.py install
```

To see available options:

```
nvchecker --help
```

Run with one or more software version files:

```
nvchecker -c config_file.toml
```

A simple config file may look like:

```
[nvchecker]
source = "github"
github = "lilydjwg/nvchecker"

[python-toml]
source = "pypi"
pypi = "toml"
```

You normally will like to specify some "version record files"; see below.

1.2.1 JSON logging

With `--logger=json` or `--logger=both`, you can get a structured logging for programmatically consuming. You can use `--json-log-fd=FD` to specify the file descriptor to send logs to (take care to do line buffering). The logging level option (`-l` or `--logging`) doesn't take effect with this.

The JSON log is one JSON string per line. The following documented events and fields are stable, undocumented ones may change without notice.

event=updated
: An update is detected. Fields `name`, `old_version` and `version` are available. `old_version` maybe `null`.

event=up-to-date
: There is no update. Fields `name` and `version` are available.

event=no-result
: No version is detected. There may be an error. Fields `name` is available.

level=error
: There is an error. Fields `name` and `exc_info` may be available to give further information.

1.2.2 Upgrade from 1.x version

There are several backward-incompatible changes from the previous 1.x version.

1. Version 2.x requires Python 3.7+ to run.
2. The command syntax changes a bit. You need to use a `-c` switch to specify your software version configuration file (or use the default).
3. The configuration file format has been changed from ini to toml. You can use the `nvchecker-ini2toml` script to convert your old configuration files. However, comments and formatting will be lost, and some options may not be converted correctly.
4. Several options have been renamed. `max_concurrent` to `max_concurrency`, and all option names have their `-` be replaced with `_`.
5. All software configuration tables need a `source` option to specify which source is to be used rather than being figured out from option names in use. This enables additional source plugins to be discovered.
6. The version record files have been changed to use JSON format (the old format will be converted on writing).
7. The `vcs` source is removed. (It's available inside lilac at the moment.) A `git` source is provided.
8. `include_tags_pattern` and `ignored_tags` are removed. Use *List Options* instead.

1.3 Version Record Files

Version record files record which version of the software you know or is available. They are a simple JSON object mapping software names to known versions.

1.3.1 The `nvtake` Command

This command helps to manage version record files. It reads both old and new version record files, and a list of names given on the commandline. It then update the versions of those names in the old version record file.

This helps when you have known (and processed) some of the updated software, but not all. You can tell nvchecker that via this command instead of editing the file by hand.

This command will help most if you specify where you version record files are in your config file. See below for how to use a config file.

1.3.2 The `nvcmp` Command

This command compares the `newver` file with the `oldver` one and prints out any differences as updates, e.g.:

```
$ nvcmp -c sample_source.toml
Sparkle Test App None -> 2.0
test 0.0 -> 0.1
```

1.4 Configuration Files

The software version source files are in toml format. The *key name* is the name of the software. Following fields are used to tell nvchecker how to determine the current version of that software.

See sample_source.toml for an example.

1.4.1 Configuration Table

A special table named `__config__` provides some configuration options.

Relative path are relative to the source files, and ~ and environmental variables are expanded.

Currently supported options are:

oldver
: Specify a version record file containing the old version info.

newver
: Specify a version record file to store the new version info.

proxy
: The HTTP proxy to use. The format is `proto://host:port`, e.g. `http://localhost:8087`. Different backends have different level support for this, e.g. with `pycurl` you can use `socks5h://host:port` proxies.

max_concurrency
: Max number of concurrent jobs. Default: 20.

http_timeout
: Time in seconds to wait for HTTP requests. Default: 20.

keyfile
: Specify a toml config file containing key (token) information. This file should contain a `keys` table, mapping key names to key values. See specific source for the key name(s) to use.

  Sample `keyfile.toml`:

```
[keys]
# https://github.com/settings/tokens
# scope: repo -> public_repo
github = "ghp_<stripped>"
```

1.4.2 Global Options

The following options apply to every check sources. You can use them in any item in your configuration file.

prefix
: Strip the prefix string if the version string starts with it. Otherwise the version string is returned as-is.

from_pattern, to_pattern
: Both are Python-compatible regular expressions. If `from_pattern` is found in the version string, it will be replaced with `to_pattern`.

  If `from_pattern` is not found, the version string remains unchanged and no error is emitted.

missing_ok
: Suppress warnings and errors if a version checking module finds nothing. Currently only `regex` supports it.

proxy
: The HTTP proxy to use. The format is `proto://host:port`, e.g. `http://localhost:8087`. Different backends have different level support for this, e.g. with `pycurl` you can use `socks5h://host:port` proxies.

  Set it to `""` (empty string) to override the global setting.

  This only works when the source implementation uses the builtin HTTP client, and doesn't work with the `aur` source because it's batched (however the global proxy config still applies).

user_agent
: The user agent string to use for HTTP requests.

tries
: Try specified times when a network error occurs. Default is `1`.

  This only works when the source implementation uses the builtin HTTP client.

httptoken
: A personal authorization token used to fetch the url with the `Authorization` header. The type of token depends on the authorization required.

  - For Bearer token set : `Bearer <Your_bearer_token>`
  - For Basic token set : `Basic <Your_base64_encoded_token>`

  In the keyfile add `httptoken_{name}` token.

verify_cert
: Whether to verify the HTTPS certificate or not. Default is `true`.

If both `prefix` and `from_pattern`/`to_pattern` are used, `from_pattern`/`to_pattern` are ignored. If you want to strip the prefix and then do something special, just use `from_pattern`/`to_pattern`. For example, the transformation of `v1_1_0 => 1.1.0` can be achieved with `from_pattern = 'v(\d+)_(\d+)_(\d+)'` and `to_pattern = '\1.\2.\3'`. (Note that in TOML it's easiler to write regexes in single quotes so you don't need to escape `\`.)

1.4.3 List Options

The following options apply to sources that return a list. See individual source tables to determine whether they are supported.

include_regex
: Only consider version strings that match the given regex. The whole string should match the regex. Be sure to use `.*` when you mean it!

exclude_regex
: Don't consider version strings that match the given regex. The whole string should match the regex. Be sure to use `.*` when you mean it! This option has higher precedence that `include_regex`; that is, if matched by this one, it's excluded even it's also matched by `include_regex`.

sort_version_key
: Sort the version string using this key function. Choose among `parse_version`, `vercmp` and `awesomeversion`. Default value is `parse_version`. `parse_version` uses an old version of `pkg_resources.parse_version`. `vercmp` uses `pyalpm.vercmp`. `awesomeversion` uses awesomeversion.

ignored
: Version strings that are explicitly ignored, separated by whitespace. This can be useful to avoid some known mis-named versions, so newer ones won't be "overridden" by the old broken ones.

1.4.4 Search in a Webpage

```
source = "regex"
```

Search through a specific webpage for the version string. This type of version finding has these fields:

url
: The URL of the webpage to fetch.

encoding
: (*Optional*) The character encoding of the webpage, if `latin1` is not appropriate.

regex
: A regular expression used to find the version string.

  It can have zero or one capture group. The capture group or the whole match is the version string.

  When multiple version strings are found, the maximum of those is chosen.

post_data
: (*Optional*) When present, a `POST` request (instead of a `GET`) will be used. The value should be a string containing the full body of the request. The encoding of the string can be specified using the `post_data_type` option.

post_data_type
: (*Optional*) Specifies the `Content-Type` of the request body (`post_data`). By default, this is `application/x-www-form-urlencoded`.

This source supports *List Options*.

1.4.5 Search in an HTTP header

```
source = "httpheader"
```

Send an HTTP request and search through a specific header.

url
: The URL of the HTTP request.

header
: (*Optional*) The header to look at. Default is `Location`. Another useful header is `Content-Disposition`.

regex
: A regular expression used to find the version string.

  It can have zero or one capture group. The capture group or the whole match is the version string.

  When multiple version strings are found, the maximum of those is chosen.

method
: (*Optional*) The HTTP method to use. Default is `HEAD`.

follow_redirects
: (*Optional*) Whether to follow 3xx HTTP redirects. Default is `false`. If you are looking at a `Location` header, you shouldn't change this.

1.4.6 Search with an HTML Parser

```
source = "htmlparser"
```

Send an HTTP request and search through the body a specific xpath.

url
: The URL of the HTTP request.

xpath
: An xpath expression used to find the version string.

post_data
: (*Optional*) When present, a `POST` request (instead of a `GET`) will be used. The value should be a string containing the full body of the request. The encoding of the string can be specified using the `post_data_type` option.

post_data_type
: (*Optional*) Specifies the `Content-Type` of the request body (`post_data`). By default, this is `application/x-www-form-urlencoded`.

Note: An additional dependency "lxml" is required.

1.4.7 Find with a Command

```
source = "cmd"
```

Use a shell command line to get the version. The output is striped first, so trailing newlines do not bother.

cmd
: The command line to use. This will run with the system's standard shell (i.e. `/bin/sh`).

1.4.8 Check AUR

```
source = "aur"
```

Check Arch User Repository for updates. Per-item proxy setting doesn't work for this because several items will be batched into one request.

aur
: The package name in AUR. If empty, use the name of software (the *table name*).

strip_release
: Strip the release part.

use_last_modified
: Append last modified time to the version.

1.4.9 Check GitHub

```
source = "github"
```

Check GitHub for updates. The version returned is in date format `%Y%m%d.%H%M%S`, e.g. `20130701.012212`, unless `use_latest_release` or `use_max_tag` is used. See below.

github
: The github repository, with author, e.g. `lilydjwg/nvchecker`.

branch
: Which branch to track? Default: the repository's default.

path
: Only commits containing this file path will be returned.

use_latest_release
: Set this to `true` to check for the latest release on GitHub.

  GitHub releases are not the same with git tags. You'll see big version names and descriptions in the release page for such releases, e.g. zfsonlinux/zfs's, and those small ones like nvchecker's are only git tags that should use `use_max_tag` below.

  Will return the release name instead of date.

use_latest_tag
: Set this to `true` to check for the latest tag on GitHub.

  This requires a token because it's using the v4 GraphQL API.

query
: When `use_latest_tag` is `true`, this sets a query for the tag. The exact matching method is not documented by GitHub.

use_max_tag
: Set this to `true` to check for the max tag on GitHub. Unlike `use_latest_release`, this option includes both annotated tags and lightweight ones, and return the largest one sorted by the `sort_version_key` option. Will return the tag name instead of date.

token
: A personal authorization token used to call the API.

An authorization token may be needed in order to use `use_latest_tag` or to request more frequently than anonymously.

To set an authorization token, you can set:

- a key named `github` in the keyfile
- the token option

This source supports *List Options* when `use_max_tag` is set.

1.4.10 Check Gitea

```
source = "gitea"
```

Check Gitea for updates. The version returned is in date format `%Y%m%d`, e.g. `20130701`, unless `use_max_tag` is used. See below.

gitea
: The gitea repository, with author, e.g. `gitea/tea`.

branch
: Which branch to track? Default: the repository's default.

use_max_tag
: Set this to `true` to check for the max tag on Gitea. Will return the biggest one sorted by old `pkg_resources.parse_version`. Will return the tag name instead of date.

host
: Hostname for self-hosted Gitea instance.

token
: Gitea authorization token used to call the API.

To set an authorization token, you can set:

- a key named `gitea_{host}` in the keyfile, where `host` is all-lowercased host name
- the token option

This source supports *List Options* when `use_max_tag` is set.

1.4.11 Check BitBucket

```
source = "bitbucket"
```

Check BitBucket for updates. The version returned is in date format `%Y%m%d`, e.g. `20130701`, unless `use_max_tag` is used. See below.

bitbucket
: The bitbucket repository, with author, e.g. `lilydjwg/dotvim`.

branch
: Which branch to track? Default: the repository's default.

use_max_tag
: Set this to `true` to check for the max tag on BitBucket. Will return the biggest one sorted by old `pkg_resources.parse_version`. Will return the tag name instead of date.

use_sorted_tags
: If `true`, tags are queried and sorted according to the `query` and `sort` keys. Will return the tag name instead of the date.

query
: A query string use to filter tags when `use_sorted_tags` set (see here for examples). The string does not need to be escaped.

sort
: A field used to sort the tags when `use_sorted_tags` is set (see here for examples). Defaults to `-target.date` (sorts tags in descending order by date).

max_page
How many pages do we search for the max tag? Default is 3. This works when `use_max_tag` is set.

This source supports *List Options* when `use_max_tag` or `use_sorted_tags` is set.

1.4.12 Check GitLab

```
source = "gitlab"
```

Check GitLab for updates. The version returned is in date format `%Y%m%d`, e.g. `20130701`, unless `use_max_tag` is used. See below.

gitlab
The gitlab repository, with author, e.g. `Deepin/deepin-music`.

branch
Which branch to track?

use_max_tag
Set this to `true` to check for the max tag on GitLab. Will return the biggest one sorted by old `pkg_resources.parse_version`. Will return the tag name instead of date.

host
Hostname for self-hosted GitLab instance.

token
GitLab authorization token used to call the API.

To set an authorization token, you can set:

- a key named `gitlab_{host}` in the keyfile, where `host` is all-lowercased host name
- the token option

This source supports *List Options* when `use_max_tag` is set.

1.4.13 Check PyPI

```
source = "pypi"
```

Check PyPI for updates.

pypi
The name used on PyPI, e.g. `PySide`.

use_pre_release
Whether to accept pre release. Default is false.

1.4.14 Check RubyGems

```
source = "gems"
```

Check RubyGems for updates.

gems
: The name used on RubyGems, e.g. `sass`.

This source supports *List Options*.

1.4.15 Check NPM Registry

```
source = "npm"
```

Check NPM Registry for updates.

npm
: The name used on NPM Registry, e.g. `coffee-script`.

To configure which registry to query, a source plugin option is available. You can specify like this:

```
[__config__.source.npm]
registry = "https://registry.npm.taobao.org"
```

1.4.16 Check Hackage

```
source = "hackage"
```

Check Hackage for updates.

hackage
: The name used on Hackage, e.g. `pandoc`.

1.4.17 Check CPAN

```
source = "cpan"
```

Check MetaCPAN for updates.

cpan
: The name used on CPAN, e.g. `YAML`.

1.4.18 Check CRAN

```
source = "cran"
```

Check CRAN for updates.

cran
: The name used on CRAN, e.g. `xml2`.

1.4.19 Check Packagist

```
source = "packagist"
```

Check Packagist for updates.

packagist
: The name used on Packagist, e.g. `monolog/monolog`.

1.4.20 Check crates.io

```
source = "cratesio"
```

Check crates.io for updates.

cratesio
: The crate name on crates.io, e.g. `tokio`.

1.4.21 Check Local Pacman Database

```
source = "pacman"
```

This is used when you run `nvchecker` on an Arch Linux system and the program always keeps up with a package in your configured repositories for Pacman.

pacman
: The package name to reference to.

strip_release
: Strip the release part.

1.4.22 Check Arch Linux official packages

```
source = "archpkg"
```

This enables you to track the update of Arch Linux official packages, without needing of pacman and an updated local Pacman databases.

archpkg
: Name of the Arch Linux package.

strip_release
: Strip the release part, only return part before `-`.

provided
Instead of the package version, return the version this package provides. Its value is what the package provides, and `strip_release` takes effect too. This is best used with libraries.

1.4.23 Check Debian Linux official packages

```
source = "debianpkg"
```

This enables you to track the update of Debian Linux official packages, without needing of apt and an updated local APT database.

debianpkg
Name of the Debian Linux source package.

suite
Name of the Debian release (jessie, wheezy, etc, defaults to sid)

strip_release
Strip the release part.

1.4.24 Check Ubuntu Linux official packages

```
source = "ubuntupkg"
```

This enables you to track the update of Ubuntu Linux official packages, without needing of apt and an updated local APT database.

ubuntupkg
Name of the Ubuntu Linux source package.

suite
Name of the Ubuntu release (xenial, zesty, etc, defaults to None, which means no limit on suite)

strip_release
Strip the release part.

1.4.25 Check Repology

```
source = "repology"
```

This enables you to track updates from Repology (repology.org).

repology
Name of the `project` to check.

repo
Check the version in this repo. This field is required.

subrepo
Check the version in this subrepo. This field is optional. When omitted all subrepos are queried.

This source supports *List Options*.

1.4.26 Check Anitya

```
source = "anitya"
```

This enables you to track updates from Anitya (release-monitoring.org).

anitya
: `distro/package`, where `distro` can be a lot of things like "fedora", "arch linux", "gentoo", etc. `package` is the package name of the chosen distribution.

1.4.27 Check Android SDK

```
source = "android_sdk"
```

This enables you to track updates of Android SDK packages listed in `sdkmanager --list`.

android_sdk
: The package path prefix. This value is matched against the `path` attribute in all <remotePackage> nodes in an SDK manifest XML. The first match is used for version comparisons.

repo
: Should be one of `addon` or `package`. Packages in `addon2-1.xml` use `addon` and packages in `repository2-1.xml` use `package`.

channel
: Choose the target channel from one of `stable`, `beta`, `dev` or `canary`. This option also accepts a comma-separated list to pick from multiple channels. For example, the latest unstable version is picked with `beta,dev,canary`. The default is `stable`.

host_os
: Choose the target OS for the tracked package from one of `linux`, `macosx`, `windows`. The default is `linux`. For OS-independent packages (e.g., Java JARs), this field is ignored.

This source supports *List Options*.

1.4.28 Check Sparkle framework

```
source = "sparkle"
```

This enables you to track updates of macOS applications which using Sparkle framework.

sparkle
: The url of the sparkle appcast.

1.4.29 Check Pagure

```
source = "pagure"
```

This enables you to check updates from Pagure.

pagure
: The project name, optionally with a namespace.

host
: Hostname of alternative instance like src.fedoraproject.org.

This source returns tags and supports *List Options*.

1.4.30 Check APT repository

```
source = "apt"
```

This enables you to track the update of an arbitrary APT repository, without needing of apt and an updated local APT database.

pkg
Name of the APT binary package.

srcpkg
Name of the APT source package.

mirror
URL of the repository.

suite
Name of the APT repository release (jessie, wheezy, etc)

repo
Name of the APT repository (main, contrib, etc, defaults to main)

arch
Architecture of the repository (i386, amd64, etc, defaults to amd64)

strip_release
Strip the release part.

Note that either pkg or srcpkg needs to be specified (but not both) or the item name will be used as pkg.

1.4.31 Check Git repository

```
source = "git"
```

This enables you to check tags or branch commits of an arbitrary git repository, also useful for scenarios like a github project having too many tags.

git
URL of the Git repository.

use_commit
Return a commit hash instead of tags.

branch
When `use_commit` is true, return the commit on the specified branch instead of the default one.

When this source returns tags (`use_commit` is not true) it supports *List Options*.

1.4.32 Check container registry

```
source = "container"
```

This enables you to check tags of images on a container registry like Docker.

container
The path for the container image. For official Docker images, use namespace `library/` (e.g. `library/python`).

registry
The container registry host. Default: `docker.io`

`registry` and `container` are the host and the path used in the pull command. Note that the `docker` command allows omitting some parts of the container name while this plugin requires the full name. If the host part is omitted, use `docker.io`, and if there is no slash in the path, prepend `library/` to the path. Here are some examples:

Pull command	registry	container
docker pull quay.io/prometheus/node-exporter	quay.io	prometheus/node-exporter
docker pull nvidia/cuda	docker.io	nvidia/cuda
docker pull python	docker.io	library/python

This source returns tags and supports *List Options*.

1.4.33 Check ALPM database

```
source = "alpm"
```

Check package updates in a local ALPM database.

alpm
Name of the package.

repo
Name of the package repository in which the package resides. If not provided, nvchecker will use `repos` value, see below.

repos
An array of possible repositories in which the package may reside in, nvchecker will use the first repository which contains the package. If not provided, `core`, `extra`, `community` and `multilib` will be used, in that order.

dbpath
Path to the ALPM database directory. Default: `/var/lib/pacman`. You need to update the database yourself.

strip_release
Strip the release part, only return the part before `-`.

provided
Instead of the package version, return the version this package provides. Its value is what the package provides, and `strip_release` takes effect too. This is best used with libraries.

1.4.34 Check Open Vsx

```
source = "openvsx"
```

Check Open Vsx for updates.

openvsx
: The extension's Unique Identifier on open-vsx.org, e.g. `ritwickdey.LiveServer`.

1.4.35 Check Visual Studio Code Marketplace

```
source = "vsmarketplace"
```

Check Visual Studio Code Marketplace for updates.

vsmarketplace
: The extension's Unique Identifier on marketplace.visualstudio.com/vscode, e.g. `ritwickdey.LiveServer`.

1.4.36 Combine others' results

```
source = "combiner"
```

This source can combine results from other entries.

from
: A list of entry names to wait results for.

format
: A format string to combine the results into the final string.

Example:

```
[entry-1]
source = "cmd"
cmd = "echo 1"

[entry-2]
source = "cmd"
cmd = "echo 2"

[entry-3]
source = "combiner"
from = ["entry-1", "entry-2"]
format = "$1-$2"
```

1.4.37 Manually updating

```
source = "manual"
```

This enables you to manually specify the version (maybe because you want to approve each release before it gets to the script).

manual
: The version string.

1.4.38 Extending

It's possible to extend the supported sources by writing plugins. See *How to develop a source plugin for nvchecker* for documentation.

CHAPTER

TWO

HOW TO DEVELOP A SOURCE PLUGIN FOR NVCHECKER

Source plugins enable nvchecker to discover software version strings in additional ways.

2.1 Where to put the plugins

They are Python modules put in any directories named `nvchecker_source` in `sys.path`. This is called namespace packages introduced by PEP 420. For local use, `~/.local/lib/pythonX.Y/site-packages/nvchecker_source` is a good place, or you can define the `PYTHONPATH` environment variable and put nvchecker source plugins there inside a `nvchecker_source` directory.

Plugins are referenced by their names in the configuration file (`source = "xxx"`). If multiple plugins have the same name, the first one in `sys.path` will be used.

2.2 How to write a simple plugin

For simple situations, you need to define an async function with the following signature:

```
async def get_version(
  name: str, conf: Entry, *,
  cache: AsyncCache, keymanager: KeyManager,
  **kwargs,
) -> VersionResult:
  ...
```

Those types are imported from *nvchecker.api*.

`name` is the table keys in the configuration file, and `conf` is a dict of the content of that table. You should not modify this dict.

`cache` is an *AsyncCache* object that caches results for you. Every plugin has its own `cache` object so that cache keys won't conflict.

`keymanager` is a *KeyManager* object that you can call *.get_key(name)* to get the key (token) from the keyfile.

There may be additional keyword arguments in the future so `**kwargs` should be used.

If you want to send an HTTP request, it's preferred to use :meth: *cache.get_json* or the :data: *nvchecker.api.session* object. It will use the auto-selected HTTP backend and handle the `proxy` option automatically.

For details about these objects, see *the API documentation*, or take existing source plugins as examples.

2.3 How to write a more powerful plugin

You may want more control in your source plugin, e.g. to do batch requests. To do this, you provide a class instead:

```
class Worker(BaseWorker):
  async def run(self) -> None:
    ...
```

You will have the following in the attributes:

```
token_q: Queue[bool],
result_q: Queue[RawResult],
tasks: List[Tuple[str, Entry]],
keymanager: KeyManager,
```

You are expected to process *tasks* and put results in *result_q*. See `nvchecker_source/none.py` for the simplest example, and `nvchecker_source/aur.py` for a complete, batching example.

For details about these objects, see *the API documentation*.

You can also receive a configuration section from the configuration as `__config__.source.SOURCE_NAME`, where `SOURCE_NAME` is what your plugin is called. This can be used to specify a mirror site for your plugin to use, e.g. the `npm` plugin accepts the following config:

```
[__config__.source.npm]
registry = "https://registry.npm.taobao.org"
```

When such a configuration exists for your plugin, you need to define a function named `configure` to receive it:

```
def configure(config):
  '''use the "config" dict in some way'''
  ...
```

CHAPTER

THREE

NVCHECKER.API — THE SOURCE PLUGIN API

exception nvchecker.api.**TemporaryError**(*code*, *message*, *response*)

A temporary error (e.g. network error) happens.

exception nvchecker.api.**HTTPError**(*code*, *message*, *response*)

An HTTP 4xx error happens

class nvchecker.api.**BaseWorker**(*task_sem: asyncio.Semaphore*, *result_q: Queue[*RawResult*]*, *tasks: List[Tuple[str, Entry]]*, *keymanager:* KeyManager)

The base class for defining `Worker` classes for source plugins.

task_sem: `asyncio.Semaphore`

This is the rate-limiting semaphore. Workers should acquire it while doing one unit of work.

result_q: `Queue[RawResult]`

Results should be put into this queue.

tasks: `List[Tuple[str, Entry]]`

A list of tasks for the `Worker` to complete. Every task consists of a tuple for the task name (table name in the configuration file) and the content of that table (as a `dict`).

keymanager: *KeyManager*

The *KeyManager* for retrieving keys from the keyfile.

abstract async run() → None

Run the *tasks*. Subclasses should implement this method.

class nvchecker.api.**RawResult**(*name: str*, *version: VersionResult*, *conf: Entry*)

The unprocessed result from a check.

Create new instance of RawResult(name, version, conf)

property name

The name (table name) of the entry.

property version

The result from the check.

property conf

The entry configuration (table content) of the entry.

class nvchecker.api.**AsyncCache**

A cache for use with async functions.

cache: `Dict[Hashable, Any]`

lock: Lock

async get_json(*url: str*, *, *headers: Dict[str, str] = {}*) → Any

Get specified `url` and return the response content as JSON.

The returned data will be cached for reuse.

async get(*key: Hashable*, *func: Callable[[Hashable], Coroutine[Any, Any, Any]]*) → Any

Run async `func` and cache its return value by `key`.

The `key` should be hashable, and the function will be called with it as its sole argument. For multiple simultaneous calls with the same key, only one will actually be called, and others will wait and return the same (cached) value.

class nvchecker.api.**KeyManager**(*file: Optional[Path]*)

Manages data in the keyfile.

get_key(*name: str*) → Optional[str]

Get the named key (token) in the keyfile.

exception nvchecker.api.**GetVersionError**(*msg: str*, ***kwargs: Any*)

An error occurred while getting version information.

Raise this when a known bad situation happens.

Parameters

- **msg** – The error message.
- **kwargs** – Arbitrary additional context for the error.

class nvchecker.api.**EntryWaiter**

async wait(*name: str*) → str

Wait on the `name` entry and return its result (the version string)

set_result(*name: str*, *value: str*) → None

set_exception(*name: str*, *e: Exception*) → None

nvchecker.api.**session: *nvchecker.httpclient.base.BaseSession***

The object to send out HTTP requests, respecting various options in the configuration entry.

class nvchecker.httpclient.base.**Response**(*headers: Mapping[str, str]*, *body: bytes*)

The response of an HTTP request.

body: bytes

headers: Mapping[str, str]

json()

Convert response content to JSON.

class nvchecker.httpclient.base.**BaseSession**

The base class for different HTTP backend.

setup(*concurreny: int = 20*, *timeout: int = 20*) → None

async head(**args*, ***kwargs*)

Shortcut for `HEAD` request.

async get(**args*, ***kwargs*)

Shortcut for GET request.

async post(**args*, ***kwargs*)

Shortcut for POST request.

async request(*url: str*, ***, *method: str*, *headers: Dict[str, str] = {}*, *follow_redirects: bool = True*, *params=()*, *json=None*, *body=None*) → *Response*

nvchecker.api.**proxy = <ContextVar name='proxy' default=None>**

nvchecker.api.**user_agent = <ContextVar name='user_agent' default='lilydjwg/nvchecker 2.11dev'>**

nvchecker.api.**tries = <ContextVar name='tries' default=1>**

nvchecker.api.**verify_cert = <ContextVar name='verify_cert' default=True>**

nvchecker.api.**entry_waiter: contextvars.ContextVar**

This ContextVar contains an *EntryWaiter* instance for waiting on other entries.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

n

INDEX

U

V

W