ahriman

Release 2.13.5

ahriman team

Apr 04, 2024

CONTENTS

Wrapper for managing custom repository inspired by repo-scripts.

CHAPTER

ONE

FEATURES

- Install-configure-forget manager for the very own repository.
- Multi architecture and repository support.
- Dependency manager.
- VCS packages support.
- Official repository support.
- Ability to patch AUR packages and even create package from local PKGBUILDs.
- Various rebuild options with ability to automatically bump package version.
- Sign support with gpg (repository, package), multiple packagers support.
- Triggers for repository updates, e.g. synchronization to remote services (rsync, S3 and GitHub) and report generation (email, html, telegram).
- Repository status interface with optional authorization and control options.

CHAPTER

TWO

LIVE DEMOS

- Build status page. You can login as `demo` user by using `demo` password. Note, however, you will not be able to run tasks. HTTP API documentation is also available.
- Repository index.
- Telegram feed.

CHAPTER

THREE

CONTENTS

3.1 Initial setup

1. Install package as usual.
2. Change settings if required, see *configuration reference* for more details.
3. Perform initial setup:

```
sudo ahriman -a x86_64 -r aur-clone service-setup ...
```

`service-setup` literally does the following steps:

1. Create `/var/lib/ahriman/.makepkg.conf` with `makepkg.conf` overrides if required (at least you might want to set `PACKAGER`):

```
echo 'PACKAGER="John Doe <john@doe.com>"' | sudo -u ahriman tee -a /var/lib/
↪ahriman/.makepkg.conf
```

2. Configure build tools (it is required for correct dependency management system):
    1. Create build command (you can choose any name for command, basically it should be `{name}-{arch}-build`):

```
ln -s /usr/bin/archbuild /usr/local/bin/aur-clone-x86_64-build
```

    2. Create configuration file (same as previous `{name}.conf`):

```
cp /usr/share/devtools/pacman.conf.d/{extra,aur-clone}.conf
```

    3. Change configuration file, add your own repository, add multilib repository etc:

```
echo '[multilib]' | tee -a /usr/share/devtools/pacman.conf.d/aur-clone-x86_
↪64.conf
echo 'Include = /etc/pacman.d/mirrorlist' | tee -a /usr/share/devtools/
↪pacman.conf.d/aur-clone-x86_64.conf

echo '[aur-clone]' | tee -a /usr/share/devtools/pacman.conf.d/aur-clone-x86_
↪64.conf
echo 'SigLevel = Optional TrustAll' | tee -a /usr/share/devtools/pacman.
↪conf.d/aur-clone-x86_64.conf
echo 'Server = file:///var/lib/ahriman/repository/$repo/$arch' | tee -a /
↪usr/share/devtools/pacman.conf.d/aur-clone-x86_64.conf
```

4. Set `build_command` option to point to your command:

```
echo '[build]' | tee -a /etc/ahriman.ini.d/build.ini
echo 'build_command = aur-clone-x86_64-build' | tee -a /etc/ahriman.ini.d/
↪build.ini
```

5. Configure `/etc/sudoers.d/ahriman` to allow running command without a password:

```
echo 'Cmnd_Alias CARCHBUILD_CMD = /usr/local/bin/aur-clone-x86_64-build *'␣
↪| tee -a /etc/sudoers.d/ahriman
echo 'ahriman ALL=(ALL) NOPASSWD:SETENV: CARCHBUILD_CMD' | tee -a /etc/
↪sudoers.d/ahriman
chmod 400 /etc/sudoers.d/ahriman
```

This command supports several arguments, kindly refer to its help message.

4. Start and enable `ahriman@.timer` via `systemctl`:

```
systemctl enable --now ahriman@x86_64-aur-clone.timer
```

5. Start and enable status page:

```
systemctl enable --now ahriman-web
```

6. Add packages by using `ahriman package-add {package}` command:

```
sudo -u ahriman ahriman package-add ahriman --now --refresh
```

The `--refresh` flag is required in order to handle local database update.

3.2 Configuration

Some groups can be specified for each architecture and/or repository separately. E.g. if there are `build` and `build:x86_64` groups it will use an option from `build:x86_64` for the `x86_64` architecture and `build` for any other (architecture specific group has higher priority). In case if both groups are presented, architecture specific options will be merged into global ones overriding them. The order which will be used for option resolution is the following:

1. Repository and architecture specific, e.g. `build:aur-clone:x86_64`.
2. Repository specific, e.g. `build:aur-clone`.
3. Architecture specific, e.g. `build:x86_64`.
4. Default section, e.g. `build`.

There are two variable types which have been added to default ones, they are paths and lists. List values will be read in the same way as shell does:

- By default, it splits value by spaces excluding empty elements.
- In case if quotation mark (" or ') will be found, any spaces inside will be ignored.
- In order to use quotation mark inside value it is required to put it to another quotation mark, e.g. `wor"'"d "with quote"` will be parsed as `["wor'd", "with quote"]` and vice versa.
- Unclosed quotation mark is not allowed and will rise an exception.

Path values, except for casting to `pathlib.Path` type, will be also expanded to absolute paths relative to the configuration path. E.g. if path is set to `ahriman.ini.d/logging.ini` and root configuration path is `/etc/ahriman.ini`, the value will be expanded to `/etc/ahriman.ini.d/logging.ini`. In order to disable path expand, use the full path, e.g. `/etc/ahriman.ini.d/logging.ini`.

Configuration allows string interpolation from environment variables, e.g.:

```
[section]
key = $SECRET
```

will try to read value from `SECRET` environment variable. In case if the required environment variable wasn't found, it will keep original value (i.e. `$SECRET` in the example). Dollar sign can be set as `$$`.

There is also additional subcommand which will allow to validate configuration and print found errors. In order to do so, run `service-config-validate` subcommand, e.g.:

```
ahriman service-config-validate
```

It will check current settings on common errors and compare configuration with known schema.

3.2.1 `settings` group

Base configuration settings.

- `apply_migrations` - perform database migrations on the application start, boolean, optional, default `yes`. Useful if you are using git version. Note, however, that this option must be changed only if you know what to do and going to handle migrations manually.
- `database` - path to the application SQLite database, string, required.
- `include` - path to directory with configuration files overrides, string, optional. Files will be read in alphabetical order.
- `logging` - path to logging configuration, string, required. Check `logging.ini` for reference.

3.2.2 `alpm:*` groups

libalpm and AUR related configuration. Group name can refer to architecture, e.g. `alpm:x86_64` can be used for x86_64 architecture specific settings.

- `database` - path to pacman system database cache, string, required.
- `mirror` - package database mirror used by pacman for synchronization, string, required. This option supports standard pacman substitutions with `$arch` and `$repo`. Note that the mentioned mirror should contain all repositories which are set by `alpm.repositories` option.
- `repositories` - list of pacman repositories, used for package search, space separated list of strings, required.
- `root` - root for alpm library, string, required. In the most cases it must point to the system root.
- `use_ahriman_cache` - use local pacman package cache instead of system one, boolean, required. With this option enabled you might want to refresh database periodically (available as additional flag for some subcommands). If set to `no`, databases must be synchronized manually.

3.2.3 auth group

Base authorization settings. `OAuth` provider requires `aioauth-client` library to be installed.

- `target` - specifies authorization provider, string, optional, default `disabled`. Allowed values are `disabled`, `configuration`, `oauth`.
- `allow_read_only` - allow requesting status APIs without authorization, boolean, required.
- `client_id` - OAuth2 application client ID, string, required in case if `oauth` is used.
- `client_secret` - OAuth2 application client secret key, string, required in case if `oauth` is used.
- `cookie_secret_key` - secret key which will be used for cookies encryption, string, optional. It must be 32 bytes URL-safe base64-encoded and can be generated as following `base64.urlsafe_b64encode(os.urandom(32)).decode("utf8")`. If not set, it will be generated automatically; note, however, that in this case, all sessions will be automatically invalidated during the service restart.
- `max_age` - parameter which controls both cookie expiration and token expiration inside the service in seconds, integer, optional, default is 7 days.
- `oauth_icon` - OAuth2 login button icon, string, optional, default is `google`. Must be valid Bootstrap icon name.
- `oauth_provider` - OAuth2 provider class name as is in `aioauth-client` (e.g. `GoogleClient`, `GithubClient` etc), string, required in case if `oauth` is used.
- `oauth_scopes` - scopes list for OAuth2 provider, which will allow retrieving user email (which is used for checking user permissions), e.g. `https://www.googleapis.com/auth/userinfo.email` for `GoogleClient` or `user:email` for `GithubClient`, space separated list of strings, required in case if `oauth` is used.
- `salt` - additional password hash salt, string, optional.

Authorized users are stored inside internal database, if any of external providers (e.g. `oauth`) are used, the password field for non-service users must be empty.

3.2.4 build:* groups

Build related configuration. Group name can refer to architecture, e.g. `build:x86_64` can be used for x86_64 architecture specific settings.

- `archbuild_flags` - additional flags passed to `archbuild` command, space separated list of strings, optional.
- `build_command` - default build command, string, required.
- `ignore_packages` - list packages to ignore during a regular update (manual update will still work), space separated list of strings, optional.
- `include_debug_packages` - distribute debug packages, boolean, optional, default `yes`.
- `makepkg_flags` - additional flags passed to `makepkg` command, space separated list of strings, optional.
- `makechrootpkg_flags` - additional flags passed to `makechrootpkg` command, space separated list of strings, optional.
- `triggers` - list of `ahriman.core.triggers.Trigger` class implementation (e.g. `ahriman.core.report.ReportTrigger ahriman.core.upload.UploadTrigger`) which will be loaded and run at the end of processing, space separated list of strings, optional. You can also specify triggers by their paths, e.g. `/usr/lib/python3.10/site-packages/ahriman/core/report/report.py.ReportTrigger`. Triggers are run in the order of definition.
- `triggers_known` - optional list of `ahriman.core.triggers.Trigger` class implementations which are not run automatically and used only for trigger discovery and configuration validation.

- `vcs_allowed_age` - maximal age in seconds of the VCS packages before their version will be updated with its remote source, integer, optional, default is 7 days.
- `workers` - list of worker nodes addresses used for build process, space separated list of strings, optional. Each worker address must be valid and reachable URL, e.g. `https://10.0.0.1:8080`. If none set, the build process will be run on the current node. There is also special trigger which loads this value based on the list of the discovered nodes.

3.2.5 `repository` group

Base repository settings.

- `root` - root path for application, string, required.

3.2.6 `sign:*` groups

Settings for signing packages or repository. Group name can refer to architecture, e.g. `sign:x86_64` can be used for x86_64 architecture specific settings.

- `target` - configuration flag to enable signing, space separated list of strings, required. Allowed values are `package` (sign each package separately), `repository` (sign repository database file).
- `key` - default PGP key, string, required. This key will also be used for database signing if enabled.

3.2.7 `status` group

Reporting to web service related settings. In most cases there is fallback to web section settings.

- `enabled` - enable reporting to web service, boolean, optional, default `yes` for backward compatibility.
- `address` - remote web service address with protocol, string, optional. In case of websocket, the `http+unix` scheme and URL encoded address (e.g. `%2Fvar%2Flib%2Fahriman` for `/var/lib/ahriman`) must be used, e.g. `http+unix://%2Fvar%2Flib%2Fahriman%2Fsocket`. In case if none set, it will be guessed from `web` section.
- `password` - password to authorize in web service in order to update service status, string, required in case if authorization enabled.
- `suppress_http_log_errors` - suppress HTTP log errors, boolean, optional, default `no`. If set to `yes`, any HTTP log errors (e.g. if web server is not available, but HTTP logging is enabled) will be suppressed.
- `timeout` - HTTP request timeout in seconds, integer, optional, default is `30`.
- `username` - username to authorize in web service in order to update service status, string, required in case if authorization enabled.

3.2.8 `web` group

Web server settings. This feature requires `aiohttp` libraries to be installed.

- `address` - optional address in form `proto://host:port` (`port` can be omitted in case of default `proto` ports), will be used instead of `http://{host}:{port}` in case if set, string, optional. This option is required in case if `OAuth` provider is used.
- `enable_archive_upload` - allow to upload packages via HTTP (i.e. call of `/api/v1/service/upload` uri), boolean, optional, default `no`.

- `host` - host to bind, string, optional.
- `index_url` - full URL of the repository index page, string, optional.
- `max_body_size` - max body size in bytes to be validated for archive upload, integer, optional. If not set, validation will be disabled.
- `port` - port to bind, integer, optional.
- `service_only` - disable status routes (including logs), boolean, optional, default `no`.
- `static_path` - path to directory with static files, string, required.
- `templates` - path to templates directories, space separated list of strings, required.
- `unix_socket` - path to the listening unix socket, string, optional. If set, server will create the socket on the specified address which can (and will) be used by application. Note, that unlike usual host/port configuration, unix socket allows to perform requests without authorization.
- `unix_socket_unsafe` - set unsafe (o+w) permissions to unix socket, boolean, optional, default `yes`. This option is enabled by default, because it is supposed that unix socket is created in safe environment (only web service is supposed to be used in unsafe), but it can be disabled by configuration.
- `wait_timeout` - wait timeout in seconds, maximum amount of time to be waited before lock will be free, integer, optional.

3.2.9 `keyring` group

Keyring package generator plugin.

- `target` - list of generator settings sections, space separated list of strings, required. It must point to valid section name.

Keyring generator plugin

- `type` - type of the generator, string, optional, must be set to `keyring-generator` if exists.
- `description` - keyring package description, string, optional, default is `repo PGP keyring`, where `repo` is the repository name.
- `homepage` - URL to homepage location if any, string, optional.
- `license` - list of licenses which are applied to this package, space separated list of strings, optional, default is `Unlicense`.
- `package` - keyring package name, string, optional, default is `repo-keyring`, where `repo` is the repository name.
- `packagers` - list of packagers keys, space separated list of strings, optional, if not set, the user keys from database will be used.
- `revoked` - list of revoked packagers keys, space separated list of strings, optional.
- `trusted` - list of master keys, space separated list of strings, optional, if not set, the `key` option from `sign` group will be used.

3.2.10 `mirrorlist` group

Mirrorlist package generator plugin.

- `target` - list of generator settings sections, space separated list of strings, required. It must point to valid section name.

Mirrorlist generator plugin

- `type` - type of the generator, string, optional, must be set to `mirrorlist-generator` if exists.
- `description` - mirrorlist package description, string, optional, default is `repo mirror list for use by pacman`, where `repo` is the repository name.
- `homepage` - URL to homepage location if any, string, optional.
- `license` - list of licenses which are applied to this package, space separated list of strings, optional, default is `Unlicense`.
- `package` - mirrorlist package name, string, optional, default is `repo-mirrorlist`, where `repo` is the repository name.
- `path` - absolute path to generated mirrorlist file, string, optional, default is `/etc/pacman.d/repo-mirrorlist`, where `repo` is the repository name.
- `servers` - list of repository mirrors, space separated list of strings, required.

3.2.11 `remote-pull` group

Remote git source synchronization settings. Unlike `Upload` triggers those triggers are used for PKGBUILD synchronization - fetch from remote repository PKGBUILDs before updating process.

It supports authorization; to do so you'd need to prefix the URL with authorization part, e.g. `https://key:token@github.com/arcan1s/ahriman.git`. It is highly recommended to use application tokens instead of your user authorization details. Alternatively, you can use any other option supported by git, e.g.:

- by SSH key: generate SSH key as `ahriman` user and put public part of it to the repository keys.
- by git credentials helper: consult with the related man page.

Available options are:

- `target` - list of remote pull triggers to be used, space separated list of strings, optional, defaults to `gitremote`. It must point to valid section (or to section with architecture), e.g. `gitremote` must point to either `gitremote` or `gitremote:x86_64` (the one with architecture has higher priority).

Remote pull trigger

- `pull_url` - URL of the remote repository from which PKGBUILDs can be pulled before build process, string, required.
- `pull_branch` - branch of the remote repository from which PKGBUILDs can be pulled before build process, string, optional, default is `master`.

3.2.12 `remote-push` group

Remote git source synchronization settings. Same as remote pull triggers those triggers are used for PKGBUILD synchronization - push updated PKGBUILDs to the remote repository after build process.

It supports authorization; to do so you'd need to prefix the URL with authorization part, e.g. `https://key:token@github.com/arcan1s/ahriman.git`. It is highly recommended to use application tokens instead of your user authorization details. Alternatively, you can use any other option supported by git, e.g.:

- by SSH key: generate SSH key as `ahriman` user and put public part of it to the repository keys.
- by git credentials helper: consult with the related man page.

Available options are:

- `target` - list of remote push triggers to be used, space separated list of strings, optional, defaults to `gitremote`. It must point to valid section (or to section with architecture), e.g. `gitremote` must point to either `gitremote` or `gitremote:x86_64` (the one with architecture has higher priority).

Remote push trigger

- `commit_email` - git commit email, string, optional, default is `ahriman@localhost`.
- `commit_user` - git commit user, string, optional, default is `ahriman`.
- `push_url` - URL of the remote repository to which PKGBUILDs should be pushed after build process, string, required.
- `push_branch` - branch of the remote repository to which PKGBUILDs should be pushed after build process, string, optional, default is `master`.

3.2.13 `report` group

Report generation settings.

- `target` - list of reports to be generated, space separated list of strings, required. It must point to valid section (or to section with architecture), e.g. `somerandomname` must point to existing section, `email` must point to either `email` or `email:x86_64` (the one with architecture has higher priority).

Type will be read from several sources:

- In case if `type` option set inside the section, it will be used.
- Otherwise, it will look for type from section name removing architecture name.
- And finally, it will use section name as type.

`console` type

Section name must be either `console` (plus optional architecture name, e.g. `console:x86_64`) or random name with `type` set.

- `type` - type of the report, string, optional, must be set to `console` if exists.
- `use_utf` - use utf8 symbols in output if set and ascii otherwise, boolean, optional, default `yes`.

`email` type

Section name must be either `email` (plus optional architecture name, e.g. `email:x86_64`) or random name with `type` set.

- `type` - type of the report, string, optional, must be set to `email` if exists.
- `homepage` - link to homepage, string, optional.
- `host` - SMTP host for sending emails, string, required.
- `link_path` - prefix for HTML links, string, required.
- `no_empty_report` - skip report generation for empty packages list, boolean, optional, default `yes`.
- `password` - SMTP password to authenticate, string, optional.
- `port` - SMTP port for sending emails, integer, required.
- `receivers` - SMTP receiver addresses, space separated list of strings, required.
- `sender` - SMTP sender address, string, required.
- `ssl` - SSL mode for SMTP connection, one of `ssl`, `starttls`, `disabled`, optional, default `disabled`.
- `template` - Jinja2 template name, string, required.
- `template_full` - Jinja2 template name for full package description index, string, optional.
- `templates` - path to templates directories, space separated list of strings, required.
- `user` - SMTP user to authenticate, string, optional.

`html` type

Section name must be either `html` (plus optional architecture name, e.g. `html:x86_64`) or random name with `type` set.

- `type` - type of the report, string, optional, must be set to `html` if exists.
- `homepage` - link to homepage, string, optional.
- `link_path` - prefix for HTML links, string, required.
- `path` - path to html report file, string, required.
- `template` - Jinja2 template name, string, required.
- `templates` - path to templates directories, space separated list of strings, required.

`remote-call` type

Section name must be either `remote-call` (plus optional architecture name, e.g. `remote-call:x86_64`) or random name with `type` set.

- `type` - type of the report, string, optional, must be set to `remote-call` if exists.
- `aur` - check for AUR packages updates, boolean, optional, default `no`.
- `local` - check for local packages updates, boolean, optional, default `no`.
- `manual` - update manually built packages, boolean, optional, default `no`.
- `wait_timeout` - maximum amount of time in seconds to be waited before remote process will be terminated, integer, optional, default `-1`.

`telegram` type

Section name must be either `telegram` (plus optional architecture name, e.g. `telegram:x86_64`) or random name with `type` set.

- `type` - type of the report, string, optional, must be set to `telegram` if exists.
- `api_key` - telegram bot API key, string, required. Please refer FAQ about how to create chat and bot
- `chat_id` - telegram chat id, either string with `@` or integer value, required.
- `homepage` - link to homepage, string, optional.
- `link_path` - prefix for HTML links, string, required.
- `template` - Jinja2 template name, string, required.
- `template_type` - `parse_mode` to be passed to telegram API, one of `MarkdownV2`, `HTML`, `Markdown`, string, optional, default `HTML`.
- `templates` - path to templates directories, space separated list of strings, required.
- `timeout` - HTTP request timeout in seconds, integer, optional, default is `30`.

3.2.14 `upload` group

Remote synchronization settings.

- `target` - list of synchronizations to be used, space separated list of strings, required. It must point to valid section (or to section with architecture), e.g. `somerandomname` must point to existing section, `github` must point to one of `github` of `github:x86_64` (with architecture it has higher priority).

Type will be read from several sources:

- In case if `type` option set inside the section, it will be used.
- Otherwise, it will look for type from section name removing architecture name.
- And finally, it will use section name as type.

`github` type

This feature requires GitHub key creation (see below). Section name must be either `github` (plus optional architecture name, e.g. `github:x86_64`) or random name with `type` set.

- `type` - type of the upload, string, optional, must be set to `github` if exists.
- `owner` - GitHub repository owner, string, required.
- `password` - created GitHub API key. In order to create it do the following:
  1. Go to settings page.
  2. Switch to developers settings.
  3. Switch to personal access tokens.
  4. Generate new token. Required scope is `public_repo` (or `repo` for private repository support).
- `repository` - GitHub repository name, string, required. Repository must be created before any action and must have active branch (e.g. with readme).
- `timeout` - HTTP request timeout in seconds, integer, optional, default is `30`.

- `use_full_release_name` - if set to `yes`, the release will contain both repository name and architecture, and only architecture otherwise, boolean, optional, default `no` (legacy behavior).
- `username` - GitHub authorization user, string, required. Basically the same as `owner`.

`remote-service` type

Section name must be either `remote-service` (plus optional architecture name, e.g. `remote-service:x86_64`) or random name with `type` set.

- `type` - type of the report, string, optional, must be set to `remote-service` if exists.
- `timeout` - HTTP request timeout in seconds, integer, optional, default is `30`.

`rsync` type

Requires `rsync` package to be installed. Do not forget to configure ssh for user `ahriman`. Section name must be either `rsync` (plus optional architecture name, e.g. `rsync:x86_64`) or random name with `type` set.

- `type` - type of the upload, string, optional, must be set to `rsync` if exists.
- `command` - rsync command to run, space separated list of string, required.
- `remote` - remote server to rsync (e.g. `1.2.3.4:path/to/sync`), string, required.

`s3` type

Requires `boto3` library to be installed. Section name must be either `s3` (plus optional architecture name, e.g. `s3:x86_64`) or random name with `type` set.

- `type` - type of the upload, string, optional, must be set to `s3` if exists.
- `access_key` - AWS access key ID, string, required.
- `bucket` - bucket name (e.g. `bucket`), string, required.
- `chunk_size` - chunk size for calculating entity tags, integer, optional, default 8 * 1024 * 1024.
- `object_path` - path prefix for stored objects, string, optional. If none set, the prefix as in repository tree will be used.
- `region` - bucket region (e.g. `eu-central-1`), string, required.
- `secret_key` - AWS secret access key, string, required.

3.2.15 `worker` group

This section controls settings for `ahriman.core.distributed.WorkerTrigger` plugin.

- `address` - address of the instance, string, required. Must be reachable for the master instance.
- `identifier` - unique identifier of the instance, string, optional.
- `time_to_live` - amount of time which remote worker will be considered alive in seconds, integer, optional, default is `60`. The ping interval will be set automatically equal this value divided by 4.

3.3 Commands reference

3.3.1 ahriman

ArcH linux ReposItory MANager

```
usage: ahriman [-h] [-a ARCHITECTURE] [-c CONFIGURATION] [--force] [-l LOCK] [--log-
↪handler {console,syslog,journald}]
               [-q] [--report | --no-report] [-r REPOSITORY] [--unsafe] [-V] [--wait-
↪timeout WAIT_TIMEOUT]
               {aur-search,search,help-commands-unsafe,help,help-updates,help-version,
↪version,package-add,add,package-update,package-changes,package-changes-remove,package-
↪remove,remove,package-status,status,package-status-remove,package-status-update,status-
↪update,patch-add,patch-list,patch-remove,patch-set-add,repo-backup,repo-check,check,
↪repo-create-keyring,repo-create-mirrorlist,repo-daemon,daemon,repo-rebuild,rebuild,
↪repo-remove-unknown,remove-unknown,repo-report,report,repo-restore,repo-sign,sign,repo-
↪status-update,repo-sync,sync,repo-tree,repo-triggers,repo-update,update,service-clean,
↪clean,repo-clean,service-config,config,repo-config,service-config-validate,config-
↪validate,repo-config-validate,service-key-import,key-import,service-repositories,
↪service-run,run,service-setup,init,repo-init,repo-setup,setup,service-shell,shell,
↪service-tree-migrate,user-add,user-list,user-remove,web}
               ...
```

Named Arguments

-a, --architecture	filter by target architecture
-c, --configuration	configuration path
	Default: /etc/ahriman.ini
--force	force run, remove file lock
	Default: False
-l, --lock	lock file
	Default: /tmp/ahriman.lock
--log-handler	Possible choices: console, syslog, journald
	explicit log handler specification. If none set, the handler will be guessed from environment
-q, --quiet	force disable any logging
	Default: False
--report, --no-report	force enable or disable reporting to web service
	Default: True
-r, --repository	filter by target repository
--unsafe	allow to run ahriman as non-ahriman user. Some actions might be unavailable
	Default: False

-V, --version show program's version number and exit

--wait-timeout wait for lock to be free. Negative value will lead to immediate application run even if there is lock file. In case of zero value, the application will wait infinitely

Default: -1

command

command Possible choices: aur-search, search, help-commands-unsafe, help, help-updates, help-version, version, package-add, add, package-update, package-changes, package-changes-remove, package-remove, remove, package-status, status, package-status-remove, package-status-update, status-update, patch-add, patch-list, patch-remove, patch-set-add, repo-backup, repo-check, check, repo-create-keyring, repo-create-mirrorlist, repo-daemon, daemon, repo-rebuild, rebuild, repo-remove-unknown, remove-unknown, repo-report, report, repo-restore, repo-sign, sign, repo-status-update, repo-sync, sync, repo-tree, repo-triggers, repo-update, update, service-clean, clean, repo-clean, service-config, config, repo-config, service-config-validate, config-validate, repo-config-validate, service-key-import, key-import, service-repositories, service-run, run, service-setup, init, repo-init, repo-setup, setup, service-shell, shell, service-tree-migrate, user-add, user-list, user-remove, web

command to run

Sub-commands

aur-search (search)

search for package in AUR using API

```
ahriman aur-search [-h] [-e] [--info | --no-info]
                   [--sort-by {description,first_submitted,id,last_modified,maintainer,
↪name,num_votes,out_of_date,package_base,package_base_id,popularity,repository,
↪submitter,url,url_path,version}]
                   search [search ...]
```

Positional Arguments

search search terms, can be specified multiple times, the result will match all terms

Named Arguments

-e, --exit-code return non-zero exit status if result is empty

Default: False

--info, --no-info show additional package information

Default: False

--sort-by Possible choices: description, first_submitted, id, last_modified, maintainer, name, num_votes, out_of_date, package_base, package_base_id, popularity, repository, submitter, url, url_path, version

sort field by this field. In case if two packages have the same value of the specified field, they will be always sorted by name

Default: "name"

help-commands-unsafe

list unsafe commands as defined in default args

```
ahriman help-commands-unsafe [-h] [subcommand ...]
```

Positional Arguments

subcommand instead of showing commands, just test command line for unsafe subcommand and return 0 in case if command is safe and 1 otherwise

help

show help message for application or command and exit

```
ahriman help [-h] [subcommand]
```

Positional Arguments

subcommand show help message for specific command

help-updates

request AUR for current version and compare with current service version

```
ahriman help-updates [-h] [-e]
```

Named Arguments

-e, --exit-code return non-zero exit code if updates available

Default: False

help-version (version)

print application and its dependencies versions

```
ahriman help-version [-h]
```

package-add (add, package-update)

add existing or new package to the build queue

```
ahriman package-add [-h] [--dependencies | --no-dependencies] [-e] [--increment | --no-
↪increment] [-n] [-y]
                    [-s {auto,archive,aur,directory,local,remote,repository}] [-u␣
↪USERNAME] [-v VARIABLE]
                    package [package ...]
```

Positional Arguments

package package source (base name, path to local files, remote URL)

Named Arguments

--dependencies, --no-dependencies process missing package dependencies

Default: True

-e, --exit-code return non-zero exit status if result is empty

Default: False

--increment, --no-increment increment package release (pkgrel) version on duplicate

Default: True

-n, --now run update function after

Default: False

-y, --refresh download fresh package databases from the mirror before actions, -yy to force refresh even if up to date

Default: False

-s, --source Possible choices: auto, archive, aur, directory, local, remote, repository

explicitly specify the package source for this command

Default: auto

-u, --username build as user

-v, --variable apply specified makepkg variables to the next build

This subcommand should be used for new package addition. It also supports flag –now in case if you would like to build the package immediately. You can add new package from one of supported sources: 1) if it is already built package you can specify the path to the archive; 2) you can also add built packages from the directory (e.g. during the migration from another repository source); 3) it is also possible to add package from local PKGBUILD, but in this case it will

be ignored during the next automatic updates; 4) ahriman supports downloading archives from remote (e.g. HTTP) sources; 5) and finally you can add package from AUR.

package-changes

retrieve package changes stored in database

```
ahriman package-changes [-h] [-e] package
```

Positional Arguments

package package base

Named Arguments

-e, --exit-code return non-zero exit status if result is empty

Default: False

This feature requests package status from the web interface if it is available.

package-changes-remove

remove the package changes stored remotely

```
ahriman package-changes-remove [-h] package
```

Positional Arguments

package package base

package-remove (remove)

remove package from the repository

```
ahriman package-remove [-h] package [package ...]
```

Positional Arguments

package package name or base

package-status (status)

request status of the package

```
ahriman package-status [-h] [--ahriman] [-e] [--info | --no-info] [-s {unknown,pending,
↪building,failed,success}]
                       [package ...]
```

Positional Arguments

package	filter status by package base

Named Arguments

--ahriman	get service status itself
	Default: False
-e, --exit-code	return non-zero exit status if result is empty
	Default: False
--info, --no-info	show additional package information
	Default: False
-s, --status	Possible choices: unknown, pending, building, failed, success
	filter packages by status

This feature requests package status from the web interface if it is available.

package-status-remove

remove the package from the status page

```
ahriman package-status-remove [-h] package [package ...]
```

Positional Arguments

package	remove specified packages from status page

Please note that this subcommand does not remove the package itself, it just clears the status page.

package-status-update (status-update)

update package status on the status page

```
ahriman package-status-update [-h] [-s {unknown,pending,building,failed,success}]␣
↪[package ...]
```

Positional Arguments

package	set status for specified packages. If no packages supplied, service status will be updated

Named Arguments

-s, --status	Possible choices: unknown, pending, building, failed, success new package build status Default: success

patch-add

create or update patched PKGBUILD function or variable

```
ahriman patch-add [-h] package variable [patch]
```

Positional Arguments

package	package base
variable	PKGBUILD variable or function name. If variable is a function, it must end with ()
patch	path to file which contains function or variable value. If not set, the value will be read from stdin

Unlike `patch-set-add`, this function allows to patch only one PKGBUILD function, e.g. typing `ahriman patch-add ahriman pkgver` it will change the `pkgver` inside PKGBUILD, typing `ahriman patch-add ahriman build()` it will change `build()` function inside PKGBUILD

patch-list

list available patches for the package

```
ahriman patch-list [-h] [-e] [-v VARIABLE] [package]
```

Positional Arguments

package package base

Named Arguments

-e, --exit-code return non-zero exit status if result is empty

Default: False

-v, --variable if set, show only patches for specified PKGBUILD variables

patch-remove

remove patches for the package

```
ahriman patch-remove [-h] [-v VARIABLE] package
```

Positional Arguments

package package base

Named Arguments

-v, --variable should be used for single-function patches in case if you wold like to remove only specified PKGBUILD variables. In case if not set, it will remove all patches related to the package

patch-set-add

create or update source patches

```
ahriman patch-set-add [-h] [-t TRACK] package
```

Positional Arguments

package path to directory with changed files for patch addition/update

Named Arguments

-t, --track files which has to be tracked

Default: ['*.diff*', '.patch']

In order to add a patch set for the package you will need to clone the AUR package manually, add required changes (e.g. external patches, edit PKGBUILD) and run command, e.g. `ahriman patch-set-add path/to/directory`. By default it tracks *.patch and *.diff files, but this behavior can be changed by using –track option

repo-backup

backup repository settings and database

```
ahriman repo-backup [-h] path
```

Positional Arguments

path path of the output archive

repo-check (check)

check for packages updates. Same as repo-update –dry-run –no-manual

```
ahriman repo-check [-h] [--changes | --no-changes] [-e] [--vcs | --no-vcs] [-y] [package␣
↪...]
```

Positional Arguments

package filter check by package base

Named Arguments

--changes, --no-changes calculate changes from the latest known commit if available. Only applicable in dry run mode

Default: True

-e, --exit-code return non-zero exit status if result is empty

Default: False

--vcs, --no-vcs fetch actual version of VCS packages

Default: True

-y, --refresh download fresh package databases from the mirror before actions, -yy to force refresh even if up to date

Default: False

repo-create-keyring

create package which contains list of trusted keys as set by configuration. Note, that this action will only create package, the package itself has to be built manually

```
ahriman repo-create-keyring [-h]
```

repo-create-mirrorlist

create package which contains list of available mirrors as set by configuration. Note, that this action will only create package, the package itself has to be built manually

```
ahriman repo-create-mirrorlist [-h]
```

repo-daemon (daemon)

start process which periodically will run update process

```
ahriman repo-daemon [-h] [-i INTERVAL] [--aur | --no-aur] [--changes | --no-changes]
                    [--dependencies | --no-dependencies] [--dry-run] [--increment | --no-
↪increment]
                    [--local | --no-local] [--manual | --no-manual] [--partitions | --no-
↪partitions] [-u USERNAME]
                    [--vcs | --no-vcs] [-y]
```

Named Arguments

-i, --interval interval between runs in seconds

Default: 43200

--aur, --no-aur enable or disable checking for AUR updates

Default: True

--changes, --no-changes calculate changes from the latest known commit if available. Only applicable in dry run mode

Default: True

--dependencies, --no-dependencies process missing package dependencies

Default: True

--dry-run just perform check for updates, same as check command

Default: False

--increment, --no-increment increment package release (pkgrel) on duplicate

Default: True

--local, --no-local enable or disable checking of local packages for updates

Default: True

--manual, --no-manual include or exclude manual updates

Default: True

--partitions, --no-partitions instead of updating whole repository, split updates into chunks

Default: True

-u, --username build as user

--vcs, --no-vcs fetch actual version of VCS packages

Default: True

-y, --refresh download fresh package databases from the mirror before actions, -yy to force refresh even if up to date

Default: False

repo-rebuild (rebuild)

force rebuild whole repository

```
ahriman repo-rebuild [-h] [--depends-on DEPENDS_ON] [--dry-run] [--from-database] [--
↪increment | --no-increment] [-e]
                     [-s {unknown,pending,building,failed,success}] [-u USERNAME]
```

Named Arguments

--depends-on only rebuild packages that depend on specified packages

--dry-run just perform check for packages without rebuild process itself

Default: False

--from-database read packages from database instead of filesystem. This feature in particular is required in case if you would like to restore repository from another repository instance. Note, however, that in order to restore packages you need to have original ahriman instance run with web service and have run repo-update at least once.

Default: False

--increment, --no-increment increment package release (pkgrel) on duplicate

Default: True

-e, --exit-code return non-zero exit status if result is empty

Default: False

-s, --status Possible choices: unknown, pending, building, failed, success

filter packages by status. Requires –from-database to be set

-u, --username build as user

repo-remove-unknown (remove-unknown)

remove packages which are missing in AUR and do not have local PKGBUILDs

```
ahriman repo-remove-unknown [-h] [--dry-run]
```

Named Arguments

--dry-run just perform check for packages without removal

Default: False

repo-report (report)

generate repository report according to current settings

```
ahriman repo-report [-h]
```

Create and/or update repository report as configured.

repo-restore

restore settings and database

```
ahriman repo-restore [-h] [-o OUTPUT] path
```

Positional Arguments

path path of the input archive

Named Arguments

-o, --output root path of the extracted files

Default: /

repo-sign (sign)

(re-)sign packages and repository database according to current settings

```
ahriman repo-sign [-h] [package ...]
```

Positional Arguments

package	sign only specified packages

Sign repository and/or packages as configured.

repo-status-update

update repository status on the status page

```
ahriman repo-status-update [-h] [-s {unknown,pending,building,failed,success}]
```

Named Arguments

-s, --status	Possible choices: unknown, pending, building, failed, success new status Default: success

repo-sync (sync)

sync repository files to remote server according to current settings

```
ahriman repo-sync [-h]
```

Synchronize the repository to remote services as configured.

repo-tree

dump repository tree based on packages dependencies

```
ahriman repo-tree [-h] [-p PARTITIONS]
```

Named Arguments

-p, --partitions	also divide packages by independent partitions Default: 1

repo-triggers

run triggers on empty build result as configured by settings

```
ahriman repo-triggers [-h] [trigger ...]
```

Positional Arguments

trigger instead of running all triggers as set by configuration, just process specified ones in order of mention

repo-update (update)

check for packages updates and run build process if requested

```
ahriman repo-update [-h] [--aur | --no-aur] [--changes | --no-changes] [--dependencies |␣
↪--no-dependencies] [--dry-run]
                    [-e] [--increment | --no-increment] [--local | --no-local] [--manual␣
↪| --no-manual] [-u USERNAME]
                    [--vcs | --no-vcs] [-y]
                    [package ...]
```

Positional Arguments

package filter check by package base

Named Arguments

--aur, --no-aur enable or disable checking for AUR updates

Default: True

--changes, --no-changes calculate changes from the latest known commit if available. Only applicable in dry run mode

Default: True

--dependencies, --no-dependencies process missing package dependencies

Default: True

--dry-run just perform check for updates, same as check command

Default: False

-e, --exit-code return non-zero exit status if result is empty

Default: False

--increment, --no-increment increment package release (pkgrel) on duplicate

Default: True

--local, --no-local enable or disable checking of local packages for updates

Default: True

--manual, --no-manual include or exclude manual updates

Default: True

-u, --username build as user

--vcs, --no-vcs fetch actual version of VCS packages

Default: True

-y, --refresh download fresh package databases from the mirror before actions, -yy to force refresh even if up to date

Default: False

service-clean (clean, repo-clean)

remove local caches

```
ahriman service-clean [-h] [--cache | --no-cache] [--chroot | --no-chroot] [--manual | --
↪no-manual]
                      [--packages | --no-packages] [--pacman | --no-pacman]
```

Named Arguments

--cache, --no-cache clear directory with package caches

Default: False

--chroot, --no-chroot clear build chroot

Default: False

--manual, --no-manual clear manually added packages queue

Default: False

--packages, --no-packages clear directory with built packages

Default: False

--pacman, --no-pacman clear directory with pacman local database cache

Default: False

The subcommand clears every temporary directories (builds, caches etc). Normally you should not run this command manually. Also in case if you are going to clear the chroot directories you will need root privileges.

service-config (config, repo-config)

dump configuration for the specified architecture

```
ahriman service-config [-h] [--info | --no-info] [--secure | --no-secure] [section] [key]
```

Positional Arguments

section	filter settings by section
key	filter settings by key

Named Arguments

--info, --no-info	show additional information, e.g. configuration files Default: True
--secure, --no-secure	hide passwords and secrets from output Default: True

service-config-validate (config-validate, repo-config-validate)

validate configuration and print found errors

```
ahriman service-config-validate [-h] [-e]
```

Named Arguments

-e, --exit-code	return non-zero exit status if configuration is invalid Default: False

service-key-import (key-import)

import PGP key from public sources to the repository user

```
ahriman service-key-import [-h] [--key-server KEY_SERVER] key
```

Positional Arguments

key	PGP key to import from public server

Named Arguments

--key-server	key server for key import Default: “keyserver.ubuntu.com”

By default ahriman runs build process with package sources validation (in case if signature and keys are available in PKGBUILD). This process will fail in case if key is not known for build user. This subcommand can be used in order to import the PGP key to user keychain.

service-repositories

list all available repositories

```
ahriman service-repositories [-h] [--id-only | --no-id-only]
```

Named Arguments

--id-only, --no-id-only show machine readable identifier instead

Default: False

service-run (run)

run multiple commands on success run of the previous command

```
ahriman service-run [-h] command [command ...]
```

Positional Arguments

command command to be run (quoted) without `ahriman`

Commands must be quoted by using usual bash rules. Processes will be spawned under the same user as this command

service-setup (init, repo-init, repo-setup, setup)

create initial service configuration, requires root

```
ahriman service-setup [-h] [--build-as-user BUILD_AS_USER] [--from-configuration FROM_
↪CONFIGURATION]
                      [--generate-salt | --no-generate-salt] [--makeflags-jobs | --no-
↪makeflags-jobs] [--mirror MIRROR]
                      [--multilib | --no-multilib] --packager PACKAGER [--server SERVER]␣
↪[--sign-key SIGN_KEY]
                      [--sign-target {disabled,packages,repository}] [--web-port WEB_
↪PORT]
                      [--web-unix-socket WEB_UNIX_SOCKET]
```

Named Arguments

--build-as-user force makepkg user to the specific one

--from-configuration path to default devtools pacman configuration

Default: /usr/share/devtools/pacman.conf.d/extra.conf

--generate-salt, --no-generate-salt generate salt for user passwords

Default: False

--makeflags-jobs, --no-makeflags-jobs append MAKEFLAGS variable with parallelism set to number of cores

Default: True

--mirror use the specified explicitly mirror instead of including mirrorlist

--multilib, --no-multilib add or do not multilib repository

Default: True

--packager packager name and email

--server server to be used for devtools. If none set, local files will be used

--sign-key sign key id

--sign-target Possible choices: disabled, packages, repository

sign options

--web-port port of the web service

--web-unix-socket path to unix socket used for interprocess communications

Create _minimal_ configuration for the service according to provided options.

service-shell (shell)

drop into python shell

```
ahriman service-shell [-h] [code]
```

Positional Arguments

code instead of dropping into shell, just execute the specified code

service-tree-migrate

migrate repository tree between versions

```
ahriman service-tree-migrate [-h]
```

user-add

update user for web services with the given password and role. In case if password was not entered it will be asked interactively

```
ahriman user-add [-h] [--key KEY] [--packager PACKAGER] [-p PASSWORD] [-R {unauthorized,
↪read,reporter,full}] username
```

Positional Arguments

username	username for web service

Named Arguments

--key	optional PGP key used by this user. The private key must be imported
--packager	optional packager id used for build process in form of *Name Surname <mail@example.com>*
-p, --password	user password. Blank password will be treated as empty password, which is in particular must be used for OAuth2 authorization type.
-R, --role	Possible choices: unauthorized, read, reporter, full user access level Default: read

user-list

list users from the user mapping and their roles

```
ahriman user-list [-h] [-e] [-R {unauthorized,read,reporter,full}] [username]
```

Positional Arguments

username	filter users by username

Named Arguments

-e, --exit-code	return non-zero exit status if result is empty Default: False
-R, --role	Possible choices: unauthorized, read, reporter, full filter users by role

user-remove

remove user from the user mapping and update the configuration

```
ahriman user-remove [-h] username
```

Positional Arguments

username username for web service

web

start web server

```
ahriman web [-h]
```

Argument list can also be read from file by using @ prefix.

3.4 FAQ

3.4.1 General topics

What is the purpose of the project

This project has been created in order to maintain self-hosted Arch Linux user repository without manual intervention - checking for updates and building packages.

How to install ahriman

TL;DR

```
yay -S ahriman
ahriman -a x86_64 -r aur-clone service-setup --packager "ahriman bot <ahriman@example.
↪com>"
systemctl enable --now ahriman@x86_64-aur-clone.timer
```

Long answer

The idea is to install the package as usual, create working directory tree, create configuration for `sudo` and `devtools`. Detailed description of the setup instruction can be found *here*.

Run as daemon

The alternative way (though not recommended) is to run service instead of timer:

```
systemctl enable --now ahriman-daemon@x86_64-aur-clone
```

How to validate settings

There is special command which can be used in order to validate current configuration:

```
ahriman service-config-validate --exit-code
```

This command will print found errors, based on cerberus, e.g.:

```
auth
                ssalt: unknown field
                target: none or more than one rule validate
                        oneof definition 0: unallowed value mapping
                        oneof definition 1: field 'salt' is required
                        oneof definition 2: unallowed value mapping
                        oneof definition 2: field 'salt' is required
                        oneof definition 2: field 'client_id' is required
                        oneof definition 2: field 'client_secret' is required
gitremote
                pull_url: unknown field
```

If an additional flag `--exit-code` is supplied, the application will return non-zero exit code, which can be used partially in scripts.

What does "architecture specific" mean / How to configure for different architectures

Some sections can be configured per architecture. The service will merge architecture specific values into common settings. In order to specify settings for specific architecture you must point it in section name.

For example, the section

```
[build]
build_command = extra-x86_64-build
```

states that default build command is `extra-x86_64-build`. But if there is section

```
[build:i686]
build_command = extra-i686-build
```

the `extra-i686-build` command will be used for `i686` architecture. You can also override settings for different repositories and architectures; in this case section names will be `build:aur-clone` (repository name only) and `build:aur-clone:i686` (both repository name and architecture).

How to generate build reports

Normally you would probably like to generate only one report for the specific type, e.g. only one email report. In order to do so you will need to have the following configuration:

```
[report]
target = email

[email]
...
```

or in case of multiple architectures and *different* reporting settings:

```
[report]
target = email

[email:i686]
...

[email:x86_64]
...
```

But for some cases you would like to have multiple different reports with the same type (e.g. sending different templates to different addresses). For these cases you will need to specify section name in target and type in section, e.g. the following configuration can be used:

```
[report]
target = email_1 email_2

[email_1]
type = email
...

[email_2]
type = email
...
```

How to add new package

```
sudo -u ahriman ahriman package-add ahriman --now
```

`--now` flag is totally optional and just run `repo-update` subcommand after the registering the new package. Thus the extended flow is the following:

```
sudo -u ahriman ahriman package-add ahriman
sudo -u ahriman ahriman repo-update
```

How to build package from local PKGBUILD

TL;DR

```
sudo -u ahriman ahriman package-add /path/to/local/directory/with/PKGBUILD --now
```

Before using this command you will need to create local directory, put `PKGBUILD` there and generate `.SRCINFO` by using `makepkg --printsrcinfo > .SRCINFO` command. These packages will be stored locally and *will be ignored* during automatic update; in order to update the package you will need to run `package-add` command again.

How to copy package from another repository

As simple as add package from archive. Considering case when you would like to copy package `package` with version `ver-rel` from repository `source-repository` to `target-respository` (same architecture), the command will be following:

```
sudo -u ahriman ahriman -r target-repository package-add /var/lib/ahriman/repository/
↪source-repository/x86_64/package-ver-rel-x86_64.pkg.tar.zst
```

In addition, you can remove source package as usual later.

This feature in particular useful if for managing multiple repositories like `[testing]` and `[extra]`.

How to fetch PKGBUILDs from remote repository

For that purpose you could use `RemotePullTrigger` trigger. To do so you will need to configure trigger as following:

```
[remote-pull]
target = gitremote

[gitremote]
pull_url = https://github.com/username/repository
```

During the next application run it will fetch repository from the specified URL and will try to find packages there which can be used as local sources.

This feature can be also used to build packages which are not listed in AUR, the example of the feature use can be found here.

How to push updated PKGBUILDs to remote repository

For that purpose you'd need to use another trigger called `RemotePushTrigger`. Configure trigger as following:

```
[remote-push]
target = gitremote

[gitremote]
push_url = https://github.com/username/repository
```

Unlike `RemotePullTrigger` trigger, the `RemotePushTrigger` more likely will require authorization. It is highly recommended to use application tokens for that instead of using your password (e.g. for GitHub you can generate tokens here with scope `public_repo`). Authorization can be supplied by using authorization part of the URL, e.g. `https://key:token@github.com/username/repository`.

How to change PKGBUILDs before build

Well it is supported also. The recommended way is to patch specific function, e.g. by running

```
sudo -u ahriman ahriman patch-add ahriman version
```

This command will prompt for new value of the PKGBUILD variable `version`. You can also write it to file and read from it:

```
sudo -u ahriman ahriman patch-add ahriman version version.patch
```

Alternatively you can create full-diff patches, which are calculated by using `git diff` from current PKGBUILD master branch:

1. Clone sources from AUR.
2. Make changes you would like to (e.g. edit `PKGBUILD`, add external patches).
3. Run command

   ```
   sudo -u ahriman ahriman patch-set-add /path/to/local/directory/with/PKGBUILD
   ```

The last command will calculate diff from current tree to the `HEAD` and will store it locally. Patches will be applied on any package actions (e.g. it can be used for dependency management).

It is also possible to create simple patch during package addition, e.g.:

```
sudo -u ahriman ahriman package-add ahriman --variable PKGEXT=.pkg.tar.xz
```

The `--variable` argument accepts variables in shell like format: quotation and lists are supported as usual, but functions are not. This feature is useful in particular in order to override specific makepkg variables during build.

How to build package from official repository

It is the same as adding any other package, but due to restrictions you must specify source explicitly, e.g.:

```
sudo -u ahriman ahriman package-add pacman --source repository
```

This feature is heavily depends on local pacman cache. In order to use this feature it is recommended to either run `pacman -Sy` before the interaction or use internal application cache with `--refresh` flag.

Package build fails because it cannot validate PGP signature of source files

TL;DR

```
sudo -u ahriman ahriman service-key-import ...
```

How to update VCS packages

Normally the service handles VCS packages correctly, however it requires additional dependencies:

```
pacman -S breezy darcs mercurial subversion
```

How to review changes before build

In this scenario, the update process must be separated into several stages. First, it is required to check updates:

```
sudo -u ahriman ahriman repo-check
```

During the check process, the service will generate changes from the last known commit and will send it to remote service. In order to verify source files changes, the web interface or special subcommand can be used:

```
ahriman package-changes ahriman
```

After validation, the operator can run update process with approved list of packages, e.g.:

```
sudo -u ahriman ahriman repo-update ahriman
```

How to remove package

```
sudo -u ahriman ahriman package-remove ahriman
```

Also, there is command `repo-remove-unknown` which checks packages in AUR and local storage and removes ones which have been removed.

Remove commands also remove any package files (patches, caches etc).

How to sign repository

Repository sign feature is available in several configurations. The recommended way is just to sign repository database file by single key instead of trying to sign each package. However, the steps are pretty same, just configuration is a bit different. For more details about options kindly refer to *configuration reference*.

1. First you would need to create the key on your local machine:

   ```
   gpg --full-generate-key
   ```

   This command will prompt you for several questions. Most of them may be left default, but you will need to fill real name and email address with some data. Because at the moment the service doesn't support passphrases, it must be left blank.

2. The command above will generate key and print its fingerprint, something like `8BE91E5A773FB48AC05CC1EDBED105AED6246B39`. Copy it.

3. Export your private key by using the fingerprint above:

   ```
   gpg --export-secret-keys -a 8BE91E5A773FB48AC05CC1EDBED105AED6246B39 > repository-
   ↪key.gpg
   ```

4. Copy the specified key to the build machine (i.e. where the service is running).

5. Import the specified key to the service user:

```
sudo -u ahriman gpg --import repository-key.gpg
```

   Don't forget to remove the key from filesystem after import.

6. Change trust level to `ultimate`:

```
sudo -u ahriman gpg --edit-key 8BE91E5A773FB48AC05CC1EDBED105AED6246B39
```

   The command above will drop you into gpg shell, in which you will need to type `trust`, choose `5 = I trust ultimately`, confirm and exit `quit`.

7. Proceed with service configuration according to the *configuration*:

```
[sign]
target = repository
key = 8BE91E5A773FB48AC05CC1EDBED105AED6246B39
```

How to rebuild packages after library update

TL;DR

```
sudo -u ahriman ahriman repo-rebuild --depends-on python
```

You can even rebuild the whole repository (which is particular useful in case if you would like to change packager) if you do not supply `--depends-on` option. This action will automatically increment `pkgrel` value; in case if you don't want to, the `--no-increment` option has to be supplied.

However, note that you do not need to rebuild repository in case if you just changed signing option, just use `repo-sign` command instead.

How to install built packages

Add the following lines to your `pacman.conf`:

```
[repository]
Server = file:///var/lib/ahriman/repository/$repo/$arch
```

(You might need to add `SigLevel` option according to the pacman documentation.)

How to serve repository

Easy. For example, nginx configuration (without SSL) will look like:

```
server {
    listen 80;
    server_name repo.example.com;

    location / {
        autoindex on;
        root /var/lib/ahriman/repository;
    }
}
```

Example of the status page configuration is the following (status service is using 8080 port):

```
server {
    listen 80;
    server_name builds.example.com;

    location / {
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarder-Proto $scheme;

        proxy_pass http://127.0.0.1:8080;
    }
}
```

Some more examples can be found in configuration recipes.

3.4.2 Docker image

We provide official images which can be found under:

- docker registry `arcan1s/ahriman`;
- ghcr.io registry `ghcr.io/arcan1s/ahriman`.

These images are totally identical.

Docker image is being updated on each commit to master as well as on each version. If you would like to use last (probably unstable) build you can use `edge` tag or `latest` for any tagged versions; otherwise you can use any version tag available.

The default action (in case if no arguments provided) is `repo-update`. Basically the idea is to run container, e.g.:

```
docker run --privileged -v /path/to/local/repo:/var/lib/ahriman arcan1s/ahriman:latest
```

`--privileged` flag is required to make mount possible inside container. In order to make data available outside of container, you would need to mount local (parent) directory inside container by using `-v /path/to/local/repo:/var/lib/ahriman` argument, where `/path/to/local/repo` is a path to repository on local machine. In addition, you can pass own configuration overrides by using the same `-v` flag, e.g.:

```
docker run --privileged -v /path/to/local/repo:/var/lib/ahriman -v /path/to/overrides/
↪overrides.ini:/etc/ahriman.ini.d/10-overrides.ini arcan1s/ahriman:latest
```

The action can be specified during run, e.g.:

```
docker run --privileged -v /path/to/local/repo:/var/lib/ahriman arcan1s/ahriman:latest␣
↪package-add ahriman --now
```

For more details please refer to the docker FAQ.

Environment variables

The following environment variables are supported:

- `AHRIMAN_ARCHITECTURE` - architecture of the repository, default is `x86_64`.
- `AHRIMAN_DEBUG` - if set all commands will be logged to console.
- `AHRIMAN_FORCE_ROOT` - force run ahriman as root instead of guessing by subcommand.
- `AHRIMAN_HOST` - host for the web interface, default is `0.0.0.0`.
- `AHRIMAN_MULTILIB` - if set (default) multilib repository will be used, disabled otherwise.
- `AHRIMAN_OUTPUT` - controls logging handler, e.g. `syslog`, `console`. The name must be found in logging configuration. Note that if `syslog` handler is used you will need to mount `/dev/log` inside container because it is not available there.
- `AHRIMAN_PACKAGER` - packager name from which packages will be built, default is `ahriman bot <ahriman@example.com>`.
- `AHRIMAN_PACMAN_MIRROR` - override pacman mirror server if set.
- `AHRIMAN_PORT` - HTTP server port if any, default is empty.
- `AHRIMAN_POSTSETUP_COMMAND` - if set, the command which will be called (as root) after the setup command, but before any other actions.
- `AHRIMAN_PRESETUP_COMMAND` - if set, the command which will be called (as root) right before the setup command.
- `AHRIMAN_REPOSITORY` - repository name, default is `aur-clone`.
- `AHRIMAN_REPOSITORY_SERVER` - optional override for the repository URL. Useful if you would like to download packages from remote instead of local filesystem.
- `AHRIMAN_REPOSITORY_ROOT` - repository root. Because of filesystem rights it is required to override default repository root. By default, it uses `ahriman` directory inside ahriman's home, which can be passed as mount volume.
- `AHRIMAN_UNIX_SOCKET` - full path to unix socket which is used by web server, default is empty. Note that more likely you would like to put it inside `AHRIMAN_REPOSITORY_ROOT` directory (e.g. `/var/lib/ahriman/ahriman/ahriman-web.sock`) or to `/tmp`.
- `AHRIMAN_USER` - ahriman user, usually must not be overwritten, default is `ahriman`.
- `AHRIMAN_VALIDATE_CONFIGURATION` - if set (default) validate service configuration.

You can pass any of these variables by using `-e` argument, e.g.:

```
docker run --privileged -e AHRIMAN_PORT=8080 -v /path/to/local/repo:/var/lib/ahriman␣
↪arcan1s/ahriman:latest
```

Daemon service

There is special `repo-daemon` subcommand which emulates systemd timer and will perform repository update periodically:

```
docker run --privileged -v /path/to/local/repo:/var/lib/ahriman arcan1s/ahriman:latest␣
↪repo-daemon
```

This command uses same rules as `repo-update`, thus, e.g. requires `--privileged` flag. Check also examples.

Web service setup

For that you would need to have web container instance running forever; it can be achieved by the following command:

```
docker run --privileged -p 8080:8080 -e AHRIMAN_PORT=8080 -e AHRIMAN_UNIX_SOCKET=/var/
↪lib/ahriman/ahriman/ahriman-web.sock -v /path/to/local/repo:/var/lib/ahriman arcan1s/
↪ahriman:latest
```

Note about `AHRIMAN_PORT` environment variable which is required in order to enable web service. An additional port bind by `-p 8080:8080` is required to pass docker port outside of container.

The `AHRIMAN_UNIX_SOCKET` variable is not required, however, highly recommended as it can be used for interprocess communications. If you set this variable you would like to be sure that this path is available outside of container if you are going to use multiple docker instances.

If you are using `AHRIMAN_UNIX_SOCKET` variable, for every next container run it has to be passed also, e.g.:

```
docker run --privileged -e AHRIMAN_UNIX_SOCKET=/var/lib/ahriman/ahriman/ahriman-web.sock␣
↪-v /path/to/local/repo:/var/lib/ahriman arcan1s/ahriman:latest
```

Otherwise, you would need to pass `AHRIMAN_PORT` and mount container network to the host system (`--net=host`), e.g.:

```
docker run --privileged --net=host -e AHRIMAN_PORT=8080 -v /path/to/local/repo:/var/lib/
↪ahriman arcan1s/ahriman:latest
```

Simple server with authentication can be found in examples too.

Mutli-repository web service

Idea is pretty same as to just run web service. However, it is required to run setup commands for each repository, except for one which is specified by `AHRIMAN_REPOSITORY` and `AHRIMAN_ARCHITECTURE` variables.

In order to create configuration for additional repositories, the `AHRIMAN_POSTSETUP_COMMAND` variable should be used, e.g.:

```
docker run --privileged -p 8080:8080 -e AHRIMAN_PORT=8080 -e AHRIMAN_UNIX_SOCKET=/var/
↪lib/ahriman/ahriman/ahriman-web.sock -e AHRIMAN_POSTSETUP_COMMAND="ahriman --
↪architecture x86_64 --repository aur-clone-v2 service-setup --build-as-user ahriman --
↪packager 'ahriman bot <ahriman@example.com>'" -v /path/to/local/repo:/var/lib/ahriman␣
↪arcan1s/ahriman:latest
```

The command above will also create configuration for the repository named `aur-clone-v2`.

Note, however, that the command above is only required in case if the service is going to be used to run subprocesses. Otherwise, everything else (web interface, status, etc) will be handled as usual.

Configuration example.

3.4.3 Non-x86_64 architecture setup

The following section describes how to setup ahriman with architecture different from x86_64, as example i686. For most cases you have base repository available, e.g. archlinux32 repositories for i686 architecture; in case if base repository is not available, steps are a bit different, however, idea remains the same.

The example of setup with docker compose can be found here.

Physical server setup

In this example we are going to use files and packages which are provided by official repositories of the used architecture. Note, that versions might be different, thus you need to find correct versions on the distribution web site, e.g. archlinux32 packages.

1. First, considering having base Arch Linux system, we need to install keyring for the specified repositories, e.g.:

   ```
   wget https://pool.mirror.archlinux32.org/i686/core/archlinux32-keyring-20230705-1.0-
   ↪any.pkg.tar.zst
   pacman -U archlinux32-keyring-20230705-1.0-any.pkg.tar.zst
   ```

2. In order to run `devtools` scripts for custom architecture they also need specific `makepkg` configuration, it can be retrieved by installing the `devtools` package of the distribution, e.g.:

   ```
   wget https://pool.mirror.archlinux32.org/i686/extra/devtools-20221208-1.2-any.pkg.
   ↪tar.zst
   pacman -U devtools-20221208-1.2-any.pkg.tar.zst
   ```

   Alternatively, you can create your own `makepkg` configuration and save it as `/usr/share/devtools/makepkg.conf.d/i686.conf`.

3. Setup repository as usual:

   ```
   ahriman -a i686 service-setup --mirror 'https://de.mirror.archlinux32.org/$arch/
   ↪$repo'--no-multilib ...
   ```

   In addition to usual options, you need to specify the following options:

   - `--mirror` - link to the mirrors which will be used instead of official repositories.
   - `--no-multilib` - in the example we are using i686 architecture for which multilib repository doesn't exist.

4. That's all Folks!

Docker container setup

There are two possible ways to achieve same setup, by using docker container. The first one is just mount required files inside container and run it as usual (with specific environment variables). Another one is to create own container based on official one:

1. Clone official container as base:

```
FROM arcan1s/ahriman:latest
```

2. Init pacman keys. This command is required in order to populate distribution keys:

```
RUN pacman-key --init
```

3. Install packages as it was described above:

```
RUN pacman --noconfirm -Sy wget
RUN wget https://pool.mirror.archlinux32.org/i686/extra/devtools-20221208-1.2-any.
↪pkg.tar.zst && pacman --noconfirm -U devtools-20221208-1.2-any.pkg.tar.zst
RUN wget https://pool.mirror.archlinux32.org/i686/core/archlinux32-keyring-20230705-
↪1.0-any.pkg.tar.zst && pacman --noconfirm -U archlinux32-keyring-20230705-1.0-any.
↪pkg.tar.zst
```

4. At that point you should have full `Dockerfile` like:

```
FROM arcan1s/ahriman:latest

RUN pacman-key --init

RUN pacman --noconfirm -Sy wget
RUN wget https://pool.mirror.archlinux32.org/i686/extra/devtools-20221208-1.2-any.
↪pkg.tar.zst && pacman --noconfirm -U devtools-20221208-1.2-any.pkg.tar.zst
RUN wget https://pool.mirror.archlinux32.org/i686/core/archlinux32-keyring-20230705-
↪1.0-any.pkg.tar.zst && pacman --noconfirm -U archlinux32-keyring-20230705-1.0-any.
↪pkg.tar.zst
```

5. After that you can build you own container, e.g.:

```
docker build --tag ahriman-i686:latest
```

6. Now you can run locally built container as usual with passing environment variables for setup command:

```
docker run --privileged -p 8080:8080 -e AHRIMAN_ARCHITECTURE=i686 -e AHRIMAN_PACMAN_
↪MIRROR='https://de.mirror.archlinux32.org/$arch/$repo' -e AHRIMAN_MULTILIB=␣
↪ahriman-i686:latest
```

3.4.4 Remote synchronization

How to sync repository to another server

There are several choices:

1. Easy and cheap, just share your local files through the internet, e.g. for `nginx`:

```
server {
    location / {
        autoindex on;
        root /var/lib/ahriman/repository/;
    }
}
```

2. You can also upload your packages using `rsync` to any available server. In order to use it you would need to configure ahriman first:

```
[upload]
target = rsync

[rsync]
remote = 192.168.0.1:/srv/repo
```

   After that just add `/srv/repo` to the `pacman.conf` as usual. You can also upload to S3 (`Server = https://s3.eu-central-1.amazonaws.com/repository/aur-clone/x86_64`) or to GitHub (`Server = https://github.com/ahriman/repository/releases/download/aur-clone-x86_64`).

How to sync to S3

1. Install dependencies:

```
pacman -S python-boto3
```

2. Create a bucket (e.g. `repository`).
3. Create an user with write access to the bucket:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListObjectsInBucket",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::repository"
            ]
        },
        {
            "Sid": "AllObjectActions",
            "Effect": "Allow",
```

(continues on next page)

(continued from previous page)

```
            "Action": "s3:*Object",
            "Resource": [
                "arn:aws:s3:::repository/*"
            ]
        }
    ]
}
```

4. Create an API key for the user and store it.
5. Configure the service as following:

```
[upload]
target = s3

[s3]
access_key = ...
bucket = repository
region = eu-central-1
secret_key = ...
```

S3 with SSL

In order to configure S3 on custom domain with SSL (and some other features, like redirects), the CloudFront should be used.

1. Configure S3 as described above.
2. In bucket properties, enable static website hosting with hosting type "Host a static website".
3. Go to AWS Certificate Manager and create public certificate on your domain. Validate domain as suggested.
4. Go to CloudFront and create distribution. The following settings are required:
    - Origin domain choose S3 bucket.
    - Tick use website endpoint.
    - Disable caching.
    - Select issued certificate.
5. Point DNS record to CloudFront address.

How to sync to GitHub releases

1. Create a repository.
2. Create API key with scope `public_repo`.
3. Configure the service as following:

```
[upload]
target = github

[github]
```

(continues on next page)

(continued from previous page)

```
owner = ahriman
password = ...
repository = repository
username = ahriman
```

3.4.5 Reporting

How to report by email

1. Install dependencies:

```
yay -S --asdeps python-jinja
```

2. Configure the service:

```
[report]
target = email

[email]
host = smtp.example.com
link_path = http://example.com/aur-clone/x86_64
password = ...
port = 465
receivers = me@example.com
sender = me@example.com
user = me@example.com
```

How to generate index page for S3

1. Install dependencies:

```
yay -S --asdeps python-jinja
```

2. Configure the service:

```
[report]
target = html

[html]
path = /var/lib/ahriman/repository/aur-clone/x86_64/index.html
link_path = http://example.com/aur-clone/x86_64
```

After these steps `index.html` file will be automatically synced to S3.

How to post build report to telegram

1. It still requires additional dependencies:

   ```
   yay -S --asdeps python-jinja
   ```

2. Register bot in telegram. You can do it by starting chat with @BotFather. For more details please refer to official documentation.
3. Optionally (if you want to post message in chat):
   1. Create telegram channel.
   2. Invite your bot into the channel.
   3. Make your channel public
4. Get chat id if you want to use by numerical id or just use id prefixed with `@` (e.g. `@ahriman`). If you are not using chat the chat id is your user id. If you don't want to make channel public you can use this guide.
5. Configure the service:

   ```
   [report]
   target = telegram

   [telegram]
   api_key = aaAAbbBBccCC
   chat_id = @ahriman
   link_path = http://example.com/aur-clone/x86_64
   ```

   `api_key` is the one sent by @BotFather, `chat_id` is the value retrieved from previous step.

If you did everything fine you should receive the message with the next update. Quick credentials check can be done by using the following command:

```
curl 'https://api.telegram.org/bot{api_key}/sendMessage?chat_id={chat_id}&text=hello'
```

(replace `{chat_id}` and `{api_key}` with the values from configuration).

3.4.6 Distributed builds

The service allows to run build on multiple machines and collect packages on main node. There are several ways to achieve it, this section describes officially supported methods.

Remote synchronization and remote server call

This setup requires at least two instances of the service:

1. Web service (with opt-in authorization enabled), later will be referenced as `master` node.
2. Application instances responsible for build, later will be referenced as `worker` nodes.

In this example the following settings are assumed:

- Repository architecture is `x86_64`.
- Master node address is `master.example.com`.

Master node configuration

The only requirements for the master node is that API must be available for worker nodes to call (e.g. port must be exposed to internet, or local network in case of VPN, etc) and file upload must be enabled:

```
[web]
enable_archive_upload = yes
```

In addition, the following settings are recommended for the master node:

- As it has been mentioned above, it is recommended to enable authentication (see *How to enable basic authorization*) and create system user which will be used later. Later this user (if any) will be referenced as `worker-user`.
- In order to be able to spawn multiple processes at the same time, wait timeout must be configured:

  ```
  [web]
  wait_timeout = 0
  ```

Worker nodes configuration

1. First of all, in this setup you need to split your repository into chunks manually, e.g. if you have repository on master node with packages `A`, `B` and `C`, you need to split them between all available workers, as example:

   - Worker #1: `A`.
   - Worker #2: `B` and `C`.

   Hint: `repo-tree` subcommand provides `--partitions` argument.

2. Each worker must be configured to upload files to master node:

   ```
   [upload]
   target = remote-service

   [remote-service]
   ```

3. Worker must be configured to access web on master node:

   ```
   [status]
   address = https://master.example.com
   username = worker-user
   password = very-secure-password
   ```

   As it has been mentioned above, `status.address` must be available for workers. In case if unix socket is used, it can be passed in the same option as usual. Optional `status.username`/`status.password` can be supplied in case if authentication was enabled on master node.

4. Each worker must call master node on success:

   ```
   [report]
   target = remote-call

   [remote-call]
   manual = yes
   ```

   After success synchronization (see above), the built packages will be put into directory, from which they will be read during manual update, thus `remote-call.manual` flag is required.

5. Change order of trigger runs. This step is required, because by default the report trigger is called before the upload trigger and we would like to achieve the opposite:

```
[build]
triggers = ahriman.core.gitremote.RemotePullTrigger ahriman.core.upload.
↪UploadTrigger ahriman.core.report.ReportTrigger ahriman.core.gitremote.
↪RemotePushTrigger
```

In addition, the following settings are recommended for workers:

- You might want to wait until report trigger will be completed; in this case the following option must be set:

```
[remote-call]
wait_timeout = 0
```

Dependency management

By default worker nodes don't know anything about master nodes packages, thus it will try to build each dependency by its own. However, using `AHRIMAN_REPOSITORY_SERVER` docker variable (or `--server` flag for setup command), it is possible to specify address of the master node for devtools configuration.

Repository and packages signing

You can sign packages on worker nodes and then signatures will be synced to master node. In order to do so, you need to configure worker node as following, e.g.:

```
[sign]
target = package
key = 8BE91E5A773FB48AC05CC1EDBED105AED6246B39
```

Note, however, that in this case, signatures will not be validated on master node and just will be copied to repository tree.

If you would like to sign only database files (aka repository sign), it has to be configured only on master node as usual, e.g.:

```
[sign]
target = repository
key = 8BE91E5A773FB48AC05CC1EDBED105AED6246B39
```

Double node minimal docker example

Master node config (`master.ini`) as:

```
[auth]
target = configuration

[web]
enable_archive_upload = yes
wait_timeout = 0
```

Command to run master node:

```
docker run --privileged -p 8080:8080 -e AHRIMAN_PORT=8080 -v master.ini:/etc/ahriman.ini.
↪d/overrides.ini arcan1s/ahriman:latest web
```

The user `worker-user` has been created additionally. Worker node config (`worker.ini`) as:

```
[status]
address = http://172.17.0.1:8080
username = worker-user
password = very-secure-password

[upload]
target = remote-service

[remote-service]

[report]
target = remote-call

[remote-call]
manual = yes
wait_timeout = 0

[build]
triggers = ahriman.core.gitremote.RemotePullTrigger ahriman.core.upload.UploadTrigger␣
↪ahriman.core.report.ReportTrigger ahriman.core.gitremote.RemotePushTrigger
```

The address above (`http://172.17.0.1:8080`) is somewhat available for worker container.

Command to run worker node:

```
docker run --privileged -v worker.ini:/etc/ahriman.ini.d/overrides.ini -it arcan1s/
↪ahriman:latest package-add ahriman --now
```

The command above will successfully build `ahriman` package, upload it on master node and, finally, will update master node repository.

Check proof-of-concept setup here.

Addition of new package and repository update

Just run on worker command as usual, the built packages will be automatically uploaded to master node. Note that automatic update process must be disabled on master node.

Package removal

This action must be done in two steps:

1. Remove package on worker.
2. Remove package on master node.

Delegate builds to remote workers

This setup heavily uses upload feature described above and, in addition, also delegates build process automatically to build machines. Same as above, there must be at least two instances available (`master` and `worker`), however, all `worker` nodes must be run in the web service mode.

Master node configuration

In addition to the configuration above, the worker list must be defined in configuration file (`build.workers` option), i.e.:

```
[build]
workers = https://worker1.example.com https://worker2.example.com

[web]
enable_archive_upload = yes
wait_timeout = 0
```

In the example above, `https://worker1.example.com` and `https://worker2.example.com` are remote `worker` node addresses available for `master` node.

In case if authentication is required (which is recommended way to setup it), it can be set by using `status` section as usual.

Worker nodes configuration

It is required to point to the master node repository, otherwise internal dependencies will not be handled correctly. In order to do so, the `--server` argument (or `AHRIMAN_REPOSITORY_SERVER` environment variable for docker images) can be used.

Also, in case if authentication is enabled, the same user with the same password must be created for all workers.

It is also recommended to set `web.wait_timeout` to infinite in case of multiple conflicting runs and `service_only` to `yes` in order to disable status endpoints.

Other settings are the same as mentioned above.

Triple node minimal docker example

In this example, all instances are run on the same machine with address `172.17.0.1` with ports available outside of container. Master node config (`master.ini`) as:

```
[auth]
target = configuration

[status]
username = builder-user
password = very-secure-password

[build]
workers = http://172.17.0.1:8081 http://172.17.0.1:8082

[web]
```

(continues on next page)

(continued from previous page)

```
enable_archive_upload = yes
wait_timeout = 0
```

Command to run master node:

```
docker run --privileged -p 8080:8080 -e AHRIMAN_PORT=8080 -v master.ini:/etc/ahriman.ini.
↪d/overrides.ini arcan1s/ahriman:latest web
```

Worker nodes (applicable for all workers) config (`worker.ini`) as:

```
[auth]
target = configuration

[status]
address = http://172.17.0.1:8080
username = builder-user
password = very-secure-password

[upload]
target = remote-service

[remote-service]

[report]
target = remote-call

[remote-call]
manual = yes
wait_timeout = 0

[web]
service_only = yes

[build]
triggers = ahriman.core.upload.UploadTrigger ahriman.core.report.ReportTrigger
```

Command to run worker nodes (considering there will be two workers, one is on `8081` port and other is on `8082`):

```
docker run --privileged -p 8081:8081 -e AHRIMAN_PORT=8081 -v worker.ini:/etc/ahriman.ini.
↪d/overrides.ini arcan1s/ahriman:latest web
docker run --privileged -p 8082:8082 -e AHRIMAN_PORT=8082 -v worker.ini:/etc/ahriman.ini.
↪d/overrides.ini arcan1s/ahriman:latest web
```

Unlike the previous setup, it doesn't require to mount repository root for `worker` nodes, because they don't use it anyway.

Check proof-of-concept setup here.

Addition of new package, package removal, repository update

In all scenarios, update process must be run only on `master` node. Unlike the manually distributed packages described above, automatic update must be enabled only for `master` node.

Automatic worker nodes discovery

Instead of setting `build.workers` option it is also possible to configure services to load worker list dynamically. To do so, the `ahriman.core.distributed.WorkerLoaderTrigger` and `ahriman.core.distributed.WorkerTrigger` must be used for `master` and `worker` nodes repsectively. See recipes for more details.

Known limitations

- Workers don't support local packages. However, it is possible to build custom packages by providing sources by using `ahriman.core.gitremote.RemotePullTrigger` trigger.
- No dynamic nodes discovery. In case if one of worker nodes is unavailable, the build process will fail.
- No pkgrel bump on conflicts.
- The identical user must be created for all workers. However, the `master` node user can be different from this one.

3.4.7 Maintenance packages

Generate keyring package

The application provides special plugin which generates keyring package. This plugin heavily depends on `sign` group settings, however it is possible to override them. The minimal package can be generated in the following way:

1. Edit configuration:

   ```
   [keyring]
   target = keyring-generator
   ```

   By default it will use `sign.key` as trusted key and all other keys as packagers ones. For all available options refer to *configuration*.

2. Create package source files:

   ```
   sudo -u ahriman ahriman repo-create-keyring
   ```

   This command will generate PKGBUILD, revoked and trusted listings and keyring itself and will register the package in database.

3. Build new package as usual:

   ```
   sudo -u ahriman ahriman package-add aur-clone-keyring --source local --now
   ```

   where `aur-clone` is your repository name.

This plugin might have some issues, in case of any of them, kindly create new issue.

Generate mirrorlist package

The application provides special plugin which generates mirrorlist package also. It is possible to distribute this package as usual later. The package can be generated in the following way:

1. Edit configuration:

```
[mirrorlist]
target = mirrorlist-generator

[mirrorlist-generator]
servers = https://repo.example.com/$arch
```

   The `mirrorlist-generator.servers` must contain list of available mirrors, the `$arch` and `$repo` variables are supported. For more options kindly refer to *configuration*.

2. Create package source files:

```
sudo -u ahriman ahriman repo-create-mirrorlist
```

   This command will generate PKGBUILD and mirrorlist file and will register the package in database.

3. Build new package as usual:

```
sudo -u ahriman ahriman package-add aur-clone-mirrorlist --source local --now
```

   where `aur-clone` is your repository name.

3.4.8 Web service

How to setup web service

1. Install dependencies:

```
yay -S --asdeps python-aiohttp python-aiohttp-jinja2 python-aiohttp-apispec>=3.0.0␣
↪python-aiohttp-cors
```

2. Configure service:

```
[web]
port = 8080
```

3. Start the web service `systemctl enable --now ahriman-web`.

How to enable basic authorization

1. Install dependencies :

```
yay -S --asdeps python-aiohttp-security python-aiohttp-session python-cryptography
```

2. Configure the service to enable authorization:

```
[auth]
target = configuration
salt = somerandomstring
```

The `salt` parameter is optional, but recommended, and can be set to any (random) string.

3. In order to provide access for reporting from application instances you can (the recommended way) use unix sockets by the following configuration (note, that it requires `python-requests-unixsocket` package to be installed):

```
[web]
unix_socket = /var/lib/ahriman/ahriman-web.sock
```

This socket path must be available for web service instance and must be available for all application instances (e.g. in case if you are using docker container - see above - you need to make sure that the socket is passed to the root filesystem).

By the way, unix socket variable will be automatically set in case if `--web-unix-socket` argument is supplied to the `setup` subcommand.

Alternatively, you need to create user for the service:

```
sudo -u ahriman ahriman user-add -r full api
```

This command will ask for the password, just type it in stdin; **do not** leave the field blank, user will not be able to authorize, and finally configure the application:

```
[status]
username = api
password = pa55w0rd
```

4. Create end-user with password:

```
sudo -u ahriman ahriman user-add -r full my-first-user
```

5. Restart web service `systemctl restart ahriman-web`.

How to enable OAuth authorization

1. Create OAuth web application, download its `client_id` and `client_secret`.
2. Guess what? Install dependencies:

```
yay -S --asdeps python-aiohttp-security python-aiohttp-session python-cryptography␣
↪python-aioauth-client
```

3. Configure the service:

```
[auth]
target = oauth
client_id = ...
client_secret = ...

[web]
address = https://example.com
```

Configure `oauth_provider` and `oauth_scopes` in case if you would like to use different from Google provider. Scope must grant access to user email. `web.address` is required to make callback URL available from internet.

4. If you are not going to use unix socket, you also need to create service user (remember to set `auth.salt` option before if required):

```
sudo -u ahriman ahriman user-add --as-service -r full api
```

5. Create end-user:

```
sudo -u ahriman ahriman user-add -r full my-first-user
```

When it will ask for the password leave it blank.

6. Restart web service `systemctl restart ahriman-web`.

How to implement own interface

You can write your own interface by using API which is provided by the web service. Full autogenerated API documentation is available at `http://localhost:8080/api-docs`.

3.4.9 Backup and restore

The service provides several commands aim to do easy repository backup and restore. If you would like to move repository from the server `server1.example.com` to another `server2.example.com` you have to perform the following steps:

1. On the source server `server1.example.com` run `repo-backup` command, e.g.:

```
ahriman repo-backup /tmp/repo.tar.gz
```

This command will pack all configuration files together with database file into the archive specified as command line argument (i.e. `/tmp/repo.tar.gz`). In addition it will also archive `cache` directory (the one which contains local clones used by e.g. local packages) and `.gnupg` of the `ahriman` user.

2. Copy created archive from source server `server1.example.com` to target `server2.example.com`.
3. Install package as usual on the target server `server2.example.com` if you didn't yet.
4. Extract archive e.g. by using subcommand:

```
ahriman repo-restore /tmp/repo.tar.gz
```

An additional argument `-o`/`--output` can be used to specify extraction root (`/` by default).

5. Rebuild repository:

```
sudo -u ahriman ahriman repo-rebuild --from-database
```

3.4.10 Use cases

There is a collection of some specific recipes which can be found in the repository.

Most of them can be run (`AHRIMAN_PASSWORD` environment variable is required in the most setups) as simple as:

```
AHRIMAN_PASSWORD=demo docker compose up
```

Note, however, they are just an examples of specific configuration for specific cases and they are never intended to be used as is in real environment.

3.4.11 Other topics

How does it differ from %another-manager%?

Short answer - I do not know. Also for some references credits to Alad, he did really good investigation of existing alternatives.

arch-repo-manager

Looks actually pretty good, in case if I would find it, I would probably didn't start this project; the most of features (like web interface or additional helpers) are already implemented or planned to be. However, this project seems to be at early alpha stage (as for Nov 2022), written in C++ (not pro or con) and misses documentation.

archrepo2

Don't know, haven't tried it. But it lacks of documentation at least.

- `ahriman` has web interface.
- `archrepo2` doesn't have synchronization and reporting.
- `archrepo2` actively uses direct shell calls and `yaourt` components.
- `archrepo2` has constantly running process instead of timer process (it is not pro or con).

repoctl

- `ahriman` has web interface.
- `repoctl` does not have reporting feature.
- `repoctl` does not support local packages and patches.
- Some actions are not fully automated in `repoctl` (e.g. package update still requires manual intervention for the build itself).
- `repoctl` has better AUR interaction features. With colors!
- `repoctl` has much easier configuration and even completion.
- `repoctl` is able to store old packages.
- Ability to host repository from same command in `repoctl` vs external services (e.g. nginx) in `ahriman`.

repod

Official tool provided by distribution, has clean logic, but it is just a helper for `repo-add`, e.g. it doesn't work with AUR and all packages builds have to be handled separately.

repo-scripts

Though originally I've created ahriman by trying to improve the project, it still lacks a lot of features:

- `ahriman` has web interface.
- `ahriman` has better reporting with template support.
- `ahriman` has more synchronization features (there was only `rsync` based).
- `ahriman` supports local packages and patches.
- `repo-scripts` doesn't have dependency management.

...and so on. `repo-scripts` also has bad architecture and bad quality code and uses out-of-dated `yaourt` and `package-query`.

toolbox

It is automation tools for `repoctl` mentioned above. Except for using shell it looks pretty cool and also offers some additional features like patches, remote synchronization (isn't it?) and reporting.

How to check service logs

By default, the service writes logs to `journald` which can be accessed by using `journalctl` command (logs are written to the journal of the user under which command is run). In order to retrieve logs for the process you can use the following command:

```
sudo journalctl SYSLOG_IDENTIFIER=ahriman
```

You can also ask to forward logs to `stderr`, just set `--log-handler` flag, e.g.:

```
ahriman --log-handler console ...
```

You can even configure logging as you wish, but kindly refer to python `logging` module configuration.

The application uses java concept to log messages, e.g. class `Application` imported from `ahriman.application.application` package will have logger called `ahriman.application.application.Application`. In order to e.g. change logger name for whole application package it is possible to change values for `ahriman.application` package; thus editing `ahriman` logger configuration will change logging for whole application (unless there are overrides for another logger).

Html customization

It is possible to customize html templates. In order to do so, create files somewhere (refer to Jinja2 documentation and the service source code for available parameters) and prepend `templates` with value pointing to this directory.

In addition, default html templates supports style customization out-of-box. In order to customize style, just put file named `user-style.jinja2` to the templates directory.

Web API extension

The application loads web views dynamically, so it is possible relatively easy extend its API. In order to do so:

1. Create view class which is derived from `ahriman.web.views.base.BaseView` class.
2. Create implementation for this class.
3. Put file into `ahriman.web.views` package.
4. Restart application.

For more details about implementation and possibilities, kindly refer to module documentation and source code and aiohttp documentation.

I did not find my question

Create an issue with type **Question**.

3.5 Manual migrations

Normally the most of migrations are handled automatically after application start, however, some upgrades require manual interventions; this document describes them.

3.5.1 Upgrades to breakpoints

To 2.9.0

This release includes major upgrade for the newest devtools and archlinux repository structure. In order to upgrade package need to:

1. Upgrade to the latest major release of python (3.11) (required by other changes).
2. Upgrade devtools to the latest release.
3. Backup local settings, `/etc/ahriman.ini.d/00-setup-overrides.ini` by default.
4. Run setup command (i.e. `ahriman service-setup`) again with the same arguments as used before. This step can be done manually by moving `devtools` configuration (something like `/usr/share/devtools/pacman-ahriman*.conf`) to new location `/usr/share/devtools/pacman.conf.d/` under name `ahriman.conf`. After that make sure to remove any `community` mentions from configurations (e.g. `/usr/share/devtools/pacman.conf.d/ahriman.conf`, `/etc/ahriman.ini`) if there were any. The only thing which will change is `devtools` configuration.
5. Remove build chroot as it is incompatible, e.g. `sudo ahriman service-clean --chroot`.
6. Run `sudo -u ahriman ahriman update --no-aur --no-local --no-manual -yy` in order to update local databases.

To 2.12.0

This release includes paths migration. Unlike usual case, no automatic migration is performed because it might break user configuration. The following noticeable changes have been made:

- Path to pre-built packages now includes repository name, i.e. it has been changed from `/var/lib/ahriman/packages/x86_64` to `/var/lib/ahriman/packages/aur-clone/x86_64`.
- Path to pacman databases now includes repository name too, it has been changed from `/var/lib/ahriman/pacman/x86_64` to `/var/lib/ahriman/pacman/aur-clone/x86_64`.
- Path to repository itself also includes repository name, from `/var/lib/ahriman/repository/x86_64` to `/var/lib/ahriman/repository/aur-clone/x86_64`.

In order to migrate to the new filesystem tree the following actions are required:

1. Stop and disable all services, e.g. timer and web service:

   ```
   sudo systemctl disable --now ahriman@x86_64.timer
   sudo systemctl disable --now ahriman-web@x86_64
   ```

2. Create directory tree. It can be done by running `ahriman service-tree-migrate` subcommand. It performs copying between the old repository tree and the new one. Alternatively directories can be copied by hands.

3. Edit configuration in case if anything is pointing to the old path, e.g. HTML report generation, in the way in which it will point to the directory inside repository specific one, e.g. `/var/lib/ahriman/repository/x86_64` to `/var/lib/ahriman/repository/aur-clone/x86_64`.

4. Run setup command (i.e. `ahriman service-setup`) again with the same arguments as used before. This step can be done manually by editing devtools pacman configuration (`/usr/share/devtools/pacman.conf.d/ahriman-x86_64.conf` by default) replacing `Server` with path to the repository, e.g.:

   ```
   [aur-clone]
   SigLevel = Optional TrustAll
   Server = file:///var/lib/ahriman/repository/aur-clone/x86_64
   ```

   In case of manual interventions make sure to remove architecture reference from `web` sections (if any) to avoid ambiguity.

5. Make sure to update remote synchronization services if any. Almost all of them rely on current repository tree by default, so it is required to setup either redirects or configure to synchronize to the old locations (e.g. `object_path` option for S3 synchronization).

6. Enable and start services again. Unit template parameter should include both repository architecture and name, dash separated, e.g. `x86_64-aur-clone`, where `x86_64` is the repository architecture and `aur-clone` is the repository name:

   ```
   sudo systemctl enable --now ahriman@x86_64-aur-clone.timer
   sudo systemctl enable --now ahriman-web
   ```

3.6 Architecture

3.6.1 Package structure

Packages have strict rules of importing:

- `ahriman.application` package must not be used outside of this package.
- `ahriman.core` and `ahriman.models` packages don't have any import restriction. Actually we would like to totally restrict importing of `core` package from `models`, but it is impossible at the moment.
- `ahriman.web` package is allowed to be imported from `ahriman.application` (web handler only, only `ahriman.web.web` methods). It also must not be imported globally, only local import is allowed.

Full dependency diagram:

`ahriman.application` package

This package contains application (aka executable) related classes and everything for it. It also contains package called `ahriman.application.handlers` in which all available subcommands are described as separated classes derived from the base `ahriman.application.handlers.Handler` class.

`ahriman.application.application.Application` (god class) is used for any interaction from parsers with repository. It is divided into multiple traits by functions (package related and repository related) in the same package.

`ahriman.application.application.workers` package contains specific wrappers for local and remote build processes.

`ahriman.application.ahriman` contains only command line parses and executes specified `Handler` on success, `ahriman.application.lock.Lock` is additional class which provides file-based lock and also performs some common checks.

`ahriman.core` package

This package contains everything required for the most of application actions and it is separated into several packages:

- `ahriman.core.alpm` package controls pacman related functions. It provides wrappers for `pyalpm` library and safe calls for repository tools (`repo-add` and `repo-remove`). Also this package contains `ahriman.core.alpm.remote` package which provides wrapper for remote sources (e.g. AUR RPC and official repositories RPC).
- `ahriman.core.auth` package provides classes for authorization methods used by web mostly. Base class is `ahriman.core.auth.Auth` which must be instantiated by `load` method.
- `ahriman.core.build_tools` is a package which provides wrapper for `devtools` commands.
- `ahriman.core.configuration` contains extension for standard `configparser` library and some validation related classes.
- `ahriman.core.database` is everything for database, including data and schema migrations.
- `ahriman.core.distributed` package with triggers and helpers for distributed build system.
- `ahriman.core.formatters` package provides `Printer` sub-classes for printing data (e.g. package properties) to stdout which are used by some handlers.
- `ahriman.core.gitremote` is a package with remote PKGBUILD triggers. Should not be called directly.
- `ahriman.core.http` package provides HTTP clients which can be used later by other classes.

- `ahriman.core.log` is a log utils package. It includes logger loader class, custom HTTP based logger and some wrappers.
- `ahriman.core.report` is a package with reporting triggers. Should not be called directly.
- `ahriman.core.repository` contains several traits and base repository (`ahriman.core.repository.Repository` class) implementation.
- `ahriman.core.sign` package provides sign feature (only gpg calls are available).
- `ahriman.core.status` contains helpers and watcher class which are required for web application. Reporter must be initialized by using `ahriman.core.status.client.Client.load` method.
- `ahriman.core.support` provides plugins for support packages (mirrorlist and keyring) generation.
- `ahriman.core.triggers` package contains base trigger classes. Classes from this package must be imported in order to implement user extensions. In fact, `ahriman.core.report`, `ahriman.core.upload` and other built-in triggers use this package.
- `ahriman.core.upload` package provides sync feature, should not be called directly.

This package also provides some generic functions and classes which may be used by other packages:

- `ahriman.core.exceptions` provides custom exceptions.
- `ahriman.core.spawn.Spawn` is a tool which can spawn another `ahriman` process. This feature is used by web application.
- `ahriman.core.tree` is a dependency tree implementation.

`ahriman.models` package

It provides models for any other part of application. Unlike `ahriman.core` package classes from here provide only conversion methods (e.g. create class from another or convert to). It is mostly presented by case classes and enumerations.

`ahriman.web` package

Web application. It is important that this package is isolated from any other to allow it to be optional feature (i.e. dependencies which are required by the package are optional).

- `ahriman.web.middlewares` provides middlewares for request handlers.
- `ahriman.web.schemas` provides schemas (actually copy paste from dataclasses) used by swagger documentation.
- `ahriman.web.views` contains web views derived from aiohttp view class.
- `ahriman.web.apispec` provides generators for swagger documentation.
- `ahriman.web.cors` contains helpers for cross origin resource sharing middlewares.
- `ahriman.web.routes` creates routes for web application.
- `ahriman.web.web` provides main web application functions (e.g. start, initialization).

3.6.2 Application run

1. Parse command line arguments, find subcommand and related handler which is set by the parser.
2. Call `Handler.execute` method.
3. Define list of architectures to run. In case if there is more than one architecture specified run several subprocesses or continue in current process otherwise. Class attribute `ALLOW_MULTI_ARCHITECTURE_RUN` controls whether the application can be run in multiple processes or not - this feature is required for some handlers (e.g. `Web`, which should be able to spawn child process in daemon mode; it is impossible to do from daemonic processes).
4. In each child process call lock functions.
5. After success checks pass control to `Handler.run` method defined by specific handler class.
6. Return result (success or failure) of each subprocess and exit from application.
7. Some handlers may override their status and throw `ExitCode` exception. This exception is just silently suppressed and changes application exit code to `1`.

In the most cases handlers spawn god class `ahriman.application.application.Application` class and call required methods.

The application is designed to run from `systemd` services and provides parametrized by repository identifier timer and service file for that.

Subcommand design

All subcommands are divided into several groups depending on the role they are doing:

- `aur` (`aur-search`) group is for AUR operations.
- `help` (e.g. `help`) are system commands.
- `package` subcommands (e.g. `package-add`) allow to perform single package actions.
- `patch` subcommands (e.g. `patch-list`) are the special case of `package` subcommands introduced in order to control patches for packages.
- `repo` subcommands (e.g. `repo-check`) usually perform actions on whole repository.
- `service` subcommands (e.g. `service-setup`) perform actions which are related to whole service managing: create repository, show configuration.
- `user` subcommands (`user-add`) are intended for user management.
- `web` subcommands are related to web service management.

For historical reasons and in order to keep backward compatibility some subcommands have aliases to their shorter forms or even other groups, but the application doesn't guarantee that they will remain unchanged.

3.6.3 Filesystem tree

The application supports two types of trees, one is for the legacy configuration (when there were no explicit repository name configuration available) and another one is the new-style tree. This document describes only new-style tree in order to avoid deprecated structures.

Having default root as `/var/lib/ahriman` (differs from container though), the directory structure is the following:

```
/var/lib/ahriman/
├── ahriman.db
├── cache
├── chroot
│   └── aur-clone
├── packages
│   └── aur-clone
│       └── x86_64
├── pacman
│   └── aur-clone
│       └── x86_64
│           ├── local
│           │   └── ALPM_DB_VERSION
│           └── sync
│               ├── core.db
│               ├── extra.db
│               └── multilib.db
└── repository
    └── aur-clone
        └── x86_64
            ├── aur-clone.db -> aur-clone.db.tar.gz
            ├── aur-clone.db.tar.gz
            ├── aur-clone.files -> aur-clone.files.tar.gz
            └── aur-clone.files.tar.gz
```

There are multiple subdirectories, some of them are commons for any repository, but some of them are not.

- `cache` is a directory with locally stored PKGBUILD's and VCS packages. It is common for all repositories and architectures.
- `chroot/{repository}` is a chroot directory for `devtools`. It is specific for each repository, but shared for different architectures inside (the `devtools` handles architectures automatically).
- `packages/{repository}/{architecture}` is a directory with prebuilt packages. When a package is built, first it will be uploaded to this directory and later will be handled by update process. It is architecture and repository specific.
- `pacman/{repository}/{architecture}` is the repository and architecture specific caches for pacman's databases.
- `repository/{repository}/{architecture}` is a repository packages directory.

Normally you should avoid direct interaction with the application tree. For tree migration process refer to the *migration notes*.

3.6.4 Database

The service uses SQLite database in order to store some internal info.

Database instance

All methods related to the specific part of database (basically operations per table) are split into different traits located inside `ahriman.core.database.operations` package. The base trait `ahriman.core.database.operations.Operations` also provides generic methods for database access (e.g. row converters and transactional support).

The `ahriman.core.database.SQLite` class itself derives from all of these traits and implements methods for initialization, including migrations.

Schema and data migrations

The schema migrations are applied according to current `pragma user_info` values, located at `ahriman.core.database.migrations` package and named as `m000_migration_name.py` (the preceding `m` is required in order to import migration content for tests). Additional class `ahriman.core.database.migrations.Migrations` reads all migrations automatically and applies them in alphabetical order.

These migrations can also contain data migrations. Though the recommended way is to migrate data directly from SQL queries, sometimes it is required to have external data (like packages list) in order to set correct data. To do so, special method `migrate_data` is used.

Type conversions

By default, it parses rows into python dictionary. In addition, the following pseudo-types are supported:

- `dict[str, Any]`, `list[Any]` - for storing JSON data structures in database (technically there is no restriction on types for dictionary keys and values, but it is recommended to use only string keys). The type is stored as `json` data type and `json.loads` and `json.dumps` methods are used in order to read and write from/to database respectively.

3.6.5 Basic flows

By default package build operations are performed with `PACKAGER` which is specified in `makepkg.conf`, however, it is possible to override this variable from command line; in this case service performs lookup in the following way:

- If packager is not set, it reads environment variables (e.g. `SUDO_USER` and `USER`), otherwise it uses value from command line.
- It checks users for the specified username and tries to extract packager variable from it.
- If packager value has been found, it will be passed as `PACKAGER` system variable (additional sudo configuration might be required).

Add new packages or rebuild existing

Idea is to add package to a build queue from which it will be handled automatically during the next update run. Different variants are supported:

- If supplied argument is file, then application moves the file to the directory with built packages. Same rule applies for directory, but in this case it copies every package-like file from the specified directory.
- If supplied argument is directory and there is `PKGBUILD` file there, it will be treated as local package. In this case it will queue this package to build and copy source files (`PKGBUILD` and `.SRCINFO`) to caches.
- If supplied argument is not file then application tries to lookup for the specified name in AUR and clones it into the directory with manual updates. This scenario can also handle package dependencies which are missing in repositories.

This logic can be overwritten by specifying the `source` parameter, which is partially useful if you would like to add package from AUR, but there is local directory cloned from AUR. Also official repositories calls are hidden behind explicit source definition.

Rebuild packages

Same as add function for every package in repository. Optional filters by reverse dependency or build status can be supplied.

Remove packages

This flow removes package from filesystem, updates repository database and also runs synchronization and reporting methods.

Update packages

This feature is divided into to the following stages: check AUR for updates and run rebuild for required packages. Whereas check does not do anything except for check itself, update flow is the following:

1. Process every built package first. Those packages are usually added manually.
2. Run sync and report methods.
3. Generate dependency tree for packages to be built.
4. For each level of tree it does:
    1. Download package data from AUR.
    2. Bump `pkgrel` if there is duplicate version in the local repository (see explanation below).
    3. Build every package in clean chroot.
    4. Sign packages if required.
    5. Add packages to database and sign database if required.
    6. Process triggers.

After any step any package data is being removed.

In case if there are configured workers, the build process itself will be delegated to the remote instances. Packages will be partitioned to the chunks according to the amount of configured workers.

Distributed builds

This feature consists of two parts:

- Upload built packages to the node.
- Delegate packages building to separated nodes.

The upload process is performed via special API endpoint, which is disabled by default, and is performed in several steps:

1. Upload package to temporary file.
2. Copy content from temporary file to the built package directory with dot (`.`) prefix.
3. Rename copied file, removing preceding dot.

After success upload, the update process must be called as usual in order to copy built packages to the main repository tree.

On the other side, the delegation uses upload feature, but in addition it also calls external services in order to trigger build process. The packages are separated to chunks based on the amount of the configured workers and their dependencies.

pkgrel bump rules

The application is able to automatically bump package release (`pkgrel`) during build process if there is duplicate version in repository. The version will be incremented as following:

1. Get version of the remote package.
2. Get version of the local package if available.
3. If local version is not set, proceed with remote one.
4. If local version is set and epoch or package version (`pkgver`) are different, proceed with remote version.
5. If local version is set and remote version is newer than local one, proceed with remote.
6. Extract `pkgrel` value.
7. If it has `major.minor` notation (e.g. `1.1`), then increment last part by 1, e.g. `1.1 -> 1.2`, `1.0.1 -> 1.0.2`.
8. If `pkgrel` is a number (e.g. `1`), then append 1 to the end of the string, e.g. `1 -> 1.1`.

3.6.6 Core functions reference

Configuration

`ahriman.core.configuration.Configuration` class provides some additional methods (e.g. `getpath` and `getlist`) and also combines multiple files into single configuration dictionary using repository identifier overrides. It is the recommended way to deal with settings.

Enumerations

All enumerations are derived from `enum.StrEnum`. Integer enumerations in general are not allowed, because most of operations require conversions from string variable. Derivation from string based enumeration is required to make json conversions implicitly (e.g. during calling `json.dumps` methods).

In addition, some enumerations provide `from_option` class methods in order to allow some flexibility while reading configuration options.

Utils

For every external command run (which is actually not recommended if possible) custom wrapper for `subprocess` is used. Additional functions `ahriman.core.auth.helpers` provide safe calls for `aiohttp_security` methods and are required to make this dependency optional.

Context variables

Package provides implicit global variables which can be accessed from `ahriman.core` package as `context` variable, wrapped by `contextvars.ContextVar` class. The value of the variable is defaulting to private `_Context` class which is defined in the same module. The default values - such as `database` and `sign` - are being set on the service initialization.

The `_Context` class itself mimics default collection interface (as is `Mapping`) and can be modified by `_Context.set` method. The stored variables can be achieved by `_Context.get` method, which is unlike default `Mapping` interface also performs type and presence checks.

In order to provide statically typed interface, the `ahriman.models.context_key.ContextKey` class is used for both `_Content.get` and `_Content.set` methods; the context instance itself, however, does not store information about types.

Submodules

Some packages provide different behaviour depending on configuration settings. In these cases inheritance is used and recommended way to deal with them is to call class method `load` from base classes.

Authorization

The package provides several authorization methods: disabled, based on configuration and OAuth2.

Disabled (default) authorization provider just allows everything for everyone and does not have any specific configuration (it uses some default configuration parameters though). It also provides generic interface for derived classes.

Mapping (aka configuration) provider uses hashed passwords with optional salt from the database in order to authenticate users. This provider also enables user permission checking (read/write) (authorization). Thus, it defines the following methods:

- `check_credentials` - user password validation (authentication).
- `verify_access` - user permission validation (authorization).

Passwords must be stored in database as `hash(password + salt)`, where `password` is user defined password (taken from user input), `salt` is random string (any length) defined globally in configuration and `hash` is secure hash function. Thus, the following configuration

```
"username","password","access"
"username","$6$rounds=656000$mWBiecMPrHAL1VgX$oU4Y5HH8HzlvMaxwkNEJjK13ozElyU1wAHBoO/
↪WW5dAaE4YEfnB0X3FxbynKMl4FBdC3Ovap0jINz4LPkNADg0","read"
```

means that there is user `username` with `read` access and password `password` hashed by `sha512` with salt `salt`.

OAuth provider uses library definitions (`aioauth-client`) in order *authenticate* users. It still requires user permission to be set in database, thus it inherits mapping provider without any changes. Whereas we could override `check_credentials` (authentication method) by something custom, OAuth flow is a bit more complex than just forward request, thus we have to implement the flow in login form.

OAuth's implementation also allows authenticating users via username + password (in the same way as mapping does) though it is not recommended for end-users and password must be left blank. In particular this feature can be used by service reporting (aka robots).

In addition, web service checks the source socket used. In case if it belongs to `socket.AF_UNIX` family, it will skip any further checks considering the request to be performed in safe environment (e.g. on the same physical machine). This feature, in particular is being used by the reporter instances in case if socket address is set in configuration.

In order to configure users there are special subcommands.

Triggers

Triggers are extensions which can be used in order to perform any actions on application start, after the update process and, finally, before the application exit.

The main idea is to load classes by their full path (e.g. `ahriman.core.upload.UploadTrigger`) by using `importlib`: get the last part of the import and treat it as class name, join remain part by `.` and interpret as module path, import module and extract attribute from it.

The loaded triggers will be called with `ahriman.models.result.Result` and `list[Packages]` arguments, which describes the process result and current repository packages respectively. Any exception raised will be suppressed and will generate an exception message in logs.

In addition triggers can implement `on_start` and `on_stop` actions which will be called on the application start and right before the application exit respectively. The `on_start` action is usually being called from handlers directly in order to make sure that no trigger will be run when it is not required (e.g. on user management). As soon as `on_start` action is called, the additional flag will be set; `ahriman.core.triggers.TriggerLoader` class implements `__del__` method in which, if the flag is set, the `on_stop` actions will be called.

For more details how to deal with the triggers, refer to *documentation* and modules descriptions.

Remote synchronization

There are several supported synchronization providers, currently they are `rsync`, `s3`, `github`.

`rsync` provider does not have any specific logic except for running external rsync application with configured arguments. The service does not handle SSH configuration, thus it has to be configured before running application manually.

`s3` provider uses `boto3` package and implements sync feature. The files are stored in architecture specific directory (e.g. if bucket is `repository`, packages will be stored in `repository/aur-clone/x86_64` for the `aur-clone` repository and `x86_64` architecture), bucket must be created before any action and API key must have permissions to write to the bucket. No external configuration required. In order to upload only changed files the service compares calculated hashes with the Amazon ETags, the implementation used is described here.

`github` provider authenticates through basic auth, API key with repository write permissions is required. There will be created a release with the name of the architecture in case if it does not exist; files will be uploaded to the release

assets. It also stores array of files and their MD5 checksums in release body in order to upload only changed ones. According to the GitHub API in case if there is already uploaded asset with the same name (e.g. database files), asset will be removed first.

Additional features

Some features require optional dependencies to be installed:

- Version control executables (e.g. `git`, `svn`) for VCS packages.
- `gnupg` application for package and repository sign feature.
- `rsync` application for rsync based repository sync.
- `boto3` python package for `S3` sync.
- `Jinja2` python package for HTML report generation (it is also used by web application).

3.6.7 Web application

Web application requires the following python packages to be installed:

- Core part requires `aiohttp` (application itself), `aiohttp_jinja2` and `Jinja2` (HTML generation from templates).
- Additional web features also require `aiohttp-apispec` (autogenerated documentation), `aiohttp_cors` (CORS support, required by documentation).
- In addition, authorization feature requires `aiohttp_security`, `aiohttp_session` and `cryptography`.
- In addition to base authorization dependencies, OAuth2 also requires `aioauth-client` library.
- In addition if you would like to disable authorization for local access (recommended way in order to run the application itself with reporting support), the `requests-unixsocket` library is required.

Middlewares

Service provides some custom middlewares, e.g. logging every exception (except for user ones) and user authorization.

HEAD and OPTIONS requests

`HEAD` request is automatically generated by `ahriman.web.views.base.BaseView` class. It just calls `GET` method, removes any data from body and returns the result. In case if no `GET` method available for this view, the `aiohttp.web.HTTPMethodNotAllowed` exception will be raised.

On the other side, `OPTIONS` method is implemented in the `ahriman.web.middlewares.exception_handler.exception_handler` middleware. In case if `aiohttp.web.HTTPMethodNotAllowed` exception is raised and original method was `OPTIONS`, the middleware handles it, converts to valid request and returns response to user.

Web views

All web views are defined in separated package and derived from `ahriman.web.views.base.Base` class which provides typed interfaces for web application.

REST API supports only JSON data.

Different APIs are separated into different packages:

- `ahriman.web.views.api` not a real API, but some views which provide OpenAPI support.
- `ahriman.web.views.*.service` provides views for application controls.
- `ahriman.web.views.*.status` package provides REST API for application reporting.
- `ahriman.web.views.*.user` package provides login and logout methods which can be called without authorization.

The views are also divided by supporting API versions (e.g. `v1`, `v2`).

Templating

Package provides base jinja templates which can be overridden by settings. Vanilla templates actively use bootstrap library.

Requests and scopes

Service provides optional authorization which can be turned on in settings. In order to control user access there are two levels of authorization - read-only (only GET-like requests) and write (anything), settings for which are provided by each web view directly.

If this feature is configured any request will be prohibited without authentication. In addition, configuration flag `auth.allow_read_only` can be used in order to allow read-only operations - reading index page and packages - without authorization.

For authenticated users it uses encrypted session cookies to store tokens; encryption key is generated each time at the start of the application. It also stores expiration time of the session inside.

External calls

Web application provides external calls to control main service. It spawns child process with specific arguments and waits for its termination. This feature must be used either with authorization or in safe (i.e. when status page is not available world-wide) environment.

For most actions it also extracts user from authentication (if provided) and passes it to the underlying process.

3.7 Advanced usage

Depending on the goal the package can be used in different ways. Nevertheless, in the most cases you will need some basic classes

```
from pathlib import Path

from ahriman.core.configuration import Configuration
from ahriman.core.database import SQLite
```

(continues on next page)

(continued from previous page)

```
from ahriman.models.repository_id import RepositoryId

repository_id = RepositoryId("x86_64", "aur-clone")
configuration = Configuration.from_path(Path("/etc/ahriman.ini"), repository_id)
database = SQLite.load(configuration)
```

At this point there are `configuration` and `database` instances which can be used later at any time anywhere, e.g.

```
# instance of ``RepositoryPaths`` class
paths = configuration.repository_paths
```

Almost all actions are wrapped by `ahriman.core.repository.Repository` class

```
from ahriman.core.repository import Repository
from ahriman.models.pacman_synchronization import PacmanSynchronization

repository = Repository(repository_id, configuration, database,
                        report=True, refresh_pacman_database=PacmanSynchronization.
↪Disabled)
```

And the `repository` instance can be used to perform repository maintenance

```
build_result = repository.process_build(known_packages)
built_packages = repository.packages_built()
update_result = repository.process_update(built_packages)

repository.triggers.on_result(update_result, repository.packages())
```

For the more info please refer to the classes documentation.

3.8 Triggers

The package provides ability to write custom extensions which will be run on (the most) actions, e.g. after updates. By default ahriman provides three types of extensions - reporting, files uploading and PKGBUILD synchronization. Each extension must derive from the `ahriman.core.triggers.Trigger` class and should implement at least one of the abstract methods:

- `on_result` - trigger action which will be called after build process, the build result and the list of repository packages will be supplied as arguments.
- `on_start` - trigger action which will be called right before the start of the application process.
- `on_stop` - action which will be called right before the exit.

Note, it isn't required to implement all of those methods (or even one of them), however, it is highly recommended to avoid trigger actions in `__init__` method as it will be run on any application start (e.g. even if you are just searching in AUR).

3.8.1 Built-in triggers

For the configuration details and settings explanation kindly refer to the *documentation*.

ahriman.core.distributed.WorkerLoaderTrigger

Special trigger to be used to load workers from database on the start of the application rather than configuration. If the option is already set, it will skip processing.

ahriman.core.distributed.WorkerTrigger

Another trigger for the distributed system, which registers itself as remote worker, calling remote service periodically.

ahriman.core.gitremote.RemotePullTrigger

This trigger will be called before any action (`on_start`) and pulls remote PKGBUILD repository locally; after that it copies found PKGBUILDs from the cloned repository to the local cache. It is useful in case if you have patched PGKBUILDs (or even missing in AUR) which you would like to use for package building and, technically, just simplifies the local package building.

In order to update those packages you would need to clone your repository separately, make changes in PKGBUILD (e.g. bump version and update checksums), commit them and push back. On the next ahriman's repository update, it will pull changes you committed and will perform package update.

ahriman.core.gitremote.RemotePushTrigger

This trigger will be called right after build process (`on_result`). It will pick PKGBUILDs for the updated packages, pull them (together with any other files) and commit and push changes to remote repository. No real use cases, but the most of user repositories do it.

ahriman.core.report.ReportTrigger

Trigger which can be used for reporting. It implements `on_result` method and thus being called on each build update and generates report (e.g. html, telegram etc) according to the current settings.

ahriman.core.support.KeyringTrigger

Generator for keyring package. This trigger will extract keys from local keychain and pack them into keyring specific format. This trigger will generate sources including PKGBUILD, which can be used later for package building.

ahriman.core.support.MirrorlistTrigger

Simple generator for mirrorlist package, based on the URLs which were set by configuration. This trigger will generate sources including PKGBUILD, which can be used later for package building.

ahriman.core.upload.UploadTrigger

This trigger takes build result (`on_result`) and performs syncing of the local packages to the remote mirror (e.g. S3 or just by rsync).

3.8.2 Context variables

By default, only configuration and architecture are passed to triggers. However, some triggers might want to have access to other high-level wrappers. In order to provide such ability and avoid (double) initialization, the service provides a global context variables, which can be accessed from `ahriman.core` package:

```
from ahriman.core import context

ctx = context.get()
```

Just because context is wrapped inside `contexvars.ContextVar`, you need to explicitly extract variable by `get()` method. Later you can extract any variable if it is set, e.g.:

```
from ahriman.core.database import SQLite
from ahriman.models.context_key import ContextKey

database = ctx.get(ContextKey("database", SQLite))
```

In order to provide typed API, all variables are stored together with their type. The `get(ContextKey)` method will throw `KeyError` in case if key is missing. Alternatively you can set your own variable inside context:

```
ctx.set(ContextKey("answer", int), 42)
context.set(ctx)
```

Note, however, that there are several limitations:

- Context variables are immutable, thus you cannot override value if the key already presented.
- The `return_type` of `ContextKey` should match the value type, otherwise exception will be thrown.

The `context` also implements collection methods such as `__iter__` and `__len__`.

3.8.3 Trigger example

Lets consider example of reporting trigger (e.g. slack, which provides easy HTTP API for integration triggers).

In order to post message to slack we will need a specific trigger URL (something like `https://hooks.slack.com/services/company_id/trigger_id`), channel (e.g. `#archrepo`) and username (`repo-bot`).

As it has been mentioned, our trigger must derive from specific class:

```
from ahriman.core.triggers import Trigger

class SlackReporter(Trigger):
```

(continues on next page)

(continued from previous page)

```
    def __init__(self, repository_id, configuration):
        Trigger.__init__(self, repository_id, configuration)
        self.slack_url = configuration.get("slack", "url")
        self.channel = configuration.get("slack", "channel")
        self.username = configuration.get("slack", "username")
```

By now we have class with all required variables. Lets implement run method. Slack API requires positing data with specific payload by HTTP, thus:

```
import json
import requests

def notify(result, slack_url, channel, username):
    text = f"""Build has been completed with packages: {", ".join([package.name for␣
↪package in result.success])}"""
    payload = {"channel": channel, "username": username, "text": text}
    response = requests.post(slack_url, data={"payload": json.dumps(payload)})
    response.raise_for_status()
```

Obviously you can implement the specified method in class, but for guide purpose it has been done as separated method. Now we can merge this method into the class:

```
class SlackReporter(Trigger):

    def __init__(self, repository_id, configuration):
        Trigger.__init__(self, repository_id, configuration)
        self.slack_url = configuration.get("slack", "url")
        self.channel = configuration.get("slack", "channel")
        self.username = configuration.get("slack", "username")

    def on_result(self, result, packages):
        notify(result, self.slack_url, self.channel, self.username)
```

Setup the trigger

First, put the trigger in any path it can be exported, e.g. by packing the resource into python package (which will lead to import path as `package.slack_reporter.SlackReporter`) or just put file somewhere it can be accessed by application (e.g. `/usr/local/lib/slack_reporter.SlackReporter`).

After that run application as usual and receive notification in your slack channel.

Trigger configuration schema

Triggers can expose their configuration schema. It can be achieved by implementing `CONFIGURATION_SCHEMA` class variable according to cerberus documentation. For more details and examples, please refer to built-in triggers implementations.

3.9 Modules

3.9.1 ahriman package

Subpackages

ahriman.application package

Subpackages

ahriman.application.application package

Subpackages

ahriman.application.application.workers package

Submodules

ahriman.application.application.workers.local_updater module

class LocalUpdater(*repository:* Repository)

Bases: *Updater*

local build process implementation

repository

repository instance

Type
Repository

default constructor

Parameters
repository (Repository) – repository instance

partition(*packages: Iterable[*Package*]*) → list[list[*Package*]]

split packages into partitions to be processed by this worker

Parameters
packages (*Iterable[*Package*]*) – list of packages to partition

Returns
packages partitioned by this worker type

Return type
list[list[*Package*]]

update(*updates: Iterable[Package]*, *packagers: Packagers | None = None*, ***, *bump_pkgrel: bool = False*) → *Result*

run package updates

Parameters

- **updates** (*Iterable[Package]*) – list of packages to update
- **packagers** (*Packagers | None, optional*) – optional override of username for build process (Default value = None)
- **bump_pkgrel** (*bool, optional*) – bump pkgrel in case of local version conflict (Default value = False)

Returns
update result

Return type
Result

ahriman.application.application.workers.remote_updater module

class RemoteUpdater(*workers: list[Worker]*, *repository_id: RepositoryId*, *configuration: Configuration*)

Bases: *Updater*

remote update worker

configuration

configuration instance

Type
Configuration

repository_id

repository unique identifier

Type
RepositoryId

workers

worker identifiers

Type
list[*Worker*]

default constructor

Parameters

- **workers** (*list[Worker]*) – worker identifiers
- **repository_id** (*RepositoryId*) – repository unique identifier
- **configuration** (*Configuration*) – configuration instance

next_worker() → tuple[*Worker*, *SyncAhrimanClient*]

generate next not-used web client. In case if all clients have been already used, it yields next not used client

Returns
worker and constructed client instance for the web

Return type
tuple[*Worker*, *SyncAhrimanClient*]

partition(*packages: Iterable[*Package*]*) → list[list[*Package*]]

split packages into partitions to be processed by this worker

Parameters
packages (*Iterable[*Package*]*) – list of packages to partition

Returns
packages partitioned by this worker type

Return type
list[list[*Package*]]

update(*updates: Iterable[*Package*]*, *packagers:* Packagers | *None* = *None*, *, *bump_pkgrel: bool* = *False*) → *Result*

run package updates

Parameters

- **updates** (*Iterable[*Package*]*) – list of packages to update
- **packagers** (*Packagers | None, optional*) – optional override of username for build process (Default value = None)
- **bump_pkgrel** (*bool, optional*) – bump pkgrel in case of local version conflict (Default value = False)

Returns
update result

Return type
Result

property clients: dict[*Worker*, *SyncAhrimanClient*]

extract loaded clients. Note that this method yields only workers which have been already loaded

Returns
map of the worker to the related web client

Return type
dict[*Worker*, *SyncAhrimanClient*]

ahriman.application.application.workers.updater module

class Updater

Bases: *LazyLogging*

updater handler interface

static load(*repository_id:* RepositoryId, *configuration:* Configuration, *repository:* Repository, *workers: list[*Worker*]* | *None* = *None*) → *Updater*

construct updaters from parameters

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **repository** (Repository) – repository instance

- **workers** (*list[Worker] | None, optional*) – worker identifiers if any (Default value = None)

Returns
constructed updater worker

Return type
Updater

partition(*packages: Iterable[Package]*) → list[list[*Package*]]

split packages into partitions to be processed by this worker

Parameters
packages (*Iterable[Package]*) – list of packages to partition

Returns
packages partitioned by this worker type

Return type
list[list[*Package*]]

Raises
NotImplementedError – not implemented method

update(*updates: Iterable[Package]*, *packagers:* Packagers | *None* = *None*, ***, *bump_pkgrel: bool* = *False*) → *Result*

run package updates

Parameters

- **updates** (*Iterable[Package]*) – list of packages to update
- **packagers** (*Packagers | None, optional*) – optional override of username for build process (Default value = None)
- **bump_pkgrel** (*bool, optional*) – bump pkgrel in case of local version conflict (Default value = False)

Returns
update result

Return type
Result

Raises
NotImplementedError – not implemented method

Module contents

Submodules

ahriman.application.application.application module

class Application(*repository_id:* RepositoryId, *configuration:* Configuration, ***, *report: bool*, *refresh_pacman_database:* PacmanSynchronization = *PacmanSynchronization.Disabled*)

Bases: *ApplicationPackages*, *ApplicationRepository*

base application class

Examples

This class groups *ahriman.core.repository.repository.Repository* methods into specific method which process all supposed actions caused by underlying action. E.g.:

```
>>> from ahriman.core.configuration import Configuration
>>> from ahriman.models.package_source import PackageSource
>>> from ahriman.models.repository_id import RepositoryId
>>>
>>> configuration = Configuration()
>>> application = Application(RepositoryId("x86_64", "x86_64"), configuration,␣
↪report=True)
>>> # add packages to build queue
>>> application.add(["ahriman"], PackageSource.AUR)
>>>
>>> # check for updates
>>> updates = application.updates([], aur=True, local=True, manual=True, vcs=True)
>>> # updates for specified packages
>>> application.update(updates)
```

In case if specific actions or their order are required, the direct access to *ahriman.core.repository.repository.Repository* must be used instead.

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting
- **refresh_pacman_database** (PacmanSynchronization, *optional*) – pacman database synchronization level (Default value = PacmanSynchronization.Disabled)

on_result(*result:* Result) → None

generate report and sync to remote server

Parameters

result (Result) – build result

on_start() → None

run triggers on start of the application

on_stop() → None

run triggers on stop of the application. Note, however, that in most cases this method should not be called directly as it will be called after on_start action

print_updates(*packages: list[*Package*]*, ***, *log_fn: Callable[[str], None]*) → None

print list of packages to be built. This method will build dependency tree and print updates accordingly

Parameters

- **packages** (*list[*Package*]*) – package list to be printed
- **log_fn** (*Callable[[str], None]*) – logger function to log updates

with_dependencies(*packages: list[*Package*]*, ***, *process_dependencies: bool*) → list[*Package*]

add missing dependencies to list of packages. This will extract known packages, check dependencies of the supplied packages and add packages which are not presented in the list of known packages.

Parameters

- **packages** (*list[Package]*) – list of source packages of which dependencies have to be processed
- **process_dependencies** (*bool*) – if no set, dependencies will not be processed

Returns
updated packages list. Packager for dependencies will be copied from original package

Return type
list[*Package*]

Examples

In the most cases, in order to avoid build failure, it is required to add missing packages, which can be done by calling:

```
>>> application = ...
>>> packages = application.with_dependencies(packages, process_
↪dependencies=True)
>>> application.print_updates(packages, log_fn=print)
```

ahriman.application.application.application_packages module

class ApplicationPackages(*repository_id: RepositoryId*, *configuration: Configuration*, *, *report: bool*, *refresh_pacman_database: PacmanSynchronization = PacmanSynchronization.Disabled*)

Bases: *ApplicationProperties*

package control class

default constructor

Parameters

- **repository_id** (*RepositoryId*) – repository unique identifier
- **configuration** (*Configuration*) – configuration instance
- **report** (*bool*) – force enable or disable reporting
- **refresh_pacman_database** (*PacmanSynchronization, optional*) – pacman database synchronization level (Default value = PacmanSynchronization.Disabled)

add(*names: Iterable[str]*, *source: PackageSource*, *username: str | None = None*) → None

add packages for the next build

Parameters

- **names** (*Iterable[str]*) – list of package bases to add
- **source** (*PackageSource*) – package source to add
- **username** (*str | None, optional*) – optional override of username for build process (Default value = None)

on_result(*result:* Result) → None

generate report and sync to remote server

Parameters
result (Result) – build result

Raises
NotImplementedError – not implemented method

remove(*names: Iterable[str]*) → *Result*

remove packages from repository

Parameters
names (*Iterable[str]*) – list of packages (either base or name) to remove

Returns
removal result

Return type
Result

ahriman.application.application.application_properties module

class ApplicationProperties(*repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*, *refresh_pacman_database:* PacmanSynchronization = *PacmanSynchronization.Disabled*)

Bases: *LazyLogging*

application base properties class

configuration

configuration instance

Type
Configuration

database

database instance

Type
SQLite

repository

repository instance

Type
Repository

repository_id

repository unique identifier

Type
RepositoryId

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance

- **report** (*bool*) – force enable or disable reporting
- **refresh_pacman_database** (PacmanSynchronization, *optional*) – pacman database synchronization level (Default value = PacmanSynchronization.Disabled)

property architecture: str

repository architecture for backward compatibility

Returns
repository architecture

Return type
str

ahriman.application.application.application_repository module

class ApplicationRepository(*repository_id: RepositoryId*, *configuration: Configuration*, *, *report: bool*, *refresh_pacman_database: PacmanSynchronization = PacmanSynchronization.Disabled*)

Bases: *ApplicationProperties*

repository control class

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting
- **refresh_pacman_database** (PacmanSynchronization, *optional*) – pacman database synchronization level (Default value = PacmanSynchronization.Disabled)

changes(*packages: Iterable[Package]*) → None

generate and update package changes

Parameters
packages (*Iterable[Package]*) – list of packages to retrieve changes

clean(*, *cache: bool*, *chroot: bool*, *manual: bool*, *packages: bool*, *pacman: bool*) → None

run all clean methods. Warning: some functions might not be available for non-root user

Parameters

- **cache** (*bool*) – clear directory with package caches
- **chroot** (*bool*) – clear build chroot
- **manual** (*bool*) – clear directory with manually added packages' bases
- **packages** (*bool*) – clear directory with built packages
- **pacman** (*bool*) – clear directory with pacman databases

on_result(*result: Result*) → None

generate report and sync to remote server

Parameters
result (Result) – build result

Raises
NotImplementedError – not implemented method

sign(*packages: Iterable[str]*) → None
sign packages and repository

Parameters
packages (*Iterable[str]*) – only sign specified packages

unknown() → list[str]
get packages which were not found in AUR

Returns
unknown package archive list

Return type
list[str]

update(*updates: Iterable[*Package*]*, *packagers:* Packagers | *None* = *None*, *, *bump_pkgrel: bool* = *False*) → *Result*
run package updates. This method will separate update in the several steps:

1. Check already built packages.
2. Construct builder instance.
3. Delegate build process to the builder instance (either remote or local).

Parameters
- **updates** (*Iterable[*Package*]*) – list of packages to update
- **packagers** (Packagers | *None, optional*) – optional override of username for build process (Default value = None)
- **bump_pkgrel** (*bool, optional*) – bump pkgrel in case of local version conflict (Default value = False)

Returns
update result

Return type
Result

updates(*filter_packages: Iterable[str]*, *, *aur: bool*, *local: bool*, *manual: bool*, *vcs: bool*) → list[*Package*]
get list of packages to run update process

Parameters
- **filter_packages** (*Iterable[str]*) – do not check every package just specified in the list
- **aur** (*bool*) – enable or disable checking for AUR updates
- **local** (*bool*) – enable or disable checking of local packages for updates
- **manual** (*bool*) – include or exclude manual updates
- **vcs** (*bool*) – enable or disable checking of VCS packages

Returns
list of out-of-dated packages

Return type
list[*Package*]

ahriman.application.application.updates_iterator module

class FixedUpdatesIterator(*application: Application*, *interval: int*)

Bases: *UpdatesIterator*

implementation of the *UpdatesIterator* which always emits empty list, which is the same as update all

default constructor

Parameters

- **application** (*Application*) – application instance
- **interval** (*int*) – predefined interval for updates

select_packages() → tuple[list[str] | None, int]

select next packages partition for updates

Returns
packages partition for updates if any and total amount of partitions.

Return type
tuple[list[str] | None, int]

class UpdatesIterator(*application: Application*, *interval: int*)

Bases: Iterator[list[str] | None]

class-helper for iteration over packages to check for updates. It yields list of packages which were not yet updated

application

application instance

Type
Application

interval

predefined interval for updates. The updates will be split into chunks in the way in which all packages will be updated in the specified interval

Type
int

updated_packages

list of packages which have been already updated

Type
set[str]

Examples

Typical usage of this class is something like:

```
>>> application = ...
>>> iterator = UpdatesIterator(application, None)
>>>
>>> for updates in iterator:
>>>     print(updates)
```

default constructor

Parameters

- **application** (Application) – application instance
- **interval** (*int*) – predefined interval for updates

select_packages() → tuple[list[str] | None, int]

select next packages partition for updates

Returns
packages partition for updates if any and total amount of partitions.

Return type
tuple[list[str] | None, int]

Module contents

ahriman.application.handlers package

Submodules

ahriman.application.handlers.add module

class Add

Bases: *Handler*

add packages handler

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.backup module

class Backup

Bases: *Handler*

backup packages handler

static get_paths(*configuration:* Configuration) → set[Path]

extract paths to back up

Parameters
configuration (Configuration) – configuration instance

Returns
map of the filesystem paths

Return type
set[Path]

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.change module

class Change

Bases: *Handler*

package changes handler

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.clean module

class Clean

Bases: *Handler*

clean caches handler

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.daemon module

class Daemon

Bases: *Handler*

daemon packages handler

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.dump module

class Dump

Bases: *Handler*

dump configuration handler

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance

- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.handler module

class Handler

Bases: object

base handler class for command callbacks

ALLOW_MULTI_ARCHITECTURE_RUN

(class attribute) allow running with multiple architectures

Type
bool

Examples

Wrapper for all command line actions, though each derived class implements *run()* method, it usually must not be called directly. The recommended way is to call *execute()* class method, e.g.:

```
>>> from ahriman.application.handlers import Add
>>>
>>> Add.execute(args)
```

classmethod call(*args: Namespace*, *repository_id: RepositoryId*) → bool

additional function to wrap all calls for multiprocessing library

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (*RepositoryId*) – repository unique identifier

Returns
True on success, False otherwise

Return type
bool

static check_if_empty(*enabled: bool*, *predicate: bool*) → None

check condition and flag and raise ExitCode exception in case if it is enabled and condition match

Parameters

- **enabled** (*bool*) – if False no check will be performed
- **predicate** (*bool*) – indicates condition on which exception should be thrown

Raises
ExitCode – if result is empty and check is enabled

classmethod execute(*args: Namespace*) → int

execute function for all aru

Parameters
args (*argparse.Namespace*) – command line args

Returns
0 on success, 1 otherwise

Return type
int

Raises
MultipleArchitecturesError – if more than one architecture supplied and no multi architecture supported

static repositories_extract(*args: Namespace*) → list[*RepositoryId*]

get known architectures

Parameters
args (*argparse.Namespace*) – command line args

Returns
list of repository names and architectures for which tree is created

Return type
list[*RepositoryId*]

Raises
MissingArchitectureError – if no architecture set and automatic detection is not allowed or failed

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting

Raises
NotImplementedError – not implemented method

ahriman.application.handlers.help module

class Help

Bases: *Handler*

help handler

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.key_import module

class KeyImport

Bases: *Handler*

key import packages handler

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.patch module

class Patch

Bases: *Handler*

patch control handler

static patch_create_from_diff(*sources_dir: Path*, *architecture: str*, *track: list[str]*) → tuple[str, *PkgbuildPatch*]

create PKGBUILD plain diff patches from sources directory

Parameters

- **sources_dir** (*Path*) – path to directory with the package sources
- **architecture** (*str*) – repository architecture
- **track** (*list[str]*) – track files which match the glob before creating the patch

Returns
package base and created PKGBUILD patch based on the diff from master HEAD to current files

Return type
tuple[str, *PkgbuildPatch*]

static patch_create_from_function(*variable: str*, *patch_path: Path | None*) → *PkgbuildPatch*

create single-function patch set for the package base

Parameters

- **variable** (*str*) – function or variable name inside PKGBUILD
- **patch_path** (*Path | None*) – optional path to patch content. If not set, it will be read from stdin

Returns
created patch for the PKGBUILD function

Return type
PkgbuildPatch

static patch_set_create(*application: Application*, *package_base: str*, *patch: PkgbuildPatch*) → None

create patch set for the package base

Parameters

- **application** (*Application*) – application instance
- **package_base** (*str*) – package base
- **patch** (*PkgbuildPatch*) – patch descriptor

static patch_set_list(*application: Application*, *package_base: str | None*, *variables: list[str] | None*, *exit_code: bool*) → None

list patches available for the package base

Parameters

- **application** (*Application*) – application instance
- **package_base** (*str | None*) – package base
- **variables** (*list[str] | None*) – extract patches only for specified PKGBUILD variables
- **exit_code** (*bool*) – exit with error on empty search result

static patch_set_remove(*application: Application*, *package_base: str*, *variables: list[str] | None*) → None

remove patch set for the package base

Parameters

- **application** (*Application*) – application instance
- **package_base** (*str*) – package base
- **variables** (*list[str] | None*) – remove patches only for specified PKGBUILD variables

classmethod run(*args: Namespace*, *repository_id: RepositoryId*, *configuration: Configuration*, ***, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (*RepositoryId*) – repository unique identifier
- **configuration** (*Configuration*) – configuration instance
- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.rebuild module

class Rebuild

Bases: *Handler*

make world handler

static extract_packages(*application:* Application, *status:* BuildStatusEnum | *None*, *, *from_database:* *bool*) → list[*Package*]

extract packages from database file

Parameters

- **application** (*Application*) – application instance
- **status** (*BuildStatusEnum | None*) – optional filter by package status
- **from_database** (*bool*) – extract packages from database instead of repository filesystem

Returns
list of packages which were stored in database

Return type
list[*Package*]

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report:* *bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (*RepositoryId*) – repository unique identifier
- **configuration** (*Configuration*) – configuration instance
- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.remove module

class Remove

Bases: *Handler*

remove packages handler

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report:* *bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (*RepositoryId*) – repository unique identifier
- **configuration** (*Configuration*) – configuration instance
- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.remove_unknown module

class RemoveUnknown

Bases: *Handler*

remove unknown packages handler

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.repositories module

class Repositories

Bases: *Handler*

repositories listing handler

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.restore module

class Restore

Bases: *Handler*

restore packages handler

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance

- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.run module

class Run

Bases: *Handler*

multicommand handler

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting

static run_command(*command: list[str]*, *parser: ArgumentParser*) → bool

run command specified by the argument

Parameters

- **command** (*list[str]*) – command to run
- **parser** (*argparse.ArgumentParser*) – generated argument parser

Returns
status of the command

Return type
bool

ahriman.application.handlers.search module

class Search

Bases: *Handler*

packages search handler

SORT_FIELDS

(class attribute) allowed fields to sort the package list

Type
set[str]

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier

- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting

static sort(*packages: Iterable[*AURPackage*]*, *sort_by: str*) → list[*AURPackage*]

sort package list by specified field

Parameters

- **packages** (*Iterable[*AURPackage*]*) – packages list to sort
- **sort_by** (*str*) – AUR package field name to sort by

Returns
sorted list for packages

Return type
list[*AURPackage*]

Raises
OptionError – if search fields is not in list of allowed ones

ahriman.application.handlers.service_updates module

class ServiceUpdates

Bases: *Handler*

service updates handler

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.setup module

class Setup

Bases: *Handler*

setup handler

ARCHBUILD_COMMAND_PATH

(class attribute) default devtools command

Type
Path

MIRRORLIST_PATH

(class attribute) path to pacman default mirrorlist (used by multilib repository)

Type
Path

SUDOERS_DIR_PATH

(class attribute) path to sudoers.d includes directory

Type
Path

static build_command(*root: Path*, *repository_id:* RepositoryId) → Path

generate build command name

Parameters

- **root** (*Path*) – root directory for the build command (must be root of the repository)
- **repository_id** (RepositoryId) – repository unique identifier

Returns
valid devtools command name

Return type
Path

static configuration_create_ahriman(*args: Namespace*, *repository_id:* RepositoryId, *root:* Configuration) → None

create service specific configuration

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **root** (Configuration) – root configuration instance

static configuration_create_devtools(*repository_id:* RepositoryId, *source: Path*, *mirror: str | None*, *multilib: bool*, *repository_server: str*) → None

create configuration for devtools based on `source` configuration

Notes

devtools does not allow to specify the pacman configuration, thus we still have to use configuration in /usr

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **source** (*Path*) – path to source configuration file
- **mirror** (*str | None*) – link to package server mirror
- **multilib** (*bool*) – add or do not multilib repository to the configuration
- **repository_server** (*str*) – url of the repository

static configuration_create_makepkg(*packager: str*, *makeflags_jobs: bool*, *paths:* RepositoryPaths) → None

create configuration for makepkg

Parameters

- **packager** (*str*) – packager identifier (e.g. name, email)
- **makeflags_jobs** (*bool*) – set MAKEFLAGS variable to number of cores
- **paths** (RepositoryPaths) – repository paths instance

static configuration_create_sudo(*paths:* RepositoryPaths, *repository_id:* RepositoryId) → None

create configuration to run build command with sudo without password

Parameters

- **paths** (RepositoryPaths) – repository paths instance
- **repository_id** (RepositoryId) – repository unique identifier

static executable_create(*paths:* RepositoryPaths, *repository_id:* RepositoryId) → None

create executable for the service

Parameters

- **paths** (RepositoryPaths) – repository paths instance
- **repository_id** (RepositoryId) – repository unique identifier

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.shell module

class Shell

Bases: *Handler*

python shell handler

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.sign module

class Sign

Bases: *Handler*

(re-)sign handler

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.status module

class Status

Bases: *Handler*

package status handler

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.status_update module

class StatusUpdate

Bases: *Handler*

status update handler

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance

- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.structure module

class Structure

Bases: *Handler*

dump repository structure handler

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.tree_migrate module

class TreeMigrate

Bases: *Handler*

tree migration handler

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting

static tree_move(*from_tree:* RepositoryPaths, *to_tree:* RepositoryPaths) → None

move files between trees. Trees must be created in advance

Parameters

- **from_tree** (RepositoryPaths) – old repository tree
- **to_tree** (RepositoryPaths) – new repository tree

ahriman.application.handlers.triggers module

class Triggers

Bases: *Handler*

triggers handlers

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.unsafe_commands module

class UnsafeCommands

Bases: *Handler*

unsafe command help parser

static check_unsafe(*command: list[str]*, *unsafe_commands: list[str]*, *parser: ArgumentParser*) → None

check if command is unsafe

Parameters

- **command** (*str*) – command to check
- **unsafe_commands** (*list[str]*) – list of unsafe commands
- **parser** (*argparse.ArgumentParser*) – generated argument parser

static get_unsafe_commands(*parser: ArgumentParser*) → list[str]

extract unsafe commands from argument parser

Parameters
parser (*argparse.ArgumentParser*) – generated argument parser

Returns
list of commands with default unsafe flag

Return type
list[str]

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance

- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.update module

class Update

Bases: *Handler*

package update handler

static log_fn(*application: Application*, *dry_run: bool*) → Callable[[str], None]

package updates log function

Parameters

- **application** (*Application*) – application instance
- **dry_run** (*bool*) – do not perform update itself

Returns

in case if dry_run is set it will return print, logger otherwise

Return type

Callable[[str], None]

classmethod run(*args: Namespace*, *repository_id: RepositoryId*, *configuration: Configuration*, ***, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (*RepositoryId*) – repository unique identifier
- **configuration** (*Configuration*) – configuration instance
- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.users module

class Users

Bases: *Handler*

user management handler

classmethod run(*args: Namespace*, *repository_id: RepositoryId*, *configuration: Configuration*, ***, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (*RepositoryId*) – repository unique identifier
- **configuration** (*Configuration*) – configuration instance
- **report** (*bool*) – force enable or disable reporting

static user_create(*args: Namespace*) → *User*

create user descriptor from arguments

Parameters
args (*argparse.Namespace*) – command line args

Returns
built user descriptor

Return type
User

Raises
PasswordError – password input is invalid

ahriman.application.handlers.validate module

class Validate

Bases: *Handler*

configuration validator handler

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting

static schema(*repository_id:* RepositoryId, *configuration:* Configuration) → dict[str, dict[str, Any]]

get schema with triggers

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance

Returns
configuration validation schema

Return type
ConfigurationSchema

static schema_erase_required(*schema: dict[str, dict[str, Any]]*) → dict[str, dict[str, Any]]

recursively remove required field from supplied cerberus schema

Parameters
schema (*ConfigurationSchema*) – source schema from which required field must be removed

Returns
schema without required fields. Note, that source schema will be modified in-place

Return type
ConfigurationSchema

static schema_merge(*source: dict[str, Any]*, *schema: dict[str, Any]*) → dict[str, Any]

merge child schema into source. In case if source already contains values, new keys will be added
(possibly with overrides - in case if such key already set also)

Parameters

- **source** (*dict[str, Any]*) – source (current) schema into which will be merged
- **schema** (*dict[str, Any]*) – new schema to be merged

Returns
schema with added elements from source schema if they were set before and not presented in the new one. Note, that schema will be modified in-place

Return type
dict[str, Any]

ahriman.application.handlers.versions module

class Versions

Bases: *Handler*

version handler

PEP423_PACKAGE_NAME

(class attribute) special regex for valid PEP423 package name

Type
str

static package_dependencies(*root: str*) → Generator[tuple[str, str], None, None]

extract list of ahriman package dependencies installed into system with their versions

Parameters
root (*str*) – root package name

Yields
tuple[str, str] – map of installed dependency to its version

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting

ahriman.application.handlers.web module

class Web

Bases: *Handler*

web server handler

static extract_arguments(*args: Namespace*, *configuration:* Configuration) → Generator[str, None, None]

extract list of arguments used for current command, except for command specific ones

Parameters

- **args** (*argparse.Namespace*) – command line args
- **configuration** (Configuration) – configuration instance

Yields

str – command line arguments which were used for this specific command

classmethod run(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration, *, *report: bool*) → None

callback for command line

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting

Module contents

Submodules

ahriman.application.ahriman module

ahriman.application.lock module

class Lock(*args: Namespace*, *repository_id:* RepositoryId, *configuration:* Configuration)

Bases: *LazyLogging*

wrapper for application lock file

force

remove lock file on start if any

Type
bool

path

path to lock file if any

Type
Path

reporter

build status reporter instance

Type
Client

paths

repository paths instance

Type
RepositoryPaths

unsafe

skip user check

Type
bool

wait_timeout

wait in seconds until lock will free

Type
int

Examples

Instance of this class except for controlling file-based lock is also required for basic applications checks. The common flow is to create instance in `with` block and handle exceptions after all:

```
>>> from ahriman.core.configuration import Configuration
>>> from ahriman.models.repository_id import RepositoryId
>>>
>>> configuration = Configuration()
>>> try:
>>>     with Lock(args, RepositoryId("x86_64", "aur-clone"), configuration):
>>>         perform_actions()
>>> except Exception as exception:
>>>     handle_exceptions(exception)
```

default constructor

Parameters

- **args** (*argparse.Namespace*) – command line args
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance

check_user() → None

check if current user is actually owner of ahriman root

check_version() → None

check web server version

clear() → None

remove lock file

create() → None
create lock file

Raises
DuplicateRunError – if lock exists and no force flag supplied

watch() → None
watch until lock disappear

Module contents

ahriman.core package

Subpackages

ahriman.core.alpm package

Subpackages

ahriman.core.alpm.remote package

Submodules

ahriman.core.alpm.remote.aur module

class AUR(*configuration: Configuration | None = None*, *section: str | None = None*, *, *suppress_errors: bool = False*)

Bases: *Remote*

AUR RPC wrapper

DEFAULT_AUR_URL
(class attribute) default AUR url

Type
str

DEFAULT_RPC_URL
(class attribute) default AUR RPC url

Type
str

DEFAULT_RPC_VERSION
(class attribute) default AUR RPC version

Type
str

default constructor

Parameters

- **configuration** (*Configuration | None, optional*) – configuration instance (Default value = None)

- **section** (*str | None, optional*) – settings section name (Default value = None)
- **suppress_errors** (*bool, optional*) – suppress logging of request errors (Default value = False)

aur_request(*request_type: str*, **args: str*, ***kwargs: str*) → list[*AURPackage*]

perform request to AUR RPC

Parameters

- **request_type** (*str*) – AUR request type, e.g. search, info
- ***args** (*str*) – list of arguments to be passed as args query parameter
- ****kwargs** (*str*) – list of additional named parameters like by

Returns
response parsed to package list

Return type
list[*AURPackage*]

package_info(*package_name: str*, ***, *pacman:* Pacman | *None*) → *AURPackage*

get package info by its name

Parameters

- **package_name** (*str*) – package name to search
- **pacman** (Pacman | *None*) – alpm wrapper instance, required for official repositories search

Returns
package which match the package name

Return type
AURPackage

Raises
UnknownPackageError – package doesn't exist

package_search(**keywords: str*, *pacman:* Pacman | *None*) → list[*AURPackage*]

search package in AUR web

Parameters

- ***keywords** (*str*) – keywords to search
- **pacman** (Pacman | *None*) – alpm wrapper instance, required for official repositories search

Returns
list of packages which match the criteria

Return type
list[*AURPackage*]

static parse_response(*response: dict[str, Any]*) → list[*AURPackage*]

parse RPC response to package list

Parameters
response (*dict[str, Any]*) – RPC response json

Returns
list of parsed packages

Return type
list[*AURPackage*]

Raises
PackageInfoError – for error API response

classmethod remote_git_url(*package_base: str*, *repository: str*) → str

generate remote git url from the package base

Args
package_base(str): package base repository(str): repository name

Returns
git url for the specific base

Return type
str

classmethod remote_web_url(*package_base: str*) → str

generate remote web url from the package base

Args
package_base(str): package base

Returns
web url for the specific base

Return type
str

ahriman.core.alpm.remote.official module

class Official(*configuration: Configuration | None = None*, *section: str | None = None*, ***, *suppress_errors: bool = False*)

Bases: *Remote*

official repository RPC wrapper

DEFAULT_ARCHLINUX_URL

(class attribute) default archlinux url

Type
str

DEFAULT_ARCHLINUX_GIT_URL

(class attribute) default url for git packages

Type
str

DEFAULT_SEARCH_REPOSITORIES

(class attribute) default list of repositories to search

Type
list[str]

DEFAULT_RPC_URL

(class attribute) default archlinux repositories RPC url

Type
str

default constructor

Parameters

- **configuration** (*Configuration | None, optional*) – configuration instance (Default value = None)
- **section** (*str | None, optional*) – settings section name (Default value = None)
- **suppress_errors** (*bool, optional*) – suppress logging of request errors (Default value = False)

arch_request(**args: str*, *by: str*) → list[*AURPackage*]

perform request to official repositories RPC

Parameters

- ***args** (*str*) – list of arguments to be passed as args query parameter
- **by** (*str*) – search by the field

Returns
response parsed to package list

Return type
list[*AURPackage*]

package_info(*package_name: str*, ***, *pacman:* Pacman | *None*) → *AURPackage*

get package info by its name

Parameters

- **package_name** (*str*) – package name to search
- **pacman** (*Pacman | None*) – alpm wrapper instance, required for official repositories search

Returns
package which match the package name

Return type
AURPackage

Raises
UnknownPackageError – package doesn't exist

package_search(**keywords: str*, *pacman:* Pacman | *None*) → list[*AURPackage*]

search package in AUR web

Parameters

- ***keywords** (*str*) – keywords to search
- **pacman** (*Pacman | None*) – alpm wrapper instance, required for official repositories search

Returns
list of packages which match the criteria

Return type
list[*AURPackage*]

static parse_response(*response: dict[str, Any]*) → list[*AURPackage*]

parse RPC response to package list

Parameters
response (*dict[str, Any]*) – RPC response json

Returns
list of parsed packages

Return type
list[*AURPackage*]

Raises
PackageInfoError – for error API response

classmethod remote_git_url(*package_base: str*, *repository: str*) → str

generate remote git url from the package base

Args
package_base(str): package base repository(str): repository name

Returns
git url for the specific base

Return type
str

classmethod remote_web_url(*package_base: str*) → str

generate remote web url from the package base

Args
package_base(str): package base

Returns
web url for the specific base

Return type
str

ahriman.core.alpm.remote.official_syncdb module

class OfficialSyncdb(*configuration: Configuration | None = None*, *section: str | None = None*, ***, *suppress_errors: bool = False*)

Bases: *Official*

official repository wrapper based on synchronized databases.

Despite the fact that official repository provides an API for the interaction according to the comment in issue https://github.com/arcan1s/ahriman/pull/59#issuecomment-1106412297 we might face rate limits while requesting updates.

This approach also has limitations, because we don't require superuser rights (neither going to download database separately), the database file might be outdated and must be handled manually (or kind of). This behaviour might be changed in the future.

Still we leave search function based on the official repositories RPC.

default constructor

Parameters

- **configuration** (*Configuration | None, optional*) – configuration instance (Default value = None)
- **section** (*str | None, optional*) – settings section name (Default value = None)
- **suppress_errors** (*bool, optional*) – suppress logging of request errors (Default value = False)

package_info(*package_name: str*, *, *pacman: Pacman | None*) → *AURPackage*

get package info by its name

Parameters

- **package_name** (*str*) – package name to search
- **pacman** (*Pacman | None*) – alpm wrapper instance, required for official repositories search

Returns
package which match the package name

Return type
AURPackage

Raises
UnknownPackageError – package doesn't exist

ahriman.core.alpm.remote.remote module

class Remote(*configuration: Configuration | None = None*, *section: str | None = None*, *, *suppress_errors: bool = False*)

Bases: *SyncHttpClient*

base class for remote package search

Examples

These classes are designed to be used without instancing. In order to achieve it several class methods are provided: *info()*, *multisearch()* and *search()*. Thus, the basic flow is the following:

```
>>> from ahriman.core.alpm.remote import AUR, Official
>>>
>>> package = AUR.info("ahriman")
>>> search_result = Official.multisearch("pacman", "manager", pacman=pacman)
```

Differnece between *search()* and *multisearch()* is that *search()* passes all arguments to underlying wrapper directly, whereas *multisearch()* splits search one by one and finds intersection between search results.

default constructor

Parameters

- **configuration** (*Configuration | None, optional*) – configuration instance (Default value = None)
- **section** (*str | None, optional*) – settings section name (Default value = None)

- **suppress_errors** (*bool, optional*) – suppress logging of request errors (Default value = False)

classmethod info(*package_name: str*, *, *pacman:* Pacman | *None* = *None*) → *AURPackage*

get package info by its name

Parameters

- **package_name** (*str*) – package name to search
- **pacman** (Pacman | *None, optional*) – alpm wrapper instance, required for official repositories search (Default value = None)

Returns
package which match the package name

Return type
AURPackage

classmethod multisearch(**keywords: str*, *pacman:* Pacman | *None* = *None*) → list[*AURPackage*]

search in remote repository by using API with multiple words. This method is required in order to handle https://bugs.archlinux.org/task/49133. In addition, short words will be dropped

Parameters

- ***keywords** (*str*) – search terms, e.g. “ahriman”, “is”, “cool”
- **pacman** (Pacman | *None, optional*) – alpm wrapper instance, required for official repositories search (Default value = None)

Returns
list of packages each of them matches all search terms

Return type
list[*AURPackage*]

package_info(*package_name: str*, *, *pacman:* Pacman | *None*) → *AURPackage*

get package info by its name

Parameters

- **package_name** (*str*) – package name to search
- **pacman** (Pacman | *None*) – alpm wrapper instance, required for official repositories search

Returns
package which match the package name

Return type
AURPackage

Raises
NotImplementedError – not implemented method

package_search(**keywords: str*, *pacman:* Pacman | *None*) → list[*AURPackage*]

search package in AUR web

Parameters

- ***keywords** (*str*) – keywords to search
- **pacman** (Pacman | *None*) – alpm wrapper instance, required for official repositories search

Returns
list of packages which match the criteria

Return type
list[*AURPackage*]

Raises
NotImplementedError – not implemented method

classmethod remote_git_url(*package_base: str*, *repository: str*) → str

generate remote git url from the package base

Args
package_base(str): package base repository(str): repository name

Returns
git url for the specific base

Return type
str

Raises
NotImplementedError – not implemented method

classmethod remote_web_url(*package_base: str*) → str

generate remote web url from the package base

Args
package_base(str): package base

Returns
web url for the specific base

Return type
str

Raises
NotImplementedError – not implemented method

classmethod search(**keywords: str*, *pacman:* Pacman | *None* = *None*) → list[*AURPackage*]

search package in AUR web

Parameters

- ***keywords** (*str*) – search terms, e.g. “ahriman”, “is”, “cool”
- **pacman** (*Pacman | None, optional*) – alpm wrapper instance, required for official repositories search (Default value = None)

Returns
list of packages which match the criteria

Return type
list[*AURPackage*]

Module contents

Submodules

ahriman.core.alpm.pacman module

class Pacman(*repository_id: RepositoryId*, *configuration: Configuration*, *, *refresh_database: PacmanSynchronization*)

Bases: *LazyLogging*

alpm wrapper

default constructor

Parameters

- **repository_id** (*RepositoryId*) – repository unique identifier
- **configuration** (*Configuration*) – configuration instance
- **refresh_database** (*PacmanSynchronization*) – synchronize local cache to remote

database_copy(*handle: pyalpm.Handle*, *database: pyalpm.DB*, *pacman_root: Path*, *paths: RepositoryPaths*, *, *use_ahriman_cache: bool*) → None

copy database from the operating system root to the ahriman local home

Parameters

- **handle** (*Handle*) – pacman handle which will be used for database copying
- **database** (*DB*) – pacman database instance to be copied
- **pacman_root** (*Path*) – operating system pacman root
- **paths** (*RepositoryPaths*) – repository paths instance
- **use_ahriman_cache** (*bool*) – use local ahriman cache instead of system one

database_init(*handle: pyalpm.Handle*, *repository: str*, *mirror: str*, *architecture: str*) → pyalpm.DB

create database instance from pacman handler and set its properties

Parameters

- **handle** (*Handle*) – pacman handle which will be used for database initializing
- **repository** (*str*) – pacman repository name (e.g. core)
- **mirror** (*str*) – arch linux mirror url
- **architecture** (*str*) – repository architecture

Returns
loaded pacman database instance

Return type
DB

database_sync(*handle: pyalpm.Handle*, *, *force: bool*) → None

sync local database

Parameters

- **handle** (*Handle*) – pacman handle which will be used for database sync

- **force** (*bool*) – force database synchronization (same as `pacman -Syy`)

package_get(*package_name: str*) → Generator[pyalpm.Package, None, None]

retrieve list of the packages from the repository by name

Parameters
package_name (*str*) – package name to search

Yields
Package – list of packages which were returned by the query

packages() → set[str]

get list of packages known for alpm

Returns
list of package names

Return type
set[str]

property handle: pyalpm.Handle

pyalpm handle

Returns
generated pyalpm handle instance

Return type
Handle

ahriman.core.alpm.repo module

class Repo(*name: str*, *paths:* RepositoryPaths, *sign_args: list[str]*)

Bases: *LazyLogging*

repo-add and repo-remove wrapper

name

repository name

Type
str

paths

repository paths instance

Type
RepositoryPaths

sign_args

additional args which have to be used to sign repository archive

Type
list[str]

uid

uid of the repository owner user

Type
int

default constructor

Parameters

- **name** (*str*) – repository name
- **paths** (RepositoryPaths) – repository paths instance
- **sign_args** (*list[str]*) – additional args which have to be used to sign repository archive

add(*path: Path*) → None

add new package to repository

Parameters
path (*Path*) – path to archive to add

init() → None

create empty repository database. It just calls add with empty arguments

remove(*package: str*, *filename: Path*) → None

remove package from repository

Parameters

- **package** (*str*) – package name to remove
- **filename** (*Path*) – package filename to remove

property repo_path: Path

get full path to the repository database

Returns
path to repository database

Return type
Path

Module contents

ahriman.core.auth package

Submodules

ahriman.core.auth.auth module

class Auth(*configuration: Configuration*, *provider: AuthSettings = AuthSettings.Disabled*)

Bases: *LazyLogging*

helper to deal with user authorization

enabled

indicates if authorization is enabled

Type
bool

max_age

session age in seconds. It will be used for both client side and server side checks

Type
int

allow_read_only

allow read only access to APIs

Type
bool

default constructor

Parameters

- **configuration** (Configuration) – configuration instance
- **provider** (AuthSettings, *optional*) – authorization type definition (Default value = AuthSettings.Disabled)

async check_credentials(*username: str | None*, *password: str | None*) → bool

validate user password

Parameters

- **username** (*str | None*) – username
- **password** (*str | None*) – entered password

Returns
True in case if password matches, False otherwise

Return type
bool

async known_username(*username: str | None*) → bool

check if user is known

Parameters
username (*str | None*) – username

Returns
True in case if user is known and can be authorized and False otherwise

Return type
bool

static load(*configuration:* Configuration, *database:* SQLite) → *Auth*

load authorization module from settings

Parameters

- **configuration** (Configuration) – configuration instance
- **database** (SQLite) – database instance

Returns
authorization module according to current settings

Return type
Auth

async verify_access(*username: str*, *required:* UserAccess, *context: str | None*) → bool

validate if user has access to requested resource

Parameters

- **username** (*str*) – username
- **required** (UserAccess) – required access level

- **context** (*str | None*) – URI request path

Returns
True in case if user is allowed to do this request and False otherwise

Return type
bool

property auth_control: str

This workaround is required to make different behaviour for login interface. In case of internal authentication it must provide an interface (modal form) to log in with button sends POST request. But for an external providers behaviour can be different: e.g. OAuth provider requires sending GET request to external resource

Returns
login control as html code to insert

Return type
str

ahriman.core.auth.helpers module

async authorized_userid(**args: Any*, ***kwargs: Any*) → Any

handle aiohttp security methods

Parameters

- ***args** (*Any*) – argument list as provided by authorized_userid function
- ****kwargs** (*Any*) – named argument list as provided by authorized_userid function

Returns
None in case if no aiohttp_security module found and function call otherwise

Return type
Any

async check_authorized(**args: Any*, ***kwargs: Any*) → Any

handle aiohttp security methods

Parameters

- ***args** (*Any*) – argument list as provided by check_authorized function
- ****kwargs** (*Any*) – named argument list as provided by authorized_userid function

Returns
None in case if no aiohttp_security module found and function call otherwise

Return type
Any

async forget(**args: Any*, ***kwargs: Any*) → Any

handle aiohttp security methods

Parameters

- ***args** (*Any*) – argument list as provided by forget function
- ****kwargs** (*Any*) – named argument list as provided by authorized_userid function

Returns
None in case if no aiohttp_security module found and function call otherwise

Return type
Any

async remember(**args: Any*, ***kwargs: Any*) → Any

handle disabled auth

Parameters

- ***args** (*Any*) – argument list as provided by remember function
- ****kwargs** (*Any*) – named argument list as provided by authorized_userid function

Returns
None in case if no aiohttp_security module found and function call otherwise

Return type
Any

ahriman.core.auth.mapping module

class Mapping(*configuration: Configuration*, *database: SQLite*, *provider: AuthSettings = AuthSettings.Configuration*)

Bases: *Auth*

user authorization based on mapping from configuration file

salt

random generated string to salted password

Type
str

database

database instance

Type
SQLite

default constructor

Parameters

- **configuration** (*Configuration*) – configuration instance
- **database** (*SQLite*) – database instance
- **provider** (*AuthSettings, optional*) – authorization type definition (Default value = AuthSettings.Configuration)

async check_credentials(*username: str | None*, *password: str | None*) → bool

validate user password

Parameters

- **username** (*str | None*) – username
- **password** (*str | None*) – entered password

Returns
True in case if password matches, False otherwise

Return type
bool

get_user(*username: str*) → *User* | None

retrieve user from in-memory mapping

Parameters
username (*str*) – username

Returns
user descriptor if username is known and None otherwise

Return type
User | None

async known_username(*username: str | None*) → bool

check if user is known

Parameters
username (*str | None*) – username

Returns
True in case if user is known and can be authorized and False otherwise

Return type
bool

async verify_access(*username: str*, *required:* UserAccess, *context: str | None*) → bool

validate if user has access to requested resource

Parameters

- **username** (*str*) – username
- **required** (UserAccess) – required access level
- **context** (*str | None*) – URI request path

Returns
True in case if user is allowed to do this request and False otherwise

Return type
bool

ahriman.core.auth.oauth module

class OAuth(*configuration:* Configuration, *database:* SQLite, *provider:* AuthSettings = *AuthSettings.OAuth*)

Bases: *Mapping*

User authorization implementation via OAuth. It is required to create application first and put application credentials.

client_id

application client id

Type
str

client_secret

application client secret key

Type
str

icon

icon to be used in login control

Type
str

provider

provider class, should be one of aiohttp-client provided classes

Type
aioauth_client.OAuth2Client

redirect_uri

redirect URI registered in provider

Type
str

scopes

list of scopes required by the application

Type
str

default constructor

Parameters

- **configuration** (Configuration) – configuration instance
- **database** (SQLite) – database instance
- **provider** (AuthSettings, *optional*) – authorization type definition (Default value = AuthSettings.OAuth)

get_client() → OAuth2Client

load client from parameters

Returns
generated client according to current settings

Return type
aioauth_client.OAuth2Client

get_oauth_url() → str

get authorization URI for the specified settings

Returns
authorization URI as a string

Return type
str

async get_oauth_username(*code: str*) → str | None

extract OAuth username from remote

Parameters
code (*str*) – authorization code provided by external service

Returns
username as is in OAuth provider

Return type
str | None

static get_provider(*name: str*) → type[OAuth2Client]

load OAuth2 provider by name

Parameters
name (*str*) – name of the provider. Must be valid class defined in aioauth-client library

Returns
loaded provider type

Return type
type[aioauth_client.OAuth2Client]

Raises
OptionError – in case if invalid OAuth provider name supplied

property auth_control: str

get authorization html control

Returns
login control as html code to insert

Return type
str

Module contents

ahriman.core.build_tools package

Submodules

ahriman.core.build_tools.sources module

class Sources

Bases: *LazyLogging*

helper to download package sources (PKGBUILD etc…) and perform some operations with git

DEFAULT_BRANCH

(class attribute) default branch to process git repositories. Must be used only for local stored repositories, use RemoteSource descriptor instead for real packages

Type
str

DEFAULT_COMMIT_AUTHOR

(class attribute) default commit author to be used if none set

Type
tuple[str, str]

add(*sources_dir: Path*, **pattern: str*, *intent_to_add: bool = False*) → None

track found files via git

Parameters

- **sources_dir** (*Path*) – local path to git repository

- ***pattern** (*str*) – glob patterns
- **intent_to_add** (*bool, optional*) – record only the fact that it will be added later, acts as –intent-to-add git flag (Default value = False)

static changes(*source_dir: Path*, *last_commit_sha: str | None*) → str | None

extract changes from the last known commit if available

Parameters

- **source_dir** (*Path*) – local path to directory with source files
- **last_commit_sha** (*str | None*) – last known commit hash

Returns
changes from the last commit if available or None otherwise

Return type
str | None

commit(*sources_dir: Path*, *message: str | None = None*, *commit_author: tuple[str, str] | None = None*) → bool

commit changes

Parameters

- **sources_dir** (*Path*) – local path to git repository
- **message** (*str | None, optional*) – optional commit message if any. If none set, message will be generated according to the current timestamp (Default value = None)
- **commit_author** (*tuple[str, str] | None, optional*) – optional commit author if any (Default value = None)

Returns
True in case if changes have been committed and False otherwise

Return type
bool

diff(*sources_dir: Path*, *sha: str | None = None*) → str

generate diff from the current version and write it to the output file

Parameters

- **sources_dir** (*Path*) – local path to git repository
- **sha** (*str | None, optional*) – optional commit sha to calculate diff (Default value = None)

Returns
patch as plain string

Return type
str

static extend_architectures(*sources_dir: Path*, *architecture: str*) → list[*PkgbuildPatch*]

extend existing PKGBUILD with repository architecture

Parameters

- **sources_dir** (*Path*) – local path to directory with source files
- **architecture** (*str*) – repository architecture

Returns
generated patch for PKGBUILD architectures if required

Return type
list[*PkgbuildPatch*]

static fetch(*sources_dir: Path*, *remote:* RemoteSource) → str | None

either clone repository or update it to origin/`remote.branch`

Parameters

- **sources_dir** (*Path*) – local path to fetch
- **remote** (RemoteSource) – remote target (from where to fetch)

Returns
current commit sha if available

Return type
str | None

fetch_until(*sources_dir: Path*, ***, *branch: str | None = None*, *commit_sha: str | None = None*) → None

fetch repository until commit sha

Parameters

- **sources_dir** (*Path*) – local path to git repository
- **branch** (*str | None, optional*) – use specified branch (Default value = None)
- **commit_sha** (*str | None, optional*) – commit hash to fetch. If none set, only one will be fetched (Default value = None)

has_changes(*sources_dir: Path*) → bool

check if there are changes in current git tree

Parameters
sources_dir (*Path*) – local path to git repository

Returns
True if there are uncommitted changes and False otherwise

Return type
bool

static has_remotes(*sources_dir: Path*) → bool

check if there are remotes for the repository

Parameters
sources_dir (*Path*) – local path to git repository

Returns
True in case if there is any remote and false otherwise

Return type
bool

head(*sources_dir: Path*, *ref_name: str = 'HEAD'*) → str

extract HEAD reference for the current git repository

Parameters

- **sources_dir** (*Path*) – local path to git repository
- **ref_name** (*str, optional*) – reference name (Default value = “HEAD”)

Returns
HEAD commit hash

Return type
str

static init(*sources_dir: Path*) → None
create empty git repository at the specified path

Parameters
sources_dir (*Path*) – local path to sources

static load(*sources_dir: Path*, *package:* Package, *patches: list[*PkgbuildPatch*]*, *paths:* RepositoryPaths) → str | None
fetch sources from remote and apply patches

Parameters
- **sources_dir** (*Path*) – local path to fetch
- **package** (Package) – package definitions
- **patches** (*list[*PkgbuildPatch*]*) – optional patch to be applied
- **paths** (RepositoryPaths) – repository paths instance

Returns
current commit sha if available

Return type
str | None

move(*pkgbuild_dir: Path*, *sources_dir: Path*) → None
move content from pkgbuild_dir to sources_dir

Parameters
- **pkgbuild_dir** (*Path*) – path to directory with pkgbuild from which need to move
- **sources_dir** (*Path*) – path to target directory

patch_apply(*sources_dir: Path*, *patch:* PkgbuildPatch) → None
apply patches if any

Parameters
- **sources_dir** (*Path*) – local path to directory with git sources
- **patch** (PkgbuildPatch) – patch to be applied

static patch_create(*sources_dir: Path*, **pattern: str*) → str
create patch set for the specified local path

Parameters
- **sources_dir** (*Path*) – local path to git repository
- ***pattern** (*str*) – glob patterns

Returns
patch as plain text

Return type
str

static push(*sources_dir: Path*, *remote:* RemoteSource, **pattern: str*, *commit_author: tuple[str, str] | None = None*) → None

commit selected changes and push files to the remote repository

Parameters

- **sources_dir** (*Path*) – local path to git repository
- **remote** (RemoteSource) – remote target, branch and url
- ***pattern** (*str*) – glob patterns
- **commit_author** (*tuple[str, str] | None, optional*) – commit author if any (Default value = None)

ahriman.core.build_tools.task module

class Task(*package:* Package, *configuration:* Configuration, *architecture: str*, *paths:* RepositoryPaths)

Bases: *LazyLogging*

base package build task

archbuild_flags

command flags for archbuild command

Type
list[str]

architecture

repository architecture

Type
str

build_command

build command

Type
str

include_debug_packages

whether to include debug packages or not

Type
bool

makechrootpkg_flags

command flags for makechrootpkg command

Type
list[str]

makepkg_flags

command flags for makepkg command

Type
list[str]

package

package definitions

Type
Package

paths

repository paths instance

Type
RepositoryPaths

uid

uid of the repository owner user

Type
int

default constructor

Parameters

- **package** (Package) – package definitions
- **configuration** (Configuration) – configuration instance
- **architecture** (*str*) – repository architecture
- **paths** (RepositoryPaths) – repository paths instance

build(*sources_dir: Path*, ***kwargs: str | None*) → list[Path]

run package build

Parameters

- **sources_dir** (*Path*) – path to where sources are
- ****kwargs** (*str | None*) – environment variables to be passed to build processes

Returns
paths of produced packages

Return type
list[Path]

init(*sources_dir: Path*, *database:* SQLite, *local_version: str | None*) → str | None

fetch package from git

Parameters

- **sources_dir** (*Path*) – local path to fetch
- **database** (SQLite) – database instance
- **local_version** (*str | None*) – local version of the package. If set and equal to current version, it will automatically bump pkgrel

Returns
current commit sha if available

Return type
str | None

Module contents

ahriman.core.configuration package

Submodules

ahriman.core.configuration.configuration module

class Configuration(*allow_no_value: bool = False*)

Bases: RawConfigParser

extension for built-in configuration parser

ARCHITECTURE_SPECIFIC_SECTIONS

(class attribute) known sections which can be architecture specific. Required by dump and merging functions

Type
list[str]

SYSTEM_CONFIGURATION_PATH

(class attribute) default system configuration path distributed by package

Type
Path

includes

list of includes which were read

Type
list[Path]

path

path to root configuration file

Type
Path | None

repository_id

repository unique identifier

Type
RepositoryId | None

Examples

Configuration class provides additional method in order to handle application configuration. Since this class is derived from built-in `configparser.RawConfigParser` class, the same flow is applicable here. Nevertheless, it is recommended to use *from_path()* class method which also calls initialization methods:

```
>>> from pathlib import Path
>>>
>>> configuration = Configuration.from_path(Path("/etc/ahriman.ini"), RepositoryId(
↪"x86_64", "aur-clone"))
>>> repository_name = configuration.get("repository", "name")
>>> makepkg_flags = configuration.getlist("build", "makepkg_flags")
```

The configuration instance loaded in this way will contain only sections which are defined for the specified architecture according to the merge rules. Moreover, the architecture names will be removed from section names.

In order to get current settings, the *check_loaded()* method can be used. This method will raise an *ahriman.core.exceptions.InitializeError* in case if configuration was not yet loaded:

```
>>> path, repository_id = configuration.check_loaded()
```

default constructor. In the most cases must not be called directly

Parameters

allow_no_value (*bool, optional*) – copies `configparser.RawConfigParser` behaviour. In case if it is set to `True`, the keys without values will be allowed (Default value = False)

check_loaded() → tuple[Path, *RepositoryId*]

check if service was actually loaded

Returns

configuration root path and architecture if loaded

Return type

tuple[Path, *RepositoryId*]

Raises

InitializeError – in case if architecture and/or path are not set

dump() → dict[str, dict[str, str]]

dump configuration to dictionary

Returns

configuration dump for specific architecture

Return type

dict[str, dict[str, str]]

classmethod from_path(*path: Path*, *repository_id:* RepositoryId) → Self

constructor with full object initialization

Parameters

- **path** (*Path*) – path to root configuration file
- **repository_id** (RepositoryId) – repository unique identifier

Returns

configuration instance

Return type

Self

gettype(*section: str*, *repository_id:* RepositoryId, *, *fallback: str | None = None*) → tuple[str, str]

get type variable with fallback to old logic. Despite the fact that it has same semantics as other get* methods, but it has different argument list

Parameters

- **section** (*str*) – section name
- **repository_id** (RepositoryId) – repository unique identifier
- **fallback** (*str | None, optional*) – optional fallback type if any. If set, second element of the tuple will be always set to this value (Default value = None)

Returns
section name and found type name

Return type
tuple[str, str]

Raises
configparser.NoSectionError – in case if no section found

load(*path: Path*) → None

fully load configuration

Parameters
path (*Path*) – path to root configuration file

load_includes(*path: Path | None = None*) → None

load configuration includes from specified path

Parameters
path (*Path | None, optional*) – path to directory with include files. If none set, the default path will be used (Default value = None)

merge_sections(*repository_id:* RepositoryId) → None

merge architecture and repository specific sections into main configuration

Parameters
repository_id (RepositoryId) – repository unique identifier

override_sections(*section: str*, *repository_id:* RepositoryId) → list[str]

extract override sections

Parameters

- **section** (*str*) – section name
- **repository_id** (RepositoryId) – repository unique identifier

Returns
architecture and repository specific sections in correct order

Return type
list[str]

reload() → None

reload configuration if possible or raise exception otherwise

static section_name(*section: str*, **suffixes: str | None*) → str

generate section name for sections which depends on context

Parameters

- **section** (*str*) – section name
- ***suffixes** (*str | None*) – session suffix, e.g. repository architecture

Returns
correct section name for repository specific section

Return type
str

set_option(*section: str*, *option: str*, *value: str*) → None

set option. Unlike default `configparser.RawConfigParser.set()` it also creates section if it does not exist

Parameters

- **section** (*str*) – section name
- **option** (*str*) – option name
- **value** (*str*) – option value as string in parsable format

property architecture: str

repository architecture for backward compatibility

Returns
repository architecture

Return type
str

property include: Path

get full path to include directory

Returns
path to directory with configuration includes

Return type
Path

property logging_path: Path

get full path to logging configuration

Returns
path to logging configuration

Return type
Path

property repository_name: str

repository name for backward compatibility

Returns
repository name

Return type
str

property repository_paths: *RepositoryPaths*

construct RepositoryPaths instance based on the configuration

Returns
repository paths instance

Return type
RepositoryPaths

ahriman.core.configuration.schema module

ahriman.core.configuration.shell_interpolator module

class ShellInterpolator

Bases: Interpolation

custom string interpolator, because we cannot use defaults argument due to config validation

before_get(*parser: MutableMapping[str, Mapping[str, str]]*, *section: str*, *option: str*, *value: str*, *defaults: Mapping[str, str]*) → str

interpolate option value

Parameters

- **parser** (*MutableMapping[str, Mapping[str, str]]*) – option parser
- **section** (*str*) – section name
- **option** (*str*) – option name
- **value** (*str*) – source (not-converted) value
- **defaults** (*Mapping[str, str]*) – default values

Returns
substituted value

Return type
str

ahriman.core.configuration.validator module

class Validator(**args: Any*, ***kwargs: Any*)

Bases: Validator

class which defines custom validation methods for the service configuration

configuration

configuration instance

Type
Configuration

default constructor

Parameters

- **configuration** (Configuration) – configuration instance used for extraction
- ***args** (*Any*) – positional arguments to be passed to base validator
- ****kwargs** (*Any*) – keyword arguments to be passed to base validator

types_mapping = {'binary': ('binary', (<class 'bytes'>, <class 'bytearray'>), ()), 'boolean': ('boolean', (<class 'bool'>,), ()), 'container': ('container', (<class 'collections.abc.Container'>,), (<class 'str'>,)), 'date': ('date', (<class 'datetime.date'>,), ()), 'datetime': ('datetime', (<class 'datetime.datetime'>,), ()), 'dict': ('dict', (<class 'collections.abc.Mapping'>,), ()), 'float': ('float', (<class 'float'>, (<class 'int'>,)), ()), 'integer': ('integer', ((<class 'int'>,),), ()), 'list': ('list', (<class 'collections.abc.Sequence'>,), (<class 'str'>,)), 'number': ('number', ((<class 'int'>,), <class 'float'>), (<class 'bool'>,)), 'path': ('path', (<class 'pathlib.Path'>,), ()), 'set': ('set', (<class 'set'>,), ()), 'string': ('string', (<class 'str'>,), ())}

This mapping holds all available constraints for the type rule and their assigned `TypeDefinition`.

Module contents

ahriman.core.database package

Subpackages

ahriman.core.database.migrations package

Submodules

ahriman.core.database.migrations.m000_initial module

migrate_data(*connection: Connection*, *configuration:* Configuration) → None

perform data migration

Parameters

- **connection** (*Connection*) – database connection
- **configuration** (Configuration) – configuration instance

ahriman.core.database.migrations.m001_package_source module

migrate_data(*connection: Connection*, *configuration:* Configuration) → None

perform data migration

Parameters

- **connection** (*Connection*) – database connection
- **configuration** (Configuration) – configuration instance

ahriman.core.database.migrations.m002_user_access module

ahriman.core.database.migrations.m003_patch_variables module

ahriman.core.database.migrations.m004_logs module

ahriman.core.database.migrations.m005_make_opt_depends module

migrate_data(*connection: Connection*, *configuration:* Configuration) → None
perform data migration

Parameters

- **connection** (*Connection*) – database connection
- **configuration** (Configuration) – configuration instance

ahriman.core.database.migrations.m006_packages_architecture_required module

ahriman.core.database.migrations.m007_check_depends module

migrate_data(*connection: Connection*, *configuration:* Configuration) → None
perform data migration

Parameters

- **connection** (*Connection*) – database connection
- **configuration** (Configuration) – configuration instance

ahriman.core.database.migrations.m008_packagers module

migrate_data(*connection: Connection*, *configuration:* Configuration) → None
perform data migration

Parameters

- **connection** (*Connection*) – database connection
- **configuration** (Configuration) – configuration instance

ahriman.core.database.migrations.m009_local_source module

ahriman.core.database.migrations.m010_version_based_logs_removal module

ahriman.core.database.migrations.m011_repository_name module

migrate_data(*connection: Connection*, *configuration:* Configuration) → None
perform data migration

Parameters

- **connection** (*Connection*) – database connection

- **configuration** (Configuration) – configuration instance

ahriman.core.database.migrations.m012_last_commit_sha module

Module contents

class Migrations(*connection: Connection*, *configuration:* Configuration)

Bases: *LazyLogging*

simple migration wrapper for the sqlite idea comes from https://www.ash.dev/blog/simple-migration-system-in-sqlite/

configuration

configuration instance

Type
Configuration

connection

database connection

Type
Connection

default constructor

Parameters

- **connection** (*Connection*) – database connection
- **configuration** (Configuration) – configuration instance

static migrate(*connection: Connection*, *configuration:* Configuration) → *MigrationResult*

perform migrations implicitly

Parameters

- **connection** (*Connection*) – database connection
- **configuration** (Configuration) – configuration instance

Returns
current schema version

Return type
MigrationResult

migration(*cursor: Cursor*, *migration:* Migration) → None

perform single migration

Parameters

- **cursor** (*Cursor*) – connection cursor
- **migration** (Migration) – single migration to perform

migrations() → list[*Migration*]

extract all migrations from the current package idea comes from https://julienharbulot.com/python-dynamical-import.html

Returns
list of found migrations

Return type
list[*Migration*]

run() → *MigrationResult*

perform migrations

Returns
current schema version

Return type
MigrationResult

user_version() → int

get schema version from sqlite database

Returns
current schema version

Return type
int

ahriman.core.database.operations package

Submodules

ahriman.core.database.operations.auth_operations module

class AuthOperations(*path: Path*, *repository_id:* RepositoryId)

Bases: *Operations*

authorization operations

default constructor

Parameters

- **path** (*Path*) – path to the database file
- **repository_id** (RepositoryId) – repository unique identifier

user_get(*username: str*) → *User* | None

get user by username

Parameters
username (*str*) – username

Returns
user if it was found

Return type
User | None

user_list(*username: str* | *None*, *access:* UserAccess | *None*) → list[*User*]

get users by filter

Parameters

- **username** (*str | None*) – optional filter by username
- **access** (UserAccess | *None*) – optional filter by role

Returns
list of users who match criteria

Return type
list[*User*]

user_remove(*username: str*) → None
remove user from storage

Parameters
username (*str*) – username

user_update(*user:* User) → None
update user by username

Parameters
user (User) – user descriptor

ahriman.core.database.operations.build_operations module

class BuildOperations(*path: Path*, *repository_id:* RepositoryId)
Bases: *Operations*

operations for build queue functions

default constructor

Parameters
- **path** (*Path*) – path to the database file
- **repository_id** (RepositoryId) – repository unique identifier

build_queue_clear(*package_base: str | None*, *repository_id:* RepositoryId | *None* = *None*) → None
remove packages from build queue

Parameters
- **package_base** (*str | None*) – optional filter by package base
- **repository_id** (RepositoryId, *optional*) – repository unique identifier override (Default value = None)

build_queue_get(*repository_id:* RepositoryId | *None* = *None*) → list[*Package*]
retrieve packages from build queue

Parameters
repository_id (RepositoryId, *optional*) – repository unique identifier override (Default value = None)

Returns
list of packages to be built

Return type
list[*Package*]

build_queue_insert(*package:* Package, *repository_id:* RepositoryId | *None* = *None*) → None
insert packages to build queue

Parameters
- **package** (Package) – package to be inserted

- **repository_id** (RepositoryId, *optional*) – repository unique identifier override (Default value = None)

ahriman.core.database.operations.changes_operations module

class ChangesOperations(*path: Path*, *repository_id:* RepositoryId)

Bases: *Operations*

operations for source files changes

default constructor

Parameters

- **path** (*Path*) – path to the database file
- **repository_id** (RepositoryId) – repository unique identifier

changes_get(*package_base: str*, *repository_id:* RepositoryId | *None* = *None*) → *Changes*

get changes for the specific package base if available

Parameters

- **package_base** (*str*) – package base to search
- **repository_id** (RepositoryId, *optional*) – repository unique identifier override (Default value = None)

Returns

changes for the package base if available

Return type

Changes

changes_insert(*package_base: str*, *changes:* Changes, *repository_id:* RepositoryId | *None* = *None*) → None

insert packages to build queue

Parameters

- **package_base** (*str*) – package base to insert
- **changes** (Changes) – package changes (as in patch format)
- **repository_id** (RepositoryId, *optional*) – repository unique identifier override (Default value = None)

changes_remove(*package_base: str | None*, *repository_id:* RepositoryId | *None* = *None*) → None

remove packages changes

Parameters

- **package_base** (*str | None*) – optional filter by package base
- **repository_id** (RepositoryId, *optional*) – repository unique identifier override (Default value = None)

hashes_get(*repository_id:* RepositoryId | *None* = *None*) → dict[str, str]

extract last commit hashes if available

Parameters

repository_id (RepositoryId, *optional*) – repository unique identifier override (Default value = None)

Returns
map of package base to its last commit hash

Return type
dict[str, str]

ahriman.core.database.operations.logs_operations module

class LogsOperations(*path: Path*, *repository_id:* RepositoryId)

Bases: *Operations*

logs operations

default constructor

Parameters

- **path** (*Path*) – path to the database file
- **repository_id** (RepositoryId) – repository unique identifier

logs_get(*package_base: str*, *limit: int = -1*, *offset: int = 0*, *repository_id:* RepositoryId | *None = None*) → list[tuple[float, str]]

extract logs for specified package base

Parameters

- **package_base** (*str*) – package base to extract logs
- **limit** (*int, optional*) – limit records to the specified count, -1 means unlimited (Default value = -1)
- **offset** (*int, optional*) – records offset (Default value = 0)
- **repository_id** (RepositoryId, *optional*) – repository unique identifier override (Default value = None)

Returns
sorted package log records and their timestamps

Return type
list[tuple[float, str]]

logs_insert(*log_record_id:* LogRecordId, *created: float*, *record: str*, *repository_id:* RepositoryId | *None = None*) → None

write new log record to database

Parameters

- **log_record_id** (LogRecordId) – current log record id
- **created** (*float*) – log created timestamp from log record attribute
- **record** (*str*) – log record
- **repository_id** (RepositoryId, *optional*) – repository unique identifier override (Default value = None)

logs_remove(*package_base: str*, *version: str | None*, *repository_id:* RepositoryId | *None = None*) → None

remove log records for the specified package

Parameters

- **package_base** (*str*) – package base to remove logs
- **version** (*str | None*) – package version. If set it will remove only logs belonging to another version
- **repository_id** (RepositoryId, *optional*) – repository unique identifier override (Default value = None)

ahriman.core.database.operations.operations module

class Operations(*path: Path*, *repository_id:* RepositoryId)

Bases: *LazyLogging*

base operation class

path

path to the database file

Type
Path

default constructor

Parameters

- **path** (*Path*) – path to the database file
- **repository_id** (RepositoryId) – repository unique identifier

static factory(*cursor: Cursor*, *row: tuple[Any, ...]*) → dict[str, Any]

dictionary factory based on official documentation

Parameters

- **cursor** (*Cursor*) – cursor descriptor
- **row** (*tuple[Any, ...]*) – fetched row

Returns
row converted to dictionary

Return type
dict[str, Any]

with_connection(*query: Callable[[Connection], T]*, *, *commit: bool = False*) → T

perform operation in connection

Parameters

- **query** (*Callable[[Connection], T]*) – function to be called with connection
- **commit** (*bool, optional*) – if True commit() will be called on success (Default value = False)

Returns
result of the `query` call

Return type
T

ahriman.core.database.operations.package_operations module

class PackageOperations(*path: Path*, *repository_id:* RepositoryId)

Bases: *Operations*

package operations

default constructor

Parameters

- **path** (*Path*) – path to the database file
- **repository_id** (RepositoryId) – repository unique identifier

package_base_update(*package:* Package, *repository_id:* RepositoryId | *None* = *None*) → None

update package base only

Parameters

- **package** (Package) – package properties
- **repository_id** (RepositoryId, *optional*) – repository unique identifier override (Default value = None)

package_remove(*package_base: str*, *repository_id:* RepositoryId | *None* = *None*) → None

remove package from database

Parameters

- **package_base** (*str*) – package base name
- **repository_id** (RepositoryId, *optional*) – repository unique identifier override (Default value = None)

package_update(*package:* Package, *status:* BuildStatus, *repository_id:* RepositoryId | *None* = *None*) → None

update package status

Parameters

- **package** (Package) – package properties
- **status** (BuildStatus) – new build status
- **repository_id** (RepositoryId, *optional*) – repository unique identifier override (Default value = None)

packages_get(*repository_id:* RepositoryId | *None* = *None*) → list[tuple[*Package*, *BuildStatus*]]

get package list and their build statuses from database

Parameters

repository_id (RepositoryId, *optional*) – repository unique identifier override (Default value = None)

Returns

list of package properties and their statuses

Return type

list[tuple[*Package*, *BuildStatus*]]

remotes_get(*repository_id:* RepositoryId | *None* = *None*) → dict[str, *RemoteSource*]

get packages remotes based on current settings

Parameters

repository_id (*RepositoryId, optional*) – repository unique identifier override (Default value = None)

Returns

map of package base to its remote sources

Return type

dict[str, *RemoteSource*]

ahriman.core.database.operations.patch_operations module

class PatchOperations(*path: Path*, *repository_id:* RepositoryId)

Bases: *Operations*

operations for patches

default constructor

Parameters

- **path** (*Path*) – path to the database file
- **repository_id** (RepositoryId) – repository unique identifier

patches_get(*package_base: str*) → list[*PkgbuildPatch*]

retrieve patches for the package

Parameters

package_base (*str*) – package base to search for patches

Returns

plain text patch for the package

Return type

list[*PkgbuildPatch*]

patches_insert(*package_base: str*, *patches: list[*PkgbuildPatch*]*) → None

insert or update patch in database

Parameters

- **package_base** (*str*) – package base to insert
- **patches** (*list[*PkgbuildPatch*]*) – patch content

patches_list(*package_base: str | None*, *variables: list[str] | None*) → dict[str, list[*PkgbuildPatch*]]

extract all patches

Parameters

- **package_base** (*str | None*) – optional filter by package base
- **variables** (*list[str] | None*) – extract patches only for specified PKGBUILD variables

Returns

map of package base to patch content

Return type
dict[str, list[*PkgbuildPatch*]]

patches_remove(*package_base: str*, *variables: list[str] | None*) → None

remove patch set

Parameters

- **package_base** (*str*) – package base to clear patches
- **variables** (*list[str] | None*) – remove patches only for specified PKGBUILD variables

Module contents

Submodules

ahriman.core.database.sqlite module

class SQLite(*path: Path*, *repository_id:* RepositoryId)

Bases: *AuthOperations*, *BuildOperations*, *ChangesOperations*, *LogsOperations*, *PackageOperations*, *PatchOperations*

wrapper for sqlite3 database

Examples

Database wrapper must be used together with migration and SQLite3 setup. In order to perform it there is *load()* class method:

```
>>> from ahriman.core.configuration import Configuration
>>>
>>> configuration = Configuration()
>>> database = SQLite.load(configuration)
>>> packages = database.packages_get()
```

default constructor

Parameters

- **path** (*Path*) – path to the database file
- **repository_id** (RepositoryId) – repository unique identifier

static database_path(*configuration:* Configuration) → Path

read database from configuration

Parameters
configuration (Configuration) – configuration instance

Returns
database path according to the configuration

Return type
Path

init(*configuration:* Configuration) → None

perform database migrations

Parameters
configuration (Configuration) – configuration instance

classmethod load(*configuration:* Configuration) → Self

construct instance from configuration

Parameters
configuration (Configuration) – configuration instance

Returns
fully initialized instance of the database

Return type
Self

Module contents

ahriman.core.distributed package

Submodules

ahriman.core.distributed.distributed_system module

class DistributedSystem(*repository_id:* RepositoryId, *configuration:* Configuration)

Bases: *Trigger*, *WebClient*

simple class to (un)register itself as a distributed worker

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance

classmethod configuration_sections(*configuration:* Configuration) → list[str]

extract configuration sections from configuration

Parameters
configuration (Configuration) – configuration instance

Returns
read configuration sections belong to this trigger

Return type
list[str]

register() → None

register itself in remote system

workers() → list[*Worker*]

retrieve list of available remote workers

Returns
currently registered workers

Return type
list[*Worker*]

property worker: *Worker*

load and set worker. Lazy property loaded because it is not always required

Returns
unique self worker identifier

Return type
Worker

ahriman.core.distributed.worker_loader_trigger module

class WorkerLoaderTrigger(*repository_id:* RepositoryId, *configuration:* Configuration)

Bases: *DistributedSystem*

remote worker processor trigger (server side)

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance

on_start() → None

trigger action which will be called at the start of the application

ahriman.core.distributed.worker_trigger module

class WorkerTrigger(*repository_id:* RepositoryId, *configuration:* Configuration)

Bases: *DistributedSystem*

remote worker processor trigger (client side)

ping_interval

interval to call remote service in seconds, defined as `worker.time_to_live / 4`

Type
float

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance

create_timer() → None

create timer object and put it to queue

on_start() → None

trigger action which will be called at the start of the application

on_stop() → None

trigger action which will be called before the stop of the application

ping() → None
register itself as alive worker and update the timer

ahriman.core.distributed.workers_cache module

class WorkersCache(*configuration:* Configuration)
Bases: *LazyLogging*

cached storage for healthy workers

time_to_live
maximal amount of time in seconds to keep worker alive

Type
int

default constructor

Parameters
configuration (Configuration) – configuration instance

workers_remove() → None
remove all workers from the cache

workers_update(*worker:* Worker) → None
register or update remote worker

Parameters
worker (Worker) – worker to register

property workers: list[*Worker*]
extract currently healthy workers

Returns
list of currently registered workers which have been seen not earlier than *time_to_live*

Return type
list[*Worker*]

Module contents

ahriman.core.formatters package

Submodules

ahriman.core.formatters.aur_printer module

class AurPrinter(*package:* AURPackage)
Bases: *StringPrinter*

print content of the AUR package

package
AUR package description

Type
AURPackage

default constructor

Parameters
package (AURPackage) – AUR package description

properties() → list[*Property*]

convert content into printable data

Returns
list of content properties

Return type
list[*Property*]

ahriman.core.formatters.build_printer module

class BuildPrinter(*package: Package*, *is_success: bool*, *use_utf: bool*)

Bases: *StringPrinter*

print content of the build result

default constructor

Parameters

- **package** (Package) – built package
- **is_success** (*bool*) – True in case if build has success status and False otherwise
- **use_utf** (*bool*) – use utf instead of normal symbols

static sign(*is_success: bool*, *use_utf: bool*) → str

generate sign according to settings

Parameters

- **is_success** (*bool*) – True in case if build has success status and False otherwise
- **use_utf** (*bool*) – use utf instead of normal symbols

Returns
sign symbol according to current settings

Return type
str

ahriman.core.formatters.changes_printer module

class ChangesPrinter(*changes: Changes*)

Bases: *Printer*

print content of the changes object

changes

package changes

Type
Changes

default constructor

Parameters
changes (Changes) – package changes

properties() → list[*Property*]

convert content into printable data

Returns
list of content properties

Return type
list[*Property*]

title() → str | None

generate entry title from content

Returns
content title if it can be generated and None otherwise

Return type
str | None

ahriman.core.formatters.configuration_paths_printer module

class ConfigurationPathsPrinter(*root: Path*, *includes: list[Path]*)

Bases: *StringPrinter*

print configuration paths

includes

list of include files

Type
list[Path]

default constructor

Parameters

- **root** (*Path*) – path to root configuration file
- **includes** (*list[Path]*) – list of include files

properties() → list[*Property*]

convert content into printable data

Returns
list of content properties

Return type
list[*Property*]

ahriman.core.formatters.configuration_printer module

class ConfigurationPrinter(*section: str*, *values: dict[str, str]*)

Bases: *StringPrinter*

print content of the configuration section

HIDE_KEYS

(class attribute) hide values for mentioned keys. This list must be used in order to hide passwords from outputs

Type
list[str]

values

configuration values dictionary

Type
dict[str, str]

default constructor

Parameters

- **section** (*str*) – section name
- **values** (*dict[str, str]*) – configuration values dictionary

properties() → list[*Property*]

convert content into printable data

Returns
list of content properties

Return type
list[*Property*]

ahriman.core.formatters.package_printer module

class PackagePrinter(*package:* Package, *status:* BuildStatus)

Bases: *StringPrinter*

print content of the internal package object

package

package description

Type
Package

status

build status

Type
BuildStatus

default constructor

Parameters

- **package** (Package) – package description

- **status** (BuildStatus) – build status

properties() → list[*Property*]

convert content into printable data

Returns
list of content properties

Return type
list[*Property*]

ahriman.core.formatters.patch_printer module

class PatchPrinter(*package_base: str*, *patches: list[PkgbuildPatch]*)

Bases: *StringPrinter*

print content of the PKGBUILD patch

patches

PKGBUILD patch object

Type
list[*PkgbuildPatch*]

default constructor

Parameters

- **package_base** (*str*) – package base
- **patches** (*list[PkgbuildPatch]*) – PKGBUILD patch object

properties() → list[*Property*]

convert content into printable data

Returns
list of content properties

Return type
list[*Property*]

ahriman.core.formatters.printer module

class Printer

Bases: object

base class for formatters

print(**, verbose: bool, log_fn: ~collections.abc.Callable[[str], None] = <built-in function print>, separator: str = ': '*) → None

print content

Parameters

- **verbose** (*bool*) – print all fields
- **log_fn** (*Callable[[str], None], optional*) – logger function to log data (Default value = print)

- **separator** (*str, optional*) – separator for property name and property value (Default value = ": ")

properties() → list[*Property*]

convert content into printable data

Returns
list of content properties

Return type
list[*Property*]

title() → str | None

generate entry title from content

Returns
content title if it can be generated and None otherwise

Return type
str | None

ahriman.core.formatters.repository_printer module

class RepositoryPrinter(*repository_id:* RepositoryId)

Bases: *StringPrinter*

print repository unique identifier

repository_id

repository unique identifier

Type
RepositoryId

default constructor

Parameters
repository_id (RepositoryId) – repository unique identifier

properties() → list[*Property*]

convert content into printable data

Returns
list of content properties

Return type
list[*Property*]

ahriman.core.formatters.status_printer module

class StatusPrinter(*status:* BuildStatus)

Bases: *StringPrinter*

print content of the status object

default constructor

Parameters
status (BuildStatus) – build status

ahriman.core.formatters.string_printer module

class StringPrinter(*content: str*)

Bases: *Printer*

print content of the random string

content

any content string

Type
str

default constructor

Parameters
content (*str*) – any content string

title() → str | None

generate entry title from content

Returns
content title if it can be generated and None otherwise

Return type
str | None

ahriman.core.formatters.tree_printer module

class TreePrinter(*level: int*, *packages: list[*Package*]*)

Bases: *StringPrinter*

print content of the package tree level

packages

packages which belong to this level

Type
list[*Package*]

default constructor

Parameters

- **level** (*int*) – dependencies tree level
- **packages** (*list[*Package*]*) – packages which belong to this level

properties() → list[*Property*]

convert content into printable data

Returns
list of content properties

Return type
list[*Property*]

ahriman.core.formatters.update_printer module

class UpdatePrinter(*remote:* Package, *local_version: str | None*)

Bases: *StringPrinter*

print content of the package update

package

remote (new) package object

Type
Package

local_version

local version of the package if any

Type
str | None

default constructor

Parameters

- **remote** (Package) – remote (new) package object
- **local_version** (*str | None*) – local version of the package if any

properties() → list[*Property*]

convert content into printable data

Returns
list of content properties

Return type
list[*Property*]

ahriman.core.formatters.user_printer module

class UserPrinter(*user:* User)

Bases: *StringPrinter*

print properties of user

user

stored user

Type
User

default constructor

Parameters
user (User) – user to print

properties() → list[*Property*]

convert content into printable data

Returns
list of content properties

Return type
list[*Property*]

ahriman.core.formatters.validation_printer module

class ValidationPrinter(*node: str*, *errors: list[str | dict[str, Any]]*)

Bases: *StringPrinter*

print content of the validation errors

node

root level name

Type
str

errors

validation errors

Type
list[str | dict[str, Any]]

default constructor

Parameters

- **node** (*str*) – root level name
- **errors** (*list[str | dict[str, Any]]*) – validation errors

static get_error_messages(*node: str*, *errors: list[str | dict[str, Any]]*, *current_level: int = 1*) → Generator[*Property*, None, None]

extract default error message from cerberus class

Parameters

- **node** (*str*) – current node level name
- **errors** (*list[str | dict[str, Any]]*) – current node validation errors
- **current_level** (*int, optional*) – current level number (Default value = 1)

Yields
Property – error messages from error tree

properties() → list[*Property*]

convert content into printable data

Returns
list of content properties

Return type
list[*Property*]

ahriman.core.formatters.version_printer module

class VersionPrinter(*title: str*, *packages: dict[str, str]*)

Bases: *StringPrinter*

print content of the python package versions

packages

map of package name to its version

Type
dict[str, str]

default constructor

Parameters

- **title** (*str*) – title of the message
- **packages** (*dict[str, str]*) – map of package name to its version

properties() → list[*Property*]

convert content into printable data

Returns
list of content properties

Return type
list[*Property*]

Module contents

ahriman.core.gitremote package

Submodules

ahriman.core.gitremote.remote_pull module

class RemotePull(*repository_id: RepositoryId*, *configuration: Configuration*, *section: str*)

Bases: *LazyLogging*

fetch PKGBUILDs from remote repository and use them for following actions

architecture

repository architecture

Type
str

remote_source

repository remote source (remote pull url and branch)

Type
RemoteSource

repository_paths

repository paths instance

Type
RepositoryPaths

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **section** (*str*) – settings section name

package_copy(*pkgbuild_path: Path*) → None

copy single PKGBUILD content to the repository tree

Parameters
pkgbuild_path (*Path*) – path to PKGBUILD to copy

repo_clone() → None

clone repository from remote source

repo_copy(*clone_dir: Path*) → None

copy directories from cloned remote source to local cache

Parameters
clone_dir (*Path*) – path to temporary cloned directory

run() → None

run git pull action

Raises
GitRemoteError – pull processing error

ahriman.core.gitremote.remote_pull_trigger module

class RemotePullTrigger(*repository_id:* RepositoryId, *configuration:* Configuration)

Bases: *Trigger*

trigger based on pulling PKGBUILDs before the actions

targets

git remote target list

Type
list[str]

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance

classmethod configuration_sections(*configuration:* Configuration) → list[str]

extract configuration sections from configuration

Parameters
configuration (Configuration) – configuration instance

Returns
read configuration sections belong to this trigger

Return type
list[str]

on_start() → None
trigger action which will be called at the start of the application

ahriman.core.gitremote.remote_push module

class RemotePush(*database: SQLite*, *configuration: Configuration*, *section: str*)
Bases: *LazyLogging*

sync PKGBUILDs to remote repository after actions

commit_author
optional commit author in form of git config

Type
tuple[str, str] | None

database
database instance

Type
SQLite

remote_source
repository remote source (remote pull url and branch)

Type
RemoteSource

default constructor

Parameters
- **database** (SQLite) – database instance
- **configuration** (Configuration) – configuration instance
- **section** (*str*) – settings section name

package_update(*package: Package*, *target_dir: Path*) → str
clone specified package and update its content in cloned PKGBUILD repository

Parameters
- **package** (Package) – built package to update pkgbuild repository
- **target_dir** (*Path*) – path to the cloned PKGBUILD repository

Returns
relative path to be added as new file

Return type
str

packages_update(*result: Result*, *target_dir: Path*) → Generator[str, None, None]

update all packages from the build result

Parameters

- **result** (Result) – build result
- **target_dir** (*Path*) – path to the cloned PKGBUILD repository

Yields

str – path to updated files

run(*result: Result*) → None

run git pull action

Parameters

result (Result) – build result

Raises

GitRemoteError – push processing error

ahriman.core.gitremote.remote_push_trigger module

class RemotePushTrigger(*repository_id: RepositoryId*, *configuration: Configuration*)

Bases: *Trigger*

trigger for syncing PKGBUILDs to remote repository

targets

git remote target list

Type

list[str]

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance

classmethod configuration_sections(*configuration: Configuration*) → list[str]

extract configuration sections from configuration

Parameters

configuration (Configuration) – configuration instance

Returns

read configuration sections belong to this trigger

Return type

list[str]

on_result(*result: Result*, *packages: list[Package]*) → None

trigger action which will be called after build process with process result

Parameters

- **result** (Result) – build result
- **packages** (*list[Package]*) – list of all available packages

Raises
GitRemoteError – if database is not set in context

Module contents

ahriman.core.http package

Submodules

ahriman.core.http.sync_ahriman_client module

class SyncAhrimanClient(*configuration: Configuration | None = None*, *section: str | None = None*, *, suppress_errors: bool = False*)

Bases: *SyncHttpClient*

wrapper for ahriman web service

address
address of the web service

Type
str

default constructor

Parameters

- **configuration** (*Configuration | None, optional*) – configuration instance (Default value = None)
- **section** (*str | None, optional*) – settings section name (Default value = None)
- **suppress_errors** (*bool, optional*) – suppress logging of request errors (Default value = False)

property session: Session
get or create session

Returns
created session object

Return type
request.Session

ahriman.core.http.sync_http_client module

class SyncHttpClient(*configuration: Configuration | None = None*, *section: str | None = None*, *, suppress_errors: bool = False*)

Bases: *LazyLogging*

wrapper around requests library to reduce boilerplate

auth
HTTP basic auth object if set

Type
tuple[str, str] | None

suppress_errors

suppress logging of request errors

Type
bool

timeout

HTTP request timeout in seconds

Type
int

default constructor

Parameters

- **configuration** (*Configuration | None, optional*) – configuration instance (Default value = None)
- **section** (*str | None, optional*) – settings section name (Default value = None)
- **suppress_errors** (*bool, optional*) – suppress logging of request errors (Default value = False)

static exception_response_text(*exception: RequestException*) → str

safe response exception text generation

Parameters
exception (*requests.RequestException*) – exception raised

Returns
text of the response if it is not None and empty string otherwise

Return type
str

make_request(*method: Literal['DELETE', 'GET', 'POST', 'PUT']*, *url: str*, ***, *headers: dict[str, str] | None = None*, *params: list[tuple[str, str]] | None = None*, *data: Any | None = None*, *json: dict[str, Any] | None = None*, *files: dict[str, tuple[str, IO[bytes], str, dict[str, str]]] | None = None*, *session: Session | None = None*, *suppress_errors: bool | None = None*) → Response

perform request with specified parameters

Parameters

- **method** (*Literal["DELETE", "GET", "POST", "PUT"]*) – HTTP method to call
- **url** (*str*) – remote url to call
- **headers** (*dict[str, str] | None, optional*) – request headers (Default value = None)
- **params** (*list[tuple[str, str]] | None, optional*) – request query parameters (Default value = None)
- **data** (*Any | None, optional*) – request raw data parameters (Default value = None)
- **json** (*dict[str, Any] | None, optional*) – request json parameters (Default value = None)
- **files** (*dict[str, MultipartType] | None, optional*) – multipart upload (Default value = None)
- **session** (*requests.Session | None, optional*) – session object if any (Default value = None)

- **suppress_errors** (*bool | None, optional*) – suppress logging errors (e.g. if no web server available). If none set, the instance-wide value will be used (Default value = None)

Returns
response object

Return type
requests.Response

property session: Session

get or create session

Returns
created session object

Return type
request.Session

Module contents

ahriman.core.log package

Submodules

ahriman.core.log.http_log_handler module

class HttpLogHandler(*repository_id: RepositoryId*, *configuration: Configuration*, ***, *report: bool*, *suppress_errors: bool*)

Bases: Handler

handler for the http logging. Because default logging.handlers.HTTPHandler does not support cookies authorization, we have to implement own handler which overrides the logging.handlers.HTTPHandler.emit() method

reporter

build status reporter instance

Type
Client

suppress_errors

suppress logging errors (e.g. if no web server available)

Type
bool

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting
- **suppress_errors** (*bool*) – suppress logging errors (e.g. if no web server available)

emit(*record: LogRecord*) → None

emit log records using reporter client

Parameters
record (*logging.LogRecord*) – log record to log

classmethod load(*repository_id: RepositoryId*, *configuration: Configuration*, *, *report: bool*) → Self

install logger. This function creates handler instance and adds it to the handler list in case if no other http handler found

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **report** (*bool*) – force enable or disable reporting

Returns
logger instance with loaded settings

Return type
Self

ahriman.core.log.journal_handler module

JournalHandler

alias of _JournalHandler

ahriman.core.log.lazy_logging module

class LazyLogging

Bases: object

wrapper for the logger library inspired by scala lazy logging module

in_package_context(*package_base: str*, *version: str | None*) → Generator[None, None, None]

execute function while setting package context

Parameters

- **package_base** (*str*) – package base to set context in
- **version** (*str | None*) – package version if available

Examples

This function is designed to be called as context manager with package_base argument, e.g.:

```
>>> with self.in_package_context(package.base, package.version):
>>>     build_package(package)
```

property logger: Logger

get class logger instance

Returns
class logger instance

Return type
logging.Logger

property logger_name: str

extract logger name for the class

Returns
logger name as combination of module name and class name

Return type
str

ahriman.core.log.log_loader module

class LogLoader

Bases: object

simple static method class which setups application loggers

DEFAULT_LOG_FORMAT

(class attribute) default log format (in case of fallback)

Type
str

DEFAULT_LOG_LEVEL

(class attribute) default log level (in case of fallback)

Type
int

DEFAULT_SYSLOG_DEVICE

(class attribute) default path to syslog device

Type
Path

static handler(*selected:* LogHandler | *None*) → *LogHandler*

try to guess default log handler. In case if `selected` is set, it will return specified value with appended _handler suffix. Otherwise, it will try to import journald handler and returns *ahriman.models.log_handler.LogHandler.Journald* if library is available. Otherwise, it will check if there is `/dev/log` device and returns *ahriman.models.log_handler.LogHandler.Syslog* in this case. And, finally, it will fall back to *ahriman.models.log_handler.LogHandler.Console* if none were found

Parameters
selected (LogHandler | *None*) – user specified handler if any

Returns
selected log handler

Return type
LogHandler

static load(*repository_id:* RepositoryId, *configuration:* Configuration, *handler:* LogHandler, *, *quiet:* *bool*, *report:* *bool*) → None

setup logging settings from configuration

Parameters

- **repository_id** (RepositoryId) – repository unique identifier

- **configuration** (Configuration) – configuration instance
- **handler** (LogHandler) – selected default log handler, which will be used if no handlers were set
- **quiet** (*bool*) – force disable any log messages
- **report** (*bool*) – force enable or disable reporting

Module contents

ahriman.core.report package

Submodules

ahriman.core.report.console module

class Console(*repository_id:* RepositoryId, *configuration:* Configuration, *section: str*)

Bases: *Report*

html report generator

use_utf

print utf8 symbols instead of ASCII

Type
bool

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **section** (*str*) – settings section name

generate(*packages: list[*Package*]*, *result:* Result) → None

generate report for the specified packages

Parameters

- **packages** (*list[*Package*]*) – list of packages to generate report
- **result** (Result) – build result

ahriman.core.report.email module

class Email(*repository_id:* RepositoryId, *configuration:* Configuration, *section: str*)

Bases: *Report*, *JinjaTemplate*

email report generator

host

SMTP host to connect

Type
str

no_empty_report
skip empty report generation

Type
bool

password
password to authenticate via SMTP

Type
str | None

port
SMTP port to connect

Type
int

receivers
list of receivers emails

Type
list[str]

sender
sender email address

Type
str

ssl
SSL mode for SMTP connection

Type
SmtpSSLSettings

template
path or name to template for built packages

Type
Path | str

template_full
path or name to template for full package list

Type
Path | str | None

user
username to authenticate via SMTP

Type
str | None

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **section** (*str*) – settings section name

generate(*packages: list[Package]*, *result: Result*) → None

generate report for the specified packages

Parameters

- **packages** (*list[Package]*) – list of packages to generate report
- **result** (*Result*) – build result

ahriman.core.report.html module

class HTML(*repository_id: RepositoryId*, *configuration: Configuration*, *section: str*)

Bases: *Report*, *JinjaTemplate*

html report generator

report_path

output path to html report

Type
Path

template

name or path to template for full package list

Type
Path | str

default constructor

Parameters

- **repository_id** (*RepositoryId*) – repository unique identifier
- **configuration** (*Configuration*) – configuration instance
- **section** (*str*) – settings section name

generate(*packages: list[Package]*, *result: Result*) → None

generate report for the specified packages

Parameters

- **packages** (*list[Package]*) – list of packages to generate report
- **result** (*Result*) – build result

ahriman.core.report.jinja_template module

class JinjaTemplate(*repository_id: RepositoryId*, *configuration: Configuration*, *section: str*)

Bases: object

jinja based report generator

It uses jinja2 templates for report generation, the following variables are allowed:

- homepage - link to homepage, string, optional
- link_path - prefix fo packages to download, string, required
- has_package_signed - True in case if package sign enabled, False otherwise, required

- has_repo_signed - True in case if repository database sign enabled, False otherwise, required
- **packages - sorted list of packages properties, required**
  - architecture, string
  - archive_size, pretty printed size, string
  - build_date, pretty printed datetime, string
  - depends, sorted list of strings
  - description, string
  - filename, string,
  - groups, sorted list of strings
  - installed_size, pretty printed datetime, string
  - licenses, sorted list of strings
  - name, string
  - url, string
  - version, string
- pgp_key - default PGP key ID, string, optional
- repository - repository name, string, required

`default_pgp_key`
: default PGP key

  Type
  : str | None

`homepage`
: homepage link if any (for footer)

  Type
  : str | None

`link_path`
: prefix fo packages to download

  Type
  : str

`name`
: repository name

  Type
  : str

`sign_targets`
: targets to sign enabled in configuration

  Type
  : set[*SignSettings*]

`templates`
: list of directories with templates

Type
list[Path]

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **section** (*str*) – settings section name

make_html(*result: Result*, *template_name: Path | str*) → str

generate report for the specified packages

Parameters

- **result** (Result) – build result
- **template_name** (*Path | str*) – name of the template or path to it (legacy configuration)

ahriman.core.report.remote_call module

class RemoteCall(*repository_id: RepositoryId*, *configuration: Configuration*, *section: str*)

Bases: *Report*

trigger implementation which call remote service with update

client

web client instance

Type
WebClient

update_aur

check for AUR updates

Type
bool

update_local

check for local packages update

Type
bool

update_manual

check for manually built packages

Type
bool

wait_timeout

timeout to wait external process

Type
int

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **section** (*str*) – settings section name

generate(*packages: list[Package]*, *result:* Result) → None

generate report for the specified packages

Parameters

- **packages** (*list[Package]*) – list of packages to generate report
- **result** (Result) – build result

is_process_alive(*process_id: str*) → bool

check if process is alive

Parameters
process_id (*str*) – remote process id

Returns
True in case if remote process is alive and False otherwise

Return type
bool

remote_update() → str

call remote server for update

Returns
remote process id

Return type
str

remote_wait(*process_id: str*) → None

wait for remote process termination

Parameters
process_id (*str*) – remote process id

ahriman.core.report.report module

class Report(*repository_id:* RepositoryId, *configuration:* Configuration)

Bases: *LazyLogging*

base report generator

configuration

configuration instance

Type
Configuration

repository_id

repository unique identifier

Type
RepositoryId

Examples

Report subclasses provide several method in order to operate with the report generation and additional class method *load()* which can be used in order to determine right report instance:

```
>>> configuration = Configuration()
>>> report = Report.load(RepositoryId("x86_64", "aur-clone"), configuration, "email
↪")
```

The *generate()* method can be used in order to perform the report itself, whereas *run()* method handles exception and raises *ahriman.core.exceptions.ReportError* instead:

```
>>> try:
>>>     report.generate([], Result())
>>> except Exception as exception:
>>>     handle_exceptions(exception)
>>>
>>> report.run(Result(), [])
```

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance

generate(*packages: list[*Package*]*, *result:* Result) → None

generate report for the specified packages

Parameters

- **packages** (*list[*Package*]*) – list of packages to generate report
- **result** (Result) – build result

static load(*repository_id:* RepositoryId, *configuration:* Configuration, *target: str*) → *Report*

load client from settings

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **target** (*str*) – target to generate report aka section name (e.g. html)

Returns
client according to current settings

Return type
Report

run(*result:* Result, *packages: list[*Package*]*) → None

run report generation

Parameters

- **result** (Result) – build result
- **packages** (*list[*Package*]*) – list of packages to generate report

Raises
ReportError – in case of any report unmatched exception

ahriman.core.report.report_trigger module

class ReportTrigger(*repository_id:* RepositoryId, *configuration:* Configuration)

Bases: *Trigger*

report trigger

targets

report target list

Type
list[str]

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance

classmethod configuration_sections(*configuration:* Configuration) → list[str]

extract configuration sections from configuration

Parameters
configuration (Configuration) – configuration instance

Returns
read configuration sections belong to this trigger

Return type
list[str]

on_result(*result:* Result, *packages: list[*Package*]*) → None

run trigger

Parameters

- **result** (Result) – build result
- **packages** (*list[*Package*]*) – list of all available packages

ahriman.core.report.telegram module

class Telegram(*repository_id:* RepositoryId, *configuration:* Configuration, *section: str*)

Bases: *Report*, *JinjaTemplate*, *SyncHttpClient*

telegram report generator

TELEGRAM_API_URL

(class attribute) telegram api base url

Type
str

TELEGRAM_MAX_CONTENT_LENGTH

(class attribute) max content length of the message

Type
int

api_key

bot api key

Type
str

chat_id

chat id to post message, either string with @ or integer

Type
str

template

name or path to template for built packages

Type
Path | str

template_type

template message type to be used in parse mode, one of MarkdownV2, HTML, Markdown

Type
str

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **section** (*str*) – settings section name

generate(*packages: list[*Package*]*, *result:* Result) → None

generate report for the specified packages

Parameters

- **packages** (*list[*Package*]*) – list of packages to generate report
- **result** (Result) – build result

Raises
ValueError – impossible to split message by chunks

Module contents

ahriman.core.repository package

Submodules

ahriman.core.repository.cleaner module

class Cleaner(*repository_id:* RepositoryId, *configuration:* Configuration, *database:* SQLite, *, *report: bool*, *refresh_pacman_database:* PacmanSynchronization)

Bases: *RepositoryProperties*

trait to clean common repository objects

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **database** (SQLite) – database instance
- **report** (*bool*) – force enable or disable reporting
- **refresh_pacman_database** (PacmanSynchronization) – pacman database synchronization level

clear_cache() → None

clear cache directory

clear_chroot() → None

clear cache directory. Warning: this method is architecture independent and will clear every chroot

clear_packages() → None

clear directory with built packages (NOT repository itself)

clear_pacman() → None

clear directory with pacman databases

clear_queue() → None

clear packages which were queued for the update

packages_built() → list[Path]

get list of files in built packages directory

Returns
list of filenames from the directory

Return type
list[Path]

Raises
NotImplementedError – not implemented method

ahriman.core.repository.executor module

class **Executor**(*repository_id:* RepositoryId, *configuration:* Configuration, *database:* SQLite, *, *report: bool*, *refresh_pacman_database:* PacmanSynchronization)

Bases: *PackageInfo*, *Cleaner*

trait for common repository update processes

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **database** (SQLite) – database instance
- **report** (*bool*) – force enable or disable reporting
- **refresh_pacman_database** (PacmanSynchronization) – pacman database synchronization level

process_build(*updates: Iterable[*Package*]*, *packagers:* Packagers | *None = None*, *, *bump_pkgrel: bool = False*) → *Result*

build packages

Parameters

- **updates** (*Iterable[*Package*]*) – list of packages properties to build
- **packagers** (Packagers | *None, optional*) – optional override of username for build process (Default value = None)
- **bump_pkgrel** (*bool, optional*) – bump pkgrel in case of local version conflict (Default value = False)

Returns
build result

Return type
Result

process_remove(*packages: Iterable[str]*) → *Result*

remove packages from list

Parameters
packages (*Iterable[str]*) – list of package names or bases to remove

Returns
remove result

Return type
Result

process_update(*packages: Iterable[Path]*, *packagers:* Packagers | *None = None*) → *Result*

sign packages, add them to repository and update repository database

Parameters

- **packages** (*Iterable[Path]*) – list of filenames to run
- **packagers** (Packagers | *None, optional*) – optional override of username for build process (Default value = None)

Returns
path to repository database

Return type
Result

ahriman.core.repository.package_info module

class PackageInfo(*repository_id:* RepositoryId, *configuration:* Configuration, *database:* SQLite, *, *report:* *bool*, *refresh_pacman_database:* PacmanSynchronization)

Bases: *RepositoryProperties*

handler for the package information

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **database** (SQLite) – database instance
- **report** (*bool*) – force enable or disable reporting
- **refresh_pacman_database** (PacmanSynchronization) – pacman database synchronization level

load_archives(*packages: Iterable[Path]*) → list[*Package*]

load packages from list of archives

Parameters
packages (*Iterable[Path]*) – paths to package archives

Returns
list of read packages

Return type
list[*Package*]

package_changes(*package:* Package, *last_commit_sha: str | None*) → *Changes*

extract package change for the package since last commit if available

Parameters

- **package** (Package) – package properties
- **last_commit_sha** (*str | None*) – last known commit hash

Returns
changes if available

Return type
Changes

packages() → list[*Package*]

generate list of repository packages

Returns
list of packages properties

Return type
list[*Package*]

packages_built() → list[Path]

get list of files in built packages directory

Returns
list of filenames from the directory

Return type
list[Path]

packages_depend_on(*packages: list[Package]*, *depends_on: Iterable[str] | None*) → list[*Package*]

extract list of packages which depends on specified package

Parameters

- **packages** (*list[Package]*) – list of packages to be filtered
- **depends_on** (*Iterable[str] | None*) – dependencies of the packages

Returns
list of repository packages which depend on specified packages

Return type
list[*Package*]

ahriman.core.repository.repository module

class Repository(*repository_id: RepositoryId*, *configuration: Configuration*, *database: SQLite*, ***, *report: bool*, *refresh_pacman_database: PacmanSynchronization*)

Bases: *Executor*, *UpdateHandler*

base repository control class

Examples

This class along with traits provides access to local repository actions, e.g. remove packages, update packages, sync local repository to remote, generate report, etc.:

```
>>> from ahriman.core.configuration import Configuration
>>> from ahriman.core.database import SQLite
>>>
>>> configuration = Configuration()
>>> database = SQLite.load(configuration)
>>> repository = Repository.load(RepositoryId("x86_64", "x86_64"), configuration,␣
↪database, report=True)
>>> known_packages = repository.packages()
>>>
>>> build_result = repository.process_build(known_packages)
>>> built_packages = repository.packages_built()
>>> update_result = repository.process_update(built_packages)
>>>
>>> repository.triggers.on_result(update_result, repository.packages())
```

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **database** (SQLite) – database instance
- **report** (*bool*) – force enable or disable reporting
- **refresh_pacman_database** (PacmanSynchronization) – pacman database synchronization level

classmethod load(*repository_id:* RepositoryId, *configuration:* Configuration, *database:* SQLite, *, *report: bool*, *refresh_pacman_database:* PacmanSynchronization = *PacmanSynchronization.Disabled*) → Self

load instance from argument list

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **database** (SQLite) – database instance
- **report** (*bool*) – force enable or disable reporting
- **refresh_pacman_database** (PacmanSynchronization, *optional*) – pacman database synchronization level (Default value = PacmanSynchronization.Disabled)

Returns
fully loaded repository class instance

Return type
Self

ahriman.core.repository.repository_properties module

class RepositoryProperties(*repository_id:* RepositoryId, *configuration:* Configuration, *database:* SQLite, *, *report: bool*, *refresh_pacman_database:* PacmanSynchronization)

Bases: *LazyLogging*

repository internal objects holder

configuration
configuration instance

Type
Configuration

database
database instance

Type
SQLite

ignore_list
package bases which will be ignored during auto updates

Type
list[str]

pacman

alpm wrapper instance

Type
Pacman

paths

repository paths instance

Type
RepositoryPaths

repo

repo commands wrapper instance

Type
Repo

reporter

build status reporter instance

Type
Client

repository_id

repository unique identifier

Type
RepositoryId

sign

GPG wrapper instance

Type
GPG

triggers

triggers holder

Type
TriggerLoader

vcs_allowed_age

maximal age of the VCS packages before they will be checked

Type
int

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **database** (SQLite) – database instance
- **report** (*bool*) – force enable or disable reporting
- **refresh_pacman_database** (PacmanSynchronization) – pacman database synchronization level

packager(*packagers: Packagers*, *package_base: str*) → *User*

extract packager from configuration having username

Parameters

- **packagers** (Packagers) – packagers override holder
- **package_base** (*str*) – package base to lookup

Returns
user found in database if any and empty object otherwise

Return type
User | None

property architecture: str

repository architecture for backward compatibility

Returns
repository architecture

Return type
str

property name: str

repository name for backward compatibility

Returns
repository name

Return type
str

ahriman.core.repository.update_handler module

class UpdateHandler(*repository_id: RepositoryId*, *configuration: Configuration*, *database: SQLite*, ***, *report: bool*, *refresh_pacman_database: PacmanSynchronization*)

Bases: *PackageInfo*, *Cleaner*

trait to get package update list

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **database** (SQLite) – database instance
- **report** (*bool*) – force enable or disable reporting
- **refresh_pacman_database** (PacmanSynchronization) – pacman database synchronization level

updates_aur(*filter_packages: Iterable[str]*, ***, *vcs: bool*) → list[*Package*]

check AUR for updates

Parameters

- **filter_packages** (*Iterable[str]*) – do not check every package just specified in the list

- **vcs** (*bool*) – enable or disable checking of VCS packages

Returns
list of packages which are out-of-dated

Return type
list[*Package*]

updates_local(*, *vcs: bool*) → list[*Package*]
check local packages for updates

Parameters
vcs (*bool*) – enable or disable checking of VCS packages

Returns
list of local packages which are out-of-dated

Return type
list[*Package*]

updates_manual() → list[*Package*]
check for packages for which manual update has been requested

Returns
list of packages which are out-of-dated

Return type
list[*Package*]

Module contents

ahriman.core.sign package

Submodules

ahriman.core.sign.gpg module

class GPG(*configuration:* Configuration)
Bases: *SyncHttpClient*

gnupg wrapper

configuration
configuration instance

Type
Configuration

default_key
default PGP key ID to use

Type
str | None

targets
list of targets to sign (repository, package etc.)

Type
set[*SignSettings*]

default constructor

Parameters
configuration (Configuration) – configuration instance

key_download(*server: str*, *key: str*) → str

download key from public PGP server

Parameters

- **server** (*str*) – public PGP server which will be used to download data
- **key** (*str*) – key ID to download

Returns
key as plain text

Return type
str

key_export(*key: str*) → str

export public key from stored keychain

Parameters
key (*str*) – key ID to export

Returns
PGP key in .asc format

Return type
str

key_fingerprint(*key: str*) → str

get full key fingerprint from short key id

Parameters
key (*str*) – key ID to lookup

Returns
full PGP key fingerprint

Return type
str

key_import(*server: str*, *key: str*) → None

import key to current user and sign it locally

Parameters

- **server** (*str*) – public PGP server which will be used to download data
- **key** (*str*) – key ID to import

process(*path: Path*, *key: str*) → list[Path]

gpg command wrapper

Parameters

- **path** (*Path*) – path to file to sign
- **key** (*str*) – PGP key ID

Returns
list of generated files including original file

Return type
list[Path]

process_sign_package(*path: Path*, *packager_key: str | None*) → list[Path]
sign package if required by configuration and signature doesn't exist

Parameters

- **path** (*Path*) – path to file to sign
- **packager_key** (*str | None*) – optional packager key to sign

Returns
list of generated files including original file

Return type
list[Path]

process_sign_repository(*path: Path*) → list[Path]
sign repository if required by configuration

Notes

More likely you just want to pass *repository_sign_args* to repo wrapper

Parameters
path (*Path*) – path to repository database

Returns
list of generated files including original file

Return type
list[Path]

static sign_command(*path: Path*, *key: str*) → list[str]
gpg command to run

Parameters

- **path** (*Path*) – path to file to sign
- **key** (*str*) – PGP key ID

Returns
gpg command with all required arguments

Return type
list[str]

static sign_options(*configuration:* Configuration) → tuple[set[*SignSettings*], str | None]
extract default sign options from configuration

Parameters
configuration (Configuration) – configuration instance

Returns
tuple of sign targets and default PGP key

Return type
tuple[set[*SignSettings*], str | None]

static signature(*filepath: Path*) → Path

generate signature name for the file

Parameters
filepath (*Path*) – path to the file which will be signed

Returns
path to signature file

Return type
str

property repository_sign_args: list[str]

get command line arguments based on settings

Returns
command line arguments for repo-add command to sign database

Return type
list[str]

Module contents

ahriman.core.status package

Submodules

ahriman.core.status.client module

class Client

Bases: object

base build status reporter client

static load(*repository_id: RepositoryId*, *configuration: Configuration*, *, *report: bool*) → *Client*

load client from settings

Parameters

- **repository_id** (*RepositoryId*) – repository unique identifier
- **configuration** (*Configuration*) – configuration instance
- **report** (*bool*) – force enable or disable reporting

Returns
client according to current settings

Return type
Client

package_add(*package: Package*, *status: BuildStatusEnum*) → None

add new package with status

Parameters

- **package** (*Package*) – package properties
- **status** (*BuildStatusEnum*) – current package build status

package_changes_get(*package_base: str*) → *Changes*

get package changes

Parameters
package_base (*str*) – package base to retrieve

Returns
package changes if available and empty object otherwise

Return type
Changes

package_changes_set(*package_base: str*, *changes:* Changes) → None

update package changes

Parameters

- **package_base** (*str*) – package base to update
- **changes** (Changes) – changes descriptor

package_get(*package_base: str | None*) → list[tuple[*Package*, *BuildStatus*]]

get package status

Parameters
package_base (*str | None*) – package base to get

Returns
list of current package description and status if it has been found

Return type
list[tuple[*Package*, *BuildStatus*]]

package_logs(*log_record_id:* LogRecordId, *record: LogRecord*) → None

post log record

Parameters

- **log_record_id** (LogRecordId) – log record id
- **record** (*logging.LogRecord*) – log record to post to api

package_remove(*package_base: str*) → None

remove packages from watcher

Parameters
package_base (*str*) – package base to remove

package_update(*package_base: str*, *status:* BuildStatusEnum) → None

update package build status. Unlike *package_add()* it does not update package properties

Parameters

- **package_base** (*str*) – package base to update
- **status** (BuildStatusEnum) – current package build status

set_building(*package_base: str*) → None

set package status to building

Parameters
package_base (*str*) – package base to update

set_failed(*package_base: str*) → None

set package status to failed

Parameters
package_base (*str*) – package base to update

set_pending(*package_base: str*) → None

set package status to pending

Parameters
package_base (*str*) – package base to update

set_success(*package: Package*) → None

set package status to success

Parameters
package (Package) – current package properties

set_unknown(*package: Package*) → None

set package status to unknown

Parameters
package (Package) – current package properties

status_get() → *InternalStatus*

get internal service status

Returns
current internal (web) service status

Return type
InternalStatus

status_update(*status: BuildStatusEnum*) → None

update ahriman status itself

Parameters
status (BuildStatusEnum) – current ahriman status

ahriman.core.status.watcher module

class Watcher(*repository_id: RepositoryId*, *database: SQLite*)

Bases: *LazyLogging*

package status watcher

database

database instance

Type
SQLite

repository_id

repository unique identifier

Type
RepositoryId

status

daemon status

Type
BuildStatus

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **database** (SQLite) – database instance

load() → None

load packages from local database

logs_get(*package_base: str*, *limit: int = -1*, *offset: int = 0*) → list[tuple[float, str]]

extract logs for the package base

Parameters

- **package_base** (*str*) – package base
- **limit** (*int, optional*) – limit records to the specified count, -1 means unlimited (Default value = -1)
- **offset** (*int, optional*) – records offset (Default value = 0)

Returns
package logs

Return type
list[tuple[float, str]]

logs_remove(*package_base: str*, *version: str | None*) → None

remove package related logs

Parameters

- **package_base** (*str*) – package base
- **version** (*str*) – package versio

logs_update(*log_record_id:* LogRecordId, *created: float*, *record: str*) → None

make new log record into database

Parameters

- **log_record_id** (LogRecordId) – log record id
- **created** (*float*) – log created timestamp
- **record** (*str*) – log record

package_changes_get(*package_base: str*) → *Changes*

retrieve package changes

Parameters
package_base (*str*) – package base

Returns
package changes if available

Return type
Changes

package_get(*package_base: str*) → tuple[*Package*, *BuildStatus*]

get current package base build status

Parameters
package_base (*str*) – package base

Returns
package and its status

Return type
tuple[*Package*, *BuildStatus*]

Raises
UnknownPackageError – if no package found

package_remove(*package_base: str*) → None

remove package base from known list if any

Parameters
package_base (*str*) – package base

package_update(*package_base: str*, *status:* BuildStatusEnum, *package:* Package | *None*) → None

update package status and description

Parameters

- **package_base** (*str*) – package base to update
- **status** (BuildStatusEnum) – new build status
- **package** (Package | *None*) – optional package description. In case if not set current properties will be used

patches_get(*package_base: str*, *variable: str* | *None*) → list[*PkgbuildPatch*]

get patches for the package

Parameters

- **package_base** (*str*) – package base
- **variable** (*str* | *None*) – patch variable name if any

Returns
list of patches which are stored for the package

Return type
list[*PkgbuildPatch*]

patches_remove(*package_base: str*, *variable: str*) → None

remove package patch

Parameters

- **package_base** (*str*) – package base
- **variable** (*str*) – patch variable name

patches_update(*package_base: str*, *patch:* PkgbuildPatch) → None

update package patch

Parameters

- **package_base** (*str*) – package base
- **patch** (PkgbuildPatch) – package patch

status_update(*status:* BuildStatusEnum) → None

update service status

Parameters
status (BuildStatusEnum) – new service status

property packages: list[tuple[*Package*, *BuildStatus*]]

get current known packages list

Returns
list of packages together with their statuses

Return type
list[tuple[*Package*, *BuildStatus*]]

ahriman.core.status.web_client module

class WebClient(*repository_id:* RepositoryId, *configuration:* Configuration)

Bases: *Client*, *SyncAhrimanClient*

build status reporter web client

repository_id

repository unique identifier

Type
RepositoryId

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance

package_add(*package:* Package, *status:* BuildStatusEnum) → None

add new package with status

Parameters

- **package** (Package) – package properties
- **status** (BuildStatusEnum) – current package build status

package_changes_get(*package_base: str*) → *Changes*

get package changes

Parameters
package_base (*str*) – package base to retrieve

Returns
package changes if available and empty object otherwise

Return type
Changes

package_changes_set(*package_base: str*, *changes:* Changes) → None

update package changes

Parameters

- **package_base** (*str*) – package base to update
- **changes** (Changes) – changes descriptor

package_get(*package_base: str | None*) → list[tuple[*Package*, *BuildStatus*]]

get package status

Parameters
package_base (*str | None*) – package base to get

Returns
list of current package description and status if it has been found

Return type
list[tuple[*Package*, *BuildStatus*]]

package_logs(*log_record_id:* LogRecordId, *record: LogRecord*) → None

post log record

Parameters

- **log_record_id** (LogRecordId) – log record id
- **record** (*logging.LogRecord*) – log record to post to api

package_remove(*package_base: str*) → None

remove packages from watcher

Parameters
package_base (*str*) – basename to remove

package_update(*package_base: str*, *status:* BuildStatusEnum) → None

update package build status. Unlike *package_add()* it does not update package properties

Parameters

- **package_base** (*str*) – package base to update
- **status** (BuildStatusEnum) – current package build status

static parse_address(*configuration:* Configuration) → tuple[str, str]

parse address from legacy configuration

Parameters
configuration (Configuration) – configuration instance

Returns
tuple of section name and server address

Return type
tuple[str, str]

status_get() → *InternalStatus*

get internal service status

Returns
current internal (web) service status

Return type
InternalStatus

status_update(*status:* BuildStatusEnum) → None

update ahriman status itself

Parameters
status (BuildStatusEnum) – current ahriman status

Module contents

ahriman.core.support package

Subpackages

ahriman.core.support.pkgbuild package

Submodules

ahriman.core.support.pkgbuild.keyring_generator module

class KeyringGenerator(*database:* SQLite, *sign:* GPG, *repository_id:* RepositoryId, *configuration:* Configuration, *section: str*)

Bases: *PkgbuildGenerator*

generator for keyring PKGBUILD

sign

GPG wrapper instance

Type
GPG

name

repository name

Type
str

packagers

list of packagers PGP keys

Type
list[str]

pkgbuild_license

keyring package license

Type
list[str]

pkgbuild_pkgdesc

keyring package description

Type
str

pkgbuild_pkgname

keyring package name

Type
str

pkgbuild_url

keyring package home page

Type
str

revoked

list of revoked PGP keys

Type
list[str]

trusted

lif of trusted PGP keys

Type
list[str]

default constructor

Parameters

- **database** (SQLite) – database instance
- **sign** (GPG) – GPG wrapper instance
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **section** (*str*) – settings section name

install() → str | None

content of the .install functions

Returns
content of the .install functions if any

Return type
str | None

package() → str

package function generator

Returns
package() function for PKGBUILD

Return type
str

sources() → dict[str, Callable[[Path], None]]

return list of sources for the package

Returns
map of source identifier (e.g. filename) to its generator function

Return type
dict[str, Callable[[Path], None]]

property license: list[str]

package licenses list

Returns
package licenses as PKGBUILD property

Return type
list[str]

property pkgdesc: str

package description

Returns
package description as PKGBUILD property

Return type
str

property pkgname: str

package name

Returns
package name as PKGBUILD property

Return type
str

property url: str

package upstream url

Returns
package upstream url as PKGBUILD property

Return type
str

ahriman.core.support.pkgbuild.mirrorlist_generator module

class MirrorlistGenerator(*repository_id:* RepositoryId, *configuration:* Configuration, *section: str*)

Bases: *PkgbuildGenerator*

generator for mirrorlist PKGBUILD

path

path to mirrorlist relative to /

Type
Path

pkgbuild_license

mirrorlist package license

Type
list[str]

pkgbuild_pkgdesc

mirrorlist package description

Type
str

pkgbuild_pkgname

mirrorlist package name

Type
str

pkgbuild_url

mirrorlist package home page

Type
str

servers

list of mirror servers

Type
list[str]

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **section** (*str*) – settings section name

package() → str

package function generator

Returns
package() function for PKGBUILD

Return type
str

patches() → list[*PkgbuildPatch*]

list of additional PKGBUILD properties

Returns
list of patches which generate PKGBUILD content

Return type
list[*PkgbuildPatch*]

sources() → dict[str, Callable[[Path], None]]

return list of sources for the package

Returns
map of source identifier (e.g. filename) to its generator function

Return type
dict[str, Callable[[Path], None]]

property license: list[str]

package licenses list

Returns
package licenses as PKGBUILD property

Return type
list[str]

property pkgdesc: str

package description

Returns
package description as PKGBUILD property

Return type
str

property pkgname: str

package name

Returns
package name as PKGBUILD property

Return type
str

property url: str

package upstream url

Returns
package upstream url as PKGBUILD property

Return type
str

ahriman.core.support.pkgbuild.pkgbuild_generator module

class PkgbuildGenerator

Bases: object

main class for generating PKGBUILDs

PKGBUILD_STATIC_PROPERTIES

(class attribute) list of default pkgbuild static properties

Type
list[*PkgbuildPatch*]

install() → str | None

content of the .install functions

Returns
content of the .install functions if any

Return type
str | None

package() → str

package function generator

Returns
package() function for PKGBUILD

Return type
str

Raises
NotImplementedError – not implemented method

patches() → list[*PkgbuildPatch*]

list of additional PKGBUILD properties

Returns
list of patches which generate PKGBUILD content

Return type
list[*PkgbuildPatch*]

sources() → dict[str, Callable[[Path], None]]

return list of sources for the package

Returns
map of source identifier (e.g. filename) to its generator function

Return type
dict[str, Callable[[Path], None]]

write_install(*source_dir: Path*) → list[*PkgbuildPatch*]

generate content of install file

Parameters
source_dir (*Path*) – path to directory in which sources must be generated

Returns
patch for the pkgbuild if install file exists and empty list otherwise

Return type
list[*PkgbuildPatch*]

write_pkgbuild(*source_dir: Path*) → None

generate PKGBUILD content to the specified path

Parameters
source_dir (*Path*) – path to directory in which sources must be generated

write_sources(*source_dir: Path*) → list[*PkgbuildPatch*]

write sources and returns valid PKGBUILD properties for them

Parameters
source_dir (*Path*) – path to directory in which sources must be generated

Returns
list of patches to be applied to the PKGBUILD

Return type
list[*PkgbuildPatch*]

property license: list[str]

package licenses list

Returns
package licenses as PKGBUILD property

Return type
list[str]

property pkgdesc: str

package description

Returns
package description as PKGBUILD property

Return type
str

Raises
NotImplementedError – not implemented method

property pkgname: str

package name

Returns
package name as PKGBUILD property

Return type
str

Raises
NotImplementedError – not implemented method

property pkgver: str

package version

Returns
package version as PKGBUILD property

Return type
str

property url: str

package upstream url

Returns
package upstream url as PKGBUILD property

Return type
str

Module contents

Submodules

ahriman.core.support.keyring_trigger module

class KeyringTrigger(*repository_id:* RepositoryId, *configuration:* Configuration)

Bases: *Trigger*

keyring generator trigger

targets

git remote target list

Type
list[str]

default constructor

Parameters
- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance

classmethod configuration_sections(*configuration:* Configuration) → list[str]

extract configuration sections from configuration

Parameters
configuration (Configuration) – configuration instance

Returns
read configuration sections belong to this trigger

Return type
list[str]

on_start() → None

trigger action which will be called at the start of the application

ahriman.core.support.mirrorlist_trigger module

class MirrorlistTrigger(*repository_id:* RepositoryId, *configuration:* Configuration)

Bases: *Trigger*

mirrorlist generator trigger

targets

git remote target list

Type
list[str]

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance

classmethod configuration_sections(*configuration:* Configuration) → list[str]

extract configuration sections from configuration

Parameters
configuration (Configuration) – configuration instance

Returns
read configuration sections belong to this trigger

Return type
list[str]

on_start() → None

trigger action which will be called at the start of the application

ahriman.core.support.package_creator module

class PackageCreator(*configuration:* Configuration, *generator:* PkgbuildGenerator)

Bases: object

helper which creates packages based on pkgbuild generator

configuration

configuration instance

Type
Configuration

generator

PKGBUILD generator instance

Type
PkgbuildGenerator

default constructor

Parameters

- **configuration** (Configuration) – configuration instance
- **generator** (PkgbuildGenerator) – PKGBUILD generator instance

run() → None

create new local package

Module contents

ahriman.core.triggers package

Submodules

ahriman.core.triggers.trigger module

class Trigger(*repository_id:* RepositoryId, *configuration:* Configuration)

Bases: *LazyLogging*

trigger base class

CONFIGURATION_SCHEMA

(class attribute) configuration schema template

Type
ConfigurationSchema

CONFIGURATION_SCHEMA_FALLBACK

(class attribute) optional fallback option for defining configuration schema type used

Type
str | None

configuration
configuration instance

Type
Configuration

repository_id
repository unique identifier

Type
RepositoryId

Examples

This class must be used in order to create own extension. Basically idea is the following:

```
>>> class CustomTrigger(Trigger):
>>>     def on_result(self, result: Result, packages: list[Package]) -> None:
>>>         perform_some_action()
```

Having this class you can pass it to `configuration` and it will be run on action:

```
>>> from ahriman.core.triggers import TriggerLoader
>>>
>>> configuration = Configuration()
>>> configuration.set_option("build", "triggers", "my.awesome.package.CustomTrigger
↪")
>>>
>>> loader = TriggerLoader.load(RepositoryId("x86_64", "aur-clone"), configuration)
>>> loader.on_result(Result(), [])
```

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance

classmethod configuration_schema(*repository_id:* RepositoryId, *configuration:* Configuration | *None*) → dict[str, dict[str, Any]]

configuration schema based on supplied service configuration

Notes

Schema must be in cerberus format, for details and examples you can check built-in triggers.

Parameters

- **repository_id** (*str*) – repository unique identifier
- **configuration** (Configuration | *None*) – configuration instance. If set to None, the default schema should be returned

Returns
configuration schema in cerberus format

Return type
ConfigurationSchema

classmethod configuration_sections(*configuration: Configuration*) → list[str]

extract configuration sections from configuration

Parameters
configuration (Configuration) – configuration instance

Returns
read configuration sections belong to this trigger

Return type
list[str]

Examples

This method can be used in order to extract specific configuration sections which are set by user, e.g. from sources:

```
>>> @classmethod
>>> def configuration_sections(cls, configuration: Configuration) -> list[str]:
>>>     return configuration.getlist("report", "target", fallback=[])
```

on_result(*result: Result*, *packages: list[Package]*) → None

trigger action which will be called after build process with process result

Parameters

- **result** (Result) – build result
- **packages** (*list[Package]*) – list of all available packages

on_start() → None

trigger action which will be called at the start of the application

on_stop() → None

trigger action which will be called before the stop of the application

property architecture: str

repository architecture for backward compatibility

Returns
repository architecture

Return type
str

ahriman.core.triggers.trigger_loader module

class TriggerLoader

Bases: *LazyLogging*

trigger loader class

triggers
list of loaded triggers according to the configuration

Type
list[*Trigger*]

Examples

This class more likely must not be used directly, but the usual workflow is the following:

```
>>> configuration = Configuration()  # create configuration
>>> configuration.set_option("build", "triggers", "ahriman.core.report.ReportTrigger
↪")  # set class for load
```

Having such configuration you can create instance of the loader:

```
>>> loader = TriggerLoader.load(RepositoryId("x86_64", "aur-clone"), configuration)
>>> print(loader.triggers)
```

After that you are free to run triggers:

```
>>> loader.on_result(Result(), [])
```

default constructor

static known_triggers(*configuration:* Configuration) → list[str]
read configuration and return list of known triggers. Unlike *selected_triggers()* this option is used mainly for configuration and validation and mentioned triggers are not being executed automatically

Parameters
configuration (Configuration) – configuration instance

Returns
list of registered, but not enabled, triggers

Return type
list[str]

classmethod load(*repository_id:* RepositoryId, *configuration:* Configuration) → Self
create instance from configuration

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance

Returns
fully loaded trigger instance

Return type
Self

load_trigger(*module_path: str*, *repository_id:* RepositoryId, *configuration:* Configuration) → *Trigger*
load trigger by module path

Parameters

- **module_path** (*str*) – module import path to load

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance

Returns
loaded trigger based on settings

Return type
Trigger

Raises
ExtensionError – in case if trigger could not be instantiated

load_trigger_class(*module_path: str*) → type[*Trigger*]

load trigger class by module path

Parameters
module_path (*str*) – module import path to load

Returns
loaded trigger type by module path

Return type
type[*Trigger*]

Raises
ExtensionError – in case if module cannot be loaded from the specified module path or is not a trigger

on_result(*result: Result*, *packages: list[Package]*) → None

run trigger with result of application run

Parameters

- **result** (Result) – build result
- **packages** (*list[Package]*) – list of all available packages

on_start() → None

run triggers on load

on_stop() → None

run triggers before the application exit

static selected_triggers(*configuration: Configuration*) → list[str]

read configuration and return triggers which are set by settings

Parameters
configuration (Configuration) – configuration instance

Returns
list of triggers according to configuration

Return type
list[str]

Module contents

ahriman.core.upload package

Submodules

ahriman.core.upload.github module

class GitHub(*repository_id:* RepositoryId, *configuration:* Configuration, *section: str*)

Bases: *Upload*, *HttpUpload*

upload files to GitHub releases

github_owner

GitHub repository owner

Type
str

github_release_tag

GitHub release tag

Type
str

github_release_tag_name

GitHub release tag name

Type
str

github_repository

GitHub repository name

Type
str

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **section** (*str*) – settings section name

asset_remove(*release: dict[str, Any]*, *name: str*) → None

remove asset from the release by name

Parameters

- **release** (*dict[str, Any]*) – release object
- **name** (*str*) – asset name

asset_upload(*release: dict[str, Any]*, *path: Path*) → None

upload asset to the release

Parameters

- **release** (*dict[str, Any]*) – release object
- **path** (*Path*) – path to local file

files_remove(*release: dict[str, Any]*, *local_files: dict[Path, str]*, *remote_files: dict[str, str]*) → None

remove files from GitHub

Parameters

- **release** (*dict[str, Any]*) – release object
- **local_files** (*dict[Path, str]*) – map of local file paths to its checksum
- **remote_files** (*dict[str, str]*) – map of the remote files and its checksum

files_upload(*release: dict[str, Any]*, *local_files: dict[Path, str]*, *remote_files: dict[str, str]*) → None

upload files to GitHub

Parameters

- **release** (*dict[str, Any]*) – release object
- **local_files** (*dict[Path, str]*) – map of local file paths to its checksum
- **remote_files** (*dict[str, str]*) – map of the remote files and its checksum

get_local_files(*path: Path*) → dict[Path, str]

get all local files and their calculated checksums

Parameters
path (*Path*) – local path to sync

Returns
map of path objects to its checksum

Return type
dict[Path, str]

release_create() → dict[str, Any]

create empty release

Returns
GitHub API release object for the new release

Return type
dict[str, Any]

release_get() → dict[str, Any] | None

get release object if any

Returns
GitHub API release object if release found and None otherwise

Return type
dict[str, Any] | None

release_update(*release: dict[str, Any]*, *body: str*) → None

update release

Parameters

- **release** (*dict[str, Any]*) – release object
- **body** (*str*) – new release body

sync(*path: Path*, *built_packages: list[Package]*) → None

sync data to remote server

Parameters

- **path** (*Path*) – local path to sync
- **built_packages** (*list[Package]*) – list of packages which has just been built

ahriman.core.upload.http_upload module

class HttpUpload(*configuration: Configuration | None = None*, *section: str | None = None*, *, *suppress_errors: bool = False*)

Bases: *SyncHttpClient*

helper for the http based uploads

default constructor

Parameters

- **configuration** (*Configuration | None, optional*) – configuration instance (Default value = None)
- **section** (*str | None, optional*) – settings section name (Default value = None)
- **suppress_errors** (*bool, optional*) – suppress logging of request errors (Default value = False)

static calculate_hash(*path: Path*) → str

calculate file checksum

Parameters
path (*Path*) – path to local file

Returns
calculated checksum of the file

Return type
str

static get_body(*local_files: dict[Path, str]*) → str

generate release body from the checksums as returned from HttpUpload.get_hashes method

Parameters
local_files (*dict[Path, str]*) – map of the paths to its checksum

Returns
body to be inserted into release

Return type
str

static get_hashes(*body: str*) → dict[str, str]

get checksums of the content from the repository

Parameters
body (*str*) – release string body object

Returns
map of the filename to its checksum as it is written in body

Return type
dict[str, str]

ahriman.core.upload.remote_service module

class RemoteService(*repository_id: RepositoryId*, *configuration: Configuration*, *section: str*)

Bases: *Upload*, *HttpUpload*

upload files to another server instance

client

web client instance

Type
WebClient

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **section** (*str*) – settings section name

package_upload(*path: Path*, *package: Package*) → None

upload single package to remote

Parameters

- **path** (*Path*) – local path to sync
- **package** (Package) – package to upload

sync(*path: Path*, *built_packages: list[Package]*) → None

sync data to remote server

Parameters

- **path** (*Path*) – local path to sync
- **built_packages** (*list[Package]*) – list of packages which has just been built

property session: Session

get or create session

Returns
created session object

Return type
request.Session

ahriman.core.upload.rsync module

class Rsync(*repository_id:* RepositoryId, *configuration:* Configuration, *section: str*)

Bases: *Upload*

rsync wrapper

command

command arguments for sync

Type
list[str]

remote

remote address to sync

Type
str

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **section** (*str*) – settings section name

sync(*path: Path*, *built_packages: list[*Package*]*) → None

sync data to remote server

Parameters

- **path** (*Path*) – local path to sync
- **built_packages** (*list[*Package*]*) – list of packages which has just been built

ahriman.core.upload.s3 module

class S3(*repository_id:* RepositoryId, *configuration:* Configuration, *section: str*)

Bases: *Upload*

boto3 wrapper

Attributes

bucket(Any): boto3 S3 bucket object chunk_size(int): chunk size for calculating checksums object_path(Path): relative path to which packages will be uploaded

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **section** (*str*) – settings section name

static calculate_etag(*path: Path*, *chunk_size: int*) → str

calculate amazon s3 etag credits to https://teppen.io/2018/10/23/aws_s3_verify_etags/ For this method we have to define nosec because it is out of any security context and provided by AWS

Parameters

- **path** (*Path*) – path to local file
- **chunk_size** (*int*) – read chunk size, which depends on client settings

Returns
calculated entity tag for local file

Return type
str

static files_remove(*local_files: dict[Path, str]*, *remote_objects: dict[Path, Any]*) → None

remove files which have been removed locally

Parameters

- **local_files** (*dict[Path, str]*) – map of local path object to its checksum
- **remote_objects** (*dict[Path, Any]*) – map of remote path object to the remote s3 object

files_upload(*path: Path*, *local_files: dict[Path, str]*, *remote_objects: dict[Path, Any]*) → None

upload changed files to s3

Parameters

- **path** (*Path*) – local path to sync
- **local_files** (*dict[Path, str]*) – map of local path object to its checksum
- **remote_objects** (*dict[Path, Any]*) – map of remote path object to the remote s3 object

static get_bucket(*configuration: Configuration*, *section: str*) → Any

create resource client from configuration

Parameters

- **configuration** (*Configuration*) – configuration instance
- **section** (*str*) – settings section name

Returns
amazon client

Return type
Any

get_local_files(*path: Path*) → dict[Path, str]

get all local files and their calculated checksums

Parameters
path (*Path*) – local path to sync

Returns
map of path object to its checksum

Return type
dict[Path, str]

get_remote_objects() → dict[Path, Any]

get all remote objects and their checksums

Returns

map of path object to the remote s3 object

Return type

dict[Path, Any]

sync(*path: Path*, *built_packages: list[*Package*]*) → None

sync data to remote server

Parameters

- **path** (*Path*) – local path to sync
- **built_packages** (*list[*Package*]*) – list of packages which has just been built

ahriman.core.upload.upload module

class Upload(*repository_id:* RepositoryId, *configuration:* Configuration)

Bases: *LazyLogging*

base remote sync class

configuration

configuration instance

Type

Configuration

repository_id

repository unique identifier

Type

RepositoryId

Examples

These classes provide the way to upload packages to remote sources as it is described in their implementations. Basic flow includes class instantiating by using the *load()* method and then calling the *run()* method which wraps any internal exceptions into the *ahriman.core.exceptions.SynchronizationError* exception:

```
>>> configuration = Configuration()
>>> upload = Upload.load(RepositoryId("x86_64", "aur-clone"), configuration, "s3")
>>> upload.run(configuration.repository_paths.repository, [])
```

Or in case if direct access to exception is required, the *sync()* method can be used:

```
>>> try:
>>>     upload.sync(configuration.repository_paths.repository, [])
>>> except Exception as ex:
>>>     handle_exceptions(ex)
```

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance

static load(*repository_id:* RepositoryId, *configuration:* Configuration, *target: str*) → *Upload*

load client from settings

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance
- **target** (*str*) – target to run sync (e.g. s3)

Returns
client according to current settings

Return type
Upload

run(*path: Path*, *built_packages: list[*Package*]*) → None

run remote sync

Parameters

- **path** (*Path*) – local path to sync
- **built_packages** (*list[*Package*]*) – list of packages which has just been built

Raises
SynchronizationError – in case of any synchronization unmatched exception

sync(*path: Path*, *built_packages: list[*Package*]*) → None

sync data to remote server

Parameters

- **path** (*Path*) – local path to sync
- **built_packages** (*list[*Package*]*) – list of packages which has just been built

ahriman.core.upload.upload_trigger module

class UploadTrigger(*repository_id:* RepositoryId, *configuration:* Configuration)

Bases: *Trigger*

synchronization trigger

targets

upload target list

Type
list[str]

default constructor

Parameters

- **repository_id** (RepositoryId) – repository unique identifier
- **configuration** (Configuration) – configuration instance

classmethod configuration_sections(*configuration: Configuration*) → list[str]

extract configuration sections from configuration

Parameters
configuration (Configuration) – configuration instance

Returns
read configuration sections belong to this trigger

Return type
list[str]

on_result(*result: Result*, *packages: list[Package]*) → None

run trigger

Parameters

- **result** (Result) – build result
- **packages** (*list[Package]*) – list of all available packages

Module contents

Submodules

ahriman.core.exceptions module

exception BuildError(*package_base: str*, *stderr: str | None = None*)

Bases: RuntimeError

base exception for failed builds

default constructor

Parameters

- **package_base** (*str*) – package base raised exception
- **stderr** (*str | None, optional*) – stderr of the process if available (Default value = None)

classmethod from_process(*package_base: str*) → Callable[[int, list[str], str, str], Self]

generate exception callable from process error

Parameters
package_base (*str*) – package base raised exception

Returns
exception generator to be passed to subprocess utils

Return type
Callable[[int, list[str], str, str], Self]

exception CalledProcessError(*status_code: int*, *process: list[str]*, *stderr: str*)

Bases: CalledProcessError

like subprocess.CalledProcessError, but better

default constructor

Parameters

- **status_code** (*int*) – process return code
- **process** (*list[str]*) – process argument list
- **stderr** (*str*) – stderr of the process

exception DuplicateRunError

Bases: RuntimeError

exception which will be raised if there is another application instance

default constructor

exception ExitCode

Bases: RuntimeError

special exception which has to be thrown to return non-zero status without error message

exception ExtensionError

Bases: RuntimeError

exception being raised by trigger load in case of errors

exception GitRemoteError

Bases: RuntimeError

git remote exception

default constructor

exception InitializeError(*details: str*)

Bases: RuntimeError

base service initialization exception

default constructor

Parameters
details (*str*) – details of the exception

exception MigrationError(*details: str*)

Bases: RuntimeError

exception which will be raised on migration error

default constructor

Parameters
details (*str*) – error details

exception MissingArchitectureError(*command: str*)

Bases: ValueError

exception which will be raised if architecture is required, but missing

default constructor

Parameters
command (*str*) – command name which throws exception

exception MultipleArchitecturesError(*command: str*, *repositories: list[RepositoryId] | None = None*)

Bases: ValueError

exception which will be raised if multiple architectures are not supported by the handler

default constructor

Parameters

- **command** (*str*) – command name which throws exception
- **repositories** (*list[RepositoryId] | None, optional*) – found repository list (Default value = None)

exception OptionError(*value: Any*)

Bases: ValueError

exception which will be raised on configuration errors

default constructor

Parameters

value (*Any*) – option value

exception PackageInfoError(*details: Any*)

Bases: RuntimeError

exception which will be raised on package load errors

default constructor

Parameters

details (*Any*) – error details

exception PartitionError(*count: int*)

Bases: RuntimeError

exception raised during packages partition actions

default constructor

Parameters

count (*int*) – count of partitions

exception PasswordError(*details: Any*)

Bases: ValueError

exception which will be raised in case of password related errors

default constructor

Parameters

details (*Any*) –

exception PathError(*path: Path*, *root: Path*)

Bases: ValueError

exception which will be raised on path which is not belong to root directory

default constructor

Parameters

- **path** (*Path*) – path which raised an exception
- **root** (*Path*) – repository root (i.e. ahriman home)

exception PkgbuildGeneratorError

Bases: RuntimeError

exception class for support type triggers

default constructor

exception ReportError

Bases: RuntimeError

report generation exception

default constructor

exception SynchronizationError

Bases: RuntimeError

remote synchronization exception

default constructor

exception UnknownPackageError(*package_base: str*)

Bases: ValueError

exception for status watcher which will be thrown on unknown package

default constructor

Parameters

package_base (*str*) – package base name

exception UnsafeRunError(*current_uid: int*, *root_uid: int*)

Bases: RuntimeError

exception which will be raised in case if user is not owner of repository

default constructor

Parameters

- **current_uid** (*int*) – current user ID
- **root_uid** (*int*) – ID of the owner of root directory

ahriman.core.spawn module

class Spawn(*args_parser: ArgumentParser*, *command_arguments: list[str]*)

Bases: Thread, *LazyLogging*

helper to spawn external ahriman process MUST NOT be used directly, the only one usage allowed is to spawn process from web services

active

map of active child processes required to avoid zombies

Type

dict[str, Process]

command_arguments

base command line arguments

Type

list[str]

queue

multiprocessing queue to read updates from processes

Type

Queue[*ProcessStatus* | None]

default constructor

Parameters

- **args_parser** (*argparse.ArgumentParser*) – command line parser for the application
- **command_arguments** (*list[str]*) – base command line arguments

static boolean_action_argument(*name: str*, *value: bool*) → str

convert option of given name with value to boolean action argument

Parameters

- **name** (*str*) – command line argument name
- **value** (*bool*) – command line argument value

Returns

if `value` is True, then returns positive flag and negative otherwise

Return type

str

has_process(*process_id: str*) → bool

check if given process is alive

Parameters

process_id (*str*) – process id to be checked as returned by `_spawn_process()`

Returns

True in case if process still counts as active and False otherwise

Return type

bool

key_import(*key: str*, *server: str | None*) → str

import key to service cache

Parameters

- **key** (*str*) – key to import
- **server** (*str | None*) – PGP key server

Returns

spawned process identifier

Return type

str

packages_add(*repository_id: RepositoryId*, *packages: Iterable[str]*, *username: str | None*, ***, *patches: list[PkgbuildPatch]*, *now: bool*, *increment: bool*, *refresh: bool*) → str

add packages

Parameters

- **repository_id** (*RepositoryId*) – repository unique identifier
- **packages** (*Iterable[str]*) – packages list to add
- **username** (*str | None*) – optional override of username for build process
- **patches** (*list[PkgbuildPatch]*) – list of patches to be passed
- **now** (*bool*) – build packages now

- **increment** (*bool*) – increment pkgrel on conflict
- **refresh** (*bool*) – refresh pacman database before process

Returns
spawned process identifier

Return type
str

packages_rebuild(*repository_id: RepositoryId*, *depends_on: str*, *username: str | None*, ***, *increment: bool*) → str

rebuild packages which depend on the specified package

Parameters

- **repository_id** (*RepositoryId*) – repository unique identifier
- **depends_on** (*str*) – packages dependency
- **username** (*str | None*) – optional override of username for build process
- **increment** (*bool*) – increment pkgrel on conflict

Returns
spawned process identifier

Return type
str

packages_remove(*repository_id: RepositoryId*, *packages: Iterable[str]*) → str

remove packages

Parameters

- **repository_id** (*RepositoryId*) – repository unique identifier
- **packages** (*Iterable[str]*) – packages list to remove

Returns
spawned process identifier

Return type
str

packages_update(*repository_id: RepositoryId*, *username: str | None*, ***, *aur: bool*, *local: bool*, *manual: bool*, *increment: bool*, *refresh: bool*) → str

run full repository update

Parameters

- **repository_id** (*RepositoryId*) – repository unique identifier
- **username** (*str | None*) – optional override of username for build process
- **aur** (*bool*) – check for aur updates
- **local** (*bool*) – check for local packages updates
- **manual** (*bool*) – check for manual packages
- **increment** (*bool*) – increment pkgrel on conflict
- **refresh** (*bool*) – refresh pacman database before process

Returns
spawned process identifier

Return type
str

static process(*callback: Callable[[argparse.Namespace, RepositoryId], bool]*, *args: argparse.Namespace*, *repository_id: RepositoryId*, *process_id: str*, *queue: Queue[ProcessStatus | None]*) → None

helper to run external process

Parameters

- **callback** (*Callable[[argparse.Namespace, str], bool]*) – application run function (i.e. *ahriman.application.handlers.handler.Handler.call()* method)
- **args** (*argparse.Namespace*) – command line arguments
- **repository_id** (RepositoryId) – repository unique identifier
- **process_id** (*str*) – process unique identifier
- **queue** (*Queue[ProcessStatus | None]*) – output queue

run() → None

thread run method

stop() → None

gracefully terminate thread

ahriman.core.tree module

class Leaf(*package: Package*)

Bases: object

tree leaf implementation

dependencies

list of package dependencies

Type
set[str]

package

leaf package properties

Type
Package

default constructor

Parameters
package (Package) – package properties

is_dependency(*packages: Iterable[Leaf]*) → bool

check if the package is dependency of any other package from list or not

Parameters
packages (*Iterable[Leaf]*) – list of known leaves

Returns
True in case if package is dependency of others and False otherwise

Return type
bool

`is_root`(*packages: Iterable[Leaf]*) → bool

check if package depends on any other package from list of not

Parameters
packages (*Iterable[Leaf]*) – list of known leaves

Returns
True if any of packages is dependency of the leaf, False otherwise

Return type
bool

`property items: Iterable[str]`

extract all packages from the leaf

Returns
packages containing in this leaf

Return type
Iterable[str]

`class Tree`(*leaves: list[Leaf]*)

Bases: `object`

dependency tree implementation

`leaves`

list of tree leaves

Type
list[*Leaf*]

Examples

The most important feature here is to generate tree levels one by one which can be achieved by using class method:

```
>>> from ahriman.core.configuration import Configuration
>>> from ahriman.core.database import SQLite
>>> from ahriman.core.repository import Repository
>>> from ahriman.models.repository_id import RepositoryId
>>>
>>> configuration = Configuration()
>>> database = SQLite.load(configuration)
>>> repository = Repository.load(RepositoryId("x86_64", "aur-clone"), configuration,
↪ database, report=True)
>>> packages = repository.packages()
>>>
>>> tree = Tree.resolve(packages)
>>> for tree_level in tree:
>>>     for package in tree_level:
>>>         print(package.base)
>>>     print()
```

The direct constructor call is also possible but requires tree leaves to be instantioned in advance, e.g.:

```
>>> leaves = [Leaf(package) for package in packages]
>>> tree = Tree(leaves)
```

default constructor

Parameters
leaves (*list[Leaf]*) – leaves to build the tree

static balance(*partitions: list[list[Leaf]]*) → list[list[*Leaf*]]

balance partitions. This method tries to find the longest and the shortest lists and move free leaves between them if possible. In case if there are no free packages (i.e. the ones which don't depend on any other in partition and are not dependency of any), it will drop it as it is. This method is guaranteed to produce the same unsorted sequences for same unsorted input

Parameters
partitions (*list[list[Leaf]]*) – source unbalanced partitions

Returns
balanced partitions

Return type
list[list[*Leaf*]]

levels() → list[list[*Package*]]

get build levels starting from the packages which do not require any other package to build

Returns
sorted list of packages lists based on their dependencies

Return type
list[list[*Package*]]

static partition(*packages: Iterable[Package]*, *, *count: int*) → list[list[*Package*]]

partition tree into independent chunks of more or less equal amount of packages. The packages in produced partitions don't depend on any package from other partitions

Parameters

- **packages** (*Iterable[Package]*) – packages list
- **count** (*int*) – maximal amount of partitions

Returns
list of packages lists based on their dependencies. The amount of elements in each sublist is less or equal to `count`

Return type
list[list[*Package*]]

Raises
PartitionError – in case if it is impossible to divide tree by specified amount of partitions

partitions(*, *count: int*) → list[list[*Package*]]

partition tree into (more or less) equal chunks of packages which don't depend on each other

Parameters
count (*int*) – maximal amount of partitions

Returns
sorted list of packages partitions

Return type
list[list[*Package*]]

static resolve(*packages: Iterable[*Package*]*) → list[list[*Package*]]

resolve dependency tree

Parameters
packages (*Iterable[*Package*]*) – packages list

Returns
list of packages lists based on their dependencies

Return type
list[list[*Package*]]

static sort(*leaves: list[list[*Leaf*]]*) → list[list[*Package*]]

sort given list of leaves by package base

Parameters
leaves (*list[list[*Leaf*]]*) – leaves to sort

Returns
sorted list of packages on each level

Return type
list[list[*Package*]]

ahriman.core.util module

check_output(**args: str*, *exception: Exception | Callable[[int, list[str], str, str], Exception] | None = None*, *cwd: Path | None = None*, *input_data: str | None = None*, *logger: Logger | None = None*, *user: int | None = None*, *environment: dict[str, str] | None = None*) → str

subprocess wrapper

Parameters

- ***args** (*str*) – command line arguments
- **exception** (*Exception | Callable[[int, list[str], str, str]] | None, optional*) – exception which has to be raised instead of default subprocess exception. If callable us is supplied, the `subprocess.CalledProcessError` arguments will be passed (Default value = None)
- **cwd** (*Path | None, optional*) – current working directory (Default value = None)
- **input_data** (*str | None, optional*) – data which will be written to command stdin (Default value = None)
- **logger** (*logging.Logger | None, optional*) – logger to log command result if required (Default value = None)
- **user** (*int | None, optional*) – run process as specified user (Default value = None)
- **environment** (*dict[str, str] | None, optional*) – optional environment variables if any (Default value = None)

Returns
command output

Return type
str

Raises
CalledProcessError – if subprocess ended with status code different from 0 and no exception supplied

Examples

Simply call the function:

```
>>> check_output("echo", "hello world")
```

The more complicated calls which include result logging and input data are also possible:

```
>>> import logging
>>>
>>> logger = logging.getLogger()
>>> check_output("python", "-c", "greeting = input('say hello: '); print();␣
↪print(greeting)",
>>>              input_data="hello world", logger=logger)
```

An additional argument `exception` can be supplied in order to override the default exception:

```
>>> check_output("false", exception=RuntimeError("An exception occurred"))
```

check_user(*paths: RepositoryPaths*, *, *unsafe: bool*) → None

check if current user is the owner of the root

Parameters

- **paths** (*RepositoryPaths*) – repository paths object
- **unsafe** (*bool*) – if set no user check will be performed before path creation

Raises
UnsafeRunError – if root uid differs from current uid and check is enabled

Examples

Simply run function with arguments:

```
>>> check_user(paths, unsafe=False)
```

dataclass_view(*instance: Any*) → dict[str, Any]

convert dataclass instance to json object

Parameters
instance (*Any*) – dataclass instance

Returns
json representation of the dataclass with empty field removed

Return type
dict[str, Any]

enum_values(*enum: type[Enum]*) → list[str]

generate list of enumeration values from the source

Parameters
enum (*type[Enum]*) – source enumeration class

Returns
available enumeration values as string

Return type
list[str]

extract_user() → str | None

extract user from system environment

Returns
SUDO_USER in case if set and USER otherwise. It can return None in case if environment has been

Return type
str | None

cleared before application start

filter_json(*source: dict[str, Any]*, *known_fields: Iterable[str]*) → dict[str, Any]

filter json object by fields used for json-to-object conversion

Parameters

- **source** (*dict[str, Any]*) – raw json object
- **known_fields** (*Iterable[str]*) – list of fields which have to be known for the target object

Returns
json object without unknown and empty fields

Return type
dict[str, Any]

Examples

This wrapper is mainly used for the dataclasses, thus the flow must be something like this:

```
>>> from dataclasses import fields
>>> from ahriman.models.package import Package
>>>
>>> known_fields = [pair.name for pair in fields(Package)]
>>> properties = filter_json(dump, known_fields)
>>> package = Package(**properties)
```

full_version(*epoch: str | int | None*, *pkgver: str*, *pkgrel: str*) → str

generate full version from components

Parameters

- **epoch** (*str | int | None*) – package epoch if any
- **pkgver** (*str*) – package version
- **pkgrel** (*str*) – package release version (arch linux specific)

Returns
generated version

Return type
str

minmax(*source: Iterable[T]*, *, *key: Callable[[T], Any] | None = None*) → tuple[T, T]

get min and max value from iterable

Parameters

- **source** (*Iterable[T]*) – source list to find min and max values
- **key** (*Callable[[T], Any] | None, optional*) – key to sort (Default value = None)

Returns
min and max values for sequence

Return type
tuple[T, T]

package_like(*filename: Path*) → bool

check if file looks like package

Parameters
filename (*Path*) – name of file to check

Returns
True in case if name contains `.pkg.` and not signature, False otherwise

Return type
bool

parse_version(*version: str*) → tuple[str | None, str, str]

parse version and returns its components

Parameters
version (*str*) – full version string

Returns
epoch if any, pkgver and pkgrel variables

Return type
tuple[str | None, str, str]

partition(*source: Iterable[T]*, *predicate: Callable[[T], bool]*) → tuple[list[T], list[T]]

partition list into two based on predicate, based on https://docs.python.org/dev/library/itertools.html#itertools-recipes

Parameters

- **source** (*Iterable[T]*) – source list to be partitioned
- **predicate** (*Callable[[T], bool]*) – filter function

Returns
two lists, first is which `predicate` is `True`, second is `False`

Return type
tuple[list[T], list[T]]

pretty_datetime(*timestamp: datetime | float | int | None*) → str

convert datetime object to string

Parameters
timestamp (*datetime.datetime | float | int | None*) – datetime to convert

Returns
pretty printable datetime as string

Return type
str

pretty_size(*size: float | None*, *level: int = 0*) → str
convert size to string

Parameters

- **size** (*float | None*) – size to convert
- **level** (*int, optional*) – represents current units, 0 is B, 1 is KiB, etc. (Default value = 0)

Returns
pretty printable size as string

Return type
str

Raises
OptionError – if size is more than 1TiB

safe_filename(*source: str*) → str
convert source string to its safe representation

Parameters
source (*str*) – string to convert

Returns
result string in which all unsafe characters are replaced by dash

Return type
str

srcinfo_property(*key: str*, *srcinfo: dict[str, Any]*, *package_srcinfo: dict[str, Any]*, ***, *default: Any = None*) → Any
extract property from SRCINFO. This method extracts property from package if this property is presented in `srcinfo`. Otherwise, it looks for the same property in root srcinfo. If none found, the default value will be returned

Parameters

- **key** (*str*) – key to extract
- **srcinfo** (*dict[str, Any]*) – root structure of SRCINFO
- **package_srcinfo** (*dict[str, Any]*) – package specific SRCINFO
- **default** (*Any, optional*) – the default value for the specified key (Default value = None)

Returns
extracted value from SRCINFO

Return type
Any

srcinfo_property_list(*key: str*, *srcinfo: dict[str, Any]*, *package_srcinfo: dict[str, Any]*, ***, *architecture: str | None = None*) → list[Any]
extract list property from SRCINFO. Unlike *srcinfo_property()* it supposes that default return value is always empty list. If `architecture` is supplied, then it will try to lookup for architecture specific values and will append it at the end of result

Parameters

- **key** (*str*) – key to extract
- **srcinfo** (*dict[str, Any]*) – root structure of SRCINFO
- **package_srcinfo** (*dict[str, Any]*) – package specific SRCINFO
- **architecture** (*str | None, optional*) – package architecture if set (Default value = None)

Returns
list of extracted properties from SRCINFO

Return type
list[Any]

trim_package(*package_name: str*) → str

remove version bound and description from package name. Pacman allows to specify version bound (=, <=, >= etc.) for packages in dependencies and also allows to specify description (via :); this function removes trailing parts and return exact package name

Parameters
package_name (*str*) – source package name

Returns
package name without description or version bound

Return type
str

unquote(*source: str*) → str

like `shlex.quote()`, but opposite

Parameters
source (*str*) – source string to remove quotes

Returns
string with quotes removed

Return type
str

Raises
ValueError – if no closing quotation

utcnow() → datetime

get current time

Returns
current time in UTC

Return type
datetime.datetime

walk(*directory_path: Path*) → Generator[Path, None, None]

list all file paths in given directory Credits to https://stackoverflow.com/a/64915960

Parameters
directory_path (*Path*) – root directory path

Yields
Path – all found files in given directory with full path

Examples

Since the `pathlib` module does not provide an alternative to `os.walk()`, this wrapper can be used instead:

```
>>> from pathlib import Path
>>>
>>> for file_path in walk(Path.cwd()):
>>>     print(file_path)
```

Note, however, that unlike the original method, it does not yield directories.

Module contents

ahriman.models package

Submodules

ahriman.models.action module

class Action(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `StrEnum`

base action enumeration

List

(class attribute) list available values

Type
Action

Remove

(class attribute) remove everything from local storage

Type
Action

Update

(class attribute) update local storage or add to

Type
Action

ahriman.models.aur_package module

class AURPackage(***, *id: int*, *name: str*, *package_base_id: int*, *package_base: str*, *version: str*, *num_votes: int*, *popularity: float*, *first_submitted: ~datetime.datetime*, *last_modified: ~datetime.datetime*, *url_path: str*, *description: str = ''*, *url: str | None = None*, *out_of_date: ~datetime.datetime | None = None*, *maintainer: str | None = None*, *submitter: str | None = None*, *repository: str = 'aur'*, *depends: list[str] = <factory>*, *make_depends: list[str] = <factory>*, *opt_depends: list[str] = <factory>*, *check_depends: list[str] = <factory>*, *conflicts: list[str] = <factory>*, *provides: list[str] = <factory>*, *license: list[str] = <factory>*, *keywords: list[str] = <factory>*)

Bases: `object`

AUR package descriptor

id

package ID

Type
int

name

package name

Type
str

package_base_id

package base ID

Type
int

version

package base version

Type
str

description

package base description

Type
str

url

package upstream URL

Type
str | None

num_votes

number of votes for the package

Type
int

popularity

package popularity

Type
float

out_of_date

package out of date timestamp if any

Type
datetime.datetime | None

`maintainer`
: package maintainer

  Type
  : str | None

`submitter`
: package first submitter

  Type
  : str | None

`first_submitted`
: timestamp of the first package submission

  Type
  : datetime.datetime

`last_modified`
: timestamp of the last package submission

  Type
  : datetime.datetime

`url_path`
: AUR package path

  Type
  : str

`repository`
: repository name of the package

  Type
  : str

`depends`
: list of package dependencies

  Type
  : list[str]

`make_depends`
: list of package make dependencies

  Type
  : l[str]

`opt_depends`
: list of package optional dependencies

  Type
  : list[str]

`check_depends`
: list of package test dependencies

  Type
  : list[str]

conflicts

conflicts list for the package

Type
list[str]

provides

list of packages which this package provides

Type
list[str]

license

list of package licenses

Type
list[str]

keywords

list of package keywords

Type
list[str]

Examples

Mainly this class must be used from class methods instead of default `__init__()`:

```
>>> package = AURPackage.from_json(metadata)  # load package from json dump
>>> # ...or alternatively...
>>> package = AURPackage.from_repo(metadata)  # load package from official␣
↪repository RPC
>>> # properties of the class are built based on ones from AUR RPC, thus additional␣
↪method is required
>>>
>>> from ahriman.core.alpm.pacman import Pacman
>>> from ahriman.core.configuration import Configuration
>>> from ahriman.models.repository_id import RepositoryId
>>>
>>> configuration = Configuration()
>>> pacman = Pacman(RepositoryId("x86_64", "aur-clone"), configuration)
>>> metadata = pacman.package_get("pacman")
>>> package = AURPackage.from_pacman(next(metadata))  # load package from pyalpm␣
↪wrapper
```

static convert(*descriptor: dict[str, Any]*) → dict[str, Any]

covert AUR RPC key names to package keys

Parameters
descriptor (*dict[str, Any]*) – RPC package descriptor

Returns
package descriptor with names converted to snake case

Return type
dict[str, Any]

classmethod from_json(*dump: dict[str, Any]*) → Self

construct package descriptor from RPC properties

Parameters
dump (*dict[str, Any]*) – json dump body

Returns
AUR package descriptor

Return type
Self

classmethod from_pacman(*package: pyalpm.Package*) → Self

construct package descriptor from official repository wrapper

Parameters
package (Package) – pyalpm package descriptor

Returns
AUR package descriptor

Return type
Self

classmethod from_repo(*dump: dict[str, Any]*) → Self

construct package descriptor from official repository RPC properties

Parameters
dump (*dict[str, Any]*) – json dump body

Returns
AUR package descriptor

Return type
Self

ahriman.models.auth_settings module

class AuthSettings(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: StrEnum

web authorization type

Disabled

(class attribute) authorization is disabled

Type
AuthSettings

Configuration

(class attribute) configuration based authorization

Type
AuthSettings

OAuth

(class attribute) OAuth based provider

Type
AuthSettings

`static from_option`(*value: str*) → *AuthSettings*

construct value from configuration

Parameters
value (*str*) – configuration value

Returns
parsed value

Return type
AuthSettings

`property is_enabled: bool`

get enabled flag

Returns
False in case if authorization is disabled and True otherwise

Return type
bool

ahriman.models.build_status module

`class BuildStatus`(*status: ~ahriman.models.build_status.BuildStatusEnum = BuildStatusEnum.Unknown*, *timestamp: int = <factory>*)

Bases: `object`

build status holder

`status`

build status

Type
BuildStatusEnum

`timestamp`

build status update time

Type
int

`classmethod from_json`(*dump: dict[str, Any]*) → Self

construct status properties from json dump

Parameters
dump (*dict[str, Any]*) – json dump body

Returns
status properties

Return type
Self

`pretty_print`() → str

generate pretty string representation

Returns
print-friendly string

Return type
str

`view`() → dict[str, Any]

generate json status view

Returns
json-friendly dictionary

Return type
dict[str, Any]

`class BuildStatusEnum`(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `StrEnum`

build status enumeration

`Unknown`

(class attribute) build status is unknown

Type
BuildStatusEnum

`Pending`

(class attribute) package is out-of-dated and will be built soon

Type
BuildStatusEnum

`Building`

(class attribute) package is building right now

Type
BuildStatusEnum

`Failed`

(class attribute) package build failed

Type
BuildStatusEnum

`Success`

(class attribute) package has been built without errors

Type
BuildStatusEnum

ahriman.models.changes module

class Changes(*last_commit_sha: str | None = None*, *changes: str | None = None*)

Bases: `object`

package source files changes holder

last_commit_sha

last commit hash

Type
str | None

changes

package change since the last commit if available

Type
str | None

classmethod from_json(*dump: dict[str, Any]*) → Self

construct changes from the json dump

Parameters
dump (*dict[str, Any]*) – json dump body

Returns
changes object

Return type
Self

view() → dict[str, Any]

generate json change view

Returns
json-friendly dictionary

Return type
dict[str, Any]

property is_empty: bool

validate that changes are not empty

Returns
`True` in case if changes are not set and `False` otherwise

Return type
bool

ahriman.models.context_key module

class ContextKey(*key: str*, *return_type: type[T]*)

Bases: `Generic[T]`

ahriman context key for typing purposes

key

context key to lookup

Type
str

`return_type`
return type used for the specified context key

Type
type[T]

ahriman.models.counters module

`class Counters`(*, *total: int*, *unknown: int = 0*, *pending: int = 0*, *building: int = 0*, *failed: int = 0*, *success: int = 0*)

Bases: `object`

package counters

`total`
total packages count

Type
int

`unknown`
packages in unknown status count

Type
int

`pending`
packages in pending status count

Type
int

`building`
packages in building status count

Type
int

`failed`
packages in failed status count

Type
int

`success`
packages in success status count

Type
int

`classmethod from_json`(*dump: dict[str, Any]*) → Self
construct counters from json dump

Parameters
dump (*dict[str, Any]*) – json dump body

Returns
status counters

Return type
Self

classmethod from_packages(*packages: list[tuple[*Package, BuildStatus*]]*) → Self

construct counters from packages statuses

Parameters
packages (*list[tuple[*Package, BuildStatus*]]*) – list of package and their status as per watcher property

Returns
status counters

Return type
Self

ahriman.models.internal_status module

class InternalStatus(***, *status:* BuildStatus, *architecture: str | None = None*, *packages:* Counters = *Counters(total=0, unknown=0, pending=0, building=0, failed=0, success=0)*, *repository: str | None = None*, *version: str | None = None*)

Bases: `object`

internal server status

status
service status

Type
BuildStatus

architecture
repository architecture

Type
str | None

packages
packages statuses counter object

Type
Counters

repository
repository name

Type
str | None

version
service version

Type
str | None

classmethod from_json(*dump: dict[str, Any]*) → Self

construct internal status from json dump

Parameters
dump (*dict[str, Any]*) – json dump body

Returns
internal status

Return type
Self

view() → dict[str, Any]

generate json status view

Returns
json-friendly dictionary

Return type
dict[str, Any]

ahriman.models.log_handler module

class LogHandler(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: StrEnum

log handler as described by default configuration

Console

(class attribute) write logs to console

Type
LogHandler

Syslog

(class attribute) write logs to syslog device /dev/null

Type
LogHandler

Journald

(class attribute) write logs to journald directly

Type
LogHandler

ahriman.models.log_record_id module

class LogRecordId(*package_base: str*, *version: str*)

Bases: object

log record process identifier

package_base

package base for which log record belongs

Type
str

version
package version for which log record belongs

Type
str

ahriman.models.migration module

class Migration(*, *index: int*, *name: str*, *steps: list[str]*, *migrate_data: Callable[[Connection,* Configuration*], None]*)

Bases: `object`

migration implementation

index
migration position

Type
int

name
migration name

Type
str

steps
migration steps

Type
list[str]

migrate_data
data migration callback

Type
Callable[[Connection, *Configuration*], None]

ahriman.models.migration_result module

class MigrationResult(*, *old_version: int*, *new_version: int*)

Bases: `object`

migration result implementation model

old_version
old schema version before migrations

Type
int

new_version
new schema version after migrations

Type
int

`validate`() → None

perform version validation

Raises
MigrationError – if old version is newer than new one or negative

`property is_outdated: bool`

check migration and check if there are pending migrations

Returns
True in case if it requires migrations and False otherwise

Return type
bool

ahriman.models.package module

`class Package`(*, *base: str*, *version: str*, *remote:* RemoteSource, *packages: dict[str,* PackageDescription*]*, *packager: str | None = None*)

Bases: *LazyLogging*

package properties representation

`base`

package base name

Type
str

`packager`

package packager if available

Type
str | None

`packages`

map of package names to their properties. Filled only on load from archive

Type
dict[str, *PackageDescription*

`remote`

package remote source if applicable

Type
RemoteSource

`version`

package full version

Type
str

Examples

Different usages of this class may generate different (incomplete) data, e.g. if instantiating class from json:

```
>>> package = Package.from_json(dump)
```

it will contain every data available in the json body. Otherwise, if generate package from local archive:

```
>>> package = Package.from_archive(local_path, pacman)
```

it will probably miss file descriptions (in case if there are multiple packages which belong to the base).

The specific class load method must be defined based on the source provided. The following methods (mostly) must be used: *from_archive()*, *from_aur()*, *from_build()*, *from_official()* for sources *ahriman.models.package_source.PackageSource.Archive*, *ahriman.models.package_source.PackageSource.AUR*, *ahriman.models.package_source.PackageSource.Local* and *ahriman.models.package_source.PackageSource.Repository* repsectively:

```
>>> ahriman_package = Package.from_aur("ahriman")
>>> pacman_package = Package.from_official("pacman", pacman)
```

actual_version(*paths:* RepositoryPaths) → str

additional method to handle VCS package versions

Parameters
paths (RepositoryPaths) – repository paths instance

Returns
package version if package is not VCS and current version according to VCS otherwise

Return type
str

Raises
PackageInfoError – if there are parsing errors

classmethod from_archive(*path: Path*, *pacman:* Pacman) → Self

construct package properties from package archive

Parameters

- **path** (*Path*) – path to package archive
- **pacman** (Pacman) – alpm wrapper instance

Returns
package properties

Return type
Self

classmethod from_aur(*name: str*, *packager: str | None = None*) → Self

construct package properties from AUR page

Parameters

- **name** (*str*) – package name (either base or normal name)
- **packager** (*str | None, optional*) – packager to be used for this build (Default value = None)

Returns
package properties

Return type
Self

classmethod from_build(*path: Path*, *architecture: str*, *packager: str | None = None*) → Self

construct package properties from sources directory

Parameters

- **path** (*Path*) – path to package sources directory
- **architecture** (*str*) – load package for specific architecture
- **packager** (*str | None, optional*) – packager to be used for this build (Default value = None)

Returns
package properties

Return type
Self

Raises
PackageInfoError – if there are parsing errors

classmethod from_json(*dump: dict[str, Any]*) → Self

construct package properties from json dump

Parameters
dump (*dict[str, Any]*) – json dump body

Returns
package properties

Return type
Self

classmethod from_official(*name: str*, *pacman: Pacman*, *packager: str | None = None*, ***, *use_syncdb: bool = True*) → Self

construct package properties from official repository page

Parameters

- **name** (*str*) – package name (either base or normal name)
- **pacman** (*Pacman*) – alpm wrapper instance
- **packager** (*str | None, optional*) – packager to be used for this build (Default value = None)
- **use_syncdb** (*bool, optional*) – use pacman databases instead of official repositories RPC (Default value = True)

Returns
package properties

Return type
Self

full_depends(*pacman: Pacman*, *packages: Iterable[Package]*) → list[str]

generate full dependencies list including transitive dependencies

Parameters

- **pacman** (Pacman) – alpm wrapper instance
- **packages** (*Iterable[Package]*) – repository package list

Returns
all dependencies of the package

Return type
list[str]

is_newer_than(*timestamp: float | int*) → bool

check if package was built after the specified timestamp

Parameters
timestamp (*float | int*) – timestamp to check build date against

Returns
True in case if package was built after the specified date and False otherwise. In case if build date is not set by any of packages, it returns False

Return type
bool

is_outdated(*remote:* Package, *paths:* RepositoryPaths, *, *vcs_allowed_age: float | int = 0*, *calculate_version: bool = True*) → bool

check if package is out-of-dated

Parameters

- **remote** (Package) – package properties from remote source
- **paths** (RepositoryPaths) – repository paths instance. Required for VCS packages cache
- **vcs_allowed_age** (*float | int, optional*) – max age of the built packages before they will be forced to calculate actual version (Default value = 0)
- **calculate_version** (*bool, optional*) – expand version to actual value (by calculating git versions) (Default value = True)

Returns
True if the package is out-of-dated and False otherwise

Return type
bool

static local_files(*path: Path*) → Generator[Path, None, None]

extract list of local files

Parameters
path (*Path*) – path to package sources directory

Yields
Path –

list of paths of files which belong to the package and distributed together with this tarball.
All paths are relative to the `path`

Raises
PackageInfoError – if there are parsing errors

next_pkgrel(*local_version: str*) → str | None

generate next pkgrel variable. The package release will be incremented if `local_version` is more or equal to the *version*; in this case the function will return new pkgrel value, otherwise `None` will be returned

Parameters
local_version (*str*) – locally stored package version

Returns
new generated package release version if any. In case if the release contains dot (e.g. 1.2),
the minor part will be incremented by 1. If the release does not contain major.minor notation, the minor version equals to 1 will be appended

Return type
str | None

pretty_print() → str

generate pretty string representation

Returns
print-friendly string

Return type
str

static supported_architectures(*path: Path*) → set[str]

load supported architectures from package sources

Parameters
path (*Path*) – path to package sources directory

Returns
list of package supported architectures

Return type
set[str]

Raises
PackageInfoError – if there are parsing errors

view() → dict[str, Any]

generate json package view

Returns
json-friendly dictionary

Return type
dict[str, Any]

property depends: list[str]

get package base dependencies

Returns
sum of dependencies per each package

Return type
list[str]

property depends_build: set[str]

get full list of external dependencies which has to be installed for build process

Returns
full dependencies list used by devtools

Return type
set[str]

`property depends_check: list[str]`

get package test dependencies

Returns
sum of test dependencies per each package

Return type
list[str]

`property depends_make: list[str]`

get package make dependencies

Returns
sum of make dependencies per each package

Return type
list[str]

`property depends_opt: list[str]`

get package optional dependencies

Returns
sum of optional dependencies per each package

Return type
list[str]

`property groups: list[str]`

get package base groups

Returns
sum of groups per each package

Return type
list[str]

`property is_single_package: bool`

is it possible to transform package base to single package or not

Returns
true in case if this base has only one package with the same name

Return type
bool

`property is_vcs: bool`

get VCS flag based on the package base

Returns
True in case if package base looks like VCS package and False otherwise

Return type
bool

property licenses: list[str]

get package base licenses

Returns
sum of licenses per each package

Return type
list[str]

property packages_full: list[str]

get full packages list including provides

Returns
full list of packages which this base contains

Return type
list[str]

ahriman.models.package_description module

class PackageDescription(*, architecture: str | None = None, archive_size: int | None = None, build_date: int | None = None, depends: list[str] = <factory>, make_depends: list[str] = <factory>, opt_depends: list[str] = <factory>, check_depends: list[str] = <factory>, description: str | None = None, filename: str | None = None, groups: list[str] = <factory>, installed_size: int | None = None, licenses: list[str] = <factory>, provides: list[str] = <factory>, url: str | None = None)

Bases: object

package specific properties

architecture

package architecture

Type
str | None

archive_size

package archive size

Type
int | None

build_date

package build date

Type
int | None

check_depends

package dependencies list used for check functions

Type
list[str]

depends

package dependencies list

Type
list[str]

`opt_depends`
: optional package dependencies list

  Type
  : list[str]

`make_depends`
: package dependencies list used for building

  Type
  : list[str]

`description`
: package description

  Type
  : str | None

`filename`
: package archive name

  Type
  : str | None

`groups`
: package groups

  Type
  : list[str]

`installed_size`
: package installed size

  Type
  : int | None

`licenses`
: package licenses list

  Type
  : list[str]

`provides`
: list of provided packages

  Type
  : list[str]

`url`
: package url

  Type
  : str | None

Examples

Unlike the *ahriman.models.package.Package* class, this implementation only holds properties. The recommended way to deal with it is to read data based on the source type - either json or `pyalpm.Package` instance:

```
>>> description = PackageDescription.from_json(dump)
>>>
>>> from pathlib import Path
>>> from ahriman.core.alpm.pacman import Pacman
>>> from ahriman.core.configuration import Configuration
>>> from ahriman.models.repository_id import RepositoryId
>>>
>>> configuration = Configuration()
>>> pacman = Pacman(RepositoryId("x86_64", "aur-clone"), configuration)
>>> pyalpm_description = next(package for package in pacman.package_get("pacman"))
>>> description = PackageDescription.from_package(
>>>     pyalpm_description, Path("/var/cache/pacman/pkg/pacman-6.0.1-4-x86_64.pkg.
↪tar.zst"))
```

classmethod from_aur(*package: AURPackage*) → Self

construct properties from AUR package model

Parameters
package (AURPackage) – AUR package model

Returns
package properties based on source AUR package

Return type
Self

classmethod from_json(*dump: dict[str, Any]*) → Self

construct package properties from json dump

Parameters
dump (*dict[str, Any]*) – json dump body

Returns
package properties

Return type
Self

classmethod from_package(*package: pyalpm.Package*, *path: Path*) → Self

construct class from alpm package class

Parameters

- **package** (Package) – alpm generated object
- **path** (*Path*) – path to package archive

Returns
package properties based on tarball

Return type
Self

view() → dict[str, Any]

generate json package view

Returns
json-friendly dictionary

Return type
dict[str, Any]

property filepath: Path | None

wrapper for filename, convert it to Path object

Returns
path object for current filename

Return type
Path | None

ahriman.models.package_source module

class PackageSource(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: StrEnum

package source for addition enumeration

Auto

(class attribute) automatically determine type of the source

Type
PackageSource

Archive

(class attribute) source is a package archive

Type
PackageSource

AUR

(class attribute) source is an AUR package for which it should search

Type
PackageSource

Directory

(class attribute) source is a directory which contains packages

Type
PackageSource

Local

(class attribute) source is locally stored PKGBUILD

Type
PackageSource

Remote

(class attribute) source is remote (http, ftp etc...) link

Type
PackageSource

Repository

(class attribute) source is official repository

Type
PackageSource

Examples

In case if source is unknown the *resolve()* and the source descriptor is available method must be used:

```
>>> real_source = PackageSource.Auto.resolve("ahriman", configuration.repository_
↪paths)
```

the code above will ensure that the presudo-source *Auto* will not be processed later.

resolve(*source: str*, *paths:* RepositoryPaths) → *PackageSource*

resolve auto into the correct type

Parameters

- **source** (*str*) – source of the package
- **paths** (RepositoryPaths) – repository paths instance

Returns
non-auto type of the package source

Return type
PackageSource

ahriman.models.packagers module

class Packagers(*default: str | None = None*, *overrides: dict[str, str | None] = <factory>*)

Bases: object

holder for packagers overrides

default

default packager username if any to be used if no override for the specified base was found

Type
str | None

overrides

dict[str, str | None]: packager username override for specific package base

Type
dict[str, str | None]

for_base(*package_base: str*) → str | None

extract username for the specified package base

Parameters
package_base (*str*) – package base to lookup

Returns
package base override if set and default packager username otherwise

Return type
str | None

ahriman.models.pacman_synchronization module

class PacmanSynchronization(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: IntEnum

pacman database synchronization flag

Disabled

(class attribute) do not synchronize local database

Type
PacmanSynchronization

Enabled

(class attribute) synchronize local database (same as pacman -Sy)

Type
PacmanSynchronization

Force

(class attribute) force synchronize local database (same as pacman -Syy)

Type
PacmanSynchronization

ahriman.models.pkgbuild_patch module

class PkgbuildPatch(*key: str | None*, *value: str | list[str]*)

Bases: object

wrapper for patching PKBGUILDs

key

name of the property in PKGBUILD, e.g. version, url etc. If not set, patch will be considered as full PKGBUILD diffs

Type
str | None

value

value of the stored PKGBUILD property. It must be either string or list of string values

Type
str | list[str]

classmethod from_env(*variable: str*) → Self

construct patch from environment variable. Functions are not supported

Parameters
variable (*str*) – variable in bash form, i.e. KEY=VALUE

Returns
package properties

Return type
Self

`quote()`

Return a shell-escaped version of the string *s*.

`serialize()` → str

serialize key-value pair into PKGBUILD string. List values will be put inside parentheses. All string values (including the ones inside list values) will be put inside quotes, no shell variables expanding supported at the moment

Returns
serialized key-value pair, print-friendly

Return type
str

`view()` → dict[str, Any]

generate json patch view

Returns
json-friendly dictionary

Return type
dict[str, Any]

`write`(*pkgbuild_path: Path*) → None

write serialized value into PKGBUILD by specified path

Parameters
pkgbuild_path (*Path*) – path to PKGBUILD file

`property is_function: bool`

parse key and define whether it function or not

Returns
True in case if key ends with parentheses and False otherwise

Return type
bool

`property is_plain_diff: bool`

check if patch is full diff one or just single-variable patch

Returns
True in case key set and False otherwise

Return type
bool

ahriman.models.process_status module

class ProcessStatus(*process_id: str*, *status: bool*, *consumed_time: int*)

Bases: object

terminated process status descriptor

process_id

unique process identifier

Type
str

status

process exit code status

Type
bool

consumed_time

consumed time in ms

Type
int

ahriman.models.property module

class Property(*name: str*, *value: Any*, *indent: int = 1*, ***, *is_required: bool = False*)

Bases: object

holder of object properties descriptor

name

name of the property

Type
str

value

property value

Type
Any

is_required

if set to True then this property is required

Type
bool

indent

property indentation level

Type
int

ahriman.models.remote_source module

class RemoteSource(*, *source:* PackageSource, *git_url: str | None = None*, *web_url: str | None = None*, *path: str | None = None*, *branch: str | None = None*)

Bases: object

remote package source properties

branch

branch of the git repository

Type
str | None

git_url

url of the git repository

Type
str | None

path

path to directory with PKGBUILD inside the git repository

Type
str | None

source

package source pointer used by some parsers

Type
PackageSource

web_url

url of the package in the web interface

Type
str | None

classmethod **from_json**(*dump: dict[str, Any]*) → Self

construct remote source from the json dump (or database row)

Parameters
dump (*dict[str, Any]*) – json dump body

Returns
remote source

Return type
Self

git_source() → tuple[str, str]

get git source if available

Returns
git url and branch

Return type
tuple[str, str]

Raises
InitializeError – in case if git url and/or branch are not set

view() → dict[str, Any]

generate json package remote view

Returns
json-friendly dictionary

Return type
dict[str, Any]

property is_remote: bool

check if source is remote

Returns
True in case if package is well-known remote source (e.g. AUR) and False otherwise

Return type
bool

property pkgbuild_dir: Path | None

get path to directory with package sources (PKGBUILD etc.)

Returns
path to directory with package sources based on settings if available

Return type
Path | None

ahriman.models.report_settings module

class ReportSettings(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: StrEnum

report targets enumeration

Disabled

(class attribute) option which generates no report for testing purpose

Type
ReportSettings

HTML

(class attribute) html report generation

Type
ReportSettings

Email

(class attribute) email report generation

Type
ReportSettings

Console

(class attribute) print result to console

Type
ReportSettings

`Telegram`
(class attribute) markdown report to telegram channel

Type
ReportSettings

`RemoteCall`
(class attribute) remote ahriman server call

Type
ReportSettings

`static from_option`(*value: str*) → *ReportSettings*
construct value from configuration

Parameters
value (*str*) – configuration value

Returns
parsed value

Return type
ReportSettings

ahriman.models.repository_id module

`class RepositoryId`(*architecture: str*, *name: str*)
Bases: `object`

unique identifier of the repository

`architecture`
repository architecture

Type
str

`name`
repository name

Type
str

`query`() → list[tuple[str, str]]
generate query parameters

Returns
json view as query parameters

Return type
list[tuple[str, str]]

`view`() → dict[str, Any]
generate json package view

Returns
json-friendly dictionary

Return type
dict[str, Any]

property id: str

get repository id to be used for databases

Returns
unique id for this repository

Return type
str

property is_empty: bool

check if all data is supplied for the loading

Returns
True in case if architecture or name are not set and False otherwise

Return type
bool

ahriman.models.repository_paths module

class RepositoryPaths(*root: Path*, *repository_id:* RepositoryId, *, *_force_current_tree: bool = False*)

Bases: *LazyLogging*

repository paths holder. For the most operations with paths you want to use this object

repository_id

repository unique identifier

Type
RepositoryId

root

repository root (i.e. ahriman home)

Type
Path

Examples

This class can be used in order to access the repository tree structure:

```
>>> paths = RepositoryPaths(Path("/var/lib/ahriman"), RepositoryId("x86_64", "aur-
↪clone"))
```

Additional methods can be used in order to ensure that tree is created:

```
>>> paths.tree_create()
```

Access to directories inside can be done by either using properties or specifying the package base:

```
>>> cache_dir = paths.cache
>>> ahriman_cache_dir = paths.cache_for("ahriman")
```

cache_for(*package_base: str*) → Path

get path to cached PKGBUILD and package sources for the package base

Parameters
package_base (*str*) – package base name

Returns
full path to directory for specified package base cache

Return type
Path

chown(*path: Path*) → None

set owner of path recursively (from root) to root owner

Parameters
path (*Path*) – path to be chown

Raises
PathError – if path does not belong to root

classmethod known_architectures(*root: Path*, *name: str = ''*) → set[str]

get known architecture names

Parameters

- **root** (*Path*) – repository root
- **name** (*str, optional*) – repository name (Default value = "")

Returns
list of repository architectures for which there is created tree

Return type
set[str]

classmethod known_repositories(*root: Path*) → set[str]

get known repository names

Parameters
root (*Path*) – repository root

Returns
list of repository names for which there is created tree. Returns empty set in case if repository is loaded in legacy mode

Return type
set[str]

static owner(*path: Path*) → tuple[int, int]

retrieve owner information by path

Parameters
path (*Path*) – path for which extract ids

Returns
owner user and group ids of the directory

Return type
tuple[int, int]

tree_clear(*package_base: str*) → None

clear package specific files

Parameters
package_base (*str*) – package base name

`tree_create()` → None

create ahriman working tree

`property cache: Path`

get directory for packages cache (mainly used for VCS packages)

Returns
full path to cache directory

Return type
Path

`property chroot: Path`

get directory for devtools chroot

Returns
full patch to devtools chroot directory

Return type
Path

`property packages: Path`

get directory for built packages

Returns
full path to built packages directory

Return type
Path

`property pacman: Path`

get directory for pacman local package cache

Returns
full path to pacman local database cache

Return type
Path

`property repository: Path`

get repository directory

Returns
full path to the repository directory

Return type
Path

`property root_owner: tuple[int, int]`

get UID and GID of the root directory

Returns
owner user and group of the root directory

Return type
tuple[int, int]

ahriman.models.result module

class Result(*, *added: Iterable[Package] | None = None*, *updated: Iterable[Package] | None = None*, *removed: Iterable[Package] | None = None*, *failed: Iterable[Package] | None = None*)

Bases: object

build result class holder

STATUS_PRIORITIES

(class attribute) list of statues according to their priorities

Type
list[str]

default constructor

Parameters

- **added** (*Iterable[Package] | None, optional*) – initial list of successfully added packages (Default value = None)
- **updated** (*Iterable[Package] | None, optional*) – initial list of successfully updated packages (Default value = None)
- **removed** (*Iterable[Package] | None, optional*) – initial list of successfully removed packages (Default value = None)
- **failed** (*Iterable[Package] | None, optional*) – initial list of failed packages (Default value = None)

add_added(*package: Package*) → None

add new package to new packages list

Parameters
package (Package) – package removed

add_failed(*package: Package*) → None

add new package to failed built

Parameters
package (Package) – package with errors during build

add_removed(*package: Package*) → None

add new package to removed list

Parameters
package (Package) – package removed

add_updated(*package: Package*) → None

add new package to success built

Parameters
package (Package) – package built

merge(*other: Result*) → Self

merge other result into this one. This method assumes that other has fresh info about status and override it

Parameters
other (Result) – instance of the newest result

Returns
updated instance

Return type
Self

`refine`() → Self

merge packages between different results (e.g. remove failed from added, etc.) removing duplicates

Returns
updated instance

Return type
Self

`property failed: list[Package]`

get list of failed packages

Returns
list of packages which were failed

Return type
list[*Package*]

`property is_empty: bool`

get if build result is empty or not

Returns
True in case if success list is empty and False otherwise

Return type
bool

`property removed: list[Package]`

get list of removed packages

Returns
list of packages successfully removed

Return type
list[*Package*]

`property success: list[Package]`

get list of success builds

Returns
list of packages with success result

Return type
list[*Package*]

ahriman.models.sign_settings module

`class SignSettings`(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `StrEnum`

sign targets enumeration

`Disabled`

(class attribute) option which generates no report for testing purpose

Type
SignSettings

Packages

(class attribute) sign each package

Type
SignSettings

Repository

(class attribute) sign repository database file

Type
SignSettings

static from_option(*value: str*) → *SignSettings*

construct value from configuration

Parameters
value (*str*) – configuration value

Returns
parsed value

Return type
SignSettings

ahriman.models.smtp_ssl_settings module

class SmtpSSLSettings(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: StrEnum

SMTP SSL mode enumeration

Disabled

(class attribute) no SSL enabled

Type
SmtpSSLSettings

SSL

(class attribute) use SMTP_SSL instead of normal SMTP client

Type
SmtpSSLSettings

STARTTLS

(class attribute) use STARTTLS in normal SMTP client

Type
SmtpSSLSettings

static from_option(*value: str*) → *SmtpSSLSettings*

construct value from configuration

Parameters
value (*str*) – configuration value

Returns
parsed value

Return type
SmtpSSLSettings

ahriman.models.upload_settings module

class UploadSettings(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `StrEnum`

remote synchronization targets enumeration

Disabled
(class attribute) no sync will be performed, required for testing purpose

Type
UploadSettings

Rsync
(class attribute) sync via rsync

Type
UploadSettings

S3
(class attribute) sync to Amazon S3

Type
UploadSettings

GitHub
(class attribute) sync to GitHub releases page

Type
UploadSettings

RemoteService
(class attribute) sync to another ahriman instance

Type
UploadSettings

static from_option(*value: str*) → *UploadSettings*
construct value from configuration

Parameters
value (*str*) – configuration value

Returns
parsed value

Return type
UploadSettings

ahriman.models.user module

class User(*, username: str, password: str, access: UserAccess, packager_id: str | None = None, key: str | None = None)

Bases: object

authorized web user model

username

username

Type
str

password

hashed user password with salt

Type
str

access

user role

Type
UserAccess

packager_id

packager id to be used. If not set, the default service packager will be used

Type
str | None

key

personal packager key if any. If user id is empty, it is interpreted as default key

Type
str | None

Examples

Simply create user from database data and perform required validation:

```
>>> password = User.generate_password(24)
>>> user = User(username="ahriman", password=password, access=UserAccess.Full)
```

Since the password supplied may be plain text, the *hash_password()* method can be used to hash the password:

```
>>> user = user.hash_password("salt")
```

Having the user instance and password, it can be validated:

```
>>> if user.check_credentials(password, "salt"):
>>>     print("password is valid")
>>> else:
>>>     print("password is invalid")
```

. . . and finally access can be verified:

```
>>> if user.verify_access(UserAccess.Read):
>>>     print(f"user {user.username} has read access")
```

check_credentials(*password: str*, *salt: str*) → bool

validate user password

Parameters

- **password** (*str*) – entered password
- **salt** (*str*) – salt for hashed password

Returns
True in case if password matches, False otherwise

Return type
bool

static generate_password(*length: int*) → str

generate password with specified length

Parameters
length (*int*) – password length

Returns
random string which contains letters and numbers

Return type
str

hash_password(*salt: str*) → Self

generate hashed password from plain text

Parameters
salt (*str*) – salt for hashed password

Returns
user with hashed password to store in configuration

Return type
Self

verify_access(*required:* UserAccess) → bool

validate if user has access to requested resource

Parameters
required (UserAccess) – required access level

Returns
True in case if user is allowed to do this request and False otherwise

Return type
bool

ahriman.models.user_access module

class UserAccess(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: StrEnum

web user access enumeration

Unauthorized

(class attribute) user can access specific resources which are marked as available without authorization (e.g. login, logout, static)

Type
UserAccess

Read

(class attribute) user can read the page

Type
UserAccess

Reporter

(class attribute) user can read everything and is able to perform some modifications

Type
UserAccess

Full

(class attribute) user has full access

Type
UserAccess

permits(*other:* UserAccess) → bool

compare enumeration between each other and check if current permission allows the other

Parameters
other (UserAccess) – other permission to compare

Returns
True in case if current permission allows the operation and False otherwise

Return type
bool

ahriman.models.waiter module

class Waiter(*wait_timeout: int*, ***, *start_time: float = <factory>*, *interval: int = 10*)

Bases: object

simple waiter implementation

interval

interval in seconds between checks

Type
int

start_time
monotonic time of the waiter start. More likely must not be assigned explicitly

Type
float

wait_timeout
timeout in seconds to wait for. Negative value will result in immediate exit. Zero value

Type
int

means infinite timeout

is_timed_out() → bool
check if timer is out

Returns
True in case current monotonic time is more than *start_time* and *wait_timeout* doesn't equal to 0

Return type
bool

wait(*in_progress: ~collections.abc.Callable[[~Params], bool], *args: ~typing.~Params, **kwargs: ~typing.~Params*) → float
wait until requirements are not met

Parameters

- **in_progress** (*Callable[Params, bool]*) – function to check if timer should wait for another cycle
- ***args** (*Params.args*) – positional arguments for check call
- ****kwargs** (*Params.kwargs*) – keyword arguments for check call

Returns
consumed time in seconds

Return type
float

ahriman.models.worker module

class Worker(*address: str*, *, *identifier: str = ''*)
Bases: object

worker descriptor

address
worker address to be reachable outside

Type
str

identifier
worker unique identifier. If none set it will be automatically generated from the address

Type
str

view() → dict[str, Any]

generate json patch view

Returns
json-friendly dictionary

Return type
dict[str, Any]

Module contents

ahriman.web package

Subpackages

ahriman.web.middlewares package

Submodules

ahriman.web.middlewares.auth_handler module

setup_auth(*application: Application*, *configuration:* Configuration, *validator:* Auth) → Application

setup authorization policies for the application

Parameters

- **application** (Application) – web application instance
- **configuration** (Configuration) – configuration instance
- **validator** (Auth) – authorization module instance

Returns
configured web application

Return type
Application

ahriman.web.middlewares.exception_handler module

exception_handler(*logger: Logger*) → Callable[[Request, Callable[[Request], Awaitable[StreamResponse]]], Awaitable[StreamResponse]]

exception handler middleware. Just log any exception (except for client ones)

Parameters
logger (*logging.Logger*) – class logger

Returns
built middleware

Return type
MiddlewareType

Raises
HTTPNoContent – OPTIONS method response

Module contents

ahriman.web.schemas package

Submodules

ahriman.web.schemas.aur_package_schema module

class AURPackageSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)

Bases: Schema

response AUR package schema

ahriman.web.schemas.auth_schema module

class AuthSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)

Bases: Schema

request cookie authorization schema

ahriman.web.schemas.build_options_schema module

class BuildOptionsSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)

Bases: Schema

request build options schema

ahriman.web.schemas.changes_schema module

class ChangesSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)

Bases: Schema

response package changes schema

ahriman.web.schemas.counters_schema module

class CountersSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)

Bases: Schema

response package counters schema

ahriman.web.schemas.error_schema module

class ErrorSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)

Bases: Schema

response error schema

ahriman.web.schemas.file_schema module

class FileSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)

Bases: Schema

request file upload schema

ahriman.web.schemas.info_schema module

class InfoSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)

Bases: Schema

response service information schema

ahriman.web.schemas.internal_status_schema module

class InternalStatusSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)

Bases: *RepositoryIdSchema*

response service status schema

ahriman.web.schemas.log_schema module

class LogSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)

Bases: Schema

request package log schema

ahriman.web.schemas.login_schema module

class LoginSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)

Bases: Schema

request login schema

ahriman.web.schemas.logs_schema module

class LogsSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)

Bases: Schema

response package logs schema

ahriman.web.schemas.oauth2_schema module

class OAuth2Schema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)

Bases: Schema

request OAuth2 authorization schema

ahriman.web.schemas.package_name_schema module

class PackageNameSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)

Bases: Schema

request package name schema

ahriman.web.schemas.package_names_schema module

class PackageNamesSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)

Bases: BuildOptionsSchema

request package names schema

ahriman.web.schemas.package_patch_schema module

class PackagePatchSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)

Bases: PackageNamesSchema

response schema with packages and patches

ahriman.web.schemas.package_properties_schema module

class PackagePropertiesSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)

Bases: Schema

request and response package properties schema

ahriman.web.schemas.package_schema module

class PackageSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)

Bases: Schema

request and response package schema

ahriman.web.schemas.package_status_schema module

class PackageStatusSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)

Bases: Schema

response package status schema

class PackageStatusSimplifiedSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)

Bases: Schema

special request package status schema

ahriman.web.schemas.pagination_schema module

class PaginationSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)

Bases: *RepositoryIdSchema*

request pagination schema

ahriman.web.schemas.patch_name_schema module

class PatchNameSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)

Bases: *PackageNameSchema*

request package patch schema

ahriman.web.schemas.patch_schema module

class PatchSchema(*, *only: Sequence[str] | AbstractSet[str] | None = None*, *exclude: Sequence[str] | AbstractSet[str] = ()*, *many: bool = False*, *context: dict | None = None*, *load_only: Sequence[str] | AbstractSet[str] = ()*, *dump_only: Sequence[str] | AbstractSet[str] = ()*, *partial: bool | Sequence[str] | AbstractSet[str] | None = None*, *unknown: str | None = None*)

Bases: Schema

request and response patch schema

ahriman.web.schemas.pgp_key_id_schema module

class PGPKeyIdSchema(*, *only: Sequence[str] | AbstractSet[str] | None = None*, *exclude: Sequence[str] | AbstractSet[str] = ()*, *many: bool = False*, *context: dict | None = None*, *load_only: Sequence[str] | AbstractSet[str] = ()*, *dump_only: Sequence[str] | AbstractSet[str] = ()*, *partial: bool | Sequence[str] | AbstractSet[str] | None = None*, *unknown: str | None = None*)

Bases: Schema

request PGP key ID schema

ahriman.web.schemas.pgp_key_schema module

class PGPKeySchema(*, *only: Sequence[str] | AbstractSet[str] | None = None*, *exclude: Sequence[str] | AbstractSet[str] = ()*, *many: bool = False*, *context: dict | None = None*, *load_only: Sequence[str] | AbstractSet[str] = ()*, *dump_only: Sequence[str] | AbstractSet[str] = ()*, *partial: bool | Sequence[str] | AbstractSet[str] | None = None*, *unknown: str | None = None*)

Bases: Schema

response PGP key schema

ahriman.web.schemas.process_id_schema module

class ProcessIdSchema(*, *only: Sequence[str] | AbstractSet[str] | None = None*, *exclude: Sequence[str] | AbstractSet[str] = ()*, *many: bool = False*, *context: dict | None = None*, *load_only: Sequence[str] | AbstractSet[str] = ()*, *dump_only: Sequence[str] | AbstractSet[str] = ()*, *partial: bool | Sequence[str] | AbstractSet[str] | None = None*, *unknown: str | None = None*)

Bases: Schema

request and response spawned process id schema

ahriman.web.schemas.process_schema module

class ProcessSchema(*, *only: Sequence[str] | AbstractSet[str] | None = None*, *exclude: Sequence[str] | AbstractSet[str] = ()*, *many: bool = False*, *context: dict | None = None*, *load_only: Sequence[str] | AbstractSet[str] = ()*, *dump_only: Sequence[str] | AbstractSet[str] = ()*, *partial: bool | Sequence[str] | AbstractSet[str] | None = None*, *unknown: str | None = None*)

Bases: Schema

process status response schema

ahriman.web.schemas.remote_schema module

class RemoteSchema(*, *only: Sequence[str] | AbstractSet[str] | None = None*, *exclude: Sequence[str] | AbstractSet[str] = ()*, *many: bool = False*, *context: dict | None = None*, *load_only: Sequence[str] | AbstractSet[str] = ()*, *dump_only: Sequence[str] | AbstractSet[str] = ()*, *partial: bool | Sequence[str] | AbstractSet[str] | None = None*, *unknown: str | None = None*)

Bases: Schema

request and response package remote schema

ahriman.web.schemas.repository_id_schema module

class RepositoryIdSchema(*, *only: Sequence[str] | AbstractSet[str] | None = None*, *exclude: Sequence[str] | AbstractSet[str] = ()*, *many: bool = False*, *context: dict | None = None*, *load_only: Sequence[str] | AbstractSet[str] = ()*, *dump_only: Sequence[str] | AbstractSet[str] = ()*, *partial: bool | Sequence[str] | AbstractSet[str] | None = None*, *unknown: str | None = None*)

Bases: Schema

request and response repository unique identifier schema

ahriman.web.schemas.search_schema module

class SearchSchema(*, *only: Sequence[str] | AbstractSet[str] | None = None*, *exclude: Sequence[str] | AbstractSet[str] = ()*, *many: bool = False*, *context: dict | None = None*, *load_only: Sequence[str] | AbstractSet[str] = ()*, *dump_only: Sequence[str] | AbstractSet[str] = ()*, *partial: bool | Sequence[str] | AbstractSet[str] | None = None*, *unknown: str | None = None*)

Bases: Schema

request package search schema

ahriman.web.schemas.status_schema module

class StatusSchema(*, *only: Sequence[str] | AbstractSet[str] | None = None*, *exclude: Sequence[str] | AbstractSet[str] = ()*, *many: bool = False*, *context: dict | None = None*, *load_only: Sequence[str] | AbstractSet[str] = ()*, *dump_only: Sequence[str] | AbstractSet[str] = ()*, *partial: bool | Sequence[str] | AbstractSet[str] | None = None*, *unknown: str | None = None*)

Bases: Schema

request and response status schema

ahriman.web.schemas.update_flags_schema module

class UpdateFlagsSchema(*, *only: Sequence[str] | AbstractSet[str] | None = None*, *exclude: Sequence[str] | AbstractSet[str] = ()*, *many: bool = False*, *context: dict | None = None*, *load_only: Sequence[str] | AbstractSet[str] = ()*, *dump_only: Sequence[str] | AbstractSet[str] = ()*, *partial: bool | Sequence[str] | AbstractSet[str] | None = None*, *unknown: str | None = None*)

Bases: *BuildOptionsSchema*

update flags request schema

ahriman.web.schemas.versioned_log_schema module

class VersionedLogSchema(*, *only: Sequence[str] | AbstractSet[str] | None = None*, *exclude: Sequence[str] | AbstractSet[str] = ()*, *many: bool = False*, *context: dict | None = None*, *load_only: Sequence[str] | AbstractSet[str] = ()*, *dump_only: Sequence[str] | AbstractSet[str] = ()*, *partial: bool | Sequence[str] | AbstractSet[str] | None = None*, *unknown: str | None = None*)

Bases: *LogSchema*, *RepositoryIdSchema*

request package log schema

ahriman.web.schemas.worker_schema module

class WorkerSchema(*, *only: Sequence[str] | AbstractSet[str] | None = None*, *exclude: Sequence[str] | AbstractSet[str] = ()*, *many: bool = False*, *context: dict | None = None*, *load_only: Sequence[str] | AbstractSet[str] = ()*, *dump_only: Sequence[str] | AbstractSet[str] = ()*, *partial: bool | Sequence[str] | AbstractSet[str] | None = None*, *unknown: str | None = None*)

Bases: Schema

request and response schema for workers

Module contents

ahriman.web.views package

Subpackages

ahriman.web.views.api package

Submodules

ahriman.web.views.api.docs module

class DocsView(*request: None*)

Bases: *BaseView*

api docs view

GET_PERMISSION

(class attribute) get permissions of self

Type
UserAccess

async get() → dict[str, Any]

return static docs html

Returns
parameters for jinja template

Return type
dict[str, Any]

ahriman.web.views.api.swagger module

class SwaggerView(*request: None*)

Bases: *BaseView*

api docs specification view

GET_PERMISSION

(class attribute) get permissions of self

Type
UserAccess

async get() → Response

get api specification

Returns
200 with json api specification

Return type
Response

Module contents

ahriman.web.views.v1 package

Subpackages

ahriman.web.views.v1.distributed package

Submodules

ahriman.web.views.v1.distributed.workers module

class WorkersView(*request: None*)

Bases: *BaseView*

distributed workers view

DELETE_PERMISSION

(class attribute) delete permissions of self

Type
UserAccess

GET_PERMISSION

(class attribute) get permissions of self

Type
UserAccess

POST_PERMISSION

(class attribute) post permissions of self

Type
UserAccess

async delete() → None

unregister worker

Raises
HTTPNoContent – on success response

async get() → Response

get workers list

Returns
200 with workers list on success

Return type
Response

async post() → None

register remote worker

Raises

- **HTTPBadRequest** – if bad data is supplied
- **HTTPNoContent** – in case of success response

Module contents

ahriman.web.views.v1.packages package

Submodules

ahriman.web.views.v1.packages.changes module

class ChangesView(*request: None*)

Bases: *StatusViewGuard*, *BaseView*

package changes web view

GET_PERMISSION

(class attribute) get permissions of self

Type
UserAccess

POST_PERMISSION

(class attribute) post permissions of self

Type
UserAccess

async get() → Response

get package changes

Returns
200 with package change on success

Return type
Response

Raises
HTTPNotFound – if package base is unknown

async post() → None

insert new package changes

Raises

- **HTTPBadRequest** – if bad data is supplied
- **HTTPNoContent** – in case of success response

ahriman.web.views.v1.packages.logs module

class LogsView(*request: None*)

Bases: *StatusViewGuard*, *BaseView*

package logs web view

DELETE_PERMISSION

(class attribute) delete permissions of self

Type
UserAccess

GET_PERMISSION

(class attribute) get permissions of self

Type
UserAccess

POST_PERMISSION

(class attribute) post permissions of self

Type
UserAccess

async delete() → None

delete package logs

Raises
HTTPNoContent – on success response

async get() → Response

get last package logs

Returns
200 with package logs on success

Return type
Response

Raises
HTTPNotFound – if package base is unknown

async post() → None

create new package log record

Raises

- **HTTPBadRequest** – if bad data is supplied
- **HTTPNoContent** – in case of success response

ahriman.web.views.v1.packages.package module

class PackageView(*request: None*)

Bases: *StatusViewGuard*, *BaseView*

package base specific web view

DELETE_PERMISSION

(class attribute) delete permissions of self

Type
UserAccess

GET_PERMISSION

(class attribute) get permissions of self

Type
UserAccess

`POST_PERMISSION`

(class attribute) post permissions of self

Type
UserAccess

`async delete()` → None

delete package base from status page

Raises
HTTPNoContent – on success response

`async get()` → Response

get current package base status

Returns
200 with package description on success

Return type
Response

Raises
HTTPNotFound – if no package was found

`async post()` → None

update package build status

Raises

- **HTTPBadRequest** – if bad data is supplied
- **HTTPNoContent** – in case of success response

ahriman.web.views.v1.packages.packages module

`class PackagesView(request: None)`

Bases: *StatusViewGuard*, *BaseView*

global watcher view

`GET_PERMISSION`

(class attribute) get permissions of self

Type
UserAccess

`POST_PERMISSION`

(class attribute) post permissions of self

Type
UserAccess

`async get()` → Response

get current packages status

Returns
200 with package description on success

Return type
Response

async post() → None

reload all packages from repository

Raises
HTTPNoContent – on success response

ahriman.web.views.v1.packages.patch module

class PatchView(*request: None*)

Bases: *StatusViewGuard*, *BaseView*

package patch web view

DELETE_PERMISSION

(class attribute) delete permissions of self

Type
UserAccess

GET_PERMISSION

(class attribute) get permissions of self

Type
UserAccess

async delete() → None

delete package patch

Raises
HTTPNoContent – on success response

async get() → Response

get package patch

Returns
200 with package patch on success

Return type
Response

Raises
HTTPNotFound – if package patch is unknown

ahriman.web.views.v1.packages.patches module

class PatchesView(*request: None*)

Bases: *StatusViewGuard*, *BaseView*

package patches web view

GET_PERMISSION

(class attribute) get permissions of self

Type
UserAccess

POST_PERMISSION

(class attribute) post permissions of self

Type
UserAccess

async get() → Response

get package patches

Returns
200 with package patches on success

Return type
Response

async post() → None

update or create package patch

Raises

- **HTTPBadRequest** – if bad data is supplied
- **HTTPNoContent** – on success response

Module contents

ahriman.web.views.v1.service package

Submodules

ahriman.web.views.v1.service.add module

class AddView(*request: None*)

Bases: *BaseView*

add package web view

POST_PERMISSION

(class attribute) post permissions of self

Type
UserAccess

async post() → Response

add new package

Returns
200 with spawned process id

Return type
Response

Raises
HTTPBadRequest – if bad data is supplied

ahriman.web.views.v1.service.pgp module

class PGPView(*request: None*)

Bases: *BaseView*

pgp key management web view

GET_PERMISSION

(class attribute) get permissions of self

Type
UserAccess

POST_PERMISSION

(class attribute) post permissions of self

Type
UserAccess

async get() → Response

retrieve key from the key server

Returns
200 with key body on success

Return type
Response

Raises

- **HTTPBadRequest** – if bad data is supplied
- **HTTPNotFound** – if key wasn't found or service was unable to fetch it

async post() → Response

store key to the local service environment

Returns
200 with spawned process id

Return type
Response

Raises
HTTPBadRequest – if bad data is supplied

ahriman.web.views.v1.service.process module

class ProcessView(*request: None*)

Bases: *BaseView*

Process information web view

GET_PERMISSION

(class attribute) get permissions of self

Type
UserAccess

async get() → Response

get spawned process status

Returns
200 with process information

Return type
Response

Raises
HTTPNotFound – if no process found

ahriman.web.views.v1.service.rebuild module

class RebuildView(*request: None*)

Bases: *BaseView*

rebuild packages web view

POST_PERMISSION

(class attribute) post permissions of self

Type
UserAccess

async post() → Response

rebuild packages based on their dependency

Returns
200 with spawned process id

Return type
Response

Raises
HTTPBadRequest – if bad data is supplied

ahriman.web.views.v1.service.remove module

class RemoveView(*request: None*)

Bases: *BaseView*

remove package web view

POST_PERMISSION

(class attribute) post permissions of self

Type
UserAccess

async post() → Response

remove existing packages

Returns
200 with spawned process id

Return type
Response

Raises
HTTPBadRequest – if bad data is supplied

ahriman.web.views.v1.service.request module

class RequestView(*request: None*)

Bases: *BaseView*

request package web view. It is actually the same as AddView, but without now

POST_PERMISSION

(class attribute) post permissions of self

Type
UserAccess

async post() → Response

request to add new package

Returns
200 with spawned process id

Return type
Response

Raises
HTTPBadRequest – if bad data is supplied

ahriman.web.views.v1.service.search module

class SearchView(*request: None*)

Bases: *BaseView*

AUR search web view

GET_PERMISSION

(class attribute) get permissions of self

Type
UserAccess

async get() → Response

search packages in AUR

Returns
200 with found package bases and descriptions sorted by base

Return type
Response

Raises

- **HTTPBadRequest** – in case if bad data is supplied
- **HTTPNotFound** – if no packages found

ahriman.web.views.v1.service.update module

class UpdateView(*request: None*)

Bases: *BaseView*

update repository web view

POST_PERMISSION

(class attribute) post permissions of self

Type
UserAccess

async post() → Response

run repository update. No parameters supported here

Returns
200 with spawned process id

Return type
Response

Raises
HTTPBadRequest – if bad data is supplied

ahriman.web.views.v1.service.upload module

class UploadView(*request: None*)

Bases: *BaseView*

upload file to repository

POST_PERMISSION

(class attribute) post permissions of self

Type
UserAccess

async post() → None

upload file from another instance to the server

Raises

- **HTTPBadRequest** – if bad data is supplied
- **HTTPCreated** – on success response
- **HTTPNotFound** – method is disabled by configuration

async static save_file(*part: BodyPartReader*, *target: Path*, ***, *max_body_size: int | None = None*) → tuple[str, Path]

save file to local cache

Parameters

- **part** (*BodyPartReader*) – multipart part to be saved
- **target** (*Path*) – path to directory to which file should be saved
- **max_body_size** (*int | None, optional*) – max body size in bytes (Default value = None)

Returns
map of received filename to its local path

Return type
tuple[str, Path]

Raises
HTTPBadRequest – if bad data is supplied

Module contents

ahriman.web.views.v1.status package

Submodules

ahriman.web.views.v1.status.info module

class InfoView(*request: None*)

Bases: *BaseView*

web service information view

GET_PERMISSION

(class attribute) get permissions of self

Type
UserAccess

async get() → Response

get service information

Returns
200 with service information object

Return type
Response

ahriman.web.views.v1.status.repositories module

class RepositoriesView(*request: None*)

Bases: *BaseView*

repositories view

GET_PERMISSION

(class attribute) get permissions of self

Type
UserAccess

async get() → Response

get list of available repositories

Returns
200 with service status object

Return type
Response

ahriman.web.views.v1.status.status module

class StatusView(*request: None*)

Bases: *StatusViewGuard*, *BaseView*

web service status web view

GET_PERMISSION

(class attribute) get permissions of self

Type
UserAccess

POST_PERMISSION

(class attribute) post permissions of self

Type
UserAccess

async get() → Response

get current service status

Returns
200 with service status object

Return type
Response

async post() → None

update service status

Raises

- **HTTPBadRequest** – if bad data is supplied
- **HTTPNoContent** – in case of success response

Module contents

ahriman.web.views.v1.user package

Submodules

ahriman.web.views.v1.user.login module

class LoginView(*request: None*)

Bases: *BaseView*

login endpoint view

GET_PERMISSION

(class attribute) get permissions of self

Type
UserAccess

POST_PERMISSION

(class attribute) post permissions of self

Type
UserAccess

async get() → None

OAuth2 response handler

In case if code provided it will do a request to get user email. In case if no code provided it will redirect to authorization url provided by OAuth client.

The authentication session will be passed in `Set-Cookie` header.

Raises

- **HTTPFound** – on success response
- **HTTPMethodNotAllowed** – in case if method is used, but OAuth is disabled
- **HTTPUnauthorized** – if case of authorization error

async post() → None

login user to service. The authentication session will be passed in `Set-Cookie` header.

Raises

- **HTTPFound** – on success response
- **HTTPUnauthorized** – if case of authorization error

ahriman.web.views.v1.user.logout module

class LogoutView(*request: None*)

Bases: *BaseView*

logout endpoint view

POST_PERMISSION

(class attribute) post permissions of self

Type
UserAccess

async post() → None

logout user from the service

The server will respond with `Set-Cookie` header, in which API session cookie will be nullified.

Raises

- **HTTPFound** – on success response
- **HTTPUnauthorized** – no authorization cookie available

Module contents

Module contents

ahriman.web.views.v2 package

Subpackages

ahriman.web.views.v2.packages package

Submodules

ahriman.web.views.v2.packages.logs module

class LogsView(*request: None*)

Bases: *StatusViewGuard*, *BaseView*

package logs web view

GET_PERMISSION

(class attribute) get permissions of self

Type
UserAccess

async get() → Response

get last package logs

Returns
200 with package logs on success

Return type
Response

Raises
HTTPNotFound – if package base is unknown

Module contents

Module contents

Submodules

ahriman.web.views.base module

class BaseView(*request: None*)

Bases: View, CorsViewMixin

base web view to make things typed

OPTIONS_PERMISSION

(class attribute) options permissions of self

Type
UserAccess

ROUTES

(class attribute) list of supported routes

Type
list[str]

static get_non_empty(*extractor: Callable[[str], T | None]*, *key: str*) → T

get non-empty value from request parameters

Parameters

- **extractor** (*Callable[[str], T | None]*) – function to get value
- **key** (*str*) – key to extract value

Returns
extracted values if it is presented and not empty

Return type
T

Raises
KeyError – in case if key was not found or value is empty

async classmethod get_permission(*request: Request*) → *UserAccess*

retrieve user permission from the request

Parameters
request (*Request*) – request object

Returns
extracted permission

Return type
UserAccess

async head() → StreamResponse

HEAD method implementation based on the result of GET method

Raises
HTTPMethodNotAllowed – in case if there is no GET method implemented

page() → tuple[int, int]

parse limit and offset and return values

Returns
limit and offset from request

Return type
tuple[int, int]

Raises
HTTPBadRequest – if supplied parameters are invalid

repository_id() → *RepositoryId*

extract repository from request

Returns
repository if possible to construct and first one otherwise

Return type
RepositoryIde

classmethod routes(*configuration:* Configuration) → list[str]

extract routes list for the view

Parameters
configuration (Configuration) – configuration instance

Returns
list of routes defined for the view. By default, it tries to read *ROUTES* option if set and returns empty list otherwise

Return type
list[str]

service(*repository_id:* RepositoryId | *None* = *None*) → *Watcher*

get status watcher instance

Parameters
repository_id (*RepositoryId | None, optional*) – repository unique identifier (Default value = None)

Returns
build status watcher instance. If no repository provided, it will return the first one

Return type
Watcher

Raises
HTTPNotFound – if no repository found

async username() → str | None

extract username from request if any

Returns
authorized username if any and None otherwise (e.g. if authorization is disabled)

Return type
str | None

property configuration: *Configuration*

get configuration instance

Returns
configuration instance

Return type
Configuration

property services: dict[*RepositoryId*, *Watcher*]

get all loaded watchers

Returns
map of loaded watchers per known repository

Return type
dict[*RepositoryId*, *Watcher*]

property sign: *GPG*

get GPG control instance

Returns
GPG wrapper instance

Return type
GPG

property spawner: *Spawn*

get process spawner instance

Returns
external process spawner instance

Return type
Spawn

property validator: *Auth*

get authorization instance

Returns
authorization service instance

Return type
Auth

property workers: *WorkersCache*

get workers cache instance

Returns
workers service

Return type
WorkersCache

ahriman.web.views.index module

class IndexView(*request: None*)

Bases: *BaseView*

root view

It uses jinja2 templates for report generation, the following variables are allowed:

- **auth - authorization descriptor, required**
  - control - HTML to insert for login control, HTML string, required
  - enabled - whether authorization is enabled by configuration or not, boolean, required
  - username - authenticated username if any, string, null means not authenticated
- index_url - url to the repository index, string, optional
- **repositories - list of repositories unique identifiers, required**
  - id - unique repository identifier, string, required
  - repository - repository name, string, required

- architecture - repository architecture, string, required

GET_PERMISSION

(class attribute) get permissions of self

Type
UserAccess

async get() → dict[str, Any]

process get request. No parameters supported here

Returns
parameters for jinja template

Return type
dict[str, Any]

ahriman.web.views.static module

class StaticView(*request: None*)

Bases: *BaseView*

special workaround for static files redirection (e.g. favicon)

GET_PERMISSION

(class attribute) get permissions of self

Type
UserAccess

async get() → None

process get request. No parameters supported here

Raises

- **HTTPFound** – on success response
- **HTTPNotFound** – if path is invalid or unknown

ahriman.web.views.status_view_guard module

class StatusViewGuard

Bases: object

helper for check if status routes are enabled

classmethod routes(*configuration:* Configuration) → list[str]

extract routes list for the view

Parameters
configuration (Configuration) – configuration instance

Returns
list of routes defined for the view. By default, it tries to read ROUTES option if set and returns empty list otherwise

Return type
list[str]

Module contents

Submodules

ahriman.web.apispec module

setup_apispec(*application: Application*) → AiohttpApiSpec

setup swagger api specification

Parameters
application (Application) – web application instance

Returns
created specification instance

Return type
aiohttp_apispec.AiohttpApiSpec

ahriman.web.cors module

setup_cors(*application: Application*) → CorsConfig

setup CORS for the web application

Parameters
application (Application) – web application instance

Returns
generated CORS configuration

Return type
aiohttp_cors.CorsConfig

ahriman.web.keys module

ahriman.web.routes module

setup_routes(*application: Application*, *configuration:* Configuration) → None

setup all defined routes

Parameters

- **application** (Application) – web application instance
- **configuration** (Configuration) – configuration instance

ahriman.web.web module

run_server(*application: Application*) → None

run web application

Parameters
application (Application) – web application instance

setup_server(*configuration:* Configuration, *spawner:* Spawn, *repositories: list[*RepositoryId*]*) → Application

create web application

Parameters

- **configuration** (Configuration) – configuration instance
- **spawner** (Spawn) – spawner thread
- **repositories** (*list[*RepositoryId*]*) – list of known repositories

Returns
web application instance

Return type
Application

Raises
InitializeError – if no repositories set

Module contents

Module contents

PYTHON MODULE INDEX

a

b

c

d

e

f

g

h

t

u

v

w

INDEX

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W