Hydrilla Documentation

This is the documentation of Hydrilla, server-side software that facilitates sharing of scripts for use in Haketilo Hydrilla is used for creating and hosting a repository. If all you want is to install and run Haketilo, you don't need to care about Hydrilla.

- User manual
- Technical documentation
- Reporting bugs
- Downloads
- Roadmap

Contributing

Registrations on hydrillabugs.koszko.org are currently disabled. Please e-mail koszko@koszko.org to express interest in participating.

For information about markup used by this documentation, see Markdown on RedMine.

License

Donors

banner.png	NGI0Discovery_tag.svg

This project was funded through the NGI0 Discovery Fund, a fund established by NLnet with financial support from the European Commission's Next Generation Internet programme, under the aegis of DG Communications Networks, Content and Technology under grant agreement N° 825322.

Hydrilla Releases (prior to version 3)

This page lists releases of Hydrilla before version 3. Since version 3 Hydrilla is distributed together with Haketilo proxy. Please take a look at the Haketilo downloads page for a list of newer releases.

Public keys for verification of signatures are the ones from koszko.org.

PGP key fingerprint: **E972 7060 E3C5 637C 8A4F 4B42 4BC5 221C 5A79 FD1A**
Signify public key: **RWQSf2wUdpjAtrmt7D3t9iHrHFL/GpqXOF+NxECx8ck7swrx6tNzDkM9**

You might want to read our instructions on signature verification.

Version - 1.1-beta1 (pre-release version)

Files

- hydrilla.builder-1.1b1.tar.gz
- hydrilla.builder-1.1b1-py3-none-any.whl
- hydrilla-1.1b1.tar.gz
- hydrilla-1.1b1-py3-none-any.whl

Signify signatures

- hydrilla.builder-1.1b1.tar.gz.sig
- hydrilla.builder-1.1b1-py3-none-any.whl.sig
- hydrilla-1.1b1.tar.gz.sig
- hydrilla-1.1b1-py3-none-any.whl.sig

PGP signatures

- hydrilla.builder-1.1b1.tar.gz.asc
- hydrilla.builder-1.1b1-py3-none-any.whl.asc
- hydrilla-1.1b1.tar.gz.asc
- hydrilla-1.1b1-py3-none-any.whl.asc

Current version - 1.0

Files

- hydrilla.builder-1.0.tar.gz
- hydrilla.builder-1.0-py3-none-any.whl
- hydrilla-1.0.tar.gz
- hydrilla-1.0-py3-none-any.whl

Signify signatures

- hydrilla.builder-1.0.tar.gz.sig
- hydrilla.builder-1.0-py3-none-any.whl.sig
- hydrilla-1.0.tar.gz.sig
- hydrilla-1.0-py3-none-any.whl.sig

PGP signatures

- hydrilla.builder-1.0.tar.gz.asc
- hydrilla.builder-1.0-py3-none-any.whl.asc
- hydrilla-1.0.tar.gz.asc
- hydrilla-1.0-py3-none-any.whl.asc

Version - 1.0-beta2 (pre-release version)

Files

- hydrilla.builder-1.0b2.tar.gz

- hydrilla.builder-1.0b2-py3-none-any.whl
- hydrilla-1.0b2.tar.gz
- hydrilla-1.0b2-py3-none-any.whl

Signify signatures

- hydrilla.builder-1.0b2.tar.gz.sig
- hydrilla.builder-1.0b2-py3-none-any.whl.sig
- hydrilla-1.0b2.tar.gz.sig
- hydrilla-1.0b2-py3-none-any.whl.sig

PGP signatures

- hydrilla.builder-1.0b2.tar.gz.asc
- hydrilla.builder-1.0b2-py3-none-any.whl.asc
- hydrilla-1.0b2.tar.gz.asc
- hydrilla-1.0b2-py3-none-any.whl.asc

Version - 1.0-beta1 (pre-release version)

Files

- hydrilla.builder-1.0b1.tar.gz
- hydrilla.builder-1.0b1-py3-none-any.whl
- hydrilla-1.0b1.tar.gz
- hydrilla-1.0b1-py3-none-any.whl

Signify signatures

- hydrilla.builder-1.0b1.tar.gz.sig
- hydrilla.builder-1.0b1-py3-none-any.whl.sig
- hydrilla-1.0b1.tar.gz.sig
- hydrilla-1.0b1-py3-none-any.whl.sig

PGP signatures

- hydrilla.builder-1.0b1.tar.gz.asc
- hydrilla.builder-1.0b1-py3-none-any.whl.asc
- hydrilla-1.0b1.tar.gz.asc
- hydrilla-1.0b1-py3-none-any.whl.asc

Version - 0.1.1

Files

- hydrilla_0.1.1-1_amd64.deb
- hydrilla_0.1.1-1_arm64.deb
- hydrilla_0.1.1-1.debian.tar.xz
- hydrilla_0.1.1.orig.tar.gz
- hydrilla_0.1.1-1.dsc

PGP signatures

- hydrilla_0.1.1-1_amd64.deb.sig
- hydrilla_0.1.1-1_arm64.deb.sig
- hydrilla_0.1.1-1.dsc.sig

Version - 0.1

Files

- hydrilla_0.1-1_amd64.deb
- hydrilla_0.1-1_arm64.deb
- hydrilla_0.1-1.debian.tar.xz

- hydrilla_0.1.orig.tar.gz
- hydrilla_0.1-1.dsc

PGP signatures

- hydrilla_0.1-1_amd64.deb.sig
- hydrilla_0.1-1_arm64.deb.sig
- hydrilla_0.1-1.dsc.sig

Technical Documentation

Repository API
Hydrilla on-disk data format
Hydrilla source package format
Hydrilla Software Bill of Materials

Hydrilla on-disk data format

This page explains the format of built Hydrilla custom site resources stored in the filesystem.

You might also want to read about the format of Hydrilla source packages or about Hydrilla HTTP JSON API.

How Hydrilla loads packages to serve

Hydrilla expects a serveable directory to be specified in its configuration file (under the key "malcontent-dir"). It then processes the following of its subdirectories.

file/ subdirectory

All files used by resources (either directly or as part of their source packages' license info) are stored under file/sha256 with names being their SHA256 sums.

source/ subdirectory

Every resource and mapping in Hydrilla gets built from a source package. During building of each such source package a JSON description of it gets generated. All descriptions are stored under source with filenames of the format <source_package_identifier>.json.

JSON descriptions of sources have structure as presented in the API description. They are served directly from disk files. In the future they will be accompanied by cryptographic signatures.

Additionally, for each source package a zip archive with its contents is expected to be present under source with filename of the format <source_package_identifier>.zip.

resource/ subdirectory

All resource JSON descriptions are stored under resource with filenames of the format <resource_identifier>/<resource_version_without_revision>.json. For example, definitions of versions 0.1.0-4 and 0.1.1-1 of resource hello would be stored as hello/0.1 and hello/0.1.1, respectively. Both would be under subdirectory hello of resource.

JSON definitions of resources have structure as presented in the API description. They are served directly from disk files. In the future they will be accompanied by cryptographic signatures.

mapping/ subdirectory

All mappings JSON descriptions are stored under mapping with filenames of the format <mapping_identifier>/<mapping_version>.json. For example, definitions of versions 0.1.0 and 0.1.1 of resource hello would be stored as hello/0.1 and hello/0.1.1, respectively. Both would be under subdirectory hello of mapping.

JSON descriptions of mappings have structure as presented in the API description. They are served directly from disk files. In the future they will be accompanied by cryptographic signatures.

Populating Hydrilla serveable directory

Hydrilla package builder writes generated files in a way they are suitable to be loaded by Hydrilla. Given multiple directories already populated with built package files, it is also possible to recursively copy one's all contents into the other. In this case the user is responsible for avoiding clashes in identifiers of resources, mappings and sources.

Hydrilla source package format

This page explains the format of source packages used to prepare Hydrilla custom site resources.

You might also want to read about the data format of built Hydrilla resources.

How Hydrilla packages are built

Hydrilla package builder expects a source package directory and a destination directory to be specified on the command line. If source package directory is missing, current directory is used as default.

Desination directory

The destination directory can be considered the same as Hydrilla's "malcontent" directory - built artifacts will be written in conformance with Hydrilla data format.

Source package directory

Source package directory is expected to contain an index.json file (although Hydrilla builder can be told to use another file and behave as if it was index.json). Hydrilla builder loads it (smartly ignoring "//" comments in it) and collects the definitions of site resources and pattern->payload mappings in it.

Format of an index.json

Example

To understand the format, look into this example file with explanatory comments in it:

```
// SPDX-License-Identifier: CC0-1.0

// Copyright (C) 2021, 2022 Wojtek Kosior <koszko@koszko.org>
// Available under the terms of Creative Commons Zero v1.0 Universal.

// This is an example index.json file describing Hydrilla packages. As you can
// see, for storing this information Hydrilla utilizes JSON with an additional
// extension in the form of '//' comments support.

// An index.json file conveys definitions of site resources and pattern->payload
// mappings. The definitions may reference files under index.json's containing
// directory, using relative paths. This is how scripts, license texts, etc. are
// included.
// File reference always takes the form of an object with "file" property
// specifying path to the file. In certain contexts additional properties may be
// allowed or required. Unix paths (using '/' as separator) are assumed. It is
// not allowed for an index.json file to reference files outside its directory.

// Certain objects are allowed to contain a "comment" field. Although '//'
// comments can be used in index.json files, they will not be included in
// generated JSON definitions. If a comment should be included in the
// definitions served by Hydrilla API, it should be put in a "comment" field of
// the proper object.

// Unknown object properties will be ignored. This is for compatibility with
// possible future revisions of the format.
{
    // Once our index.json schema changes, this field's value will change. Our
    // software will be able to handle both current and older formats thanks to
    // this information present in every index.json file. Schemas that differ by
    // the first (major) number are always incompatible (e.g. a Hydrilla builder
    // instance released at the time of 1.2 being the most recent schema version
    // will not understand version 2.1).
    // Schemas that are backwards-compatible will have the same major number
    // and might differ by the second (minor) version number. The third (patch)
    // and subsequent numbers are being ignored right now.
    "$schema": "https://hydrilla.koszko.org/schemas/package_source-1.schema.json",

    // Used when referring to this source package. Should be consize, unique
    // (among other source package names) and can only use a restricted set of
    // characters. It has to match: [-0-9a-z.]+
    "source_name": "hello",

    // This property lists files that contain copyright information regarding
    // this source package as well as texts of licenses used. Although no
    // specific format of these files is mandated, it is recommended to make
    // each source package REUSE-compliant, generate an spdx report for it as
    // `report.spdx` and list this report together with all license files here.
    "copyright":  [
        {"file": "report.spdx"},
        {"file": "LICENSES/CC0-1.0.txt"}
    ],

    // Where this software/work initially comes from.
    "upstream_url": "https://git.koszko.org/hydrilla-source-package-example",

    // Additional "comment" field can be used if needed.
    // "comment": ""

    // List of actual site resources and pattern->payload mappings. Each of them
    // is represented by an object. Meta-sites and replacement site interfaces
    // will also belong here once they get implemented.
    "definitions": [
        {
            // Value of "type" can currently be one of: "resource" and
            // "mapping". The one we have here, "resource", defines a list
            // of injectable scripts that can be used as a payload or as a
            // dependency of another "resource". In the future CSS style sheets
            // and WASM modules will also be composite parts of a "resource" as
            // scripts are now.
            "type": "resource",

            // Used when referring to this resource in "dependencies" list of
            // another resource or in "payload" field of a mapping. Should
            // be consize and can only use a restricted set of characters. It
            // has to match: [-0-9a-z]+
            "identifier": "helloapple",

            // "long_name" should be used to specify a user-friendly alternative
            // to an identifier. It should generally not collide with a long
            // name of some resource with a different uuid and also shouldn't
            // change in-between versions of the same resource, although
            // exceptions to both rules might be considered. Long name is
            // allowed to contain arbitrary unicode characters (within reason!).
            "long_name": "Hello Apple",

            // Item definitions contain version information. Version is
            // represented as an array of integers, with major version number
            // being the first array item. In case of resources, version is
            // accompanied by a revision field which contains a positive
            // integer. If versions specified by arrays of different length need
            // to be compared, the shorter array gets padded with zeroes on the
            // right. This means that for example version 1.3 could be given as
            // both [1, 3] and [1, 3, 0, 0] (aka 1.3.0.0) and either would mean
            // the same.
            // Different versions (e.g. 1.0 and 1.3) of the same resource can be
            // defined in separate index.json files. This makes it easy to
            // accidently cause an identifier clash. To help detect it, we
            // require that each resource has a uuid associated with it. Attempt
            // to define multiple resources with the same identifier and
            // different uuids will result in an error being reported. Defining
            // multiple resources with different identifiers and the same uuid
            // is disallowed for now (it may be later permitted if we consider
            // it good for some use-case).
            "uuid": "a6754dcb-58d8-4b7a-a245-24fd7ad4cd68",

            // Version should match the upstream version of the resource (e.g. a
            // version of JavaScript library). Revision number starts as 1 for
            // each new resource version and gets incremented by 1 each time a
            // modification to the packaging of this version is done. Hydrilla
            // will allow multiple definitions of the same resource to load, as
            // long as their versions differ. Thanks to the "version" and
            // "revision" fields, clients will know they have to update certain
            // resource after it has been updated. If multiple definitions of
            // the same version of given resource are provided, an error is
            // generated (even if those definitions differ by revision number).
            "version": [2021, 11, 10],
            "revision": 1,

            // A short, meaningful description of what the resource is and/or
            // what it does.
            "description": "greets an apple",

            // If needed, a "comment" field can be added to provide some
            // additional information.
            // "comment": "this resource something something",

            // Resource's "dependencies" array shall contain names of other
            // resources that (in case of scripts at least) should get evaluated
            // on a page before this resource's own scripts.
            "dependencies": [{"identifier": "hello-message"}],

            // Array of JavaScript files that belong to this resource.
            "scripts": [
                {"file": "hello.js"},
                {"file":   "bye.js"}
            ]
        }, {
            "type":        "resource",
            "identifier":  "hello-message",
            "long_name":   "Hello Message",
            "uuid":        "1ec36229-298c-4b35-8105-c4f2e1b9811e",
            "version":     [2021, 11, 10],
            "revision":    2,
            "description": "define messages for saying hello and bye",
            // If "dependencies" is empty, it can also be omitted.
            // "dependencies": [],
            "scripts": [{"file": "message.js"}]
        }, {
            "type": "mapping",

            // Has similar function to resource's identifier. Should be consize
            // and can only use a restricted set of characters. It has to match:
            // [-0-9a-z]+
            // It can be the same as some resource identifier (those are
            // different entities and are treated separately).
            "identifier": "helloapple",

            // "long name" and "uuid" have the same meaning as in the case of
            // resources. Uuids of a resource and a mapping can technically be
            // the same, but it is recommended to avoid even this kind of
            // repetition.
            "long_name": "Hello Apple",
            "uuid": "54d23bba-472e-42f5-9194-eaa24c0e3ee7",

            // "version" differs from its counterpart in resource in that it has
            // no accompanying revision number.
            "version": [2021, 11, 10],

            // A short, meaningful description of what the mapping does.
            "description": "causes apple to get greeted on Hydrillabugs issue tracker",

            // A comment, if necessary.
            // "comment": "blah blah because bleh"

            // The "payloads" object specifies which payloads are to be applied
            // to which URLs.
            "payloads": {
                // Each key should be a valid Haketilo URL pattern.
                "https://hydrillabugs.koszko.org/***": {
                    // Should be the name of an existing resource. The resource
                    // may, but doesn't have to, be defined in the same
                    // index.json file.
                    "identifier": "helloapple"
                },
                // More associations may follow.
                "https://hachettebugs.koszko.org/***": {
                    "identifier": "helloapple"
                }
            }
        }
    ],
    // We can also list additional files to include in the produced source
    // archive. Hydrilla builder will then include those together with all
    // script and copyright files used.
    "additional_files": [
        {"file": "README.txt"},
        {"file": "README.txt.license"},
        {"file": ".reuse/dep5"}
    ],
    // We can optionally tell Hydrilla builder to run the REUSE tool to generate
    // report.spdx file. Using this option requires REUSE to be installed and
    // and importable in the Python virtualenv used by Hydrilla builder.
    "reuse_generate_spdx_report": true
}
```

JSON schema

Valid index.json file should conform to the schema available here.

Repository API

Note: you can try using a live Hydrilla instance at https://hydrilla.koszko.org/api_v2*; try pasting this into the URL bar:* https://hydrilla.koszko.org/api_v2/query?url=https://opencores.org/projects

Fetching resource info

"resource" is now the equivalent of what we used to call "bag" in Haketilo 0.1.

Let's assume we want to retrieve the definition of resource helloapple. We can issue a GET to https://hydrilla.example.com/resource/helloapple.json. It returns:

```
{
    "$schema": "https://hydrilla.koszko.org/schemas/api_resource_description-1.schema.json",
    "source_name": "hello",
    "source_copyright": [
        {
            "file": "report.spdx",
            "sha256": "249599a36ad9f6f33b8bf16ca6ace2413d95d8cdb481173968fa4e97fd0a5635"
        },
        {
            "file": "LICENSES/CC0-1.0.txt",
            "sha256": "a2010f343487d3f7618affe54f789f5487602331c0a8d03f49e9a7c547cf0499"
        }
    ],
    "type": "resource",
    "identifier": "helloapple",
    "long_name": "Hello Apple",
    "uuid": "a6754dcb-58d8-4b7a-a245-24fd7ad4cd68",
    "version": [
        2021,
        11,
        10
    ],
    "revision": 1,
    "description": "greets an apple",
    "dependencies": [
        {
            "identifier": "hello-message"
        }
    ],
    "scripts": [
        {
            "file": "hello.js",
            "sha256": "18dd7d7ac9f74a5ba871791ac6aa4d55530a336ae1b373538b6dd5320b330414"
        },
        {
            "file": "bye.js",
            "sha256": "cee8d88cf5e5346522fd9ee5e1f994b335fb68d2f460b27b2a2a05f0bcd2b55b"
        }
    ]
}
```

or 404 Not Found in case resource helloapple does not exists. If multiple versions are available, newest version's JSON description gets returned. A different one can be retrieved by including version in the GET request. Requested path should then take the form /resource/<resource_name>/<version> (e.g. GET https://hydrilla.example.com/resource/helloapple/2021.11.9).

File can be retrieved by issuing a GET request to https://hydrilla.example.com/file/sha256/<file's_sha256_sum>. For example, requests to https://hydrilla.example.com/file/sha256/a2010f343487d3f7618affe54f789f5487602331c0a8d03f49e9a7c547cf0499 and hydrilla.example.com/file/sha256/18dd7d7ac9f74a5ba871791ac6aa4d55530a336ae1b373538b6dd5320b330414 yield contents of LICENSES/CC0-1.0.txt and hello.js, respectively.)

A JSON schema describing resource definition format is available here.

Fetching mapping info

"mapping" is now a set of what we used to call "pattern" in Haketilo 0.1.

Let's assume we want to retrieve the definition of mapping helloapple (mapping is allowed to have the same identifier as a resource). We can issue a GET to https://hydrilla.example.com/mapping/helloapple.json. It returns:

```
{
    "$schema": "https://hydrilla.koszko.org/schemas/api_mapping_description-2.schema.json",
    "source_name": "hello",
    "source_copyright": [
        {
            "file": "report.spdx",
            "sha256": "249599a36ad9f6f33b8bf16ca6ace2413d95d8cdb481173968fa4e97fd0a5635"
        },
        {
            "file": "LICENSES/CC0-1.0.txt",
            "sha256": "a2010f343487d3f7618affe54f789f5487602331c0a8d03f49e9a7c547cf0499"
        }
    ],
    "type": "mapping",
    "identifier": "helloapple",
    "long_name": "Hello Apple",
    "uuid": "54d23bba-472e-42f5-9194-eaa24c0e3ee7",
    "version": [
        2021,
        11,
        10
    ],
    "description": "causes apple to get greeted on Hydrillabugs issue tracker",
    "payloads": {
        "https://hydrillabugs.koszko.org/***": {
            "identifier": "helloapple"
        },
        "https://hachettebugs.koszko.org/***": {
            "identifier": "helloapple"
        }
    }
}
```

or 404 Not Found in case helloapple mapping does not exist. If multiple versions are available, newest version's JSON description gets returned. A different one can be retrieved by including version in the GET request. Requested path should then take the form /mapping/<mapping_name>/<version> (e.g. GET https://hydrilla.example.com/mapping/helloapple/2021.11.8).

File can be retrieved by issuing a GET request to https://hydrilla.example.com/file/sha256/<file's_sha256_sum>. For example, request to https://hydrilla.example.com/file/sha256/a2010f343487d3f7618affe54f789f5487602331c0a8d03f49e9a7c547cf0499 yields contents of LICENSES/CC0-1.0.txt.

A JSON schema describing mapping definition format is available here.

Querying mappings that match given URL

Assume we want to query mappings with patterns matching https://example.org/a/b. We can issue a GET to https://hydrilla.example.com/query?url=https://example.org/a/b. It returns:

```
{
    "$schema": "https://hydrilla.koszko.org/schemas/api_query_result-2.schema.json",
    "mappings": [
        {
            "identifier": "example-org-minimal",
            "long_name": "Example.org Minimal",
            "version": [
                2022,
                5,
                10
            ]
        },
        {
            "identifier": "example-org-experimental",
            "long_name": "example.org fix bundle",
            "version": [
                0,
                4
            ]
        }
    ]
}
```

The value of "mappings" can be an empty array ([]) in case no mappings match given URL. In case many versions of some mapping match the URL, only the most current of those versions that do is listed. In case the URL is of wrong format, 400 Bad Request is returned.

A JSON schema describing query result format is available here.

Fetching source info

By "source" we mean description of a source package used to build Hydrilla packages.

Assume we want to get description of source package hello. We can issue a GET to https://hydrilla.example.com/source/hello.json. It returns:

```
{
    "$schema": "https://hydrilla.koszko.org/schemas/api_source_description-2.schema.json",
    "source_name": "hello",
    "source_copyright": [
        {
            "file": "report.spdx",
            "sha256": "249599a36ad9f6f33b8bf16ca6ace2413d95d8cdb481173968fa4e97fd0a5635"
        },
        {
            "file": "LICENSES/CC0-1.0.txt",
            "sha256": "a2010f343487d3f7618affe54f789f5487602331c0a8d03f49e9a7c547cf0499"
        }
    ],
    "source_archives": {
        "zip": {
            "sha256": "f61bfc5594a2603452a8c8e5f7a3f36f255bc39d5ffea02c9fdefc0de6461c74"
        }
    },
    "upstream_url": "https://git.koszko.org/hydrilla-source-package-example",
    "definitions": [
        {
            "type": "resource",
            "identifier": "helloapple",
            "long_name": "Hello Apple",
            "version": [
                2021,
                11,
                10
            ]
        },
        {
            "type": "resource",
            "identifier": "hello-message",
            "long_name": "Hello Message",
            "version": [
                2021,
                11,
                10
            ]
        },
        {
            "type": "mapping",
            "identifier": "helloapple",
            "long_name": "Hello Apple",
            "version": [
                2021,
                11,
                10
            ]
        }
    ]
}
```

Or 404 Not Found in case no such source package exists. Here zip was listed among available source archive extensions. It means we can download the zip archive of this source package by issuing GET to https://hydrilla.example.com/source/hello.zip.

With schema version beginning with major number 1, zip archive is always guaranteed to be available.

A [JSON schema](#) describing source package description format is available [here](#).

User manual

This page documents basic usage of Hydrilla. The instructions assume a POSIX shell and a UNIX-like system are being used.

Installation

Using Python wheel

You can install Hydrilla server and(or) builder using .whl files from the Releases[1] page. Please consider verifying the downloads using provided cryptographic signatures.

Installing dependencies

Python3

Hydrilla requires Python interpreter in at least version 3.7. You'd typically use Python3 as provided by you operating system distribution. For example, on Debian-based systems (including Trisquel) you can install it with:

```
sudo apt install python3
```

pip

Pip is the package manager for Python. While not a direct dependency of Hydrilla, it is needed to utilize .whl files. You most likely also want to install pip as provided by your distro, e.g. for APT-based ones:

```
sudo apt install python3-pip
```

Python libraries

Hydrilla relies on the following Python packages:

- jsonschema
- click
- flask (needed for Hydrilla server only)
- reuse (optional, only needed for Hydrilla builder to generate SPDX report)

If you don't have those dependencies installed, pip will automatically pull them from PyPI (except for reuse which would need to be installed separately with a command like python3 -m pip install reuse).

Nevertheless, you are encouraged to instead install the respective packages from your operating system's official repositories because those usually have stricter policies on stability, security and free licensing. In case of APT-based distributions the packages to install would be python3-jsonschema, python3-click, python3-flask and reuse[1].

Installing Hydrilla

Let's assume you want to install version 1.0 of Hydrilla server. First, download and verify both[2] hydrilla.builder-1.0-py3-none-any.whl and hydrilla-1.0-py3-none-any.whl. Then, run:

```
python3 -m pip install
path/to/downloaded/hydrilla.builder-1.0-py3-none-any.whl path/to/downloaded/hydrilla-1.0-py3-none-any.whl
```

This will install Hydrilla **for the current user**. The commands hydrilla and hydrilla-builder will be made available in ~/.local/bin/.

Installing in virtualenv

If for example you don't want pip to install things under ~/.local/, you might choose to create a virtual Python environment. First, make sure you have the virtualenv tool installed[3] (for example from APT package python3-virtualenv). Then, choose the folder in which you'd like to install the environment and run:

```
virtualenv -p python3 --system-site-packages path/to/chosen/folder --no-download
```

The --system-site-packages flag is not strictly necessary for it to work but is needed if you want packages inside the virtual environment to be able to see globally-installed dependencies. The --no-download flag instructs the command not to pull some basic tools like wheel and setuptools from PyPI.

Once the environment is created, you need to enter it by sourcing a script created by the virtualenv command, e.g.:

```
source path/to/chosen/folder/bin/activate
```

Afterwards, the python3 -m pip commands you enter in this shell will install packages inside this virtual environment. You can learn more about Python virtual environments from online tutorials and the virtualenv documentation.

Using APT

Hydrilla APT repository is hosted at https://hydrillarepos.koszko.org/apt2/ and is signed with Wojtek's PGP key (fingerprint **E9727060E3C5637C8A4F4B424BC5221C5A79FD1A**). It is expected to work with modern releases of most APT-based distributions (including Debian bullseye and Trisquel nabia).

This APT repository can be used to install Hydrilla server and builder system-wide and to later update the installation. It has to be said that this also requires you to trust Wojtek's repository with your system's safety (a malicious APT repository could easily take over a system that uses it).

If you've decided you want to install the APT repository on your system, the easiest way to do so is by copy-pasting the following script into your POSIX shell (and then confirming with your password). You can of course modify it according to your needs.

```
__install_hydrilla_apt_repo() {
    local TMP="$1"
    local LISTS="$(cat <<EOF
deb     https://hydrillarepos.koszko.org/apt2/ koszko/
deb-src https://hydrillarepos.koszko.org/apt2/ koszko/
EOF
)"

    if ! wget -O "$TMP/koszko-keyring.gpg"
 https://hydrillarepos.koszko.org/apt2/koszko-keyring.gpg; then
    echo "Error! Failed to download keyring file!" >&2
    return 1
    elif ! gpg --no-default-keyring --keyring "$TMP/koszko-keyring.gpg" --list-key
 E9727060E3C5637C8A4F4B424BC5221C5A79FD1A; then
    echo "Error! Invalid keyring file! Someone might be doing something nasty!" >&2
    return 1
    elif ! sudo cp "$TMP/koszko-keyring.gpg" /etc/apt/trusted.gpg.d/; then
    echo "Error!" >&2
    return 1
    elif ! printf %s "$LISTS" | sudo tee /etc/apt/sources.list.d/hydrillarepos.list > /dev/null;
then
    echo "Error!" >&2
    return 1
    fi

    sudo apt-get update
}

install_hydrilla_apt_repo() {
    local TMP="$(mktemp -d)"
    __install_hydrilla_apt_repo "$TMP"
    local RESULT="$?"
    rm -r "$TMP"

    return "$RESULT"
}

install_hydrilla_apt_repo
```

This snippet is idempotent (i.e. it can be run multiple times and the effect will be as if it was run once). In addition, it executes apt-get update command at the end so that your APT is immediately aware of the new repository and its contents.

After installing the repository you can install Hydrilla builder and server using the following commands:

```
sudo apt install python3-hydrilla.builder
```

```
sudo apt install python3-hydrilla # this alone will also pull the builder as a dependency
```

The packages install their modules under /usr/lib/python3/dist-packages/ which is seen by Python3 interpreters installed from APT. The hydrilla and hydrilla-builder commands get placed in /usr/bin/.

In addition, the python3-hydrilla package also includes sample WSGI script and Apache2 config files for Hydrilla under /usr/share/doc/python3-hydrilla/examples/.

Understanding the concepts

Hydrilla serves Haketilo packages through an HTTP interface, as described in Repository API. It takes the package files to serve from a specific directory in the system as configured by the administrator. The package files stored in that directory conform to Hydrilla on-disk data format. Since that format is inconvenient for humans to operate on, there also exists another one - the Hydrilla source package format. One would typically prepare a Haketilo site resource as a source package and then use the hydrilla-builder command to convert it to Hydrilla on-disk format.

Hydrilla builder takes a directory with a source package, processes it and (if no errors are encountered) writes the "built" package files into the requested directory. Somewhat counterintuitively, the "build" does not involve actual compilation of sources nor any similar task (in future versions of Hydrilla all these will be delegated to other software packaging systems like Guix). Rather, the purpose of this step is to save files under the desired names (which involve files' hash sums) and to generate complete JSON definitions of packages being processed.

The serveable directory can be populated by invoking Hydrilla builder multiple times to put the files of different Haketilo packages in it. However, it is also possible to "build" multiple source packages into separate directories and then combine them. It is up to you, as the administrator, to choose how you are going to manage built packages, Hydrilla doesn't impose anything in this regard. Just keep in mind that there is no facility to remove a package from a serveable directory and that brutally deleting one package's files could break the other ones.

Running

Hydrilla builder

Hydrilla software includes a hydrilla-builder command that can be used to convert Hydrilla source package into its serveable form. The command has an associated manpage (also included in the APT package) as well as a --help option.

Assuming you have both Hydrilla builder and git installed, you can clone and "build" a sample Haketilo package with the following:

```
git clone https://git.koszko.org/hydrilla-source-package-example
mkdir /tmp/tmprepo
hydrilla-builder -s ./hydrilla-source-package-example -d /tmp/tmprepo
```

If you then run:

```
find /tmp/tmprepo/
```

you should see the following list of files written by the builder:

```
/tmp/tmprepo/
/tmp/tmprepo/mapping
/tmp/tmprepo/mapping/helloapple
/tmp/tmprepo/mapping/helloapple/2021.11.10
/tmp/tmprepo/resource
/tmp/tmprepo/resource/hello-message
/tmp/tmprepo/resource/hello-message/2021.11.10
/tmp/tmprepo/resource/helloapple
/tmp/tmprepo/resource/helloapple/2021.11.10
/tmp/tmprepo/file
/tmp/tmprepo/file/sha256
/tmp/tmprepo/file/sha256/cee8d88cf5e5346522fd9ee5e1f994b335fb68d2f460b27b2a2a05f0bcd2b55b
/tmp/tmprepo/file/sha256/a6b9425a80963373d90d8fa22fca07e8626291a4f4bf5f17a4487c16ea1b8dac
/tmp/tmprepo/file/sha256/18dd7d7ac9f74a5ba871791ac6aa4d55530a336ae1b373538b6dd5320b330414
/tmp/tmprepo/file/sha256/a2010f343487d3f7618affe54f789f5487602331c0a8d03f49e9a7c547cf0499
/tmp/tmprepo/file/sha256/fd3d332844be0923b29206b32b9111767d73dd995e3ead7b7f06f72020bce650
/tmp/tmprepo/source
/tmp/tmprepo/source/hello.json
/tmp/tmprepo/source/hello.zip
```

In some cases hydrilla-builder command may fail with a message about the REUSE tool being unavailable. This will only happen when a source package actually requires REUSE (for generation of an SPDX report). The error means that either you don't have it installed or it's somewhere where Hydrilla software cannot see its Python modules.

Hydrilla development server

Hydrilla repository software includes a hydrilla command that can be used to quickly spawn a local repository server. This is unsuitable for deployment of a publicly visible Hydrilla instance but very suitable for testing of both Hydrilla itself and Haketilo packages being developed. The command has an associated manpage (also included in the APT package) as well as a --help option.

For a sample run, you're going to need a directory with some Haketilo packages. You can clone the source package example repository and perform something along the lines of:

```
mkdir /tmp/tmprepo/
hydrilla-builder -s path/to/cloned/hydrilla-source-package-example/ -d /tmp/tmprepo/
```

Then comes a typical invocation of hydrilla command:

```
hydrilla -m /tmp/tmprepo/ -p 0
```

It causes Haketilo packages from /tmp/tmprepo/ directory to be served on a random free port on localhost. Sample run generated this output:

```
 * Serving Flask app "hydrilla.server" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:46485/ (Press CTRL+C to quit)
```

One could then (in another shell) try running some commands like the ones below to confirm that the just-spawned local server is responding:

```
# The following assume that Hydrilla is loaded with the sample Haketilo package
curl http://127.0.0.1:46485/mapping/helloapple.json
curl http://127.0.0.1:46485/resource/hello-message/2021.11.10
curl http://127.0.0.1:46485/query?url=https://hydrillabugs.koszko.org/a/b/c
```

Hydrilla as WSGI application under Apache2

This section describes how to configure an Apache2 virtual host to serve a Hydrilla repository. This guide is mostly meant to be useful to people running their own web servers.

You're going to need:

- root access on the machine[4] (for writing to /etc/apache2/sites-available/ directory)
- Apache2 with mod_wsgi installed and enabled
- Hydrilla installed

First, choose a directory where you want to store your serveable Haketilo packages. The default is /var/lib/hydrilla/malcontent. You can override this by saving the following file as /etc/hydrilla/config.json:

```
{
    // Path to directory from which Hydrilla will load packages metadata and serve files.
    "malcontent_dir": "/your/chosen/dir"
}
```

Fill the directory with some package files. You might for example clone the source package example repository and build it with something along the lines of:

```
sudo hydrilla-builder -s path/to/cloned/hydrilla-source-package-example/ -d
 /var/lib/hydrilla/malcontent/
```

Once done, grab Hydrilla's sample WSGI script and save it in your chosen location (the suggested one is /var/lib/hydrilla/wsgi/hydrilla.wsgi). Follow the comments in this script to modify it according to your needs.

Now, get the sample Apache2 configuration (there is also one for TLS-less deployment), also modify it according to your needs (in particular, you'll likely want to change hydrilla.example.com to some real domain of yours) and save under /etc/apache2/sites-available/your.chosen.config.name.conf.

You can now enable the configuration with:

```
sudo a2ensite your.chosen.config.name
```

You also need to reload or restart the Apache daemon for the configuration to be picked up (the command to do that varies between init systems). Once you do so, you can verify that the server is running properly. Consider running something like the following (replacing hydrilla.example.com with the domain name you used):

```
# The following assume that Hydrilla is loaded with the sample Haketilo package
curl http://hydrilla.example.com/mapping/helloapple.json
# -v flag will let us verify that the "Content-Type: application/json" header is present
curl -v http://hydrilla.example.com/resource/hello-message/2021.11.10
curl -v http://hydrilla.example.com/query?url=https://hydrillabugs.koszko.org/a/b/c
```

If everything is working as expected (i.e. JSON documents are properly served by Hydrilla&Apache2), you can start populating the "malcontent directory" with built packages of your choice.

---

1. Reuse tool was first packaged for Debian Bookworm and is not yet available in Debian Bullseye nor in Trisquel Nabia. ↩
2. Hydrilla server also depends on Hydrilla builder. ↩
3. Since Python 3.3 a virtual environment can also be created without this tool. ↩
4. If you want to run a Hydrilla server on shared hosting without root access, this might be achievable using a .htaccess file but is not documented right now. ↩