

# progszy

---

[TODO Github badges go here - as per my other repos]

progszy is a hard-caching HTTP(S) proxy server (with programmatic cache management), designed for use as part of a data-scraping pipeline.

It is both a standalone CLI tool, and a Go package.

It is **not** suitable for use as a regular HTTP(S) caching proxy, e.g. with web browsers.

progszy should work with any client, but currently has only been tested with Go's `http.Client`.

## Caching

---

Cached content is persisted in an [SQLite](#) database, using [Zstandard](#) compression, enabling cached content to be retrieved [faster](#) than regular file system reads, while also providing convenient packaging of cached content and saving storage space.

A separate single-file database is created to cache the content of each domain (that is: content is 'binned' according to the domain's base/root name), and database filenames also contain a creation timestamp.

## Caching Strategy

progszy intentionally makes no use of HTTP headers relating to cached content control that are normally utilised by browsers and other caching proxies.

The body content and appropriate headers for all `200 ok` responses are hard-cached — unless the body matches a given filter, see below.

Content exceeding an arbitrary maximum body size of 64mb is not cached, and instead returns a `412 Precondition Failed`. We may review this decision/behaviour at a later date.

Cache eviction/management is manual-only at present. Later we will add a REST API for programmatic cache management.

## HTTP(S) Proxy

---

As a standalone server, it serves as a standard HTTP proxy server, on port 5995 by default, for which the client configuration URL would be `http://127.0.0.1:5995`.

TODO 127.0.0.1 or 0.0.0.0 or localhost ?

Incoming request URLs can be either vanilla HTTP, or can be HTTPS (using CONNECT protocol). When proxying HTTPS requests, the connection is man-in-the-middle (MITM) to allow both caching and the application of rules, and the resulting outbound stream is then re-encrypted using a private certificate, before being passed to the client. Note that clients wishing to proxy

HTTPS requests will need specific configuration to prevent certificate mismatch errors.

Outgoing HTTP requests utilise automatic retries with exponential backoff. Internal HTTP clients use a shared transport with pooling.

## HTTP Headers

progszy makes use of custom HTTP `x-*` headers to both control features and report status to the client.

### Request Headers

- `X-Cache-Reject` headers control early rejection/filtering of incoming content. Each header value is compiled into a regexp reject rule: if the content body matches any filter, the request response is not cached, and instead a `412 Precondition Failed` is returned to the client. See tests for example usage. Note that cache hits (requests for already cached content) are not currently affected by the use of this header.
- `X-Cache-SSL: INSECURE` forces use of a client that skips SSL certificate validation during the upstream/outbound request. See tests for example usage.

Incoming `x-*` headers are not copied to outgoing requests.

### Response Headers

- `x-cache` value will be `HIT` or `MISS` accordingly.
- `Content-Type`, `Content-Language`, `ETag` and `Last-Modified` headers on incoming responses all have their value persisted to the cache, and restored appropriately on outgoing responses to the client.

## Installation

---

TODO installation instructions

TODO installation instruction for binary and build?

## Go Dependencies

For reference, progszy makes use of the following packages:

- SQLite driver <https://github.com/mattn/go-sqlite3>
- Zstd wrapper <https://github.com/DataDog/zstd>
- goproxy <https://github.com/elazarl/goproxy>
- retryablehttp <https://github.com/hashicorp/go-retryablehttp>
- cleanhttp <https://github.com/hashicorp/go-cleanhttp>
- publicsuffix <https://github.com/weppos/publicsuffix-go>
- Standard library.
- [Ginkgo](#) and [Gomega](#) are used in the tests.

## Usage Examples

---

Once built/installed, progszy can be invoked via the command line, as follows...

Get help / usage instructions:

```
$ ./progszy --help
Usage of ./progszy:
  -cache string
        Cache location (default "./cache")
  -port int
        Port number to listen on (default 5595)
```

Run progszy with default settings:

```
$ ./progszy
Cache location /<path-to-progszy-binary>/cache
Listening on port 5595
```

Run using custom configuration:

```
$ ./progszy -port 8080 -cache ./store
Cache location /<path-to-progszy-binary>/store
Listening on port 8080
```

Press `control` + `c` to halt execution — progszy will attempt to cleanly complete any in-flight connections before shutting down.

## Package Documentation

---

GoDocs <https://godoc.org/github.com/jimsmart/progszy>

## Testing

---

To run the tests execute `go test` inside the project folder.

For a full coverage report, try:

```
$ go test -coverprofile=coverage.out && go tool cover -html=coverage.out
```

## License

---

TODO license BSD or MIT? What do deps use?

## History

---

TODO

- v0.0.1: Initial release.

- 2020-01-01: Work in progress.