

# Pastelyzer

## User manual

Jānis Džeriņš

January 22, 2021

## Contents

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>Introduction</b>                           | <b>2</b> |
| <b>2</b> | <b>Running pastelyzer</b>                     | <b>3</b> |
| 2.1      | Common command-line options . . . . .         | 3        |
| 2.2      | Interactive mode . . . . .                    | 3        |
| 2.3      | Standalone mode . . . . .                     | 5        |
| 2.3.1    | Prerequisites . . . . .                       | 5        |
| 2.3.2    | Options . . . . .                             | 5        |
| 2.3.3    | Environment variables . . . . .               | 5        |
| 2.3.4    | Static web server files . . . . .             | 6        |
| 2.3.5    | Submitting documents . . . . .                | 6        |
| 2.4      | Reprocessing documents . . . . .              | 7        |
| <b>3</b> | <b>Configuration</b>                          | <b>7</b> |
| 3.1      | Syntax overview . . . . .                     | 8        |
| 3.2      | User sets . . . . .                           | 8        |
| 3.2.1    | super-domains . . . . .                       | 9        |
| 3.2.2    | ipv4-networks . . . . .                       | 9        |
| 3.2.3    | cc-bins . . . . .                             | 10       |
| 3.3      | Sinks . . . . .                               | 11       |
| 3.3.1    | Specifying values . . . . .                   | 11       |
| 3.3.2    | Templates . . . . .                           | 12       |
| 3.3.3    | Document field extractors . . . . .           | 12       |
| 3.3.4    | Artefact field extractors . . . . .           | 13       |
| 3.3.5    | Extractors valid in process filters . . . . . | 13       |
| 3.3.6    | WEB-SINK . . . . .                            | 14       |

|          |                                 |           |
|----------|---------------------------------|-----------|
| 3.3.7    | SMTP-SINK . . . . .             | 14        |
| 3.3.8    | MISP-SINK . . . . .             | 14        |
| 3.3.9    | CMD-SINK . . . . .              | 15        |
| 3.4      | Filters . . . . .               | 17        |
| 3.4.1    | Process filters . . . . .       | 19        |
| 3.5      | Artefact export . . . . .       | 19        |
| 3.6      | Example configuration . . . . . | 19        |
| <b>4</b> | <b>Artefact classes</b>         | <b>23</b> |
| 4.1      | Binary artefacts . . . . .      | 24        |
| 4.1.1    | COMPRESSED-BLOB . . . . .       | 24        |
| 4.1.2    | ENCODED-STRING . . . . .        | 24        |
| 4.2      | String artefacts . . . . .      | 24        |
| 4.2.1    | BANK-CARD-NUMBER . . . . .      | 24        |
| 4.2.2    | CREDENTIAL . . . . .            | 25        |
| 4.2.3    | DOMAIN . . . . .                | 25        |
| 4.2.4    | ONION . . . . .                 | 25        |
| 4.2.5    | EMAIL . . . . .                 | 25        |
| 4.2.6    | EMBEDDED-BINARY . . . . .       | 25        |
| 4.2.7    | IP-ADDRESS . . . . .            | 26        |
| 4.2.8    | IP-SERVICE . . . . .            | 26        |
| 4.2.9    | RESOLVED-IP-ADDRESS . . . . .   | 26        |
| 4.2.10   | M3U-ENTRY . . . . .             | 26        |
| 4.2.11   | URI . . . . .                   | 26        |
| 4.2.12   | WINDOWS-INTERNAL . . . . .      | 26        |
| <b>5</b> | <b>API</b>                      | <b>27</b> |
| 5.1      | Generic value search . . . . .  | 28        |
| 5.2      | Typed search . . . . .          | 29        |
| 5.3      | Document information . . . . .  | 31        |

## 1 Introduction

pastelyzer is a tool that analyses text documents and looks for security and privacy related “items” (called *artefacts* in this document). A distinguishing feature of the tool is that embedded binary content (e.g., base64-encoded) is detected and decoded. The decoded content is then recursively processed (see section 4.1).

**IMPORTANT:** This software is a work-in-progress, and is expected to change. Make sure you consult the documentation that corresponds to the version of `pastelyzer` you're using.

## 2 Running `pastelyzer`

The simplest way to use `pastelyzer` is on the command-line and to pass it one or more file names. `pastelyzer` can also be run as a background service. These modes and applicable command-line options are described in the sections below.

### 2.1 Common command-line options

**-h, --help** Show short usage information.

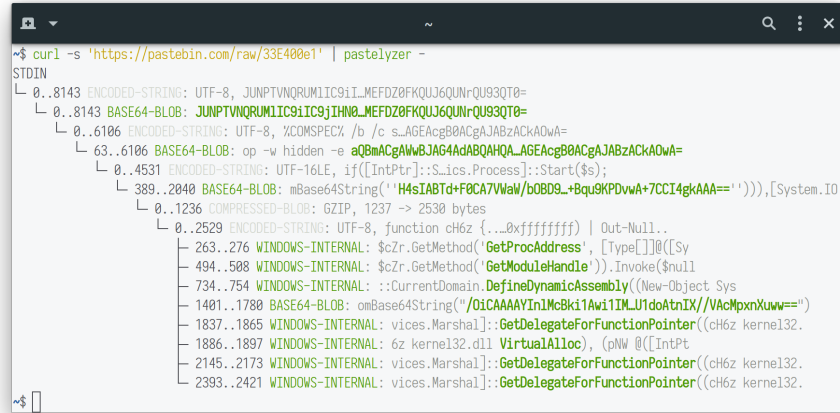
**-v, --version** Show version.

The following options affect artefact extraction and apply in all modes:

**--[no-]resolve-domains** If `--resolve-domains` option is present, domain names are resolved to IP addresses. If `--no-resolve-domains` is present, domains are not resolved (default).

### 2.2 Interactive mode

Unless specified otherwise `pastelyzer` operates in an interactive mode and processes documents specified on the command line, showing all detected artefacts in a tree structure (see figure below).



```

~$ curl -s 'https://pastebin.com/raw/33E400e1' | pastelyzer -
STDIN
├─ 0..8143 ENCODED-STRING: UTF-8, JUNPTVNQRUMLIC9iI_MEFDZ0FKQJ36QUNrQU93QT0=
│   └─ 0..8143 BASE64-BLOB: JUNPTVNQRUMLIC9iI_MEFDZ0FKQJ36QUNrQU93QT0=
│       └─ 63..6106 ENCODED-STRING: UTF-8, %COMSPEC /b /c s..AGEAcgB0ACgAJABzACKA0wA=
│           └─ 0..4531 ENCODED-STRING: UTF-16LE, if([IntPtr]::Sics.Process)::Start($s);
│               └─ 389..2040 BASE64-BLOB: mBase64String('H4sIABTd+F0CA7VWwW/b0BD9L+8qu9KPDvva+7CCI4gkAAA=='))', [System.IO
│                   └─ 0..1236 COMPRESSED-BLOB: GZIP, 1237 -> 2530 bytes
│                       └─ 0..2529 ENCODED-STRING: UTF-8, function ch6z {...0xfffff} | Out-Null..
│                           └─ 263..276 WINDOWS-INTERNAL: $Zr.GetMethod('GetProcAddress', [Type[]]0([Sy
│                               └─ 494..508 WINDOWS-INTERNAL: $Zr.GetMethod('GetModuleHandle')).Invoke($null
│                                   └─ 734..754 WINDOWS-INTERNAL: ::CurrentDomain.DefineDynamicAssembly((New-Object Sys
│                                       └─ 1401..1780 BASE64-BLOB: omBase64String("/OICAAAYInIMcBki1Aw1IM_UtdoAtnIX//VAcMpxnXuwv=="')
│                                           └─ 1837..1865 WINDOWS-INTERNAL: vices.Marshal]:GetDelegateForFunctionPointer((ch6z kernel32.
│                                               └─ 1886..1897 WINDOWS-INTERNAL: 6z kernel32.dll VirtualAlloc), (pNW 0([IntPtr
│                                                   └─ 2145..2173 WINDOWS-INTERNAL: vices.Marshal]:GetDelegateForFunctionPointer((ch6z kernel32.
│                                                       └─ 2393..2421 WINDOWS-INTERNAL: vices.Marshal]:GetDelegateForFunctionPointer((ch6z kernel32.

```

Figure 1: Sample CLI usage

Each line in the tree structure contains the following information:

- Start and end character position of the artefact in the parent object.
- Number of additional instances of this artefact (if there are any, and if the `--show-duplicates` option is not given) in the form of `(+N)`, where `N` is the number of additional occurrences.
- Artefact class.
- Short description of the artefact (depends on the class).

The following options apply to the interactive mode:

- `-C, --color` Enable colors in output (default if run interactively in a terminal).
- `+C, --no-color` Disable colors in output.
- `--extract-embedded` Embedded binary artefacts are extracted and saved in the current directory.
- `--show-duplicates` Do not detect duplicate artefacts. In this case all artefacts and their locations will be shown.

## 2.3 Standalone mode

Standalone mode is selected by passing **server** as a command-line option. In this mode **pastelyzer** receives documents from a feed<sup>1</sup>, stores them in database and analyzes them. It is possible to control what to do with the discovered artefacts using configuration. A sample **systemd** service file is provided in [support/pastelyzer.service](#) file.

### 2.3.1 Prerequisites

**pastelyzer** stores all received documents in a PostgreSQL (version **9.5** or later) database. Database connection parameters are set using environment variables (see section 2.3.3). At the moment schema updates are managed by the application itself. This means that all tables will be created automatically whenever **pastelyzer** is run in standalone mode.

### 2.3.2 Options

**-c, --config** Path to configuration file (see section 3).

**-w, --workers** Number of threads to use for document processing (default: 4).

**--server-port** Port number where to expose the built-in web server. If the port is not given then the web server is not started and it will not be possible to interact with the application using HTTP protocol.

**--server-ext-host, --server-ext-port** Web server host and port to use when generating links to documents (see **local-url** extractor). If not specified, links of the form `http://localhost/...` will be generated.

**--[no-]process-broken** Process pastes with broken UTF-8 content<sup>2</sup> (in addition to trying to fix them; defaults to yes). Specific sites can be blacklisted using an environment variable (see **IGNORED\_PASTESITES**).

### 2.3.3 Environment variables

**DB\_NAME** Database name. Default: **pastelyzer**.

**DB\_USER** Database user. Default: **pastelyzer**.

---

<sup>1</sup>Currently only CIRCL.LU AIL paste feed is supported.

<sup>2</sup>This will not be necessary once the CIRCL.LU AIL paste feed is fixed.

**DB\_PASS** Database password. Default: empty.

**DB\_HOST** Database host. Default: localhost.

**DB\_PORT** Database port. Default: 5432.

**CIRCL\_ZMQ\_ADDRESS** Paste feed endpoint<sup>3</sup>. Default: `tcp://crf.circl.lu:5556`.  
Can be set to an empty value to not connect to the feed.

**IGNORED\_PASTESITES** Comma-separated list of paste sites to not re-fetch broken<sup>4</sup> pastes from. It is a good idea to have at least `pastebin.com` in this list<sup>5</sup>.

**HTTP\_USER\_AGENT** User agent to use when fetching web pages. This will be used verbatim as the HTTP User-Agent header value.

### 2.3.4 Static web server files

The public directory in the source distribution is expected to be located in the “working directory” (or “current directory”) of the `pastelyzer` process. It is safe to make this directory read-only, or keep it on a read-only file system because `pastelyzer` does not write to any files.

### 2.3.5 Submitting documents

If the web server is started (see `--server-port` option) it is also possible to submit documents using HTTP POST requests. The following POST parameters are expected:

**data** Required. The document content.

**source** Optional. String used to identify document source. `store` is used if not provided. This is used as the value for `provider` column in the `pastes` table.

---

<sup>3</sup>You need to contact CIRCL.LU support to whitelist the external IP address of the machine running `pastelyzer`.

<sup>4</sup>This only applies to documents received from CIRCL.LU feed, and broken means that all non-ASCII characters in source document have been replaced by “REPLACEMENT CHARACTER” (U+FFFD), and therefore documents with non-ASCII characters and binary documents are corrupted.

<sup>5</sup>`pastelyzer` rate-limits requests to paste sites, currently one request per 8+ seconds, but in our experience this is still more than `pastebin.com` is willing to allow.

**id** Optional. Identifier of the document. If not provided the file name from the post data is used. This is used as the value for `provider_id` column in the `pastes` table.

Example:

```
curl 'http://localhost:8080/store' -F data=@/path/to/a/file
```

## 2.4 Reprocessing documents

There are a number of circumstances when the documents that are already in the database might need to be reprocessed. These circumstances include, but are not limited to:

- Newer `pastelyzer` versions might process documents differently and extract different set of artefacts (hopefully more or higher quality).
- User configuration affects the number and kinds of artefacts that are extracted.
- Documents have been inserted into the database manually.

The `reprocess` mode is very similar to the `server` mode described above, except instead of connecting to a feed the documents already present in the database (and not processed by the current version of `pastelyzer`) are processed. The process exits as soon as there are no documents to process.

**Important:** if `pastelyzer` configuration contains rules to send emails or interact with other systems/programs then the reprocessing might trigger these rules again. It is therefore advisable to create a separate configuration file for reprocessing and either omit these rules or change them so that the information does not go to production systems.

## 3 Configuration

Currently configuration file is used only when running in standalone (or `reprocess`) mode. By default received documents are stored in the database, processed, and extracted artefacts are stored in the database. Sinks are additional destinations where artefacts can be collected, and filters are used to control what artefacts end up in which sinks.

Current sink and filter implementation is good enough to support basic use cases. The implementation is expected to change in the future to accommodate additional use cases.

### 3.1 Syntax overview

Configuration is written using [S-expressions](#). The most unusual thing about this notation might be the prefix notation, but it can also be thought of as XML simplified (i.e., no need for closing tags). Please refer to [Wikipedia page](#) for a more verbose introduction.

Syntax used in `pastelyzer` configuration file have an extension: square brackets can be used to specify byte-vectors. The elements between opening and closing square brackets are expected to be space-separated hexadecimal 8-bit numbers (case-insensitive). Example: `[7f 45 4c 46]`.

The following configuration directives are recognized (described in following sections): `define-set`, `define-sink` and `define-artefact-filter`<sup>6</sup>. These directives, as well as other *symbolic* identifiers are **case insensitive**.

### 3.2 User sets

`define-set` defines a set of values that can then be used in filters. As an example consider a list of valid TLDs (top-level domains) that can be applied to filter domain-looking strings. The syntax of the directive is like this:

```
(define-set <name> (<type>) <attribute>*)
```

`<name>` is used in other parts of the configuration to refer to this set. `<type>` is one of the built-in set types (described below). Attributes common to all of the sets are:

**:entries** ("entry1" ("entry2" "Note for entry2") ...) Entries of the set provided in the configuration file itself. Mutually exclusive with the `:file` attribute. Each entry in the list can be:

- like `entry1` — a plain string, or
- like `entry2` — a list where first element is the entry and the second element is a note to be attached to the matching artefact (see 3.5).

**:file** "path/to-file" Path to a file where to read the set entries from. Each line of the file is read as a set entry.

---

<sup>6</sup>The word `define` is there to let text editors that have support for Lisp-like languages recognize and treat these expressions specially (i.e., for syntax highlighting and automatic indentation).



**:comment-start <string>** (Only when reading entries from file) line starting with the provided string (defaults to "#") are treated as comment lines and are not included as entries in the set. For now only a single comment start string can be provided.

**:attach-comments <bool>** When true (default is false) comments are attached to all artefacts matching the following entries as notes (see 3.5). An empty comment or an empty line can be used to clear the comment for the following entries. Only single-line comments are used as notes and each consecutive comment line will override the comment set before.

**:trim-space <bool>** (Only when reading entries from file) when true (default) leading and trailing white-space characters are removed from each line read (including comment lines).

Descriptions of defined set types follow.

### 3.2.1 super-domains

Each entry is a “super-domain” and can be used for sub-domain membership test. As an example listing 1 shows configuration to mark **domain** artefacts matching your organization as important (see filters below):

```
(define-set our-org (super-domains)
  :entries ("our.org" "our-org.com"))

(define-filter our-domain
  (and (type? domain)
        (member? our-org))
  (set-important))
```

Listing 1: Sample super-domain set

**Note:** super-domain membership tests are case-insensitive!

### 3.2.2 ipv4-networks

Each entry specifies an IPv4 network (in [CIDR notation](#)). For example if you are not interested in local IP addresses you could use the configuration shown in listing 2:

```
(define-set private-networks (ipv4-networks)
  :entries ("10.0.0.0/8" "172.16.0.0/12" "192.168.0.0/16"))

(define-filter private-ip
  (and (type? ip-address)
    (member? private-networks))
  (discard "Private network"))
```

Listing 2: Sample IPv4 networks set

### 3.2.3 cc-bins

Each entry specifies a bank card [BIN](#). The format is: digit characters followed by number placeholder characters. The number of digit and placeholder characters should match the number of bank card digits corresponding to the bin (e.g., 16 for Visa and Mastercard). Spaces are ignored (in both prefix and placeholder parts).

Listing 3 defines 5 bins for 2 banks. The first bank uses 16-digit numbers that start with 123456, 424242 and 111122. The second bank uses 15-digit numbers that start with 23 and 32. A set with the entries from this file can then be defined as shown in listing 4 (:attach-comments option defaults to true when reading bank card BINs).

```
# This is just an example. The note before each block will be
# attached to all following entries.

# National bank
1234 56xx xxxx xxxx
424242 xx xxxx xxxx
111122xxxxxxxxxxxx

# Other bank
23xx xxxxxx xxxxx
32xxxxxxxxxxxxxxxx
```

Listing 3: Sample bank card BIN file

The same bins can also be defined in-line, as shown in listing 5 (notice that the note has to be specified for each entry individually):

```
(define-set special-bins (cc-bins)
  :file "path/to/above/file.ext")
```

Listing 4: Using sample bank card BIN file

```
(define-set special-bins (cc-bins)
  :entries (("1234 56xx xxxx xxxx" "National bank")
            ("424242 xx xxxx xxxx" "National bank")
            ("111122xxxxxxxxxx" "National bank")
            ("23xx xxxxxx xxxxx" "Other bank")
            ("32xxxxxxxxxxxxxx" "Other bank")))
```

Listing 5: Sample in-line bank card BINs

### 3.3 Sinks

`define-sink` directive is used to define a sink. The syntax is as follows:

```
(define-sink <name> (<parent>) <attribute>*)
```

`<name>` is the name of sink being defined and will be used to refer to it in filters or as a parent in another sink.

`<parent>` should be either one of the built-in sink implementations (described in the following sections), or another sink that has been defined previously. Even though a sink can only have one parent it has to be specified as a single-element list (i.e., it has to be surrounded by parenthesis).

There can be zero or more `<attribute>` specifications of the following form:

```
(<name> <parameter>*)
```

Attribute `<name>` will usually be a keyword (i.e., a symbol starting with a colon). Each sink implementation has a different set of supported attributes described in corresponding section below.

#### 3.3.1 Specifying values

Attribute values have types. This means that if a string value is required, the value must be enclosed in double quotes (i.e., "this is a string").

Values can also be forms:

**"string"** A string value stands for itself.

**yes, true** A boolean true value.

**no, false** A boolean false value.

**(env <string>)** Returns the value of process environment variable named by given string.

**(file-contents <string>)** Returns the contents of a file named by the given string.

**(or <form>+)** Returns the first form from the given list that has a value. For example, the form `(or (env "SOME_VAR" "undefined"))` will return the value of `SOME_VAR` environment variable, or the string `"undefined"` if there is no such environment variable.

### 3.3.2 Templates

If an attribute is specified to be a template, the value of an attribute will be generated using forms provided in the attribute.

**:nl** New line: outputs a newline character, unconditionally.

**:fl** Fresh line: outputs a newline only if the last character was not a newline.

**(extract <field>)** Extracts `field` from the context variable. See document and artefact field extractors below.

**(fmt <format-string> <form>+)** Formats given parameters using Common-lisp [format](#) function. For advanced users only.

**(unique-count <artefact-class>)** Not really a filter function or an extractor, but can be used in sink attribute templates to calculate the number of unique artefacts of the given `artefact-class` in a document currently being processed. Note: this is a hack, and most probably will change once we come up with a more general way to access this information.

### 3.3.3 Document field extractors

**origin** Source site of the document being processed. E.g., `pastebin.com`.

**remote-url** The URL of the document being processed (if available).

**remote-raw-url** The URL to the raw contents of the document being processed (if available).

**local-url** The URL to the local copy of the document being processed. See `--server-ext-host` and `--server-ext-port` command line options.

**artefact-descriptions** Generates a summary of all artefacts collected in a sink, grouped by artefact class, one per line.

**artefact-summary-by-class** Generates a comma-separated string with artefact class and the number of artefacts with this class collected in the sink.

### 3.3.4 Artefact field extractors

**note** Note associated with an artefact.

**important** Important flag set on an artefact.

**source-string** String representation of the current artefact as it is in source document.

**source-context** The whole text (string) where current artefact has been found, usually the whole document currently being processed (might be embedded).

**context-before** Text preceding current artefact in source document.

**context-after** Text following current artefact in source document.

**bytes** (Applies to `EMBEDDED-BINARY` artefacts only) bytes of an artefact after decoding.

**digits** (Applies to `BANK-CARD-NUMBER` only) digits of a bank card number with no separators.

### 3.3.5 Extractors valid in process filters

The following extractors are only valid in process filters:

**stdout** Standard output of the finished process.

**stderr** Standard error output of the finished process.

**status** Exit status of the finished process.

### 3.3.6 WEB-SINK

This is the simplest sink. It does not have any parameters.

Example:

```
(define-sink dashboard (web-sink))
```

### 3.3.7 SMTP-SINK

This sink is used to send emails. Attributes:

(:server <string>) Sets the outgoing SMTP server to the given string.

(:from <string>) Sets the sender (“From” field) of the outgoing emails.

(:subject <form>+) Template attribute to generate subject line of the outgoing email.

(:body <form>+) Template attribute to generate the body of the outgoing email.

(:recipients <string>+) A list of strings, each being an email address to send the email to.

### 3.3.8 MISP-SINK

(:server <string>) Sets the MISP instance URL.

(:api-key <string>) The API key to use when communicating with the instance.

(:ca-cert <string>) Path to a custom CA certificate.

(:user-cert <string>) Path to PEM-encoded user certificate.

(:user-key <string>) Path to PEM encoded private key matching the certificate given in :user-cert attribute.

(:user-key-pass <string>) Passphrase to use to decrypt the private key specified with :user-key attribute.

(:alert <boolean>) Boolean value specifying whether this MISP event is or is not an alert.

(:publish <boolean>) Boolean value specifying whether to automatically publish this event.

**(:title <form>+)** Template attribute to generate the title of MISP event.

**(:sharing-group <string>)** Currently the only sharing option is a sharing group. Given value should be the name of the sharing group to use (the ID will be automatically looked up using the API).

**(:document-action <form>)** An action to perform once the MISP event is created. Currently the following actions are supported:

**(add-tags <string>+)** Tag the event with the given tags.

**(add-attribute :category <string> :type <string> :value <string> :comment <string>)**  
Add an attribute to the event with the given category, type, value and comment (optional). The context of the action is sink document.

**(:item-action <form>)** An action to perform after the MISP event is created, and all document actions are performed. Each item action is performed once for every artefact collected in the sink. The only supported action is add-attribute:

**(add-attribute :category <string> :type <string> :value <string> :comment <string>)**  
Add an attribute to the event with the given category, type, value and comment (optional). The context of the action is an artefact.

### 3.3.9 CMD-SINK

**(:action <process-filter-name>+)** Specifies one or more process filters to invoke on the result of this command.

**(:environment (<name> <value>)+)** Provides extra environment variables for the external process. The process is invoked in an otherwise empty environment, with the following variables provided by pastelyzer:

**PASTELYZER\_HOME** Directory where pastelyzer itself is running.

**PASTELYZER\_SERVER** URI of the pastelyzer web server if the web server is enabled (i.e., `--server-port` command line option is provided; see command-line options).

**(:stdout :discard|:collect-bytes|:collect-string)**

**(:stderr :discard|:collect-bytes|:collect-string)** What to do with standard output and error output of the executed process. Possible values are:

**:discard** Discard output (same as redirecting to `/dev/null`).  
**:collect-bytes** Collect the output, and treat it as a binary blob.  
**:collect-string** Collect the output, and treat it as a string (in UTF-8 encoding).

**(:stdin)** Value is fed to the standard input of the child process. Strings are converted to bytes using UTF-8 encoding.

**(:command <command> <argument>\*)** Command to execute, followed by zero or more arguments. The command is invoked in a temporary directory. `<command>` specifies the path to the program that is to be executed, as a string. Each `<argument>` can be a string or an expression that yields a string. A very useful expression that works only in `CMD-SINK` is `store-tmpfile` which writes the current context value to a temporary file and returns the path to this file.

**(:target :artefact|:document)** Specifies the context value the external process is invoked on:

**:artefact** Default if unspecified. Invokes the external process with context value being each artefact collected into this sink.  
**:document** Invokes the external process once with context value being the document currently being processed.

The following listing shows an example of invoking YARA on the binary content of artefacts added to the sink:

```
(define-sink invoke-yara (cmd-sink)
  (:command "yara" "-C" "/path/to/your/rules.yarc"
    (-> (extract bytes)
      (store-tmpfile)))
  (:stdout :collect-string)
  (:action maybe-add-note))
```

Listing 6: CMD-SINK example

A more complete example (including the `maybe-add-note` process filter) is included in the example configuration.



### 3.4 Filters

`define-artefact-filter` directive is used to define a filter. The syntax is as follows:

```
(define-artefact-filter <name> <filter-expression> <action>+)
```

`<name>` is the name of the filter. Generally unimportant, but useful for debugging. Whenever an artefact is detected, `<filter-expression>` of every filter is executed. If the result is true (i.e., the filter matches), all `<action>` items are executed in order. Currently the only supported action is to collect the artefact into a sink.

Filter expressions have an implicit argument: an artefact. The following filter expressions are supported:

**(type? <class>)** True if the artefact is of the given class (or any subclass). See section Artefact classes.

**(exact-type? <class>)** Similar to `type?`, but true only if the artefact is exactly of the given class.

**(and <filter-expression>+)** True if all enclosed filter expressions are true, false otherwise.

**(or <filter-expression>+)** True if any of the enclosed filter expressions is true, false otherwise.

**(not <filter-expression>)** False if the given filter expression is true, false otherwise.

**(extract <field>)** Extract given field from the current context value, usually an artefact (see 3.3.4).

**(-> <filter-expression>+)** Replaces the context of each consecutive filter expression with the value of the previous expression. Consider the following example:

```
(and (type? embedded-binary)
      (-> (extract bytes)
           (or (starts-with? [4D 5A])
               (starts-with? [7f 45 4c 46]))))
```

The `and` expression has two enclosed filter expressions. For it to return true both of them have to be true. So if the class of the artefact is not `EMBEDDED-BINARY`, the value of `and` will be false (and the second form will not even be considered). But if the context of the filter is indeed an `EMBEDDED-BINARY` then the value of `and` will be the value of second form: `->`.

First it calls `(extract bytes)` (see 3.3.4), and replaces the current context with returned value. The next form is `or`, and for this form the context is the binary bytes of the current artefact (which we know is an `EMBEDDED-BINARY`). The forms enclosed in `or` are executed in order; if any of them returns true the value of the `or` form is also true. In any case, the value of `->` form will be the value of the `or` form applied to the byte content of the `EMBEDDED-BINARY` artefact.

**(length)** Returns the length of current context value.

**(= <number>|<string>)** True if the current context value is equal to the given value.

**(< <number>|<string>)** True if the current context value is less than the given value.

**(> <number>|<string>)** True if the current context value is greater than the given value.

**(starts-with? <subsequence>)** True if the context value is a sequence that starts with the given subsequence.

**(ends-with? <subsequence>)** True if the context value is a sequence that ends with the given subsequence.

**(contains? <subsequence>)** True if the context value is a sequence that contains the given subsequence.

**(member? <user-set>)** True if the context value belongs to a user set defined earlier using `define-set` directive (see 3.2). If the matching set entry has a comment attached and the context value is an artefact then the comment is automatically stored in the artefact's `note` field.

**(mixed-case?)** True if the context value has both lower and upper case letters.

**(^ [<name>])** Current context value is stored as a variable `<name>`, or just `^` if name is not specified. The stored value can then be used in actions.

If an artefact matches the filter expression all actions are executed on it. The following actions are currently supported:

**(collect-into <sink-name>)** Put the artefact into a previously defined sink.

**(set-important)** Mark artefact as important.

**(set-note <string>|<variable>)** Set artefact's `note` field. The value can be a static string, or a variable name referring to value stored using the `^` expression.

**(discard)** Artefact is discarded. Filters that follow current one in the configuration file will not be applied.

### 3.4.1 Process filters

Process filters are only invoked from command sinks (using the `:action` attribute) and are used to post-process results of external programs. Extra extractors useful in process filters are described in 3.3.5. Syntax is the same as for artefact filters:

```
(define-process-filter <name> <filter-expression> <action>+)
```

## 3.5 Artefact export

After all artefacts are processed the ones that are not discarded in the process are stored in database (`artefacts` table). If the artefact has been marked as important (using `set-important` action), the `important` column is set to `true`. If the `note` field has been set (using `set-note` action or automatically by successful user membership test), it is saved in the `note` column.

Summary of what kinds of artefacts and how many is stored in `analysis` table, `summary` column. The number of discarded artefacts is stored in `discarded` column.

## 3.6 Example configuration

```
(define-sink dashboard (web-sink))
```

```
(define-sink local-misp (misp-sink)
  (:server "https://127.0.0.1:5000/")
  (:api-key (env "MISP_API_KEY"))
  (:ca-cert (or (env "MISP_CA_CERT"))
```

```

        "misp/ca.pem"))
(:user-cert (or (env "MISP_USER_CERT")
                "misp/misp.crt.pem"))
(:user-key (or (env "MISP_USER_KEY")
               "misp/misp.key.pem"))
(:user-key-pass (env "MISP_USER_KEY_PASS")))

(define-sink misp-cc-event (local-misp)
  (:alert yes)
  (:publish yes)
  (:title
   (fmt "~@[~A: ~]~A probable card number~:P"
        (extract origin)
        (unique-count bank-card-number)))
  (:sharing-group "Finance")
  (:document-action
   (add-tags "CardFraud" "tlp:amber"))
  (:document-action
   (add-attribute :category "External analysis"
                  :type "url"
                  :value (extract origin)))
  (:item-action
   (add-attribute :category "Financial fraud"
                  :type "cc-number"
                  :value (extract digits)
                  :comment (extract note))))

(define-sink email (smtp-sink)
  (:server "smtp.your.org")
  (:from "pastelyzer@your.org")
  (:subject (extract artefact-summary-by-class))
  (:body "URL: " (extract local-url) :fl
         "Origin: " (extract remote-url) :fl
         "Origin (raw): " (extract remote-raw-url) :fl
         (extract artefact-descriptions))
  (:recipients "pastelyzer-notifications@your.org"))

(define-process-filter maybe-add-note
  (-> (extract stdout)
      (^ output)

```

```

        (not (or (= "") (= "data"))))
    (set-note output))

(define-sink run-file (cmd-sink)
  (:command "file" "-b" "-")
  (:stdin (extract bytes))
  (:stdout :collect-string)
  (:action maybe-add-note))

(define-set important-ccs (cc-bins)
  :file "path/to/BIN-file")

(define-artefact-filter important-cc
  (and (type? bank-card-number)
        (member? important-ccs))
  (collect-into misp-cc-event)
  (collect-into email))

;;; Get this from https://data.iana.org/TLD/tlds-alpha-by-domain.txt
(define-set known-tlds (super-domains)
  :file "tlds-alpha-by-domain.txt")

(define-artefact-filter bad-domain
  (and (type? domain)
        (not (member? tlds)))
  (discard "Unknown TLD"))

(define-set important-tlds (super-domains)
  :entries ("important.gov.tld" "other.gov.tld"))

(define-filter important-tld
  (and (type? domain)
        (member? important-tlds))
  (set-important))

(define-artefact-filter m3u
  (type? m3u-entry)
  (discard "M3U entry"))

(define-artefact-filter png

```

```

    (and (type? embedded-binary)
         (-> (extract bytes)
              (starts-with? [89 50 4E 47])))
    (discard "PNG image"))

(define-artefact-filter small-binary
  (and (type? embedded-binary)
       (-> (extract bytes)
            (length)
            (< 500)))
  (discard "Small binary"))

(define-artefact-filter .exe
  (and (type? embedded-binary)
       (-> (extract-bytes)
            (starts-with? [4D 5A])))
  (collect-into run-file))

(define-artefact-filter keybase-proof
  (and (type? base64-blob)
       (-> (extract source-context)
            (or (starts-with? "### Keybase proof")
                (starts-with? "Keybase proof"))))
  (discard "Keybase proof"))

(define-artefact-filter html-inline-blob
  (and (type? base64-blob)
       (-> (extract context-before)
            (ends-with? ";base64,")))
  (discard "HTML inline blob"))

(define-artefact-filter ignore-ordinary-urls
  (and (type? uri)
       (not (starts-with? "hxxp")))
  (discard "Ordinary URL"))

(define-artefact-filter web
  ;; IP addresses are usually not very interesting, but we might be
  ;; interested in open-proxy lists (artefact type IP-SERVICE, which
  ;; is a sub-type of IP-ADDRESS).

```

```
;;
;; Instead of listing everything we want to include we use
;; negation, and list everything we want to exclude.
(not (or (exact-type? ip-address)
         (exact-type? resolved-ip-address)))
(collect-into dashboard))
```

## 4 Artefact classes

The list of artefact classes recognized by `pastelyzer` and their relationships are shown in listing 7. These artefact classes are described in the following sections.

```
ARTEFACT
├─ BINARY-ARTEFACT
│   ├── COMPRESSED-BLOB
│   └─ ENCODED-STRING
└─ STRING-ARTEFACT
    ├── BANK-CARD-NUMBER
    ├── CREDENTIAL
    ├── DOMAIN
    ├── EMAIL
    ├── EMBEDDED-BINARY
    │   ├── BASE64-BLOB
    │   ├── HEX-BLOB
    │   └─ BINARY-BLOB
    ├── IP-ADDRESS
    │   ├── IP-SERVICE
    │   └─ RESOLVED-IP-ADDRESS
    ├── M3U-ENTRY
    ├── ONION
    ├── URI
    └─ WINDOWS-INTERNAL
```

Listing 7: Artefact relationships

## 4.1 Binary artefacts

**BINARY-ARTEFACT** is the superclass for artefacts that are represented as binary data (i.e. sequence of 8-bit bytes). If the input data is not known to be text then it is processed as binary data. In most cases this means that the data will be detected as being text in UTF-8 encoding and processed as text.

### 4.1.1 COMPRESSED-BLOB

Created if the **BINARY-ARTEFACT** bytes can be decompressed using gzip, zlib or deflate methods.

### 4.1.2 ENCODED-STRING

Created if the **BINARY-ARTEFACT** bytes can be decoded as UTF-8 or UTF-16 (little- or big-endian) string.

## 4.2 String artefacts

**STRING-ARTEFACT** is the superclass of all artefacts represented as text (i.e. most of them, described below).

### 4.2.1 BANK-CARD-NUMBER

pastelyzer currently recognizes bank card numbers that correspond to the following patterns:

Table 1: Bank card number patterns

| Pattern             | Description                  |
|---------------------|------------------------------|
| XXXXXXXXXXXXZ       | 14 consecutive digits        |
| XXXXXXXXXXXXXZ      | 15 consecutive digits        |
| XXXXXXXXXXXXXXZ     | 16 consecutive digits        |
| AXXX-XXXX-XXXX-XXXZ | 16 digits in groups of 4     |
| AXXX-XXXXXX-XXXXZ   | 15 digits in groups of 4-6-5 |

In the latter two cases the separator can be single or double space or dash. In all cases the first digit is expected to be non-zero, and the last digit is expected to be the checksum digit (according to [Luhn algorithm](#)).

These rules are very lax and are expected to generate false-positives. This is because the [BINs](#) change, and we want to avoid updating the software when they do. cc-bins user set can be used with bank card numbers.



#### 4.2.2 CREDENTIAL

A username/passphrase pair, separated by colon (:), semicolon (;), vertical bar (|), comma (,) or tab character. At the moment regular expressions are used to match both. Even though the character repertoire for passphrases is quite generous, there are some limitations:

- Only credentials with emails (see [EMAIL](#)) as usernames are detected.
- Non-ASCII characters are not matched.
- Space (and tab) characters are not matched.

We plan to improve credential extraction to work around these limitations in the future.

#### 4.2.3 DOMAIN

Everything that looks like words separated with periods, with some additional restrictions (documented in [RFC1035](#)) and other heuristics.

We strongly suggest to provide the valid TLD file (see the `--tlds-file` option) to filter out most false positives.

#### 4.2.4 ONION

A pseudo-domain consisting of two parts separated by a period. The first part being 16 or 56 alphanumeric characters (consistency not checked), the second being onion (see [Wikipedia entry](#)).

#### 4.2.5 EMAIL

Strings of the form `username@domain` are recognized as emails using a simple regular expression. Only alphanumeric characters plus a period (.), underscore (\_), percent sign (%), plus (+) and minus (-) are considered in the username part. Period-separated sequences of alphanumeric characters are accepted as domain part.

#### 4.2.6 EMBEDDED-BINARY

All embedded binary artefacts (currently `BASE64-BLOB`, `HEX-BLOB` and `BINARY-BLOB`) are specializations of this artefact class. The textual representation is processed and resulting binary data is available as the `BYTES` field (see section 3.3.4).

The bytes are processed, and if it turns out they represent text or compressed data, the corresponding **BINARY-ARTEFACT** instance is created.

**BASE64-BLOB** Base64-encoded (see [Wikipedia](#)) sequence of bytes. Example: TVqQ.

**HEX-BLOB** A sequence of hexadecimal digit pairs, each pair representing one 8-bit byte. Example: 4D5A90.

This artefact is also created for sequences of bytes written using C-syntax for hexadecimal numbers, separated by comma and optional white-space. Example: 0x4d, 0x5a, 0x90.

**BINARY-BLOB** Just ones and zeroes in groups of 8 (i.e., the total number of digits must be a multiple of 8). Example: 010011010101101010010000.

#### 4.2.7 IP-ADDRESS

Artefact class for IPv4 addresses. Only addresses that belong to one of the “interesting networks” are considered “interesting” (see `--networks-file` option).

#### 4.2.8 IP-SERVICE

A specialization of IP-ADDRESS artefact class that is created for IPv4 address:port combination (e.g., 1.2.3.4:8080).

#### 4.2.9 RESOLVED-IP-ADDRESS

A specialization of IP-ADDRESS artefact class for IPv4 addresses that are obtained by resolving domain names using system resolver (see `DOMAIN`).

#### 4.2.10 M3U-ENTRY

Titles of playlist entries (lines starting with `#EXTINF:`).

#### 4.2.11 URI

URIs matched using the syntax provided in [RFC3986](#), [appendix A](#).

#### 4.2.12 WINDOWS-INTERNAL

One of the strings from the following table:

Table 2: Low-level Windows functions

|                               |                      |
|-------------------------------|----------------------|
| CreateProcess                 | NtUnmapViewOfSection |
| CreateRemoteThread            | NtWriteVirtualMemory |
| DefineDynamicAssembly         | PointFunctionCall    |
| GetDelegateForFunctionPointer | QueueUserAPC         |
| GetModuleHandle               | ResumeThread         |
| GetProcAddress                | SetThreadContext     |
| GetThreadContext              | VirtualAlloc         |
| NtAllocateVirtualMemory       | VirtualAllocEx       |
| NtCreateUserProcess           | WriteProcessMemory   |
| NtGetContextThread            |                      |

## 5 API

Besides the already mentioned endpoint to store documents there are two other URLs that can be used to query artefact database, described in the following sections. Requests should be done using HTTP POST method<sup>7</sup>. The values are passed through to database [LIKE](#) operator, therefore wildcard characters % and \_ can be used. It is strongly suggested to not put the wildcard characters on both ends of the query string<sup>8</sup> because that will be slow (instead of using an index a full table scan will have to be performed). As an example, the following patterns are good:

```
%@example.com
192.168.%
```

The following are **NOT** good:

```
%example.com%
%example%
%foo_bar%
```

Artefacts are stored in `artefacts` table, in `value` column. Artefacts that use `extra` column are listed in table 3. Note that the entry for `EMBEDDED-BINARY` applies to its sub-classes (`BASE64-BLOB`, `HEX-BLOB` and `BINARY-BLOB`).

---

<sup>7</sup>To protect the innocent: % is used as a wildcard character and would have to be encoded (using %-encoding) in GET requests.

<sup>8</sup>Of course you may add an index that would help in this case but there is no such index in default pastelyzer database schema.

Table 3: Artefact classes using `extra` column

| Class               | value                  | extra                      |
|---------------------|------------------------|----------------------------|
| CREDENTIAL          | username               | passphrase                 |
| EMBEDDED-BINARY     | SHA1-sum of body bytes | first 4 bytes as hex-pairs |
| IP-SERVICE          | IP address             | port number                |
| RESOLVED-IP-ADDRESS | IP address             | resolved domain            |

By default both value search endpoints limit the number of search results to `500`. A different value can be specified using `limit` parameter. To remove the limit altogether (i.e., return all matches) an empty value must be specified as the `limit` parameter value.

## 5.1 Generic value search

The first query endpoint is `/artefacts` which can be used to search all values stored in `artefacts` table. There are two parameters:

**value** Pattern to search for in the `value` column.

**extra** Pattern to search for in the `extra` column.

Return value is a list of records (JSON array of objects), each record with the following fields:

**type** One of the classes shown in listing 7.

**value** Value of the `value` column.

**extra** Value of the `extra` column, when present.

**important** true if the `important` column has a true value.

**note** Value of the `note` column, when present.

**source** URI of the document where the artefact has been extracted from (takes into account `--server-ext-host` and `--server-ext-port` command-line options).

Example requests:

```
curl 'http://pastelyzer.your.org/artefacts' -F 'value=%example.com' | jq .
```

```
[
  {
    "type": "EMAIL",
    "value": "johndoe@example.com",
    "source": "http://pastelyzer.your.org/content/12345/"
  },
  {
    "type": "DOMAIN",
    "value": "example.com",
    "source": "http://pastelyzer.your.org/content/42/"
  },
  {
    "type": "CREDENTIAL",
    "value": "fred@example.com",
    "source": "http://pastelyzer.your.org/content/54321/",
    "extra": "yabbadabbado"
  }
]
```

```
curl 'http://pastelyzer.your.org/artefacts' -F 'extra=4D5A%' | jq .
```

```
[
  {
    "type": "BASE64-BLOB",
    "value": "3FFBB74B4D9D2915EDD58AA36A65D65AB4296BCD",
    "source": "http://pastelyzer.your.org/content/4242/",
    "extra": "4D5A9000"
  }
]
```

## 5.2 Typed search

The other endpoint is `/artefacts/typed`, where artefacts can be searched by their class as summarized in table 4.

Fields in the result records are named depending on the artefact type. This is nothing more than a straight forward mapping from database columns to record fields, as summarized in the table 5. As with generic search the `important` field is only included if its value is `true`, and `note` field is only included if it has a value.

---

<sup>9</sup>Only the username part of the credential.

Table 4: Searching artefacts by type

| Parameter  | Queried artefact type(s)                    |
|------------|---|
| cc-number  | BANK-CARD-NUMBER                            |
| credential | CREDENTIAL <sup>9</sup>                     |
| domain     | DOMAIN                                      |
| email      | EMAIL                                       |
| ip         | IP-ADDRESS, IP-SERVICE, RESOLVED-IP-ADDRESS |
| onion      | ONION                                       |
| sha1       | BASE64-BLOB, HEX-BLOB, BINARY-BLOB          |
| any        | All types (similar to generic search)       |

Table 5: Typed artefact record fields

| Class               | type       | value    | extra      | More?            |
|---------------------|------------|----------|------------|------------------|
| BANK-CARD-NUMBER    | cc-number  | digits   |            |                  |
| DOMAIN              | domain     | domain   |            |                  |
| ONION               | onion      | address  |            |                  |
| EMAIL               | email      | email    |            |                  |
| CREDENTIAL          | credential | username | passphrase |                  |
| IP-ADDRESS          | ip         | address  |            |                  |
| IP-SERVICE          | service    | address  | port       |                  |
| RESOLVED-IP-ADDRESS | ip         | address  | domain     |                  |
| BASE64-BLOB         | blob       | sha1     | start      | encoding: base64 |
| HEX-BLOB            | blob       | sha1     | start      | encoding: hex    |
| BINARY-BLOB         | blob       | sha1     | start      | encoding: binary |
| other               | CLASS      | value    | extra      |                  |

Example:

```
curl 'http://pastelyzer.your.org/artefacts/typed' \  
-F 'email=jo@example.com' -F 'credential=fr@example.com' | jq .
```

```
[  
  {  
    "type": "email",  
    "email": "johndoe@example.com",  
    "source": "http://pastelyzer.your.org/content/12345/"  
  },  
  {  
    "type": "credential",  
    "username": "fred@example.com",  
    "passphrase": "yabbadabbado",  
    "source": "http://pastelyzer.your.org/content/54321/",  
  }  
]
```

### 5.3 Document information

The URIs to pastelyzer in returned data records (values of `location` and `source` fields) can also be retrieved, using HTTP GET method:

```
curl 'http://pastelyzer.your.org/content/54321/' | jq .
```

The result will look similar to one shown in listing 8.

```

{
  "type": "content",
  "id": 54321,
  "location": "http://localhost:8080/content/54321/",
  "body": "http://localhost:8080/content/54321/body",
  "sources": [
    {
      "type": "paste",
      "id": 55555,
      "location": "http://localhost:8080/paste/55555/",
      "provider": "circl",
      "provider-id": "archive/gist.github.com/2020/06/12/<user>_<hash>.gz",
      "timestamp": "2020-06-12T17:27:02.593430+03:00",
      "urls": [
        "https://gist.github.com/<user>/<hash>"
      ],
      "content": "http://localhost:8080/content/54321/"
    }
  ]
}

```

Listing 8: Document metadata