
ezPAARSE Documentation

Release

ezPAARSE team

Jun 15, 2017

1	Demo Instances	3
1.1	National Demo Instance (Stable)	3
1.2	Preproduction Demo Instance	3
2	Requirements	5
2.1	Libraries	5
2.2	MongoDB	6
3	Installation	7
3.1	Stable version	7
3.2	Development version	7
3.3	Docker and Compose	8
3.4	Uninstall ezPAARSE	8
4	How to Use ezPAARSE	9
4.1	Run the server	9
4.2	Let's Parse !	9
5	Consultation/Access Events	11
5.1	Typical Properties of an Access Event	11
5.2	Resources' identifiers	11
5.3	Resources Types (rtype)	12
5.4	Resources Formats (mime)	12
6	Set your log format	13
6.1	EZProxy syntax	13
6.2	Apache Syntax	14
6.3	Squid Syntax	14
6.4	Personalized parameters	15
7	Knowledge bases	17
7.1	What is a knowledge base?	17
7.2	How is a Knowledge Base used?	17
8	Job reports	19
8.1	General	20
8.2	Rejects	20

8.3	Statistics	20
8.4	Alerts	20
8.5	Notifications	20
8.6	Deduplicating	20
8.7	Files	20
8.8	First consultation event	20
8.9	Unknown Domains	20
9	Updates	21
9.1	Core	21
9.2	Working materials	21
10	Output Fields	23
10.1	Example	23
11	Metadata enrichment	25
11.1	Configuring Crossref Middleware Call	25
11.2	Configuring Sudoc Middleware Call	26
11.3	Configuring HAL Middleware Call	26
11.4	Configuring ISTEEX Middleware Call	26
12	Bot Filtering & Unrelevant Domains	27
12.1	Excluding the Robots' Accesses	27
12.2	Excluding Arbitrary IP Addresses	27
12.3	The Unrelevant Domains	27
13	COUNTER Reports	29
13.1	Parameters (headers)	29
13.2	CLI Usage	29
14	Alerts	31
14.1	Principles	31
14.2	How to know if alerts have been generated?	31
14.3	Available Alerts	31
15	Geolocation	33
15.1	Parameters (headers)	33
16	Double-click deduplication	35
16.1	Parameters (headers)	35
17	Access Events Qualification	37
18	Configuration options	39
18.1	EZPAARSE_ADMIN_MAIL	39
18.2	EZPAARSE_PARENT_URL	39
18.3	EZPAARSE_FEEDBACK_RECIPIENTS	39
18.4	EZPAARSE_SUBSCRIPTION_MAIL	39
18.5	EZPAARSE_MONGO_URL	40
18.6	EZPAARSE_ENV	40
18.7	EZPAARSE_NODEJS_PORT	40
18.8	EZPAARSE_NODEJS_VERSION	40
18.9	EZPAARSE_REQUIRE_AUTH	40
18.10	EZPAARSE_OUTPUT_FIELDS	40
18.11	EZPAARSE_DEMO	41
18.12	EZPAARSE_MIDDLEWARES"	41

18.13	EZPAARSE_QUALIFYING_LEVEL	41
18.14	EZPAARSE_QUALIFYING_FACTORS	41
18.15	EZPAARSE_TMP_CYCLE	42
18.16	EZPAARSE_TMP_LIFETIME	42
18.17	EZPAARSE_IGNORED_DOMAINS	42
18.18	EZPAARSE_GEOLOCALIZE_DEFAULT	42
18.19	EZPAARSE_GEOLOCALIZE_SEPARATOR	42
18.20	EZPAARSE_ALERTS	42
19	Parameters	43
19.1	Content-Encoding	43
19.2	Response-Encoding	43
19.3	Accept	43
19.4	Log-Format-xxx	43
19.5	Date-Format	44
19.6	Crypted-Fields	44
19.7	Output-Fields	44
19.8	Traces-Level	44
19.9	Reject-Files	44
19.10	Double-Click-xxx	45
19.11	Request-Charset	45
19.12	Response-Charset	45
19.13	Max-Parse-Attempts	45
19.14	Clean-Only	45
19.15	Force-Parser	45
19.16	COUNTER-Reports	45
19.17	COUNTER-Format	46
19.18	COUNTER-Customer	46
19.19	COUNTER-Vendor	46
19.20	Geoip	46
19.21	ezPAARSE-Job-Notifications	46
19.22	ezPAARSE-Middlewares	46
19.23	ezPAARSE-Enrich	47
19.24	ezPAARSE-Predefined-Settings	47
19.25	ezPAARSE-Filter-Redirects	47
19.26	ezPAARSE-Filter-Status	47
19.27	Disable-Filters	47
19.28	Force-ECField-Publisher	47
19.29	Extract	47
19.30	Metadata enrichment	48
20	Process API	49
20.1	Sending logs	49
20.2	Access the traces and rejects	51
20.3	General information	51
20.4	Administration	51
21	Administration API	53
21.1	Create an administrator	53
21.2	Get running jobs	53
21.3	Users management	54
21.4	Repositories update	55
22	Makefile	57
22.1	Node modules installation	57

22.2	Mocha tests	57
22.3	Checking coding rules	58
23	Application core	59
23.1	Technologies used by ezPAARSE	59
23.2	How does the ezPAARSE engine work?	59
23.3	ezPAARSE's monitoring	61
23.4	Launching the ezPAARSE's unit tests	61
23.5	Generate a new ezPAARSE version	61
24	Platforms	63
24.1	Prerequisites for the development: git(hub) user's guide	63
24.2	How does a parser work?	64
24.3	Writing a Parser	64
24.4	Testing Parsers	65
24.5	Description of a parser	66
24.6	Vendors' PKBs management principles	66
24.7	Running a specific test	67
25	Middlewares	69
25.1	What is a middleware ?	69
25.2	How to load a middleware ?	69
25.3	Existing Middlewares	69
25.4	How to create a middleware ?	71
26	Ecosystem	73
26.1	platform-init	73
26.2	pkb-cleaner	73
26.3	scrape	74
26.4	loginjector	74
26.5	loganonymizer	74
26.6	logextractor	75
26.7	csvextractor	75
26.8	csvtotalizer	76
26.9	logfaker	77
26.10	pkbvalidator	77
26.11	ecmaker	77
26.12	ecbulkmaker	78
26.13	hostlocalize	79
27	Documentation	81
28	Tree structure	83
29	Website languages	85
30	Changelog	87
30.1	2.9.4	87
30.2	2.9.3	87
30.3	2.9.2	87
30.4	2.9.1	87
30.5	2.9.0	88
30.6	2.8.1	88
30.7	2.8.0	88
30.8	2.7.0	88

30.9	2.6.0	88
30.10	2.5.0	89
30.11	2.4.0	89
30.12	2.3.0	89
30.13	2.2.0	89
30.14	2.1.0	89
30.15	2.0.0	90
30.16	1.9.0	90
30.17	1.8.0	90
30.18	1.7.0	90
30.19	1.6.0	90
30.20	1.5.0	91
30.21	1.4.0	91
30.22	1.3.0	91
30.23	1.2.0	91
30.24	1.1.0	91
30.25	1.0.0	92
30.26	0.9.0	92
30.27	0.8.0	92
30.28	0.7.0	92
30.29	0.6.0	92
30.30	0.5.0	93
30.31	0.4.0	93
30.32	0.3.0	93
30.33	0.2.0	93
30.34	0.1.0	93

ezPAARSE is able to mine, analyse and enrich the locally collected logs generated by reverse proxies (ezProxy, Biblio PAM, Squid, Apache) which record access to academic and scientific publishers' platforms. What you'll find here is the technical documentation for the ezPAARSE software. For a more detailed documentation on the whole project, please consult [the AnalogIST platform](#).

This documentation is organized into a few main sections :

- *Getting started*
- *Essential things to know*
- *Features*
- *Configuration*
- *Developer documentation*

Demo Instances

There are two publicly available instances of ezPAARSE:

- <http://ezparse.couperin.org> (called National Demo Instance, see below)
- <http://ezparse-preprod.couperin.org> (called Preproduction Demo Instance)

Those two instance are for test purposes only (and now identified as such: you'll see a banner indicating "This ezPAARSE instance is available as a demonstration tool") and should therefore not be used as production tools. They are very convenient to test small log files and play around with the software and discover its interface. The data you process is only kept for an hour and is then deleted.

Our advice is to use those demo instances to become familiar with the features and then install a local instance on a server hosted by your institution.

National Demo Instance (Stable)

The instance available on <http://ezparse.couperin.org> is based on the latest stable version of the ezPAARSE software. The list of parsers is regularly updated. This instance is a good mirror of what you'll get if you install ezPAARSE locally.

Preproduction Demo Instance

The instance available on <http://ezparse-preprod.couperin.org> is based on the bleeding edge version of the ezPAARSE software (can be unstable).

This is were we test the latest features (and you can too!).

ezPAARSE can be installed on many Linux distributions, with the following prerequisites :

- Unix standard tools: **bash**, **make**, **grep**, **sed**, etc.
- **python**
- **gcc** and **g++**
- **curl** (used by **nvm**)
- **git** $\geq 1.7.10$ (needed by github)
- **MongoDB** ≥ 2.4

If using the docker version of ezparse you will only need:

- **Docker** (Version ≥ 1.12)
- **Docker Compose** (Version ≥ 1.7)

Libraries

Ubuntu

Starting from a **ubuntu** image loaded in a virtual machine, with root privileges or via **sudo**.

```
apt-get install git make curl python gcc build-essential
```

Fedora

Starting from a **fedora** image loaded in a virtual machine, with root privileges or via **sudo**.

```
yum install tar git make curl python gcc-c++
```

SUSE

(to be verified)

```
zypper install git make curl python gcc-c++
```

Mac OS X

On your Mac, install **Xcode** and **git**

Windows

Windows support has been **discontinued**, however you can use a virtual machine or a Docker container to run ez-PAARSE in a linux environment.

MongoDB

We only document the procedure for Debian and Ubuntu based systems (see below). The installation instructions for other OSes are available in the [official MongoDB documentation](#).

Ubuntu 14.04 or newer

```
sudo apt-get install mongodb
```

Ubuntu 9.10 or older

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen' | sudo
tee /etc/apt/sources.list.d/mongodb.list
sudo apt-get update
sudo apt-get install -y mongodb-org
sudo service mongod start
```

Debian

Please use the [official MongoDB doc](#) for Debian

Make sure you fulfilled the requirements before going any further.

For an ezPAARSE installation on a **Windows** OS, you will have to use a dockerized container, please see below.

Stable version

To install the last stable version on a Unix system, open a console and enter:

```
git clone https://github.com/ezpaarse-project/ezpaarse.git
cd ezpaarse
git checkout `git describe --tags --abbrev=0`
make
```

Video Demonstration

This [screencast](#) demonstrates the previous instructions.

Development version

If you wish to install the development version, open a console and enter:

```
git clone https://github.com/ezpaarse-project/ezpaarse.git
cd ezpaarse
make
```

Docker and Compose

ezPAARSE is available as a [docker image](#).

To run it with docker, you will need to install [Docker](#) and [Docker-Compose](#).

Then, you can either grab the 'docker-compose.yml' file alone and start the containers:

```
mkdir ezpaarse/  
wget https://raw.githubusercontent.com/ezpaarse-project/ezpaarse/master/docker-  
→compose.yml  
docker-compose up -d
```

or simply start the containers from your local github cloned repository:

```
git clone https://github.com/ezpaarse-project/ezpaarse.git  
cd ezpaarse  
docker-compose up -d
```

Uninstall ezPAARSE

Remove the ezpaarse folder:

```
rm -rf ezpaarse
```

Delete the database:

```
mongo ezpaarse  
db.dropDatabase()
```

If using the docker version, delete the Docker containers:

```
docker rm -f ezpaarse ezpaarse_db
```

How to Use ezPAARSE

Run the server

ezPAARSE launches from the command line.

Use the following commands from the installation directory to start and stop the server:

```
make start    # start the server
make stop     # stop the server
make restart  # restart the server
make status   # check the server status
```

Let's Parse !

Visit <http://localhost:59599/> and create the first administrator of your local ezPAARSE instance. Administrators can manage the registered users and trigger updates from the web interface.

Once logged in, try drag-and-dropping a log file on the online form and processing it. If your logs are standard, you should be able to get a result immediately and see what ezPAARSE can produce for you.

Now you're up and ready to use ezPAARSE. Head onto the next section to learn about the basics.

Consultation/Access Events

A consultation event (also known as “access event”) is what ezPAARSE produces when it detects an actual consultation of e-resource in the logs. Each consultation event is generated with some generic data found in the original log line (date, user login, URL of the resource...), and is enriched with various methods.

By default, ezPAARSE produces a CSV output with a limited amount of fields. You can add or remove some fields by using the Output-Fields header. You can also choose a JSON output to get access events with all their properties.

Here is a list of fields that can be found in the consultation events :

Typical Properties of an Access Event

For more information, see all the fields you can request.

Resources’ identifiers

The identifier of a resource allows to characterize the access events associated with it. It can take the values defined in the table below (loaded from [the settings of ezPAARSE](#)). A resource can also be characterized by several identifiers at the same time (eg. a proprietary identifier and an ISBN).

UnitId

The unitid contains the most accurate identifier for a consultation event on a platform (ie. which describes it with the finest granularity). This identifier does not exclude the use of other identifiers. It is used for the deduplication of access events according to the [COUNTER](#) standard in use and provides librarians with useful indicators.

This may be the DOI or a more complex identifier that will spot as precisely as possible what has been consulted (eg. a paragraph of an article or a page of a book).

Resources Types (rtype)

The type of a resource allows to know the nature of a resource and characterize the associated access event. It can take one of the values defined in the table below (loaded from [settings of ezPAARSE](#)).

Resources Formats (mime)

The format of a resource allows to characterize the associated access event. It can take one of the values defined in the table below (loaded from the [settings of ezPAARSE](#)).

Set your log format

ezPAARSE allows its users to specify their proxy log format by using the HTTP header *Log-Format-xxx*, where *xxx* is the model of the proxy (for example: *ezproxy*, *apache*, *bibliopam*).

The different syntaxes mirror those used by the proxy. It is thus often as easy as directly copy-and-pasting the format declared in your proxy configuration. Beware though, **settings are not included in their entirety** on ezPAARSE's side.

EZProxy syntax

- %h: host IP, from where the request originates
- %u: login used during authentication
- %l: distant user name, obtained with identd (always “-”)
- %b: bytes transferred
- %U: requested URL (e.g. *http://www.somedb.com/*).
- %m: request's method (e.g. *GET*, *POST*).
- %r: complete request (e.g. *GET http://www.somedb.com HTTP/1.0*).
- %t: date/time of the request. The format can be specified in the *Date-Format* request.
- %s: HTTP request status code

Regular expressions generated for the EZProxy fields

Each of the above parameters is converted into a regular expression:

- %h (host) : `([a-zA-Z0-9\.\-]+(?:,?[a-zA-Z0-9\.\-]+)*)`
- %u (login) : `([a-zA-Z0-9@\.\-_%=#]+)`
- %l (identd) : `([a-zA-Z0-9\.\-]+)`

- %b (size) : ([0-9]+)
- %U (url) : ([^]+)
- %m (method) : ([A-Z]+)
- %r (url) : [A-Z]+ ([^]+) [^]+
- %t (datetime) : \([([^\]]+)\)
- %s (status) : ([0-9]+)

Apache Syntax

- %h : host IP (from where the request originates)
- %u : login used during authentication
- %l : distant user name, obtained with identd (always “-”)
- %b : bytes transfered
- %U : requested URL (*e.g. http://www.somedb.com/*).
- %r : complete request (*e.g. GET http://www.somedb.com HTTP/1.0*).
- %t : date/time of the request. The format can be specified in the *Date-Format* request.
- %>s : HTTP request status code

Squid Syntax

- %ts : request's timestamp (in seconds).
- %tu : timestamp's milliseconds
- %tr : server's response time
- %tl : date/time of the request. The format can be specified in the *Date-Format* request.
- %>a : host IP, from where the request originates
- %<a : IP address for the last connection
- %<A : domain name from the request
- %lp : port number from the request
- %Ss : squid status for the request (*TCP_MISS, ..*).
- %>Hs : HTTP request status code for the request
- %<st : response size (headers included).
- %rm : request method (*e.g. GET, POST*).
- %rv : protocol version number
- %ru : requested URL (*e.g. http://www.somedb.com/*).
- % [un : login used to authenticate
- %Sh : squid hierarchical status (*DEFAULT_PARENT, ..*).
- %mt : MIME type of the content

- `%ui` : distant user name, obtained with `identd`

Personalized parameters

Using custom settings allows the retrieval of information from the log lines that do not match with EZProxy standards. Conversely, it is possible to ignore some information that you don't want to see appear in the consultation events generated by ezPAARSE.

There are three ways of expressing a parameter:

- `%{field_name}<regexp>`: retrieves the field corresponding to the specified regexp and adds it to the consultation event with the given field name
- `%{field_name}`: retrieves an alphanumeric string (hyphens permitted) and adds it to the consultation event with the specified field name
- `%<regexp>`: ignores the part of the log line that matches the corresponding regexp.

Some examples with regular expressions

The following example would capture a datetime formatted as YYYY/MM/DD:hh:mm:ss

```
%{datetime}<\d{4}/\d{2}/\d{2}:\d{2}:\d{2}>
```

The following example would recognize a tabulation used as a separator between two fields(`%h` and `%u`)

```
%h%<\t>%u
```

NB: you should use non-capturing parentheses (`?:x`) for complex regexps.

Some examples of specific fields

Example of a request

```
curl -X POST --proxy "" --no-buffer -H 'Log-Format-ezproxy: %h %<[-]> %u [%t] "%r" %s  
↪%b' --data-binary @test/dataset/sd.2012-11-30.300.log http://127.0.0.1:59599 -v
```

Some concrete cases

Video Demonstration

This [screencast](#) demonstrate how to use the GUI to discover the log file format in order to let ezPAARSE correctly process your log files

What is a knowledge base?

A **PKB** (read *Publisher Knowledge Base*) is simply an organized list making the link between normalized metadata (eg. ISSN, DOI, journal title, etc.) and proprietary resource identifier(s) used by a vendor.

A **PKB** is composed of one or more **tab separated value** text files that conforms to the **KBART format**

The KBART field named **title_id** represents the vendor's identifier that will be linked to a normalized identifier like **print_identifier** (very often: print ISSN) or **online_identifier**.

The KBART files used by ezPAARSE can contain additional non-KBART fields (that will be prefixed with `pkb-`, like: `pkb-domain`), depending on the richness of the metadata available.

In order to be taken in account by ezPAARSE, the PKB files need to respect the KBART standard file naming pattern: **[ProviderName]_[Region/Consortium]_[PackageName]_[YYYY-MM-DD].txt**, and be located in a folder named after the platform's parser.

For example:

- `cairn/cairn_ebooks_2014-02-13.txt`
- `cairn/cairn_journals_part1_2014-02-13.txt`

Warning : PKB identifiers **must be unique**. If an identifier appears more than once (in one or more PKB files), **only one occurrence** will be taken in account.

How is a Knowledge Base used?

ezPAARSE < 2.1.0

When a resource carrying a vendor identifier `title_id` is met, the associated knowledge base is built from the KBART files and loaded to memory. ezPAARSE can then link the proprietary identifier with all the metadata available and add it to the access event that has been generated.

ezPAARSE since 2.1.0

As knowledge bases are growing and take too much RAM space, ezPAARSE stores them in a mongoDB database to query the metadata associated with the proprietary identifiers. For that purpose, ezPAARSE runs [CastorJS](#) in the background to keep the database and PKB files synchronized. This keeps the memory footprint of ezPAARSE at a minimum, but also requires additional startup time to perform the synchronization, especially on first startup.

Please note that processing logs without waiting for the initial synchronization to be over may result in incomplete enrichment of the access events.

Warning: some ezPAARSE output formats will force you to explicitly ask for some information you are interested in. If you don't, you will only get a minimal set of information as a result. This is the case with the CSV output. Other output formats, like JSON, will automatically return all the data available.

ezPAARSE generates an execution report, everytime it processes a log file. The various sections of this report are documented below.

- *General*: contains general information related to the processing
- *Rejects*: lists all rejects, how much they are and the links to the files containing the rejected lines
- *Statistics*: provides the first global figures
- *Alerts*: lists the active alerts
- *Notifications*: lists the email for the recipients of processing notifications
- *Duplicates*: algorithm used for deduplication
- *File*: list of processed log files
- *First consultation*: content of the first access event

There is also a special file called `domains.miss.csv`, located at the root of the `/ezparse` where unknown domains get stored (deduplicated and sorted). This file persists between every processing job. See *below* for details.

General

Rejects

Statistics

Alerts

Notifications

Deduplicating

Files

First consultation event

Unknown Domains

The `domains.miss.csv` file persists between every processing job. It is where the unknown domains (ie domains for which no parser gets started) get stored, deduplicated and sorted: if URLs present in that file correspond to a provider's platform that should be analysed by ezPAARSE, you have to check on the [Analogist platform analysis website](#) if the platform is already listed and you will also get an indication of how advanced its analysis is.

Core

The following commands update and rebuild the core of ezPAARSE. They also update the working materials (parsers, knowledge bases, scrapers, robots lists and predefined settings).

```
make update          # latest stable version
make update-latest  # latest development version
```

Video Demonstration

This [screencast](#) demonstrates the update command. It is also possible to update ezPAARSE directly from the GUI, see [this screencast](#) to learn more.

Working materials

The following commands update the materials needed by ezPAARSE. There is one distinct command per material type, see comments.

```
make platforms-update  # update parsers, knowledge bases, scrapers
make exclusions-update # update the lists of hosts identified as robots
make resources-update  # update predefined settings, default proxy formats
```


CHAPTER 10

Output Fields

The `Output-Fields` header adds or removes fields to those ezPAARSE returns by default when it generates access events.

By default, the fields returned are those present in the `EZPAARSE_OUTPUT_FIELDS` parameter configuration file (`config.json`) to which the ones already present in the log format triggered for the processing are added.

This parameter can be used to add custom fields (that some parsers would be able to extract and return). For example, the “`btype`” field is not added by default and can be used to trace back advanced information on some database consultations.

You can also add internal ezPAARSE fields as:

- **Datetime**: for the full date (hour, minute and second included) of the consultation event
- **Timestamp**: the date in a computer format for the consultation event

Please note that the personalized fields in the log format will automatically be added to the `Output-Fields` list: there is no need to declare them with this `Output-Fields` header.

The `Output-Fields` header is composed with a list of comma separated fields, each one preceded with a `+` or `-` signs, depending on whether it has to be added or removed.

Example

```
curl -X POST --proxy "" --no-buffer -H 'Output-Fields: -host,-login,+datetime' --data-binary @test/dataset/sd.2012-11-30.300.log http://127.0.0.1:59599 -v
```


Middlewares can be used to enrich access events by querying external APIs. Note that it may slow the process down, as the number of queries is limited over time.

However, the results are temporarily cached in the mongoDB database, to prevent multiple occurrences of a document from generating further requests. The actual number of requests (i.e. excluding cached ones) is available in the report under `general -> <middleware>-queries`. Where `<middleware>` is the name of the middleware involved.

For more details on middlewares, you can read the dedicated section.

By default, ezPAARSE is configured for using 4 enrichment middlewares:

- `istex`
- `crossref`
- `sudoc`
- `hal`

Configuring Crossref Middleware Call

The Crossref middleware uses the DOI found in access events to request metadata using the `node-crossref` module.

Headers

- **crossref-Enrich**: set to `false` to disable crossref enrichment. Enabled by default.
- **crossref-TTL**: lifetime of cached documents, in seconds. Defaults to `7 days (3600 * 24 * 7)`
- **crossref-throttle**: minimum time to wait between queries, in milliseconds. Defaults to `200ms`
- **crossref-paquet-size**: maximum number of identifiers to send for query in a single request. Defaults to `50`
- **crossref-buffer-size**: maximum number of memorised access events before sending a request. Defaults to `1000`

Configuring Sudoc Middleware Call

Headers

- **sudoc-Enrich**: set to `true` to enable Sudoc enrichment. Disabled by default.
- **sudoc-TTL**: lifetime of cached documents, in seconds. Defaults to `7 days (3600 * 24 * 7)`.
- **sudoc-Throttle** : minimum time to wait between queries, in milliseconds. Defaults to 500.

Configuring HAL Middleware Call

The HAL middleware uses the `hal-identifier` found in the access events to request metadata using the `node-hal`

Headers

- **HAL-Enrich**: set to `true` to enable HAL enrichment. Disabled by default.
- **HAL-TTL**: lifetime of cached documents, in seconds. Defaults to `7 days (3600 * 24 * 7)`.
- **HAL-Throttle**: minimum time to wait between queries, in milliseconds. Defaults to 500.

Configuring ISTEEX Middleware Call

The ISTEEX middleware uses the `istex-identifier` found in the access events to request metadata using the `node-istex`

ISTEEX middleware is automatically activated on ISTEEX logs

Headers

- **istex-enrich** : set to `true` to enable ISTEEX enrichment. Disabled by default.
- **istex-ttl** : lifetime of cached documents, in seconds. Defaults to `7 days (3600 * 24 * 7)`.
- **istex-throttle** : minimum time to wait between queries, in milliseconds. Defaults to 500.

Bot Filtering & Unrelevant Domains

ezPAARSE uses [various filters](#) to reduce the noise generated by the many non-pertinent log lines that are present in a typical logfile and that can represent up to 80-90% of the lines.

Excluding the Robots' Accesses

By default, a list of IP addresses is used to recognize and exclude the accesses made by **robots** (*spiders, indexing robots, etc.*)

The log lines generated by such accesses are thus **rejected**.

To complete the robots' list, you just have to create a file and place it in the `exclusions` folder. Its name has to start with `robots.`, and must contain one IP address per line.

Excluding Arbitrary IP Addresses

It is also possible to filter out some distinct IP addresses. The typical use case for this is when a training session takes place and you don't want to count those accesses: you declare the IP addresses for the computers used during the training session. The matched lines are rejected in a different file from the robots rejects.

To complete this list, you just have to create a file in the `exclusions` folder and its name must start with `hosts.`, and contain one IP address per line.

The Unrelevant Domains

Accesses to proxied domains that are of no interest for you can be **ignored** by ezPAARSE. Because they are not relevant to the electronic resources usage you need to trace, they will simply be filtered out and counted as ignored. This can help slim down the rate of rejected lines.

To complete this list of irrelevant domains, you just have to create a file in the `exclusions` folder and name it with the `domains.` prefix. It must contain only one IP address per line.

By default, we already provide you with three exclusion files :

- `domains.default.txt`: containing a list of domains related to Google
- `domains.cdn.txt`: containing a list of Content Delivery Networks (CDN) subdomains
- `domains.static.txt`: containing a list of subdomains serving static resources (mostly images)

COUNTER Reports

ezPAARSE can generate COUNTER reports based on the data collected in its results. To that effect, you use the parameter **COUNTER-Reports** and specify:

- the type of report (JR1 is the only one available for now)
- the output format with the **COUNTER-Format** parameter

Parameters (headers)

- **COUNTER-Reports:** lists the COUNTER reports you want to generate (eg: JR1, BR2). The download links are accessible in the `stats` section from the processing report.
- **COUNTER-Format:** COUNTER reports output format : XML (by default) or TSV.
- **COUNTER-Customer:** client's name and/or email address that will appear in the reports, either name, <email> or name<email>. (ezPAARSE<mail de l'administrateur> by default)
- **COUNTER-Vendor:** vendor's name and/or email address that will appear in the reports, either name, <email> or name<email>. (platform42 by default)

CLI Usage

```
curl -X POST http://localhost:59599 \  
-H "Accept:text/csv" \  
-H "Traces-Level:info" \  
-H "COUNTER-Reports:JR1" \  
-H "COUNTER-Format:csv" \  
-F "files[]=@fede.bibliovie.ezproxy.2014.06.10.log.gz;type=application/x-gzip"````
```

With cURL, the generated report can be downloaded at the URL given in the `**Job-Report-jr1**` header (see below).

```

```

```
## Usage through the online form ##
```

Using the online form, you can ask **for** the creation of COUNTER reports by adding ↳
↳specific headers in the ****Headers (advanced)**** section (see below):

```

```

When the processing of logs is finished, the COUNTER reports can be downloaded via ↳
↳the link(s) in the stats section of the processing report, look **for** the ****url-**
↳**counter-jr1**** line (see below):

```

```

```
## COUNTER reports ##
```

The COUNTER report(s) can be directly used in a spreadsheet program, depending on the ↳
↳specified output format. You can see below a TSV file imported in Excel:

```

```

Principles

When you process log files with ezPAARSE, a number of processing indicators are generated:

- number of log lines read,
- number of rejected log line,
- number of recognized platforms, etc.

Those indicators can be used to detect anomalies during the processing, based on figures considered as “normal”.

How to know if alerts have been generated?

The list of alerts is available in the processing report. If the mail notification is activated, you’ll also receive this list with the email that is sent when the processing has completed.

NB: the activation of the alert system needs a sufficient quantity of relevant log lines. The activation threshold is set in the `config.json` file, with the `activationThreshold` key. It can also be modified with the **Alerts-Activation-Threshold** header.

Available Alerts

Unknown Domains

This alert is generated when a domain frequently appears in the log lines but no associated parser has been found. The appearance rate is calculated with the sum of relevant log lines.

```
appearance_rate = appearance_sum / (total_of_loglines - ignored_lines) * 100
```

The activation threshold is set in the `config.json` file, with the `unknownDomainsRate` key. It can also be modified with the **Alerts-Unknown-Domains-Rate** header.

This alert simply means that ezPAARSE is not able to work for a certain amount of log lines you are providing it with. Most of the time, it is normal behavior because there is a lot more activity in logfiles that ezPAARSE is interested in.

The really important thing that has to be checked (especially when starting to use ezPAARSE) is the content of a file called `domains.miss.csv`. For more details on this file and its content, please refer to the corresponding Report section.

Incomplete Knowledges Bases

Lack of a Knowledge Base

An alert is generated whenever the lack (or absence of) a PKB KBART file prevents the enrichment of access events associated with a proprietary identifier (`title_id`).

Lack of an Identifier

An alert is generated when an identifier is lacking and has been repeatedly requested.

The activation threshold is set in the `config.json` file, with the `titleIdOccurrenceRate` key. It can also be modified with the **Alerts-TitleID-Occurrence-Rate** header.

General Lack

An alert is generated when a large number of searches in a PKB have proved unsuccessful.

The tolerated rate of unsuccessful searches is set in the `config.json` file, with the `pkbFailRate` key. It can also be modified with the **Alerts-PKB-Fail-Rate** header.

The geolocation is based on the IP address found in the host field from the log files. The `Geoip` header allows to choose which data is to be added to the results, or to deactivate the geolocation altogether.

The library used for this is `geoip-lite`.

Geolocation Caveats : IP address geolocation is a method that links the IP address of a terminal connected to the internet with a geographical position. The level of precision is not always the same: city in the best of cases, country otherwise.

The geolocation data is informative and depends on the country as well as the internet access provider.

Parameters (headers)

Geoip: geolocation data that can be added to the results.

By default: `geoip-longitude`, `geoip-latitude`, `geoip-country`

- `all` can be used to include all possible fields
- `none` to deactivate the geolocation.

The available fields are:

- `geoip-host`: IP address being geolocalized
- `geoip-country2` characters code indicating the country (eg: FR for France)
- `geoip-region`: 2 characters code indicating the region (eg: A8 for Île-de-France)
- `geoip-city`: complete name of the city (eg: Paris)
- `geoip-latitude`: self-explanatory
- `geoip-longitude`: self-explanatory
- `geoip-coordinates`: concatenation of latitude and longitude between brackets (eg: [48.8592,2.3417])

Usage example:

```
curl -v -X POST --proxy "" --no-buffer \  
-F "file=@test/dataset/geolocalize.log" \  
-H 'Geoip: all' \  
http://127.0.0.1:59599
```

Advanced usage example:

This example uses the “csv2geojson” and “geojsonio-cli” libraries.

```
npm install csv2geojson geojsonio-cli
```

It is then possible to directly visualize the results on a map.

```
curl -X POST http://127.0.0.1:59599 \  
--proxy "" \  
--no-buffer \  
--data-binary @./test/dataset/edp.2013-01-23.log \  
-H 'Geoip: geoip-latitude, geoip-longitude' \  
-H 'Output-Fields: -doi,-identd,-url,-status,-size,+datetime' \  
| csv2geojson --lat "geoip-latitude" --lon "geoip-longitude" --delimiter ";" 2> /  
→dev/null \  
| geojsonio
```

That opens a web browser with the following graphical representation :

Video Demonstration

This [screencast](#) demonstrates the previous usage (ie geolocation information visualized on a map)

Double-click deduplication

The `Double-Click` headers are optional. They set the deduplication applied to access events. By default, the deduplication is performed following the COUNTER algorithm (available on page 4 of Annex D)

They are written like follows (*in red: the editable zones*):

- `Double-Click-Removal: true`
- `Double-Click-HTML: 10`
- `Double-Click-PDF: 30`
- `Double-Click-MISC: 20`
- `Double-Click-Strategy: CLI`
- `Double-Click-C-field: NomDuChampCookie`
- `Double-Click-L-field: NomDuChampLogin`
- `Double-Click-I-field: NomDuChampHost`

Parameters (headers)

- **Double-Click-Removal:** COUNTER deduplication activated (*true by default*). If this header is used, it means the deduplication is not activated (with the *false* value) and the other `Double-Click-` headers are useless.
- **Double-Click-HTML:** sets the minimum delay (in seconds) between two requests considered identical to an HTML resource (*10 by default*).
- **Double-Click-PDF:** sets the minimum delay (in seconds) between two requests considered identical to a PDF resource (*30 by default*).
- **Double-Click-MISC:** sets the minimum delay (in seconds) between two requests considered identical to a MISC resource (*neither HTML, nor PDF*) (*20 by default*).

- **Double-Click-MIXED:** sets the minimum delay (in seconds) between two requests considered identical to a resource, **whatever its format** (ie. the access to a same resource in HTML then in PDF can be considered as a double-click). The delays set for each format are then ignored.
- **Double-Click-Strategy:** the strategy (in the form of a sequence of ordered letters) used to define the uniqueness of the user accessing a resource. The fields are searched sequentially. If one field is lacking, the following one is used. The letter **C** corresponds to the field containing the **cookie** (or **session ID**). The letter **L** corresponds to the **login** of the user. The letter **I** corresponds to the **IP address** contained in the host field. (*CLI by default*)
 - **Double-Click-C-field:** field name that will be looked for in the logs. This field coming from the custom log format parameters will be used to trace the cookie identifying the user (or its session ID). By default, it is not possible for ezPAARSE to know the field if it's not specified in the custom log format parameter. (*ignored by default*)
 - **Double-Click-L-field:** field name that will be looked for in the logs to identify the user login (*corresponds to %u in the log format syntax*). (*%u by default*).
 - **Double-Click-I-field:** field name that will be looked for in the logs to identify the user host (*corresponds to %h in the log format syntax*). (*%h by default*).

Usage:

```
curl -v -X POST --proxy "" --no-buffer\  
-F "file=@test/dataset/sd.duplicates.log\  
-H 'Log-Format-ezproxy: %h %u %{session}<[a-zA-Z0-9\\-]+> %t "%r" %s'\  
-H 'Double-Click-HTML: 15'\  
-H 'Double-Click-PDF: 30'\  
-H 'Double-Click-MISC: 40'\  
http://127.0.0.1:59599
```

CHAPTER 17

Access Events Qualification

The qualification level of an access event is based on the presence of so-called **qualifying** fields, weighted according to their importance.

An access event is considered **qualified** when the cumulative weight of its qualifying fields reaches a fixed threshold, ie when they possess enough information to be useful.

Niveau de qualification requis : 1

Champs qualifiants :

0,5 ← rtype

0,5 ← mime

1,0 ← identifiant de ressource (issn, isbn, doi...)

EC	Niveau de qualification
date = ... issn = ...	+ 0 + 1
Total : 1	

Qualifié

EC	Niveau de qualification
date = ... mime = ...	+ 0 + 0,5
Total : 0,5	

Non qualifié

EC	Niveau de qualification
date = ... rtype = ... mime = ...	+ 0 + 0,5 + 0,5
Total : 1	

Qualifié

Configuration options

ezPAARSE comes with a `config.json` file (located at the root of the `/ezpaarse` directory) where some the configuration options for your instance can be set or modified.

EZPAARSE_ADMIN_MAIL

The default value is set to `ezpaarse@couperin.org`

EZPAARSE_PARENT_URL

To avoid the setup of a local SMTP server, you can delegate the management of user feedback (via the on-line form) to another ezPAARSE instance (called a “parent” instance). The default value is set to `http://ezpaarse-preprod.couperin.org`

EZPAARSE_FEEDBACK_RECIPIENTS

The mail adress where the users feedback get sent. The default value is set to `ezpaarse@couperin.org`

EZPAARSE_SUBSCRIPTION_MAIL

If you wish to receive a message everytime a user opens an account on your instance, set the value to `true`. The default value is set to `false`

EZPAARSE_MONGO_URL

The default value is set to `mongodb://localhost:27017/ezpaarse`

EZPAARSE_ENV

The default value is set to `production`

EZPAARSE_NODEJS_PORT

The default value is set to `59599`

EZPAARSE_NODEJS_VERSION

The default value is set to `6.6.0`

EZPAARSE_REQUIRE_AUTH

Determines if the ezPAARSE online form is hidden (`true`) or accessible (`false`). The default value is set to `false`

EZPAARSE_OUTPUT_FIELDS

Contains an array of field names that are going to be present in the result file produced by ezPAARSE. The default array contains the following fields:

```
[
  "datetime",
  "date",
  "login",
  "platform",
  "platform_name",
  "publisher_name",
  "rtype",
  "mime",
  "print_identifier",
  "online_identifier",
  "title_id",
  "doi",
  "publication_title",
  "unitid",
  "domain"
]
```


EZPAARSE_DEMO

If `true`, it shows a warning informing users that the instance is a demo, and thus not adapted to process large log files. This warning now appears on our demo instance hosted on <http://ezpaarse.couperin.org> The default value is set to `false`.

EZPAARSE_MIDDLEWARES”

Contains an array of middleware names, in the order they are going to be launched by ezPAARSE during a process. The default array contains the following middlewares:

```
[
  "filter",
  "parser",
  "deduplicator",
  "enhancer",
  "istex",
  "crossref",
  "sudoc",
  "hal",
  "geolocalizer",
  "field-splitter",
  "qualifier",
  "cut",
  "anonymizer"
]
```

EZPAARSE_QUALIFYING_LEVEL

This sets the minimal value, under which ezPAARSE considers an EC is not qualified enough to be written to the results. The default value is set to 1

EZPAARSE_QUALIFYING_FACTORS

```
{
  "internal": {
    "rtype": 0.5,
    "mime": 0.5
  },
  "external": [
    {
      "file": "platforms/fields.json",
      "sublist": "rid",
      "attribute": "code",
      "weight": 1
    }
  ]
}
```

EZPAARSE_TMP_CYCLE

Determines how long ezPAARSE results remain accessible for downloading. The default value is set to 60min

EZPAARSE_TMP_LIFETIME

Sets the maximal duration for the storage of result files. The default value is set to 1day

EZPAARSE_IGNORED_DOMAINS

Contains an array of domains to be ignored (ie filtered out) by ezPAARSE.

```
[  
  "www.google.fr",  
  "www.google.com"  
]
```

To avoid declaring too long a list, we advise you to declare unrelevant domains in dedicated exclusion files as documented in this [section](#)

EZPAARSE_GEOLOCALIZE_DEFAULT

The default value is set to `geoup-lookup`

EZPAARSE_GEOLOCALIZE_SEPARATOR

The default value is set to `.` (dot)

EZPAARSE_ALERTS

Contains an object with 4 member properties, listed here:

```
{  
  "activationThreshold": 1000,  
  "unknownDomainsRate": 10,  
  "titleIdOccurrenceRate": 20,  
  "pkbFailRate": 20  
}
```

The properties and their values are documented in further details in the [relevant section](#)

The ezPAARSE jobs can be configured using HTTP headers. Please find the list of available headers below.

Content-Encoding

Encoding of the data sent. (*supported: gzip, deflate*)

Response-Encoding

Encoding of the data sent back by server. (*supported: gzip, deflate*)

Accept

Output format. Supported:

- text/csv (by default)
- text/tab-separated-values (for a TSV output: as CSV but tab-delimited)
- application/json
- application/jsonstream (one JSON object per line)

Log-Format-xxx

Format of the log lines in input, depends on the proxy *xxx* used. See the available formats

Date-Format

Date format used in the logs sent. Default is: 'DD/MMM/YYYY:HH:mm:ss Z'.

Crypted-Fields

Comma-separated list of fields that will be crypted in the results, or `none` to disable crypting. Defaults to `host, login`.

Caution: each job uses a random salt for crypting, so crypted values for the same acces events but from distinct jobs are not identical.

Output-Fields

To specify the fields to include in the output (if the format allows it). (More information)

Traces-Level

To specify the verbosity level from ezPAARSE's feedback. The higher levels include the lower ones.

- **error:** blocking errors, abnormal treatment termination.
- **warn:** errors not fatal to the treatment.
- **info:** general informations (requested format, ending notification, number of access events generated...).
- **verbose:** more precise than info, gives more information about each stage of the treatment.
- **silly:** every detail of the treatment (parser not found, line ignored, unsuccessful search in a pkb...).

Reject-Files

List of the reject files to create, separated by commas.

Possible values are:

- `Unknown-Formats`
- `Ignored-Domains`
- `Unknown-Domains`
- `Unqualified-ECs`
- `PKB-Miss-ECs`
- `Duplicate-ECs`
- `Unordered-ECs`
- `Filtered-ECs`
- `Ignored-Hosts`
- `Robots-ECs`

Set to `none` by default.

We recommend to set it to `all` when you start using ezPAARSE, to fully understand the filtering and exclusion system.

Double-Click-xxx

Parameters used for deduplication. (More information).

Request-Charset

Character map used for input. (see supported encodings).

Response-Charset

Character map used for output. (see supported encodings).

Max-Parse-Attempts

Maximum number of lines that ezPAARSE will attempt to parse in order to check the log format.

Clean-Only

If set to `true`, ezPAARSE will just filter out the lines we are sure are irrelevant and output only the relevant ones. The goal when using this parameter is to reduce the size of the log file, if you need to store it for further treatment.

Video Demonstration

This [screencast](#) demonstrates the usage of the Clean-Only parameter (ie the cleaning of a log file for size reduction and ease of storage)

Force-Parser

If URLs don't have 'domain' part, use this parameter to force right parser to be used. Usefull for Open Access log analysis which don't have domain part in URL (all URLs comes form the same domain). For example: Force-Parser: 'dspace'. Can be use in conjunction with Force-ECField-Publisher.

COUNTER-Reports

List of COUNTER reports to create (ex: JR1, BR2). Download links are accessible in the `stats` section of the treatment report. (More information)

COUNTER-Format

COUNTER report formats: XML (by default) ou CSV.

COUNTER-Customer

Name and/or email of the customer to include in the COUNTER reports, following the form name, <email> or name<email>. (By default :ezPAARSE<admin email>)

COUNTER-Vendor

Name and/or email of the publisher to include in the COUNTER reports, following the form name, <email> or name<email>. (By default :platform42, without email)

Geoip

Listing of the geolocation informations to be added to the results. By default geoip-longitude, geoip-latitude, geoip-country. all can be used to include every fiel available, or none to deactivate geolocation altogether. (More information)

ezPAARSE-Job-Notifications

Listing of notifications to send when treatment is done, written as action<cible> and separated by commas. Currently available: mail<adress>

ezPAARSE-Middlewares

Insert a list of middlewares that are not present in the base configuration (EZPAARSE_MIDDLEWARES). The value must be a list of middleware names separated with commas, in the order of use.

By default, they will be inserted at the end of the chain, before `qualifier`. You can prefix the list with the mention (before <middleware name>) or (after <middleware name>) to insert them at a more specific place.

Examples

```
'ezPAARSE-Middlewares': 'user-agent-parser, sudoc'
```

```
'ezPAARSE-Middlewares': '(before istex) user-agent-parser'
```

```
'ezPAARSE-Middlewares': '(after sudoc) hal, istex'
```

ezPAARSE-Enrich

Set to `false` to deactivate data enrichment (geoip and knowledge bases). Any other value will leave the data enrichment active.

ezPAARSE-Predefined-Settings

Tells ezPAARSE to use a predefined set of parameters. For example: `inist` for INIST-CNRS parameters.

ezPAARSE-Filter-Redirects

Set to `false` to prevent lines with HTTP status codes 301, 302 from being filtered and discarded.

ezPAARSE-Filter-Status

Set to `false` to disable filtering on status codes.

Disable-Filters

Disable filters applying to robots or arbitrary hosts/domains. (defaults to `none`). Possible values (separated by commas): `robots`, `ignored-hosts`, `ignored-domains`. Set to `all` to disable all above filters.

NB: when robots are not filtered, add the `robot` field to the output in order to know which consultations were made by robots.

Force-ECField-Publisher

Set the `publisher_name` field to a predefined value. For example: Force-ECField-Publisher: 'IREvues'.

Extract

Extract values from a field and dispatch them in new fields. The syntax is the following : `source_field => extract_expression => destination_fields`

Examples:

The following examples assume we have a `login` field with the value `THEODORE_MCCLURE`. Here are multiple ways to create a `firstname` field containing `THEODORE` and `lastname` field containing `MCCLURE`.

Extracting with a regular expression:

If the extract expression is a regular expression (between slashes, with optional flags after the closing slash), it's applied to the source field and the captured groups are stored in the destination fields.

The following expression applies the regular expression `/^([a-z]+)_([a-z]+)$/i` on the **login** field, and puts the captured groups in the **firstname** and **lastname** fields.

```
login => /^[a-z]+_[a-z]+$/i => firstname,lastname
```

Splitting over an expression:

If the extract expression is **split()**, then the source field will be splitted according to the expression between the parentheses.

The following splits the **login** field with the character `_` and puts the parts in the **firstname** and **lastname** fields.

```
'Extract': 'login => split(\_) => firstname,lastname'
```

The following splits the **login** field with the regular expression `/[_]+/` and puts the parts in the **firstname** and **lastname** fields.

```
'Extract': 'login => split(/[\_]+/) => firstname,lastname'
```

Metadata enrichment

The use of middlewares to enrich access events with metadata coming from external APIs is controlled by headers.

Crossref

(More information)

Sudoc

(More information)

HAL

(More information)

ISTEX

(More information)

ezPAARSE is a **RESTful** application. **REST**: Representational State Transfer

Sending logs

The main route for ezPAARSE is the **root** of the web service. The **GET** method gives access to the logs submission form, and the **POST** method allows sending logs. The submitted files are parsed and access events are sent back, as a resulting stream.

The detailed POST request

POST / HTTP/1.1

Parameters (headers)

The parameters list

Body

Log lines generated by a proxy server.

[EZProxy documentation](#) [Squid documentation](#)

Response to a POST request

Status code

- **200 OK**: the logs have been successfully processed.

- **400 Bad Request:** a request element makes the processing of logs impossible.
- **406 Not Acceptable:** encoding or output format not supported.

Headers

- **Job-ID:** unique identifier associated to the current processing job.
- **Job-Report:** URL for the detailed processing report, including all of the headers sent by ezPAARSE.
- **ezPAARSE-Status:** return code if an error is raised.
- **ezPAARSE-Status-Message:** explanation message on the return code.

Headers containing the URLs for accessing the logs :

- **Job-Traces:** traces for the current ezPAARSE job (the verbosity level can be modified with the Traces-Level header)
- **Lines-Unknown-Formats:** lines for which the format has not been recognized.
- **Lines-Ignored-Domains:** lines for which the domain is ignored.
- **Lines-Unknown-Domains:** lines for which the domain is not associated to a parser.
- **Lines-Unqualified-ECs:** lines that generated access events with too few information. (More details)
- **Lines-PKB-Miss-ECs:** lines that generated identifiers that can't be found in the PKB for the corresponding platform.
- **Lines-Duplicate-ECs:** lines filtered out by the double-clicks detection algorithm.
- **Lines-Unordered-ECs:** lines rejected because they were not chronologically ordered
- **Lines-Robots-ECs:** lines generated by non-human agents (robots, crawlers, spiders, etc.).
- **Lines-Ignored-Hosts:** lines that were filtered based on their IP address.

Body

CSV or JSON containing all of the generated access events.

Access event example:

```
{
  "host": "1234567d6b8dd5dddc87939c4a407987",
  "login": "IDEXEMPLE",
  "date": "2011-12-31T10:42:42+01:00",
  "url": "http://www.une-adresse.com/exemple.php?id=16",
  "status": "200",
  "size": "0",
  "domain": "www.une-adresse.com",
  "type": "PDF",
  "issn": "1111-1111"
}
```

Request examples

```
curl -X POST http://127.0.0.1:59599 --no-buffer --data-binary @file.log -v
curl -X POST --proxy "" --no-buffer --data-binary @test/dataset/sd.2012-11-30.log -v
↪http://127.0.0.1:59599 -v
curl -X POST --proxy "" --no-buffer -H "Accept: application/json" --data-binary @test/
↪dataset/sd.2012-11-30.log http://127.0.0.1:59599 -v
```

Access the traces and rejects

When ezPAARSE is processing a request (a job), it generates informative files bound to its activity. Those can be accessed by using the unique identifier attributed to the job.

- jobID: unique identifier attributed to the job.

General information

These routes are useful to get various information like: the list of platforms, the types of access events. They only respond to the *GET* method.

Administration

These routes are used to administrate ezPAARSE. For the most part, they can be used through the application's admin page. They require being authenticated, except for /register.

Every function listed in this page may be used directly from the administration section in the application.

Create an administrator

If no user has been registered yet, any attempt to connect triggers an admin creation form.

To create an administrator account without the help of the form, please use the following route :

Possible outputs

- **201 Created** : The admin has been created.
- **400 Bad Request** : Missing parameter.
- **409 Conflict** : There's already an admin.
- **500 Internal Server Error** : Creation failed.

Exemple curl

```
curl -X POST --data "userid=foo@foo.fr&password=bar&confirm=bar" http://  
↳localhost:59599/register
```

Get running jobs

Outputs a JSON table with the IDs of the jobs that are currently running.

Exemple curl

```
curl -X GET --proxy "" -u "admin:password" http://localhost:59599/jobs
```

Users management

List users

Outputs a JSON table with the complete list of users.

Exemple curl

```
curl -X GET --proxy "" -u "admin:password" http://localhost:59599/users
```

Add a user

Possible outputs

- **201 Created** : User has been created.
- **400 Bad Request** : Missing parameter.
- **409 Conflict** : User name already exists.
- **500 Internal Server Error** : Creation failed.

When the creation succeeds, the output contains a complete information about the user in JSON format.

Exemple curl

```
curl -X POST --proxy "" -u "admin:password" --data "userid=foo@foo.net&password=bar&group=user" http://localhost:59599/users/
```

Update a user

Possible outputs

- **200 OK** : User updated.
- **400 Bad Request** : Missing parameter.
- **404 Not Found** : User not found.
- **500 Internal Server Error** : Update failed.

When the update succeeds, the output contains the updated user in JSON format.

Exemple curl

```
curl -X POST --proxy "" -u "admin:password" --data "group=admin" http://  
↪localhost:59599/users/foo@foo.net
```

Delete a user

Possible output

- **204 No Content** : User has been deleted.
- **404 Not Found** : User not found.
- **403 Forbidden** : The admin has tried to delete the admin account.

Example curl

```
curl -v -X DELETE -u "admin:password" http://localhost:59599/users/foo@foo.net
```

Reset a password

Possible output

- **200 OK** : The password has been reset.
- **404 Not Found** : User not found.

Example curl

```
curl -v -X DELETE -u "admin:password" http://localhost:59599/users/foo@foo.net
```

Repositories update

The URLs below allow for updating the different parts of ezPAARSE.

Check the state of a repository

Possible feedbacks

- **200 OK** : Checking normally completed.
- **500 Internal Server Error** : Checking failed.

Response body

The server reply with a JSON response containing various things about the git status of the given repository.

Example:

```
{
  "current": "2.9.4-4-g9089308", # Current commit description
  "head": "2.9.4-4-g9089308",   # HEAD commit description
  "tag": "2.9.4",               # Current git tag (latest tag before the current_
↪commit)
  "from-head": "uptodate",      # State of HEAD compared to origin (can be 'uptodate
↪' or 'outdated')
  "from-tag": "upward",        # State of current tag compared to origin (can be
↪'uptodate', 'outdated' or 'upward')
  "local-commits": false,      # Is there any unpushed local commit ?
  "local-changes": false      # Is there any uncommitted local changes ?
}
```

Example curl

```
curl -X GET -u "admin:password" http://localhost:59599/platforms/status
```

Update a repository

Example curl

```
curl -X PUT -u "admin:password" http://localhost:59599/platforms/status
```

Possible outputs

- **200 OK** : Platforms have been updated
- **500 Internal Server Error** : Update failed.

Makefile

The makefile is located in the root directory and is used, among others, to launch tests, to generate the documentation, to check that the coding rules are respected.

Node modules installation

```
make nodejs
```

Download, compile and configure the modules that are necessary for the application to run.

Mocha tests

```
make test
```

Runs all the non-regression tests. It is a simple way to ensure that ezPAARSE is correctly working.

The test files are located in the `test/` folder and all the filenames follow the pattern, finishing with `-test.js`.

Note : don't forget to restart the application if the source code has been modified.

```
make test-pkb
```

Checks that the knowledge base files used by the parsers are well formed and coherent.

```
make test-pkb-verbose
```

Checks that the knowledge base files used by the parsers are well formed and coherent with a detailed output.

Checking coding rules

```
make lint
```

Checks the syntax of the javascript files with the `eslint` utility.

The coding rules can be modified with a configuration file (`.eslintrc`). All the options are documented on the [ESLint page](#).

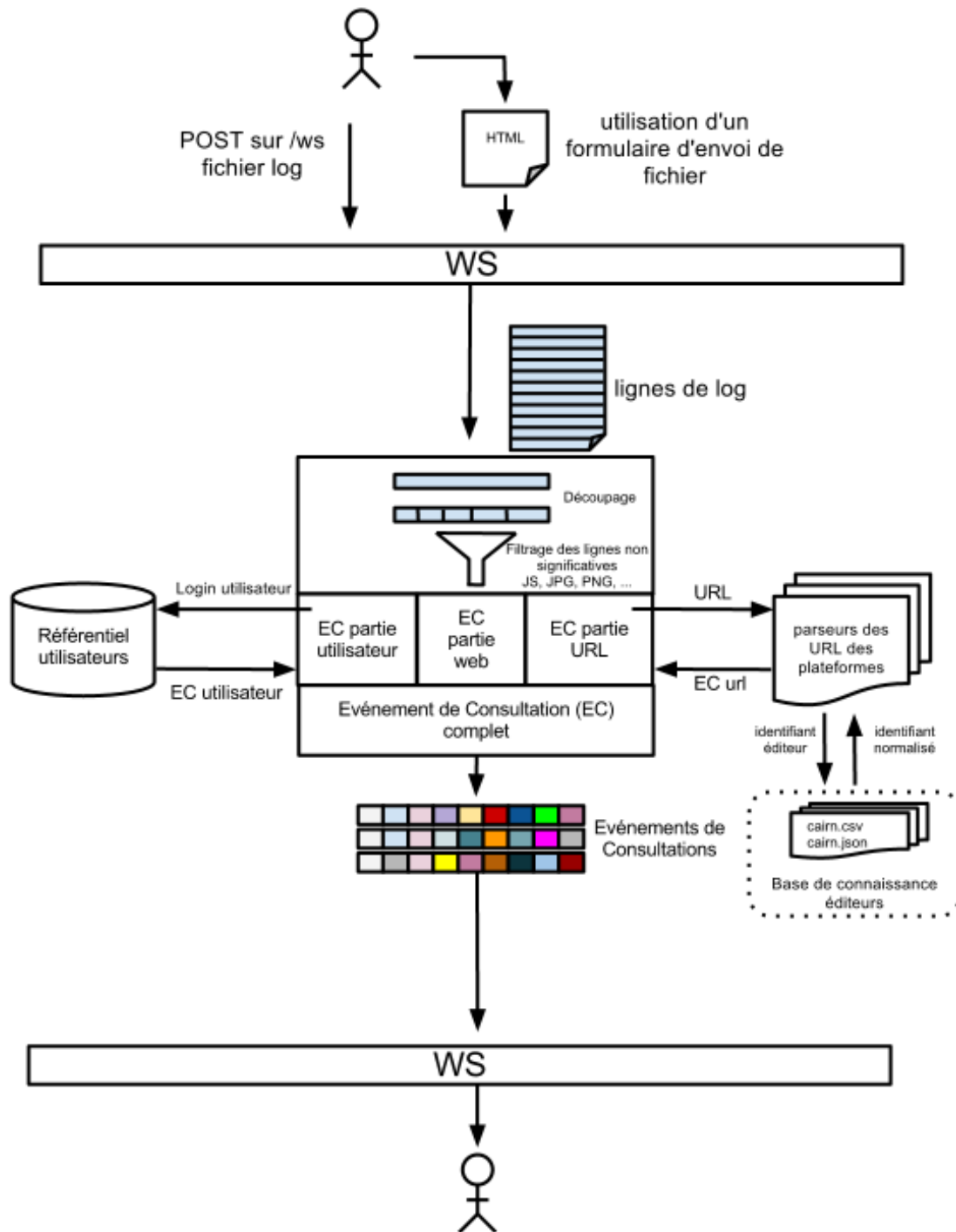
Developer-oriented documentation for ezPAARSE's core. The documentation dedicated to contributions for parsers, pkbs and scrapers can be found in this section.

Technologies used by ezPAARSE

- [nodejs](#) for the core of ezPAARSE (advanced streaming capabilities and performance).
- [git](#) to manage knowledge bases and source code.

How does the ezPAARSE engine work?

Simplified schemas of the [internal architecture](#) and [general architecture](#) are also available.



ezPAARSE's monitoring

The following options can be used to run ezPAARSE.

- `--memory`: shows the memory consumption of the ezPAARSE process every 5 seconds
- `--lsof`: displays the number of open file descriptors every 5 seconds

Example :

```
./bin/env
node app.js --memory
```

Launching the ezPAARSE's unit tests

Use the makefile to launch the tests:

```
make test
```

To test a specific function, use mocha and indicate the path of the test file as a parameter

Eg for testing custom formats:

```
./bin/env
mocha ./test/custom-formats-test
```

To perform only one functionality test, use mocha and set the path of the test file as a parameter and then specify (with `-g`) the test number (two figures) like `@xx`.

For example, for the second test about the custom formats:

```
./bin/env
mocha ./test/custom-formats-test -g @02
```

Generate a new ezPAARSE version

To generate a new version of ezPAARSE you need to be member of the ezPAARSE Team.

If you are not a member, you can submit a [pull request on github](#).

For the ezPAARSE Team:

- Check you are on the master version or run a `git checkout master`
- Use npm to generate the new version by using the appropriate options to tag the version

```
npm version [<newversion> | major | minor | patch | premajor | preminor | prepatch |
↳prerelease | from-git]
```

Example :

```
ubuntu@v-ubuntu:~/ezpaarse$ npm version patch
v2.9.4
git push
git push --tags
```


Prerequisites for the development: git(hub) user's guide

As a developer, you first have to sign up to [Github](#) to be able to contribute to ezPAARSE and:

- write a new parser
- maintain an existing parser
- contribute to improvements in the core of ezPAARSE's code

You then need to know a few `git` commands:

```
#get a local version of the github
git clone https://github.com/ezpaarse-project/ezpaarse.git

#if you already had the project, update it
cd ezpaarse/
git pull

#edit a file
cd ezpaarse/
echo "// my modification" >> ./app.js

#get an overview of which files have been modified/added/deleted (before a commit)
git status

#compare the local modifications with the local repository, line by line, before_
↳saving the modifications
git diff

#send the modifications to you local repository
git commit ./app.js -m "a comment explaining my modification"

#display the list of commits
git log
```

```
#add a new file
touch ./myexamplefile
git add ./myexamplefile
git commit ./myexamplefile -m "add an example file"

#send the (committed) modifications to the distant repository (authorization from
↳the distant repo needed).
git push
```

Note: unless you have a privileged access (from the ezPAARSE team), you must first “fork” the ezPAARSE github repository in order to work on that copy. Once you are satisfied with your changes, you can submit your work to the team by sending a “pull request”. Your job will be reviewed and integrated by the team, if no problem is detected. The team then provides write access rights to regular contributors, in order to facilitate contributions.

How does a parser work?

A parser takes the form of an executable `parser.js` file accompanied by a description file `manifest.json` and a validation structure (contained in the `test` directory, see below).

The executable can take two kinds of input:

- a stream of URLs to analyze (one per line)
- or stream of JSON objects (one per line) with the `--json` option, each containing:
 - the URL to analyze
 - other information to qualify the access event (such as the size of the download)

The parser outputs a stream of JSON objects, one per line and for each analyzed URL. The object may be empty if the parser doesn’t find anything.

Its usage is documented when you call it with the `--help` option. An [example parser](#) is available.

Usage Examples

```
echo "http://www.sciencedirect.com:80/science/bookseries/00652296" | ./parser.js
#{ "unitid": "00652296", "print_identifier": "0065-2296", "title_id": "00652296", "rtype":
↳ "BOOKSERIE", "mime": "MISC" }

echo '{ "url": "http://www.sciencedirect.com:80/science/bookseries/00652296", "status
↳ ": 200 }' | ./parser.js --json
#{ "unitid": "00652296", "print_identifier": "0065-2296", "title_id": "00652296", "rtype":
↳ "BOOKSERIE", "mime": "MISC" }
```

Writing a Parser

Parsers are written in **Javascript**. A good knowledge of the language is not really necessary to write a parser. Most of the code being outsourced in a common file for all parsers, only the URL analysis function must be adapted, making the code short and relatively simple. Most parsers still require a basic knowledge of **regular expressions**.

Writing a new parser thus consists of:

- creating the manifest.json file (see below)
- creating the test file, according to what is documented on the platform analysis page
- creating the parser so that its output is conform to the test file (see below)
- launch the validation tests (see below)

Once the parser passes the tests, it can be integrated into the github repo.

A [skeleton](#) can be used as a starting point. The directory structure and the files can be automatically generated by launching the platform-init command.

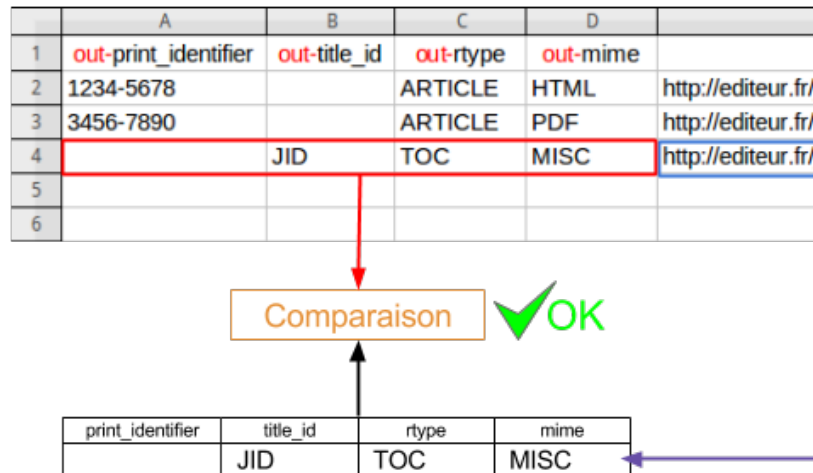
Some tools are available online to [help you in the the writing of regular expressions](#).

A [detailed procedure](#) is available on AnalogIST.

Testing Parsers

Each parser comes with the necessary to be tested: one or more files located in the `test` subdirectory. These files are in CSV format and follow the `platform.version.csv` pattern.

<platform>.<version>.csv



The test principle is represented by the following diagram:

For each row, values from columns prefixed with `in-` are sent to the parser, and the result is compared with the values coming from columns prefixed with `out-`. These must be strictly identical.

More details about the identifiers returned by the parsers are available on this page.

When the parser only takes an input URL (ie. no other fields prefixed with `in-`), it is possible to manually run the test file with the following command (from the directory of the platform):

```
#platform.version.csv is the test file
cat test/platform.version.csv | ../../bin/csvextractor --fields="in-url" -c --
↳noheader | ./parser.js
```

The tests are now integrated into ezPAARSE's platforms folder (instead of coming with the core of ezPAARSE). It means that ezPAARSE doesn't need to be running as the parsers' tests are now independent. Before you launch the platforms' tests, you need to setup the environment:

```
cd platforms/  
make install
```

Then you can either test all parsers:

```
make test
```

or test only a selection, by naming them (use the shortnames). For example, if you want to test the parsers for Nature and ScienceDirect:

```
make test sd npg
```

See the [ezparse-platforms README](#) for more information.

Description of a parser

The parser is described by a `manifest.json` file, located in its root directory. This file contains the following information:

- **name:** the short name of the parser, used as a prefix to the file names. Care should be taken not to use a name already used.
- **version:** the current version of the parser. It is used in the validation file names parsers.
- **longname:** the long name of the parser, used in the documentation.
- **describe:** the description for the parser, can be a paragraph.
- **docurl:** the URL of the documentation on the analogist website (must end by /).
- **domains:** an array of domains that the parser can handle.
- **pkb-domains:** if the platform has a PKB and if domains are present, this field is the column that contains them,

Vendors' PKBs management principles

Knowledge bases are used to:

- match the identifiers found on publishers platforms (which can be specific and proprietary) and standardized identifiers (like ISSNs)
- include the titles of accessed resources in the results

Knowledge bases are saved as text file **KBART format** and are specific to each platform. The `platform_AllTitles.txt` file contains the mappings between identifiers of a specific platform and ISSN (or other standardized identifier). The KBART field called `title_id` is used to establish this correspondence with the `print_identifier` field (for paper resources) or `online_identifier` (electronic resources). The [list of KBART fields](#) and their meaning is available.

Knowledge bases are loaded by ezPAARSE and their structure must be previously controlled by the `pkbvalidator` tool

[More on AnalogIST](#)

Running a specific test

For testing a specific feature, use mocha and give the path of the test file as a parameter. For example, to test custom formats:

```
. ./bin/env  
mocha ./test/custom-formats-test
```

For running only one feature test, use mocha and give the path of the test file and the two-digit number (@xx) of the test (with -g) as parameters.

For example, running only the second test of personalized formats will look like:

```
. ./bin/env  
mocha ./test/custom-formats-test -g @02
```


NB: The middlewares have their own dedicated repository.

What is a middleware ?

Middlewares are functions that constitute the processing chain. The middlewares are successively applied to the consultation events processed by ezPAARSE and turn them into the definitive form they will have when the events are eventually written in the result file.

How to load a middleware ?

To become part of the processing chain, a middleware must have its name (*ie* the name of its file, without the `.js` suffix) added to the `EZPAARSE_MIDDLEWARES` array in the config file. The order of declaration in the array determines the order in which middlewares are called.

Existing Middlewares

anonymizer

Anonymizes a list of fields

crossref

Enriches consultation events with `crossref` data from their API

cut

Separates any unique field into two or more distinct fields, based on a given separator or regular expression

deduplicator

Removes duplicate consultation events, based on the COUNTER algorithm for double-clicks

enhancer

Enhances consultation events with information found in a pkb (issn, eissn, doi, title_id)

field-splitter

TO BE DEPRECATED

filter

Filters irrelevant consultation events

geolocalizer

Geolocalize consultation events based on an IP address

hal

Enriches consultation events with [HAL](#) data from their [API](#)

istex

Enriches consultation events with [istex](#) data from their [API](#)

on-campus-counter

Based on the idea that all onCampus accesses will come from IP addresses contained in the private ranges (documented [here](#)), this middleware adds an `on_campus` field on each EC and increments the `on-campus-accesses` counter in the general report.

parser

Parses the URL associated with a consultation event (by calling the appropriate parser)

qualifier

Checks consultation events' qualification. See the dedicated page for more detail.

sudoc

Enriches consultation events with `Sudoc` data, especially the PPN (that identify `Sudoc` records)

throttler

Regulates the consultation events' stream

How to create a middleware ?

Specifications

Each middleware must have its own directory, with `index.js` as entrypoint, and must export a function that will serve as `initiator`. The `initiator` function must return either the actual processing function, or a `promise` that will then return it. In case of failure during the initialization, returning an `Error` object (or rejecting the `promise`) will abort the job. The error object should be extended with a `status` property that specify the status code to send back (defaults to **500**), and optionally a `code` property for the ezPAARSE-specific status (inserted in the header **ezPAARSE-Status**). The error message will be inserted in the header **ezPAARSE-Status-Message**.

The `processing` function takes the `EC` as first argument and a function to call when the `EC` should go on to the next middleware. Calling this function with an error will result in the `EC` being rejected. When there's no line left to read, the function will be called with `null` as `EC`.

The `initiator` and the `processing` function have the following properties in their context (this) :

- `request`: the request stream.
- `response`: the response stream.
- `job`: the job object.
- `logger`: a winston instance (shorthand for `job.logger`).
- `report`: a report object (shorthand for `job.report`).
- `saturate`: a function to call when the middleware is saturated.
- `drain`: a function to call when the middleware is not saturated anymore.

Example

Article counter

Here is an example of a very simple middleware that counts the articles and put the total in the report as `General -> nb-articles` :

```

module.exports = function articleCounter() {
  this.logger.verbose('Initializing article counter');

  this.report.set('general', 'nb-articles', 0);

  // Processing function
  return function count(ec, next) {
    if (!ec) { return next(); }
  };
};

```

```
    if (ec.rtype === 'ARTICLE') {
      this.report.inc('general', 'nb-articles');
    }

    next();
  };
};
```

Mandatory field

This middleware is a bit more advanced. It's activated by giving a field name in the Mandatory-Field header and it filters any EC that doesn't have a value for this field. An error is thrown on startup if the header contains a space.

```
module.exports = function mandatoryField() {
  this.logger.verbose('Initializing mandatory field');

  let mandatoryField = req.header('Mandatory-Field');

  if (mandatoryField && mandatoryField.includes(' ')) {
    let err = new Error('space not allowed in mandatory field');
    err.status = 400;
    return err;
  }

  /**
   * Actual processing function
   * @param {Object} ec the EC to process, null if no EC left
   * @param {Function} next the function to call when we are done with the given EC
   */
  return function process(ec, next) {
    if (!mandatoryField || !ec) { return next(); }

    if (ec[mandatoryField]) {
      next();
    } else {
      next(new Error(mandatoryField + ' is missing'));
    }
  };
};
```

Use of promises

```
module.exports = function mandatoryField() {

  return new Promise(function (resolve, reject) {
    if ('foo') {
      return reject(new Error('initialization failed for some reason'));
    }

    resolve(function process(ec, next) {
      // Processing function
    });
  })
};
```


platform-init

This Command Line Interactive (CLI) utility creates the structure for a platform's parser. It asks a series of questions and generates the repository structure for the parser with a manifest.json file, a parser's skeleton and an empty test file. The command is interactive and doesn't take any parameter.

Example:

```
cd ezparse/  
. ./bin/env  
platform-init
```

pkb-cleaner

Detects and deletes duplicates in the knowledge bases.

```
Usage: pkb-cleaner [-nvp] [DIR_TO_CLEAN]
```

Options:

```
--platform, -p  Name of a platform whose PKB should be cleaned. (if provided, ↪  
↪ ignore dir path)  
--norewrite, -n  If provided, do not rewrite files once the check is complete.  
--verbose, -v    Print all duplicated entries
```

Example:

```
pkb-cleaner ./path/to/some/directory  
pkb-cleaner --platform=sd
```

scrape

Launches the scrapers for one or more platforms. The scrapers are little utility programs to assemble a knowledge base by scraping a publisher's website.

```
Usage: /home/yan/ezpaarse/bin/scrape [-alvfc] [Platform] [Platform] ...
```

Options:

```
--all, -a      Execute all scrapers.
--list, -l     Only list scrapers without executing them.
--clean, -c    Clean PKB files when all scrapers has been executed.
--force, -f    Overwrite PKB files if they already exist.
--verbose, -v  Print scrapers output into the console.
```

Example:

```
scrape sd cbo # launches the scrapers for SD (ScienceDirect) and CBO
scrape -al    # lists all the existing scrapers without launching them
```

loginjector

Streams a log file to a local instance of ezPAARSE.

Example:

```
zcat monezproxy.log.gz | ./bin/loginjector
```

Usage:

Injects data into ezPAARSE **and** gets the response

```
Usage: node ./loginjector
```

Options:

```
--input, -i    a file to inject into ezPAARSE (default: stdin)
--output, -o   a file to send the result to (default: stdout)
--server, -s   the server to send the request to (ex: http://ezpaarse.com:80). If ↵
↳none, will send to a local instance.
--proxy, -p    the proxy which generated the log file
--format, -f   the format of log lines (ex: %h %u [%t] "%r")
--encoding, -e encoding of sent data (gzip, deflate)
--accept, -a   wanted type for the response (text/csv, application/json)
```

This command eases the sending of log files to an ezPAARSE instance, compared to the cURL utility.

loganonymizer

Anonymizes a log file. The sensitive elements, like the login, machine name or IP address, are replaced with random values. The log file should be sent to the system input (stdin) of the command.

Example:

```
zcat monezproxy.log.gz | ./bin/loganonymizer
```

Usage:

```
Anonymize critical data in a log file
Usage: node ./loganonymizer --input=[string] --output=[string] --proxy=[string] --
↳format [string]

Options:
  --input, -i    the input data to clean
  --output, -o   the destination where to send the result to
  --proxy, -p    the proxy which generated the log file
  --format, -f   the format of log lines (ex: %h %u [%t] "%r")
```

This is useful for generating test files by removing sensitive items (related to the protection of personal data). Each value is replaced by the same random value so keeping associations and be able to deduplicate is guaranteed.

logextractor

Retrieves one or more fields in a log file. The log file should be sent to the system input (stdin) of the command.

Examples:

```
zcat monezproxy.log.gz | ./bin/logextractor --fields=url
zcat monezproxy.log.gz | ./bin/logextractor --fields=login,url --separator="|"
```

Usage:

```
Extract specific fields from a log stream
Usage: node ./logextractor --fields=[string] --separator=";"

Options:
  --fields, -f          fields to extract from log lines (ex: url,login,host)  ↳
↳[required]
  --separator, --sep, -s character to use between each field  ↳
↳[required] [default: "\t"]
  --input, -i          a file to extract the fields from (default: stdin)
  --output, -o         a file to write the result into (default: stdout)
  --proxy, -p          the proxy which generated the log file
  --format, -t         the format of log lines (ex: %h %u [%t] "%r")
```

This is useful for manipulating log files. A common use is extracting URLs from a log file in order to analyze a platform for a publisher. For example, here's how to get the URL for the ScienceDirect platform by sorting alphabetically and deduplicating them:

```
zcat monezproxy.log.gz | ./bin/logextractor --field=url | grep "sciencedirect" | sort_
↳| uniq
```

csvextractor

Extracts content from a CSV file. The CSV file must be sent to the system input (stdin) of the command.

Example:

```
cat monfichier.csv | ./bin/csvextractor
```

Usage:

```
Parse a csv source into json.
Usage: csvextractor [-sc] [-f string | -d string | -k string] [--no-header]

Options:
  --file, -f           A csv file to parse. If absent, will read from standard input.
  --fields, -d        A list of fields to extract. Default extract all fields. (Ex: --
↳fields issn,pid)
  --key, -k           If provided, the matching field will be used as a key in the
↳resulting json.
  --silent, -s       If provided, empty values or unexisting fields won't be showed
↳in the results.
  --csv, -c          If provided, the result will be a csv.
  --json, -j         If provided, the result will be a JSON.
  --jsonstream, --js If provided, the result will be a JSON stream (one JSON per
↳line).
  --noheader         If provided, the result won't have a header line. (if csv
↳output)
```

This command is useful for testing the parser directly from the test file by extracting the URL column of the file.

Example (parser test):

```
cat ./test/npg.2013-01-16.csv | ../../bin/csvextractor --fields='url' -c --noheader |
↳./parser.js
```

csvtotalizer

Produces a summary on the content of a CSV file resulting from a processing of ezPAARSE. The CSV file must be sent to the system input (stdin) of the command.

Example:

```
cat monresultat.csv | ./bin/csvtotalizer
```

Usage:

```
Summarize fields from a CSV stream
Usage: node ./bin/csvtotalizer --fields=[string] --output="text|json"

Options:
  --output, -o output : text or json
↳[required] [default: "text"]
  --sort, -s sort : asc or desc in text mode
↳[required] [default: "desc"]
  --fields, -f fields to compute from the CSV (ex: domain;host;login;type)
↳[required] [default: "domain;host;login;type"]
```

This is useful for getting a quick overview of a processing outcome of a log file ezPAARSE. By default, domain fields, host, login and type are available in text format. Here is how to know how many different consultation events have been recognized in a sample file:

```
cat ./test/dataset/sd.2012-11-30.300.log | ./bin/loginjector | ./bin/csvtotalizer
```

logfaker

Generates an output stream matching with log lines of a platform on stdout.

Example:

```
./logfaker | ./loginjector
```

Usage:

```
Usage: node ./logfaker --platform=[string] --nb=[num] --rate=[num] --duration=[num]
```

Options:

```
--platform      the publisher platform code used as a source for generating url  ↵
↪[required]    [default: "sd"]
--nb, -n        number of lines of log to generate                               ↵
↪[required]    [default: "nolimit"]
--rate, -r      number of lines of log to generate per second (max 1000)       ↵
↪[required]    [default: 10]
--duration, -d  stop log generation after a specific number of seconds        ↵
↪[required]    [default: "nolimit"]
```

Useful to test the performance of ezPAARSE.

pkbvalidator

Checks the validity of a knowledge base for a publisher's platform. This file must conform to the KBART format.

This command checks the following:

- The presence of the .txt extension
- Uniqueness of title_id
- Minimal identification information available
- Syntax check of standardized identifiers (ISSN, ISBN, DOI)

Usage:

```
Check a platform knowledge base file.
```

```
Usage: node ./bin/pkbvalidator [-cfsv] pkb_file1.txt [pkb_file2.txt]
```

Options:

```
--silent, -s    If provided, no output generated.
--csv, -c       If provided, the error-output will be a csv.
--verbose, -v   show stats of checking.
```

ecmaker

Starts a batch processing a log file on the local instance of ezPAARSE.

Example Usage with logfaker on the standard input:

```
./logfaker -d 5 | ./ecmaker  
  
ll tmp  
ecmake-2014-01-29_11-08-15.ec.csv  
ecmake-2014-01-29_11-08-15.report.html
```

A result file and a static report file are generated by default in the destination directory.

Example of actual use with a log file:

```
./ecmaker --input=/home/ubuntu/ezparse/test/dataset/sd.2012-11-30.300.log --  
↪outpath=tmp/test  
  
ll tmp/test  
sd.2012-11-30.300.ec.csv  
sd.2012-11-30.300.report.html
```

The result file (with the ec.csv extension) and a static report file (with the report.html extension) are generated in the output directory.

```
Usage:  
Inject a file to ezPAARSE (for batch purpose)  
Usage: node ./bin/ecmaker [-hiofvH]  
  
Options:  
--input, -i      Input log file (if omitted, wait for standard input)  
--outpath, -o    If provided, output directory (default tmp).  
--force, -f      override existing result (default false).  
--headers, -H    headers parameters to use.  
--verbose, -v    Shows detailed operations.
```

ecbulkmaker

Starts a batch processing for log files in a directory on the local instance.

Exemple d'utilisation :

```
./ecbulkmaker -r /applis/stats/home/archives/fede/bibliovie/2013 /applis/stats/home/  
↪ezresults/fede/bibliovie/2013
```

A result file (.ec.csv extension) and a report in HTML format (extension.report.html) are generated in the output directory for each log file. If the destination directory is not specified, they are generated in the same directory as the file being processed. If an error occurs when processing a file, the incomplete result file is named with the .ko extension. Rejects files are not retained by ezPAARSE.

```
Inject files to ezPAARSE (for batch purpose)  
Usage: /home/yan/ezparse/bin/ecbulkmaker [-rflvh] SOURCE_DIR [RESULT_DIR]  
  
Options:  
--recursive, -r  If provided, files in subdirectories will be processed. (preserves_  
↪the file tree)  
--list, -l       If provided, only list files.  
--force, -f      override existing result (default false).  
--header, -H     header parameter to use.  
--verbose, -v    Shows detailed operations.
```

Video Demonstration

This [screencast](#) demonstrates the usage of ecbulkmaker (ie process a directory containing log files and outputting a mirror directory with the results)

hostlocalize

Enriches a csv result file containing a host name with the geolocation of the IP address

Example:

```
./hostlocalize -f ezparsedata.csv > ezparsedatalocalised.csv
```

The input file is assumed to contain a field with the ip address for the location

```
Enrich a csv with geolocalisation from host ip.  
Usage: node ./bin/hostlocalize [-s] [-f string | -k string]  
  
Options:  
--hostkey, -k the field name containing host ip (default "host").  
--file, -f A csv file to parse. If absent, will read from standard input.
```


CHAPTER 27

Documentation

In order to build the documentation locally, you need [Sphinx](#) and the Sphinx theme for [ReadTheDocs](#). For convenience, the instructions below include a live-reload server.

First, install the dependencies :

```
sudo apt-get install python python-pip python-sphinx
sudo pip install sphinx_rtd_theme sphinx-autobuild
```

Then move to the `doc` directory and type :

```
make serve
```

Finally, visit <http://localhost:8000/>.

CHAPTER 28

Tree structure

CHAPTER 29

Website languages

ezPAARSE is available in french and in english.

The `i18n-2` NodeJS module was chosen because it integrates well with `Express.js`, used to generate the HTML pages of ezPAARSE.

The language files are located in the “locales” folder in the form of json files. Those filenames follow the pattern: *country_code.json* (eg: `fr.json` or `en.json`)

The language files contain series of keys. Each key is followed by a translation in the target language.

The key is composed of a context and a french label: *context+french_label*.

The context matches the name of the HTML page in which the label appears.

2.9.4

2016/11/16

- Updated DOC

2.9.3

2016/09/27

- Updated NodeJS and dependencies

2.9.2

2016/09/12 - bug fix

- Bug fix: basic connection dialog popping up on the login page

2.9.1

2016/09/09 - bug fix

- Added compatibility with [ezMASTER](#)
- Bug fix: added missing dependency in production environments

- Bug fix: unstable results due to some problems in the read process

2.9.0

2016/08/30 - ezPLAAGE

- Bug fix: crash on startup when the PKBs had issues
- Access events have a new unique ID, created with a hash of their log line
- Similar values are now aggregated into a single field, no matter the source
- Ajax requests can now use all available headers
- Middlewares have been [externalized](#)

2.8.1

2016/06/03 - bug fix

- Serverside timeout increased and configurable

2.8.0

2016/05/31 - ezMASURE - Halte ! Kibana

- Bug fix: error on module dependencies
- Bug fix: remove timeout on diluated logs files
- New ISTEEX and cut middleware

2.7.0

2015/11/05 - Cash Enrichment

- Data caching of API requests from middlewares
- Enrichment activated by default

2.6.0

2015/10/08 - New BACON scraper tool

- Minor doc updates
- New BACON scraper tool

2.5.0

2015/09/03 - EC enrichment via APIs

- Automatic access events enrichment with crossref and elsevier
- Light Windows version

2.4.0

2015/06/11 - Optimized Installation - Docker

- Introduction of Docker for deployment process
- New Windows version
- Filtering by middleware functionality

2.3.0

2015/04/09 - Minimal Viable Product for Open Access repositories

- IP addresses exclusion for indexing robots and spiders
- New form tab for a log format helper tool
- Multi-format deduplication
- Storing predefined parameters in the local instances

2.2.0

2015/02/05 - Large volume PKB management

- Optimizing the management of NoSQL PKB via castor-js
- Improving the initialization time of the application
- Ergonomic improvements in the angular form (file name, advanced headers, alert messages)

2.1.0

2014/10/31 - embedded NoSQL

- Update of semantic-UI
- NoSQL embedded in the castor branch
- Added generic profiles in the pre-defined parameters
- Added new parsers (for legal resources)

- Downloading a deduplicated unknown-domains file now possible

2.0.0

2014/09/01 - PKB Administration

- Visualization interface for the PKBs
- PKB updates possible in the admin section
- Software update in the admin section
- Deduplicating the PKBs tool
- MVP of a “my profile” page
- Addition of publisher_name in access events (for COUNTER reports exports)

1.9.0

2014/06/23 - Parsathon

- 50 parsers are now available
- Open Access repositories first approach

1.8.0

2014/03/13 - Minimal Viable Product for anomalies alerts

- End of processing notification
- Alert on the unknown-domains while processing

1.7.0

2014/03/13 - Web interface Validation

- New Web interface validation
- Platform-plugin Refactoring

1.6.0

2014/03/13 - Web interface redesign

- Web interface redesign with AngularJS
- Access events' Geolocation

1.5.0

2014/02/01 - KBART standardization

- PKBs' KBART standardization
- LibreOffice macro rendering

1.4.0

2014/01/30 - COUNTER Export

- COUNTER JR1 export
- non-usage information (denied accesses) now taken in account

1.3.0

2013/12/19 - Multilingualism

- multilingualism (web interface, windows installer)
- MVP COUNTER export

1.2.0

2013/11/14 - Security

- securing access to the web form (password protection possibility)

1.1.0

2013/10/10 - Minimal Viable Product for synchronising the Platform Knowledge Bases

- Reorganization of the git repos
- new repo for scrapers
- Automatic synchronization of the PKBs

1.0.0

2013/09/09 - Version 1.0

- Windows packaging improvement, meeting the goal of demonstrating the steps from a log file to the macro rendering
- Browser bug correction

0.9.0

2013/08/08 - COUNTER deduplicating - candidate version 1

- Publisher knowledge base automation / ezPAARSE tools
- better code quality
- COUNTER deduplicating for consultation events

0.8.0

2013/06/27 - Minimal Viable Product for the User Knowledge Bases

- management of User-Fields
- management of advanced options and predefined values in the form
- translation to cURL for the form actions (to allow for task-automation)
- new traces and bug corrections

0.7.0

2013/06/05 - Consolidation of the software's core

- better access events definition (rtype, MIME, unitid)
- testing for large log quantities
- new parsers (BMC, lamyline, lextenso, lexisnexis)
- form redesign with dynamic metrics
- access to traces and reject files (not recognized formats, unknown domains, etc)

0.6.0

2013/04/18 - Consolidation of plafforms' recognitions

- upgrade to node v0.10

- new parsers (daloz, wiley)
- new execution traces

0.5.0

2013/03/27 - ezPAARSE usability

- a windows installer is provided
- a form for submitting logs is provided

0.4.0

2013/02/21 - More use cases

More log formats are accepted by ezPAARSE: EZproxy, Bibliopam, Squid

0.3.0

2013/01/31 - More parsers

- Clear documentation describing how accesses events are recognized
- New parsers developed for various platforms
 - sd (Science Direct) : a parser without a knowledge base because a normalized identifier is found in the URL
 - npg (Nature Publishing Group) : a parser with a knowledge base because the publisher's platform uses proprietary identifiers
 - edp (EDP Sciences) : a parser for a platform that only publishes one journal (every journal is hosted on a distinct platform but all platforms work the same way)

0.2.0

2012/12/20 - Generic installation

The MVP implemented in the first sprint can now be installed on various OSes: Ubuntu, fedora, RedHat, Suse

0.1.0

2012/12/06 - Minimum Viable Product

Accesses events are produced for ScienceDirect only from log files via a RESTful web service.