Design of the `sapupra` project

v0.50.0.98

Péter Felecan

assisted by Claude Opus 5.5 (Anthropic)

November 6, 2026

Contents

1 Introduction **4**

2 Scan devices **4**

3 Programs **4**
3.1 Running as services 5

4 Messages **6**
4.1 Topics 6
4.2 Input 6
4.3 Receipt and verdict 8
4.4 Presence of the worker 8
4.5 Delivery 8

5 The gateway **9**
5.1 Requests 9
5.2 Locations 9
5.3 Replies 10
5.4 Waiting for the verdict 11
5.5 Receipts 11
5.6 HTTP 11

6 The worker **12**
6.1 Recording first 12
6.2 Classification 12
6.3 Resolvers and their order 13
6.4 The Z39.50 resolver 15

6.5 The DNB resolver . . . 17
6.6 The source of a description . . . 17
6.7 Statuses . . . 18
6.8 Repeated scans . . . 18
6.9 Search by author and title . . . 19
6.10 Locations . . . 20

7 The entry module 21
7.1 Pages . . . 21
7.2 The look of the pages . . . 24
7.3 What it publishes . . . 24
7.4 Importing Binary Eye's history . . . 25
7.5 Access . . . 26
7.6 Access from outside the network . . . 26

8 Several users at once 28

9 Inserting texts into messages 29

10 When a part is absent 30

11 The language of the pages 31
11.1 The sentences . . . 31
11.2 The catalogues . . . 31
11.3 The reading of a catalogue . . . 32
11.4 Choosing the language . . . 32
11.5 The words of the data . . . 32
11.6 The French . . . 33

12 Location labels 34

13 Labels of our own 37
13.1 The problem . . . 37
13.2 The code . . . 37
13.3 Two ways to tie a label to an entry . . . 38
13.4 What a QR code can hold . . . 38
13.5 Where the locations are . . . 39
13.6 What a scan of a label does . . . 40
13.7 Making the labels . . . 41
13.8 The stages . . . 41

14 Removing **42**
14.1 Options . . . 42
14.2 What a removal is . . . 42
14.3 Places, editions, devices . . . 43

15 The database **43**
15.1 The files of the database . . . 43
15.2 Principles . . . 45
15.3 Tables . . . 45
15.4 Full-text index . . . 47
15.5 A view for the web front . . . 48

16 The web front **48**

17 The printed lists **51**

18 A catalogue to try the programs on **52**

19 Configuration **53**
19.1 The secrets . . . 55

20 Code layout **55**

List of Listings

1 Examples of `spplabel` . . . 36
2 The schema of the database . . . 45
3 An example of the configuration file . . . 54
4 The file of the secrets, beside the configuration file . . . 54

List of Tables

1 The programs of the prototype . . . 5
2 The topics of the broker . . . 6
3 The fields of an input . . . 6
4 The kinds of input . . . 7
5 The replies of the gateway . . . 10
6 Classification of a code . . . 12
7 Default order of the book resolvers, by registration group . . . 13
8 The statuses of an input . . . 18
9 The pages of the entry module . . . 21
10 What happens when a part is absent . . . 30

11 The options of `spplabel` 35
12 What a QR code holds: characters of the alphanumeric mode at level M, bytes at the four levels 39
13 The unit tests 56

1 Introduction

This document designs the programs of the first prototype: the gateway, the worker and the entry module, the messages they exchange through the broker, and the database. Concept.pdf gives the architecture and its reasons, and defines the terms used here (Concept.pdf, Glossary); Manual.pdf will say how to install and use what is designed here, as each program is written.

The design is written before the code, and is a proposal until the code exists; written so far: the common library and the gateway. It says what was decided; the options considered for each decision, and why one was chosen, are in the research journal (ResearchJournal.pdf), and the decisions still to take in the Decisions section of FutureResearch.pdf. The web front is the pages of the entry module that read the catalogue (0.18.0), with Datasette as an optional extra outside the build; the contract of both with the database is in A view for the web front, and the pages are in The web front.

The implementation is C++17 with Qt 6 (the modules Core, Network and Sql), libmosquitto, YAZ, and libzint for the labels; Qt's event loop drives every program, and no program needs a thread of its own but the worker, for the Z39.50 queries.

2 Scan devices

A scan device is a device that reads a barcode and transmits what it read to the broker, directly or through the gateway. Today, a phone with a scanning application that forwards each scan by HTTP to the gateway: an Android phone with Binary Eye, and later, untried, an iPhone or an iPad with AllCodeRelay. A device that could publish to the broker itself (a scanner of our own making, an application that speaks MQTT) would need no gateway. Each scan device has a scan device identifier, which it sends with every scan, and which names its topic on the broker.

3 Programs

Table 1 lists the programs, the account each uses on the broker, and what each listens on. The names follow the rule of the development standard for executables (lowercase, no extension) with the prefix `spp`. The ports are those of the configuration (Configuration); 8080 and 8081 are the defaults.

Table 1: The programs of the prototype

Program	Role	Broker account	Listens on
`sppgate`	receives the scan devices' scans	`gateway`	HTTP, port 8080
`sppwork`	records, resolves, writes the DB	`worker`	(nothing)
`sppentry`	completes failed scans, by hand	`entry`	HTTP, port 8081
`sppconsult`	consults the catalogue only	(none)	HTTP, port 8082
`spplabel`	prints labels: locations, codes	(none)	(nothing)
`spplist`	writes the printed lists (Org)	(none)	(nothing)

The programs share a library, built with them, for what they have in common: the configuration, the logging, the connection to the broker, the messages, the classification of codes, and a small HTTP server (Code layout). The gateway is the program `sppgate`; the HTTP server is a part of the library, which `sppgate` uses to receive the scans and `sppentry` to serve its pages. `spplabel` and `spplist` are commands run by hand (Location labels, The printed lists); the others are services (`sppconsult` is an option of the installation: Access from outside the network). A seventh, `sppgen`, which makes a catalogue of invented books to try the others on (A catalogue to try the programs on), is built and not installed.

3.1 Running as services

The gateway, the worker and the entry module run unattended, from the start of the machine, under the system account `sapupra`. Each program runs in the foreground; the service manager puts it in the background, restarts it if it stops, and stops it by `SIGTERM`, on which the program closes its connections and exits:

- on Debian, a systemd unit for each, as `sppgate.service` (both are made from one template, `sapupra.service.in`; so are the rc scripts, from `sapupra.rc.in`); the program logs to its standard error, which the journal keeps (`journalctl -u sppgate`);

- on FreeBSD, an rc script for each, as `sppgate`, which starts the program through `daemon(8)`: `daemon(8)` writes its own process number in the pid file (`-P`), runs the program under the account `sapupra` (`-u`), restarts it 5 seconds after it stops (`-R`), and passes the stop signal on; the program logs to syslog (`--syslog`).

Started by hand, a program stays attached to its terminal, which is what trying it needs; `SIGINT` (Ctrl-C) stops it. The files are built with the programs, and installed by hand (Manual.pdf).

4 Messages

4.1 Topics

Every topic starts with `sapupra/v1/`, so that a later, incompatible version of the messages can live beside this one. Table 2 lists them; `DEVICE` is the scan device identifier, `ID` the identifier of an input.

Table 2: The topics of the broker

Topic (after `sapupra/v1/`)	Published by	Read by	Retained
`input/DEVICE`	gateway, entry	worker	no
`receipt/ID`	worker	gateway, entry	no
`verdict/ID`	worker	gateway, entry	no
`status/worker`	worker, its will	gateway, entry	yes

One topic carries every input of a scan device, whatever it is (a scan, a location, a typed ISBN, a label): MQTT keeps the order of the messages that one client publishes on one topic, and the order matters, since a location applies to the scans that follow it. All messages are published with QoS 1.

4.2 Input

An input is a JSON object, encoded in UTF-8; Table 3 lists its fields. The gateway and the entry module make it; the worker does not need to know which application of which scan device sent the scan, only what was read. Table 4 gives the meaning of each kind.

Table 3: The fields of an input

Field	Value
`v`	1, the version of the message
`id`	a UUID, made by the publisher; the identity of the input
`kind`	`scan`, `isbn`, `location`, `label`, `place`, `device`, `remove`, `restore`, `correct`, `search`, `choice`, `record`, `copy`, `dismiss`
`device`	the scan device identifier
`origin`	`binaryeye`, `allcoderelay`, `entry` or `history`
`received`	when the publisher received it: ISO 8601, with offset
`code`	what was read or typed, as it was
`symbology`	the barcode type, as the device named it (`EAN_13`)
`addOn`	the EAN add-on, if the device reported one
`deviceTime`	the scan device's own time of the scan, as it wrote it
`for`	the `id` of the scan that this input completes
`location`	for a location: its text; for a `place` order: the place it is about

Continued on next page

Continued from previous page

Field	Value
`place`	for an item: the place of that item only; for a `place` order: the target
`action`	for a `place` order: `make`, `rename`, `merge` or `delete`
`reason`	for a `remove`: `donated`, `sold`, `lost`, `mistake` or `other`
`query`	for a search: author, title
`record`	for a choice or a record: the bibliographic record
`raw`	the request of the scan device, unchanged

Table 4: The kinds of input

Kind	Made by	Means	Fields
`scan`	gateway	a code read by a scan device	`code`
`location`	gateway, entry	the device's current location changes (to a place that is made)	`location`
`label`	gateway	a label of an item was scanned	`code`
`place`	entry	make, rename, merge or delete a place	`location`, `place`, `action`
`remove`	entry	a copy leaves the catalogue, with a reason	`code`, `reason`
`restore`	entry	a removed copy comes back	`code`
`correct`	entry	the description of an item's edition is corrected	`code` (the item), `record`
`isbn`	entry	an ISBN typed for an item, or a failed scan	`code`, `for`
`search`	entry	find records by author and title	`query`
`choice`	entry	this record, among those found, is the item	`record`, `for`
`record`	entry	a description typed by hand	`record`, `for`

A `location` from the gateway is a scanned label; from the entry module, a place chosen for a scan device. A `label` carries the number of the item in `code`. A `place` is an order of the entry module to the worker, which alone writes the database: `make` a place (with its parents), `rename` it (the paths of the places below it follow) or `merge` it into another (its items and devices move). On an `isbn`, `record` or `choice`, `place` is the place of that item and never changes the location of the device. A `search` creates no item: its verdict carries the records found, and the choice among them is the next input. Without `for`, an `isbn`, `choice` or `record` catalogues a new item, with no scan behind it (an item without barcode).

The time of the scan device is kept in `raw` only: Binary Eye sends it without time zone and with a colon before the milliseconds (`2026-10-23 14:59:12:237`); `received`, the time of the gateway, is the one the catalogue uses.

4.3 Receipt and verdict

The worker answers each input twice, on topics named after its `id`:

receipt as soon as the input is recorded in the database; carries `v` and `id` only.

verdict when the input is settled, or settled for now; carries `v`, `id`, `status` (one of the statuses of Statuses), `item` (the number of the item, if one was created), `text`, a line for the scan device: the title found, or the reason, and, for a search, `records`, the records found. A book found by a source ends with the source, " — from openlibrary" (an em dash; `Catalogue::sourcesNote`), in the verdict kept in the database, in the one published and in the log, so that one sees who answered; a second copy of a book already catalogued has no such note, no source being asked.

4.4 Presence of the worker

The worker connects with a will, `offline` on `status/worker`, retained, which the broker publishes if the worker disappears; once connected, it publishes `online` there, retained. The gateway and the entry module read it, and do not wait for a verdict that cannot come.

4.5 Delivery

Publishing the gateway (and the entry module) count an input as handed over when the broker has acknowledged it (the `PUBACK` of QoS 1, which libmosquitto reports by its publish callback), never before.

Waiting the worker connects with a fixed client identifier, the name of its account, `worker`, and a persistent session; the broker keeps on disk the inputs that arrive while it is away (the persistence of Manual.pdf).

Taking over libmosquitto acknowledges an incoming QoS 1 message to the broker before it hands the message to the program (read in its source, `handle_publish.c`, version 2.0.18 and the development branch: `send__puback` comes before the message callback), and has no manual acknowledgement. A worker that stops between the two would lose the input. Hence the receipt: the publisher keeps each input until its receipt arrives, and publishes it again, with the same `id`, if none came within a delay (30 seconds) while the worker is online.

Twice an input may then arrive twice; the worker records an `id` once only (it is the key of the table of inputs) and answers the second time with the same receipt and verdict.

5 The gateway

5.1 Requests

The gateway accepts `POST` requests on two paths, the root of the server and `/scan` (Binary Eye posts to the URL as it is given, with or without a path), with a JSON body:

from Binary Eye an object with `content`, `format`, `timestamp` and `deviceId` (as captured in Manual.pdf), and sometimes `addOn` and others, sent with the request type "POST application/json" of its settings; another request type is answered 415, with the setting to change;

from AllCodeRelay an object with one field, `code`;

the test request of Binary Eye `content test`, `format None`; answered at once, with the state of the broker and of the worker, and published nowhere.

The scan device identifier is taken, in this order, from the parameter of the path `device` (`/scan?device=NAME`), from the header `X-Sapupra-Device` (which AllCodeRelay's custom headers can set), from Binary Eye's `deviceId`; without any, it is `unknown`. It is 1 to 32 characters among letters, digits, `-` and `_`, since it is a level of a topic; an invalid one is refused (400).

An input made by the gateway keeps the request as it came in `raw`, and Binary Eye's `timestamp` in `deviceTime`: Binary Eye writes the same text in its history, which lets an import of the history recognize the scans already recorded (Importing Binary Eye's history).

A `GET` of the root of the server answers with the state of the gateway: the broker connected or not, the worker online or not, the number of inputs waiting for their receipt; a browser of the local network is enough to check it.

5.2 Locations

A shelf label and an item's barcode reach the gateway the same way, as a scanned code; the location prefix is how the gateway tells them apart. A code that starts with it, `LOC:` (configurable), is a location, not an item: the gateway publishes it as an input of kind `location`, whose text is the rest of the code. The answer is the worker's: the place is set as the device's location (`located`, 200), or it is not made and nothing changes (`unknown`, 422): a place is made on purpose, in the entry module (Where the locations are). A code that is the prefix alone is refused. A label is printed by `spplabel` (Location labels), holding, for example, `LOC:SALON B3` or, for a shelf in a room, `LOC:salon/bookcase 2/shelf 4`: a location is a path (Where the locations are), and the text after the prefix is brought to its normal form (`Code::normalizeLocation`) before anything else is done with it.

A label of an item is recognised the same way, by its own prefix (`[items] prefix=`, `ITM:`): the gateway checks the number and its check digit (`Code::checkItemLabel`), answers 422 "Label misread, scan again" for a bad one without publishing it, and publishes a good one

as an input of kind `label` (What a scan of a label does). The two prefixes must not begin alike, and each program that reads them refuses a configuration where they do.

The prefix exists because a label must never be taken for an item, nor an item for a label. The codes of items are digits only (EAN-13, UPC-A), or an ISBN; a prefix with a letter and a colon cannot begin any of them. Without one, a label holding "Salon B3" would be an unrecognized code, and a label holding only digits could be looked up as a product. `LOC:` is short, and its characters (capitals, digits, space and colon) are those of the alphanumeric mode of QR codes, the densest mode for text, which keeps the printed label small when the location is written in capitals.

5.3 Replies

The reply is short (Binary Eye shows its first 240 characters) and its status tells the scan device whether to sound the error; Table 5 lists the cases.

Table 5: The replies of the gateway

Case	HTTP	Text
resolved, found	200	the title, from the verdict
location set	200	"location: " and the location
location not made	422	the text of the verdict (`unknown`)
label of an item	200	the item and where it is put (`labelled`)
label misread	422	"Label misread, scan again"
repeated scan	200	the title, "(again)"
no verdict within the wait	202	"queued"
worker offline	202	"queued, worker offline"
a source did not answer	202	the text of the verdict
set aside by the worker	422	the text of the verdict
not handed to the broker	503	"Not recorded, scan again"
invalid request	400	the reason
not JSON	415	the setting to change

An input is set aside when it is not found, not an ISBN, ambiguous or not a code (the statuses of Statuses); it is not handed to the broker when the broker is out of reach, or does not acknowledge it in time.

Every status outside 200 to 299 makes Binary Eye vibrate and sound its error; "not recorded" is the case where the scan is lost unless it is repeated, and says so. AllCodeRelay keeps a scan whose reply is not 200 for a later retry: to it the gateway answers 200 in every case but "not recorded", with a JSON object (`code`, the kind of answer, and `codevalue`, the text), as its documentation asks.

5.4 Waiting for the verdict

The gateway first waits for the broker's acknowledgement of the input, at most 3 seconds (`wait`, configurable): without it, it answers "not recorded". Then, unless the worker is offline, it waits for the verdict, at most as long, and replies with it, or with "queued". A verdict that comes later is not lost: the scan is in the database with its status, and a failure is in the entry module's list.

An acknowledgement that comes after the wait is still possible: libmosquitto keeps the message, and may deliver it. A scan answered "not recorded" can then be recorded after all; scanned again within the repetition window, it is taken for a repetition, and creates no second item.

5.5 Receipts

The gateway keeps each input it has handed over until its receipt (or its verdict) arrives. Without a receipt within 30 seconds (`receipt_timeout`, configurable) while the worker is online, it publishes the input again, with the same identifier; the delay starts again when the worker comes back online. The inputs are kept in memory only: stopping the gateway forgets them, while the broker keeps those that the worker has not taken.

Since 0.39.0 this is not the gateway's own code: it is `InputPublisher`, the class that the entry module uses, which publishes, follows the broker's acknowledgement, the receipt, the verdict and the worker's presence, and publishes again. It tells the gateway by signals (`handedOver` with the broker's acknowledgement of the publication, `receiptReceived`, `verdictReceived`, `workerStatusReceived`). The gateway keeps what is its own, per input: the HTTP exchange, the client and the deadline of the reply. When the deadline passes before the broker's acknowledgement, the gateway answers "not recorded" and tells the publisher to `abandon` the input, which is then not published again: the scan is to be repeated by the person, who has been told so. An input that was acknowledged is kept and republished as above even when the reply was "queued".

5.6 HTTP

The HTTP server is the library's own, `HttpServer`, on Qt's `QTcpServer`: one request per connection, answered with `Connection: close`; a request line, headers, and a body of a declared length (`Content-Length`); `GET`, `HEAD` and `POST` only (a `HEAD` is routed as a `GET`, with the flag `mHead` set on the request, and the exchange leaves the body out of the reply, `Content-Length` kept: a program writes nothing for it, and cannot treat it as a `POST`); no chunked transfer, no persistent connection, no TLS. Everything received is bounded: 8 KiB of headers, 16 KiB of body, 10 seconds to receive a request, 64 connections at once; a request beyond a limit is answered with its status (431, 413, 408, 503). A reply is sent whenever it is ready, which lets the gateway wait for a verdict. The entry module uses the same server, with a larger body for the import of a history (Importing Binary Eye's history).

6 The worker

6.1 Recording first

For each input, in this order: record it (the table of inputs, with the location then current for its device and its order of arrival), commit, publish the receipt; then classify, resolve, record the result, commit, publish the verdict. Recording never waits for the Internet. The program `sppwork` does it as `Worker` (the messages), `Catalogue` (the database) and one `Resolver` per source; the sources are queried one request at a time per source, and the answers wait in the queue of the source, not in the event loop.

An input already recorded (the same identifier: the gateway publishes again an input without receipt) is not recorded twice; it is given its receipt again, and, if it is settled, its verdict again. At start, the inputs recorded and not settled (the worker stopped between the receipt and the verdict) are taken up again, and the inputs published while the worker was away are handed over by the broker, which keeps the session under the account's name.

Only one worker may run on a database. Two of them, with the same identifier, would take the session from each other at the broker, one reconnecting every few seconds, and with two identifiers each would take up the input that the other has in hand. So the worker takes a lock on a file beside the database (`catalogue.db.worker.lock`, Qt's `QLockFile`: the process that holds it is named in the file, and a lock whose process is gone is taken over), and a second worker stops at its start, saying which process holds it.

6.2 Classification

Table 6 gives the class of a code, from its digits once the spaces and hyphens are removed; a typed ISBN-10 is checked and turned into its ISBN-13.

Table 6: Classification of a code

Code	Class	Asked
13 digits, 978 or 979, not 9790	ISBN-13	book resolvers
13 digits, 9790	ISMN	BnF (printed music)
13 digits, 977	ISSN	BnF (periodicals)
13 digits, other	GTIN	record and film resolvers
12 digits (UPC-A)	GTIN, 0 prefixed	record and film resolvers
8 digits (EAN-8)	GTIN, 00000 prefixed	record and film resolvers
10 characters, valid ISBN-10	ISBN-13	book resolvers
anything else	unrecognized	nothing

A check digit that does not match makes the code unrecognized: a misread barcode is not looked up. Stored with the input, its class and its normalized form (the ISBN-13, the 13-digit GTIN) are what the repetitions and the copies are found by.

6.3 Resolvers and their order

A resolver asks one source about one code and returns nothing, one record or several. The worker asks the resolvers of the class in order and stops at the first that knows the code. For an ISBN the order depends on its registration group, and is configuration, not code; Table 7 is the proposed default.

Table 7: Default order of the book resolvers, by registration group

Group	Resolvers, in order
2 (French)	BnF SRU, Open Library
0, 1 (English)	Open Library, BnF SRU, Library of Congress
3 (German)	DNB SRU, Open Library, BnF SRU
963, 615	OSZK Z39.50, Open Library, BnF SRU
88, 979-12	SBN Z39.50, Open Library, BnF SRU
any other	Open Library, BnF SRU

The Hungarian books (groups 963 and 615) are asked of the National Széchényi Library (OSZK), whose catalogue answers Z39.50 queries (ResearchJournal.pdf); a source of their own is needed, since Open Library knows few of them. It was tried live: it finds the Kertész by its ISBN-10 and not by its ISBN-13, so the ISBN-13 of a 978 book is turned into its ISBN-10 for it. The OSZK and SBN sources exist since 0.19.0 (The Z39.50 resolver), built when YAZ is there; the table is their default order (one line of `[sources]` for the same prefix replaces it), and the default for any other ISBN is Open Library then the BnF. French books (group 2) are left to the configuration file, as before.

Since 0.32.0 two more sources exist. The German national library (DNB) answers SRU at `services.dnb.de`. The Library of Congress (LC) answers Z39.50; its address is given in The Z39.50 resolver, and it is built when YAZ is there. They are in the default orders of the table for German books (978-3: the DNB first, built in every case) and for English ones (978-0, 978-1 and 979-8: the LC third, under YAZ), unless the configuration already has a line for the same prefix. The DNB resolver is described in The DNB resolver, the LC one is a target of the Z39.50 resolver (The Z39.50 resolver). Google Books is not added (the author's decision: a key for every user for little gain, ResearchJournal.pdf).

For a GTIN: MusicBrainz, then the BnF by EAN (Discogs, when a token is configured, is not written). Google Books is not used: it refuses a request without a key (HTTP 429, a quota of zero), and a key is a registration for every user; it was tried with a key and left out (ResearchJournal.pdf). UPCitemdb's free trial answers a title, a brand and a category for a DVD, nothing to catalogue it by, and its terms are unread: it is not used (the author's decision, ResearchJournal.pdf), and a DVD is entered by hand until a library catalogue answers by UPC (FutureResearch.pdf; the Library of Congress's SRU has no index for it). The Romanian national library states a Z39.50 service in UNIMARC and does not give its address (FutureResearch.pdf).

The sources as they answer (real answers, kept as test fixtures in `FakeSources.h`):

Open Library the edition (after one redirection, which the client follows) is JSON as documented; an unknown ISBN is a 404 with an HTML body, whatever the extension asked for; so every 404 is "not found" and its body is never parsed.

The BnF SRU, `recordSchema=dublincore`, the query `bib.isbn adj "ISBN"` or `bib.ean adj "EAN"`. The BnF indexes the ISBN as it was printed: the Simondon of the sample is found by its 13 digits and not by its 10 (0 records), so an ISBN-13 is asked first and, if the BnF has no record for it and it begins with 978, its ISBN-10 is asked in a second request (a record of an older book may hold the 10 digits only). The EAN index covers books as well: the same book is found by `bib.ean` of its ISBN-13. One record per edition, but a Dublin Core one: the title ends with the author's name; a creator is "Family, Given (dates). Role" with the role in French; the publisher is "Name (Place)"; the series is a description that begins "Collection :"; the ISBN is the identifier "ISBN 9782081231412"; the barcode a description "Code à barres commercial : EAN …". `BnfResolver` turns them into the normalized record; a diagnostic of the service, a page that is not an SRU answer, or an HTTP error make the source unavailable, zero records "not found". The BnF does not hold the Mozart CD.

MusicBrainz `/ws/2/release/?query=barcode:EAN&fmt=json`: the releases that carry the barcode. A disc entered more than once comes back more than once, with other credits or labels: records with the same lower-cased title, year and medium are one. The medium is `sound`, or `video` for a DVD, a Blu-ray or the like. A UPC-A is asked with and without its leading zero.

Records that two sources would make ambiguous are not compared: the first source that knows the code decides, as for every class.

Each source has its own queue, so that its rate limit holds whatever the number of scans waiting: one request a second for Open Library and MusicBrainz, which ask for it, and the same, until known, for the others. Every request carries a `User-Agent` naming `sapupra`, its version, and the contact address set in the configuration. A Z39.50 query, which YAZ makes by a blocking call, runs in a thread of its own and reports to the event loop.

A record found is normalized (title, subtitle, contributors and their roles, publisher, place, year, languages, extent, identifiers, series, subjects); the answer of the source is kept as received, as a source record. The record is also the form of an edition written by hand (kind `record`) or chosen among several (kind `choice`): a JSON object with `title` required, and `medium` (`book`, `sound`, `video` or `other`), `subtitle`, `contributors` (objects with `name` and `role`, or names), `publisher`, `place`, `year`, `languages`, `extent`, `series`, `subjects`, `isbn13`, `gtin` and `issn`.

Open Library, the first source, is asked for `/isbn/ISBN-13.json` (it redirects to the edition), then for each author of the edition (`/authors/OL...A.json`, at most 10; an author it does not know is skipped, any other failure makes the source unavailable), one request a second. It answers 404 for an ISBN it does not know. Its `/api/books` is not used. The edition and the author as it answers are test fixtures: an author is an object with `name` (composed, `Imre Kertész`) and `personal_name` (which may not be), and the name is

read from `name`. Some editions have no `authors` at all and only `works` (the Rushdie of a trial, 9780805053081): the author is then a property of the work, so the resolver asks for `/works/OL...W.json` (one more request) and reads `authors[].author.key` there; a work that is not found leaves the edition without an author, any other failure makes the source unavailable. The edition keeps both answers as source records.

Every request of the sources reached by HTTP ends after a timeout (60 seconds, `timeout`), asks for no HTTP/2 (`http2`) and says `Connection: close` (`keepalive`): Qt warned, once, of a connection that the server had closed. These are settings, for Open Library timed out three times out of three in the trial of 0.13.0 and the cause is not known (ResearchJournal.pdf). With `--debug` each request is logged with its status, its duration and the protocol used, and a failure says how long the source was waited for.

The order is written in the configuration (`[sources]`): `isbn.default` for an ISBN, `isbn.978-2` and the like for one registration group (the first digits of the ISBN-13, hyphens or not; the longest prefix that fits the ISBN decides), `gtin`, `ismn` and `issn` for the others, with no source by default. A source that is not one of this version is an error at start. The same section sets the way every source is asked: `timeout`, `http2`, `keepalive`. When several records come from one source, or a source answers after another, the first source that knows the code decides: several records make the input `ambiguous`. The records (five at most) are in the verdict and, since 0.20.0, kept in the database (the table `candidate`, written in the same step as the status), so that the entry module can offer them at any later time, a restart of its own included. The page of the scan lists them with their year, publisher, place and extent, each with a button; the choice is sent as a `choice` input that completes the scan (never as an ISBN to look up, which would be ambiguous again), and the worker catalogues that record. A typed ISBN that turns out ambiguous shows the same list on its page of result, and the choice completes the scan that it completed. The ISBN of the scan is given to the edition when the record has none, so that the next copy is found. The source's answers as received are not kept for such an item, only the record chosen.

6.4 The Z39.50 resolver

Z39.50 is a binary protocol of the libraries, older than the web: the client connects, searches an index with an attribute (the ISBN is the Bib-1 use attribute 7) and asks for records in a syntax (MARC 21 or UNIMARC). The national libraries of Hungary and Italy speak nothing else of what they publish, and they know the books that Open Library does not. `Z3950Resolver` (a `Resolver`, so the worker asks it as the others) wraps YAZ's ZOOM interface:

- the address is `host:port/database`: the OSZK at `tagetes2.oszk.hu:1616/B1` (MARC 21), SBN at `opac.sbn.it:2100/nopac` (UNIMARC), both changeable in `[sources]` (`oszk.address`, `sbn.address`);
- a request is a connection, one search for each form of the number, and the first five records. The forms are the ISBN-13 and the ISBN-10 of the same book (978 only),

in the order the target likes: the OSZK finds the ISBN-10 and not the ISBN-13, SBN the ISBN-13 (both as seen); the second form is asked only if the first found nothing;

- ZOOM blocks, so the request runs in a thread of its own, which hands the result to the event loop (a queued call to the resolver); the resolver waits for its threads when it is destroyed, and a server that is silent is given up at the timeout of the sources (60 seconds at most). The resolver ignores `SIGPIPE`, which YAZ may raise on a closed connection;

- the records are fetched as MARCXML, which YAZ makes from either syntax after converting the character set, so that the reading of a record is one code for both; `Marc` reads MARCXML and, for the tests, ISO 2709 (the records as the servers send them), and maps a record to a `Record`;

- the mapping differs by flavour. MARC 21: the title in 245 `$a` (`$b` the subtitle), the ISBN in 020, the authors in 100 and 700 (the OSZK writes the forename in `$j`, a local subfield, where others write "Family, Given" in `$a`), the publication in 260 or 264, the extent in 300, the languages in 041 (a text of several three-letter codes written one after the other, "hunhun") or the 008, the series in 490, the subjects in 6xx. UNIMARC: the title in 200 `$a` (`$e`), the ISBN in 010, the authors in 700 to 712 (the family name in `$a`, the forename in `$b` after a comma, the role as a code in `$4`: 070 author, 730 translator), the publication in 210, the extent in 215, the languages in 101, the series in 225 or 410, the subjects in 606. The medium comes from position 6 of the leader (printed text a, sound i and j, video g). The punctuation of ISBD at the end of a value (" /", " :", ",", a full stop that does not end an abbreviation) and the non-sorting markers (C1 controls) are removed;

- a server may hold several records for one number: SBN has three editions of the Eco's *Il pendolo di Foucault* under one ISBN-13, of different years. The first is the book (it is the one SBN lists first, and the one with the fullest description in the sample), the others are kept with it as source records. Offering all of them as an ambiguous verdict would ask the author to choose at each scan of a book he owns in one edition only (since 0.20.0 the page for such a choice exists; the author decided to keep the first record, the choice being provisioned for an explicit demand, FutureResearch.pdf);

- a failure of the connection or a diagnostic of the server (the Bib-1 diagnostics, such as 114, unsupported use attribute) is "unavailable", with the reason: the worker goes on to the next source and tries the code again later if none knows it. A server that answers with no record is "not found" for that source.

The tests (`MarcTest`, `Z3950ResolverTest`) use the records that the two servers sent, kept in `FakeMarc.h`, and, for the resolver, a Z39.50 server of the tests, `Z3950Backend`, built with YAZ's server library: it serves them over the loopback with the protocol, two databases, and the refusals, the silence and the non-MARC record that the tests need. Where YAZ

has no server library, those tests skip. What was never seen is a library answering the worker: the first scan of a Hungarian or an Italian book is the trial.

6.5 The DNB resolver

The Deutsche Nationalbibliothek answers SRU at `https://services.dnb.de/sru/dnb` without a key. `DnbResolver` is a `Resolver` on `HttpResolver`, like the BnF's, and asks for `searchRetrieve` in the schema `MARC21-xml`:

- a code: `isbn=ISBN-13`, at most 10 records. It handles ISBNs only;
- a search: `tit="..." and atr="..."` (title and author), at most 50 records read and the printed ones kept, at most 10;
- the envelope is checked: a diagnostic of the service or an answer that is not SRU is "unavailable" with its text, a count of zero "not found";
- the records are MARC 21, read by `Marc` as the OSZK's and the LC's. The DNB's own habits, seen on its answers (`dnb-kafka.xml`, `dnb-none.xml`): non-sorting marks (C1 controls) in 245 `$a`, removed; several 264 (publication, production, copyright), of which the one with a place is kept; the publisher also written as a 710 with the relator `pbl`, which is not a contributor and is skipped; 653 holding noise (the publisher again), which is not a subject, so only the genre (655) is kept;
- a search drops the online editions (338 `$b cr`, or 007 beginning `cr`); a query by ISBN keeps what the ISBN is of. Records with the same title, year and publisher are one.

The answer is kept as `marc21-xml`. What was not seen is the service answering a German ISBN of the user's library, and the search by author and title live; its terms of use are unread (FutureResearch.pdf).

The LC target of the Z39.50 resolver is `lx2.loc.gov:210/LCDB`, syntax `usmarc` (MARC 21), the ISBN as attribute 7 without hyphens, the ISBN-13 first. The test record (`loc-rushdie.mrc`) is the LC's answer for 9780805053081: the ISBD punctuation is removed, the subtitle ("a novel") is read from `$b`, the subjects from 650 and 651.

6.6 The source of a description

Every answer of a source is kept in `source_record`, in the order of the lookup, so the last row of an edition is the source that knew the code. `Catalogue::description` returns the origin of the edition (`scan`, `by hand`, …) and the source and the time of that last row. The page of a copy shows them in a row "Source of the description": the name of the source (`Vocabulary::source`, in the language of the page) and the date, or, for an edition written by hand, its origin.

6.7 Statuses

Table 8 lists the statuses of an input, and what follows each.

Table 8: The statuses of an input

Status	Meaning	Next
`received`	recorded, not yet settled	classification
`resolved`	an item is catalogued	none
`repeated`	same code, same device, just before	none
`not-found`	every source answered: unknown	the entry module
`not-isbn`	a book's barcode is not its ISBN	the entry module
`ambiguous`	several records	a choice, entry module
`unavailable`	a source did not answer	retried automatically
`unrecognized`	not a code that can be looked up	the entry module
`located`	the location of the device is set	none
`labelled`	the label of an item was scanned	none
`placed`	a place was made, renamed, merged or deleted	none
`removed`	a copy was removed from the catalogue	none
`restored`	a removed copy was put back	none
`corrected`	the description of an edition was corrected	none
`unknown`	a place or an item that is not made	make the place, or check the label
`found`	a search answered, with its records	a choice, entry module
`another-copy`	a known edition scanned again, held (`copies=ask`)	confirm or dismiss, entry module
`dismissed`	set aside by hand	none

An input in `unavailable` is retried after 1 minute, 10 minutes, 1 hour, then once a day; after a week it becomes `not-found`, with its history; a source that did not answer is told to the scan device once ("queued", status `unavailable`), the retries are silent, and the verdict that settles the input is published when it comes. `not-isbn` is given to a GTIN that no record or film source knows: it is often a book whose barcode is a product code (the Romanian book of the sample), and the entry module asks first for its ISBN.

6.8 Repeated scans

Two copies of one edition are two items. But a continuous scanner can read one barcode twice; the same code from the same device within 10 seconds (configurable) is taken for a repetition, and creates no item. A second copy is scanned after another item, or after the

delay. A code whose earlier input is waiting in the to-do list, scanned again, is answered "set aside, already waiting", and adds nothing to the list: it is recorded as `repeated`, with the waiting input in its detail. The verdict of that second scan has the status of the waiting input (`not-found`, `not-isbn`, `ambiguous`, `unavailable`: the scan device sounds its error, the item is not there) and the text; in the database its status is `repeated`. A code that the history of Binary Eye holds, imported again, is found by the scan device's own time and answered "already recorded". Typing the ISBN of a book whose barcode was set aside (kind `isbn` with `for`) catalogues the edition under the ISBN, records the barcode as its GTIN, resolves the barcode's input to the same item, at the location that was current when the barcode was scanned; the next copy, scanned by its barcode, is a copy of the edition without any request.

With `[worker] copies ask` (the default), the scan of a code whose edition has an item already, outside that window, is not catalogued: it is held with the status `another-copy` and the edition (`input.edition`), and the entry module asks. An input of kind `copy` (with `for`) then catalogues it as another copy, at the location the scan had; an input of kind `dismiss` (with `for`, valid for any scan that has no item) sets the scan and itself `dismissed`, and deletes nothing. With `yes` every such scan is a copy at once; with `no` it is answered "already catalogued" (`repeated`). Only a `scan` is concerned. A scan of a code whose lookup is running, outside the window, is parked until that lookup ends, and then treated as above: without it the second scan would find no item yet and make a second one (the race of the trial of 0.15.0).

6.9 Search by author and title

An input of kind `search` carries a query (`title`, `author`, one at least). The worker asks every source of `[sources] search` that `canSearch` (Open Library by `/search.json`, the BnF by SRU with `bib.title all` and `bib.author all` joined by `and`), one after the other, gathers their records (ten at most, one ISBN-13 once) and answers `found` with them, or `not-found`. A source that fails is named only when no other found anything. Nothing is catalogued: the entry module shows the books and the chosen one comes back as an input of kind `choice`, with its record and, if a scan was being settled, `for`. A work of Open Library lists the ISBNs of all its editions; the record carries the first valid one, which may not be the edition in hand.

The second try. When no source gave a record, the title has more than two words and the query has no card number, the worker asks the sources of `search` once more with the two longest words of the title (the first of the equals, in their order; `Worker::relaxedTitle`) and the author as typed, and the verdict says ("with fewer words", or "nor with fewer words"). The input keeps the query as typed. Not a third try: a wrong surname is not guessed.

The card number. A query with `lccn` (the number as typed, alone: the entry module refuses it with a title or an author, and the worker would ignore them) is asked of the sources of `[sources] lccn` (`loc` by default) that `canSearchLccn`, that is the Z39.50 target with `mLccn`: the Library of Congress, by the use attribute 9 with `Code::normalizeLccn` of the number (letters of the prefix, the year, the serial number padded to six digits: `73-1830`

is `73001830`). A number that is not one is `unrecognized`. A worker without YAZ has no such source and answers `not-found` with that reason. The fake server of the tests knows one number (`2004431148`, the card number that Open Library gives for the Kertész). The real server was asked on 2026-11-04 for `73-1830` and gave one record.

Both queries were written after the sources' documentation and seen answered in 0.18.0 (a search for Simondon, the answers kept as test fixtures): the sources overlap, so the records are deduplicated by ISBN-13 (by title and year when neither has one), and the BnF titles end with the statement of responsibility (" / Gilbert Simondon"), which is cut. The entry module, for a chosen book that has an ISBN-13, sends an input `isbn` (with `for`) instead of `choice`: the sources then describe the edition, not the work; a book with no ISBN comes back as `choice`, with its record.

6.10 Locations

The worker keeps the current location of each device: set by an input of kind `location`, from a label or from the entry module, it applies to every input of that device that follows, until the next. A location that names a place not made is refused with `unknown`, and the current location stays. An input records the location that was current when it arrived; the item it makes is at that location, unless the input carries a `place` of its own (an item typed by hand, a search, a choice): then the item is there and the location of the device is not touched. Without a current location an item is catalogued with none.

The orders of kind `place` are done by the worker, in one transaction each (`makePlace` and `movePlace` of `Catalogue`): a place made makes its parents; a rename changes the path of the place and of every place below it, and is refused if the new path exists, or lies inside the place; a merge moves the devices, items and history of the place to another that exists, and deletes it; a delete (`deletePlace`) removes the row of a place that nothing depends on: no place below it, no device there, no copy there (a removed copy keeps its place), no scan made there (the orders of kinds `place`, `remove`, `restore`, `search` and `dismiss` do not count; their reference to the place is dropped). A place with a history is merged into another, not deleted: the history is of use, and deleting it would make the scans of the past point nowhere. A refusal is `unrecognized`, with the reason; an unknown place is `unknown`.

The order of kind `device` (0.44.0) corrects the identifier of a scan device that wrote its name wrongly: its `code` is the identifier to change, its `to` the identifier it takes. It is done by the worker in one transaction (`renameDevice` of `Catalogue`): the row of the device takes the new identifier, and so does the column `device` of every input it sent, so that its history, its duplicate detection (same device, same device time, same code) and the items it made, which point to those inputs, follow. If the new identifier is a scan device already, the two become one and the location of the one that exists stays (the location of the other is kept only if the first has none). The `message` stored with each input is not rewritten: it is what was received. The manual entry of the entry module is neither renamed nor merged into. A device that does not exist is `unknown`; an identifier that is not valid or is the same is `unrecognized`. The phone goes on sending its old identifier until it is corrected in its settings: that is a new scan device, to be merged again.

7 The entry module

7.1 Pages

The entry module serves HTML pages to any browser of the local network, with forms and no script, so that the phone's browser is enough. Table 9 lists them.

Table 9: The pages of the entry module

Path	Method	What it does
`/input/ID`	GET	a scan, the scans before and after it, the forms
`/input/ID/isbn`	POST	the ISBN typed for that scan: an input `isbn` with `for`
`/input/ID/record`	POST	the description by hand: an input `record` with `for`
`/new`	GET	an item with no code: an ISBN or a description
`/new/isbn`, `/new/record`	POST	the same inputs, without `for`
`/input/ID/copy`	POST	the confirmation of another copy: an input `copy` with `for`
`/input/ID/dismiss`	POST	the scan set aside: an input `dismiss` with `for`
`/search`	GET, POST	"Look up" by author and title, or by card number (`?for=ID` for a scan): an input `search`
`/todo/discard-copies`	POST	the scans held as `another-copy` discarded at once: one input `dismiss` for each (500 at most)
`/catalogue`	GET	the catalogue: simple search, counts, latest, results by 50 (The web front)
`/catalogue/advanced`	GET	the same, with a field for every attribute
`/catalogue/item/ID`	GET	a copy: its edition and every copy of it; `?result=ID` of an input says that this input made it
`/catalogue.csv`	GET	the copies of a search, or all, as CSV
`/result/ID/choose`	POST	the book chosen among those of a search: an input `choice` (the number `n` of the list)
`/location`	GET, POST	GET goes to `/places/devices`; POST sets the location of a device among the places made (input `location`)
`/places`	GET, POST	the places with their counts and a link to each; POST makes, renames, merges or deletes a place (inputs `place`)
`/places/manage`	GET	the forms that make, rename, merge and delete a place

Continued on next page

Continued from previous page

Path	Method	What it does
`/places/devices`	GET	the scan devices with their location, and the form that sets one
`/places/labels`	GET	for each place its code and the `spplabel` command that prints its label
`/removed`	GET	the copies removed, the latest first, with their reason
`…/ID/remove`	POST	under `/catalogue/item`: the copy removed, with a reason: an input `remove`
`…/ID/restore`	POST	under `/catalogue/item`: the copy put back: an input `restore`
`…/ID/correct`	GET, POST	under `/catalogue/item`: the form of the description, filled; the order `correct` with the record
`…/ID/move`	POST	under `/catalogue/item`: the copy put at a place made: an input `label` with a `place`
`/result/ID`	GET	the verdict of an input sent; it loads again until it comes, and leads to the page of the item when the input catalogued one (303)
`/import`	GET, POST	the history of Binary Eye: the form; the preview
`/import/go`	POST	the confirmation: the inputs are published
`/import/TOKEN`	GET	the progress of an import

to do the scans that need attention (`not-found`, `not-isbn`, `ambiguous`, `unrecognized`), oldest first, each with its location, its time, and, on its page, the scans of the same device just before and after it. `unavailable` is not in the list: the worker retries it by itself and makes it `not-found` after a week, so it is shown apart, as waiting for a source. A code typed in the search field finds the waiting scan, written as it was scanned or as an ISBN with hyphens;

an input what was read, and three ways to settle it: type an ISBN; search by author and title, with a choice among the records found (the list is on the result page, kept in memory with the latest verdicts: after a restart the search is made again); enter the record by hand. And one to set it aside (dismiss). A scan held as `another-copy` is not settled but decided, by a page whose two buttons say what each does (“Add it as copy N”; “Discard this scan, add nothing”); when several are held, the to-do page offers to discard them all (the history of Binary Eye imported again);

a new item the same two ways, for an item with no code, for the device named (a list, or another name typed), and, if wanted, a place for that item (a list of the places made, the empty choice being the device’s current location). The place is the item’s only:

it is sent as the `place` of the input and the location of the device does not change. The same choice is on the forms of an input and of a search;

search the search by author and title without a scan, for the device named;

places four pages, one idea (0.33.0; the author found the single page of 0.31.0 dense). *The places*: the places with the number of items at and below each, each a link to what is there (the address is built in one `arg()` call: see Inserting texts into messages for the fault that a chain of them once made of it). *Make, rename, merge, delete*: the four forms; make a place (its parents with it), rename one, merge one into another, delete one that nothing depends on; each is an input of kind `place`, sent to the worker. *Scan devices*: each device with its current location, and the form that sets one among the places made (with none made, the page says to make one first: it is the condition of the first scans; an input of kind `location`). *Labels*: for each place its code (the prefix of the locations and the path) and the command that prints its label, and one command for all of them. The same command is offered, folded, on the page of the result of a search by location. A line of links leads from one page to the others. Before 0.31.0 the location of a device had a page of its own, beside the places; they were joined, and are now four pages of one section, and `/location` leads to the devices;

the manual entry the pages of the module send their inputs as from a pseudo scan device, `entry` (the key `default_device`, which the pages call *manual entry* and not by the identifier, the identifier being what the database stores: the same rule as for the keywords). Its forms (new item, look up, set a location) choose it unless the person names another device: until 0.33.0 the list began with the device first in alphabetical order and the module's own was added at the end, so that a person who had set the location of the manual entry created an item that took the location of another device, or none. A device other than the default one is shown by its identifier;

look up the page does not take an ISBN, and says where to type one (*New item*, or the page of the scan it is for);

the page of a copy the number of the item (a row of the description: it is the one of the label and of the address), the edition, the source that described it, the copies of the edition (a table of the location and the day each was entered; with several copies a first column gives the number, a link for the others and *this one* for the page's own; with one copy there is nothing to tell apart and the column is not there), and the forms that act on this copy: move it to a place made, remove it with a reason. When the input that made the item was typed in the module, the page is where its result leads, and begins with a line that says so. A copy removed shows when and why, and the form to put it back. The list of the removed copies is `/removed`, reached from the home of the catalogue (the link appears once a copy was removed).

The navigation bar of every page has the buttons *Catalogue*, *To do*, *New item*, *Look up*, *Places* and *Import*, and the foot of the to-do list says what each is for: the first version of the module had them as plain links, and the author did not find them. The list of the scans in the database is read, not kept: the page shows what the worker wrote at the time of the request.

7.2 The look of the pages

(0.25.0) The look is a style sheet of about 5 kB, in `Html.cpp`, sent inside every page: no script, no image, no font file, nothing fetched from outside. The model is the calm, flat look of KDE's Breeze, which the author prefers; only its colours are taken, as the variables of a light scheme and of a dark one, the dark one chosen by the browser's own setting.

The frame a bar at the top (the name and the six entries, the entry of the page marked, fixed at the top of a long page on a wide screen, not on a phone, where the entries are two rows of three), a card with the notice, the title and the body, and the foot with the version. The page owns an entry by its title: the page of a scan is of the to-do list, a result is of the look-up, a page that no entry owns marks none.

Compact the fields of the search forms are in a grid that wraps (columns on a desktop, one on a phone); the tables have a small head in capitals, a hairline between the rows and aligned figures.

What a phone leaves out below 40 em the columns that another page repeats are hidden (the publisher and the medium in a table of copies, the device in the to-do list), and the to-do list is a list of cards (the code and the link to settle it, the status, the time and the location) where a wide screen has a table.

Large targets by pointer, not by page a touch screen gets taller buttons, links and fields, a mouse gets compact ones; the dense catalogue and the one-handed to-do list need no two designs.

Checks `HtmlTest` checks the bar, that nothing is fetched and that the sheet stays under 7 kB (past that, a route of its own with a cache header is due); `EntryTest` checks the grid, the buttons and the marked columns. The look itself is judged by eye: it was rendered in a headless browser at 390 and 1280 px, in both schemes, against a catalogue of 24 editions.

7.3 What it publishes

Inputs of kind `isbn`, `record`, `search`, `choice`, `copy`, `dismiss`, `location` and `place`, with `for` when they complete an input; `scan` and `location`, with the origin `history`, for an import. They are published like the gateway's: QoS 1, on the topic of the device, kept in memory until the receipt comes and published again with the same identifier if it does not while the worker is online (`InputPublisher`). It reads the database only to show it.

Its answer to a form is not the verdict, which the worker may take seconds to give (a lookup of an ISBN at the sources): it is a redirect, "303 See Other", to `/result/ID`, which shows the verdict if it is known (the module remembers the latest four thousand; the database, for an older one) and otherwise reloads itself every second. The reload of the page, or the back button, then resends nothing.

The identifier of the input is the token that the form carried, a UUID made when the form was displayed: a form sent twice (a double click, a reload of the answer to a POST) is the same input, which the module does not publish twice and the worker would record once. Typing an ISBN in a form that was completed by someone else is refused before it is sent: only a scan with no item yet is settled (and the worker refuses it again: `checkCompletes`).

7.4 Importing Binary Eye's history

(Since 0.30.0 the preview lists the places that the file names and that are not made, with a box, checked, that makes them when the import goes: the orders `place` are published before the scans. Unchecked, the locations in the file that name an unmade place are refused, and the scans after them stay where the device was.) When the gateway or the broker was absent, the scans that Binary Eye could not send are still in its history, which is kept by default (its setting *Save scan history*). They are recovered in four steps.

1. On the phone, in Binary Eye: *History*, then *Export to file*, then *JSON file*, and a file name; the file is written in the phone's `Download` folder. Each scan in it is an object with, among others, `_datetime` (the time of the scan, as `2026-10-26 14:59:12:237`), `format` and `content`, read in Binary Eye's source (`JsonExport.kt`, `Scan.kt`, version 1.75.5), and confirmed by the export of the trial of 0.13.0. All its values are strings, and the scans come latest first. The format has the name of the export (`EAN13`), not that of the request (`EAN_13`): the module writes the second, as the gateway receives it.

2. In the phone's browser, the entry module's page *import*: the file, chosen with the form's file field; the scan device it comes from, chosen among the known ones or typed (the history does not say); and, optionally, the time from which to import. The form is sent as `multipart/form-data`; the entry module's server takes bodies up to 4 MiB for it (`max_upload`), some ten thousand scans.

3. The entry module reads the file, sorts the scans by time, and shows what it would publish: the scans not yet recorded, and the location labels among them, each with its time, its code and its class; the scans already recorded are counted, not listed. A scan is already recorded when an input of the same device has the same `deviceTime` and the same code: the gateway writes in `deviceTime` the `timestamp` that Binary Eye sends, the same text as the `_datetime` of its history. A location label is recorded when an input of the same device has its `deviceTime`. The plan is kept in memory (the latest four) until it is confirmed.

4. Confirmed, it publishes them, in the order of their times, as inputs of the kind `scan` or `location` (a code with the location prefix), with the origin `history`, the history's object in `raw`, its `_datetime` in `deviceTime`, and as `received` the time of the scan, read in the machine's time zone. So that each scan gets the location that was current when it was taken, it first publishes the last location label of the history before the first scan it publishes, if there is one; and, at the end, the location that was the device's before the import, so that scanning goes on where it was (only if the import set a location). It publishes fifty inputs at a time, the next ones as the worker answers, so as not to flood the broker with a long history, and shows the verdicts as they come, by status; what fails joins the to-do list as any scan, and what waited for a source is retried by the worker.

The worker keeps its own guard: an input of origin `history` whose device, `deviceTime` and code are those of an input already recorded is recorded as `repeated`, and creates no item. Importing the same file twice is therefore harmless, and so is importing a file that the gateway had partly received.

7.5 Access

The prototype asks for no password: the entry module is reachable from the local network only, like everything else. A password, or the restriction to some addresses, can come later (Access from outside the network). It runs under the account of the worker, so that it reads the database as the worker's own readers do (The files of the database); it opens the file read-only and never writes the catalogue.

7.6 Access from outside the network

Everything is meant for the local network, and nothing in the programs defends against anything else. What an outsider who could reach the pages would find:

no identity the entry module and the gateway ask nobody who they are. Any visitor could settle a scan, enter an item, move, remove or restore a copy, make, rename, merge and delete places, import a file of up to 4 MiB, and publish inputs on the broker through the pages. The consultation alone is harmless to the data and still shows the whole catalogue, and *All of it as a CSV file* takes it in one request;

no encryption the pages are plain HTTP, so that on the internet everything that is typed or shown is read on the way, and a password that a proxy would ask for would be read too;

the broker it must never be reachable from outside: its passwords are the only protection of the topics, and the pages of the entry module and the gateway are its clients;

the load a pattern of the search is a regular expression tried on every copy (some 400 ms at 20000 copies, A catalogue to try the programs on); the module is single-threaded,

so that a few visitors who ask for patterns slow every page, and a pattern that backtracks a lot makes a page wait.

The solutions, from the one that asks the least of the programs:

1. A virtual private network into the local network (WireGuard on the router or on the machine): nothing is opened to the internet but the port of the VPN, authentication and encryption are those of the VPN, and from outside everything works as at home, the scans of the phone included. This is the recommendation when it can be had; no program of `sapupra` changes.

2. An SSH tunnel (`ssh -L 8081:server:8081 user@home`), for a visit from one machine: the same, with the program that is already there.

3. A program that consults and does nothing else, `sppconsult`, behind a proxy that authenticates and encrypts. A proxy alone, with a list of the paths and methods that pass, was the first idea; it protects what the list protects, and a mistake in the list publishes a page that writes, so that the program itself is made to refuse what it must not serve, and the proxy has one thing to do.

The consultation program. It is the class of the entry module in a mode that has: no connection to the broker, so no password and no file of the secrets; a route table of four paths (the catalogue, the advanced search, the page of an item, the CSV file) and the root path that leads to the catalogue, GET only, and a 404 that repeats nothing of the request for every other path; no form that changes anything (the page of an item has none, a removed copy is not found), no command that prints a label, no devices and times of the scans of an item, a bar with the catalogue alone and a footer without the version; a limit on the length of a field (200 characters) and on the duration of a search that tries patterns (3 seconds: the copies tried so far are the answer, and the page says it was cut); smaller limits of the HTTP server (a request of 4 KiB of headers and 1 KiB of body, 32 connections, 10 seconds); the headers Content-Security-Policy (nothing but its own style), nosniff, no referrer and no frame on every reply. It listens on 127.0.0.1 and warns in the log if told to listen elsewhere. It is a program of its own, `sppconsult`, not an option of `sppentry`: its service has its name, its unit shuts it in (`NoNewPrivileges` and the like), and it cannot be started with the broker's password by a slip of the command line. Because the database is readable by every account (The files of the database), it can run under an account that can do nothing else. **The proxy.** Caddy or nginx: a proxy of our own, a part of the consultation program (TLS with `QSslServer`, available since Qt 6.4, a password, a limit of the rate), was weighed and not made.

for one program and one configuration file, in the format of the others; nothing more to install, the same on Debian and FreeBSD; a part that the test suite can test, which a proxy configuration written from a manual cannot;

against the code that defends the door would be ours, young, and written to be light: the TLS and the password are the part of a program that should be written the least,

and read by the most people; the HTTP server of the programs is bounded but it is one thread and a small parser written for a local network, which the open internet tests otherwise; the certificate is the real difficulty and no program of ours removes it: a certificate from a certification authority needs the protocol ACME and a renewal every three months, which is a program of its own (certbot, or a proxy that does it), and a certificate of our own makes each browser and phone warn until it is told to trust it; the ports below 1024 (443) need a privilege or a forward of the router; the updates of the library of TLS come from the system, those of our code from us alone.

The proxy is Caddy: a file of a few lines (a name, a user and the hash of a password, the address of `sppconsult`) in which the certificate is obtained, renewed and served, and the visitors of the port 80 are sent to the port 443, by itself. It adds one program to the machine and leaves ours without a door to defend. nginx stays in the manual for whoever runs it already; it does the same with more lines, a certificate from `certbot`, and a limit of the rate that Caddy lacks without a plugin. What the password does not stop is a visitor who has it, who costs at most the duration of a search at a time, the program being single-threaded. Both configurations were run here against the program (Manual.pdf). The things to refuse in any case: opening the ports 8080, 8081 or 1883 of the router to the internet, and moving the pages to a public host with the database on it (a copy of `catalogue.db` made with `sqlite3 .backup` on a host of its own is a safer public catalogue than a door to the machine at home).

8 Several users at once

This section discusses the concurrency considerations of the design: what several people scanning and acting at the same time share, and what protects them from each other, by the shared thing.

The database one writer, the worker; the entry module, the web front and Datasette read it (WAL: a reader does not block the writer and the writer does not block a reader; the readers wait up to 5 seconds for a lock). Two workers are refused at start by a lock (Recording first; before 0.33.0 nothing prevented it).

The order the worker takes the inputs one at a time from one event loop, records each with the next sequence number and the location of its device at that moment, and settles them with the database in transactions; nothing is shared between two threads but the Z39.50 queries, which report to the loop.

Two scans of one code from one device or two, while the first lookup is under way, the second waits for it (Repeated scans) and is then a copy or not as the setting says; a code that is waiting in the to-do list makes the later scans of it wait with it. One lookup for one code. Tested (`WorkerTest::severalUsers`).

Two scans of two codes independent. Each item takes the location of the device that scanned it, not of the last one to scan.

Two decisions about one scan the first is taken, the second is answered `unrecognized` (“not waiting”): another copy and set aside at once make one or the other, never both (tested). A removal and a restoration of one copy are two orders on one row, taken in the order they arrive.

A form sent twice the identifier of the input is the token of the form: a double click or a reload is one input.

The pages the module answers from one event loop; each page is what the database held at the time of the request. Nothing is kept for a user between two requests but the latest verdicts, in memory, by the identifier of the input.

What is not protected, and cannot be by the worker:

- The device is the unit of location. Two people who use the same device identifier share its current location: one sets it to the salon and the other to the bedroom, and the next item of either goes where the last one said. Two phones have two identifiers and do not meet. The pages of the module are one device for everybody, the manual entry, and its location is everybody’s. What each browser has of its own, since 0.38.0, is a place: a cookie (`sapupra_place`, the path of the place percent-encoded, kept a year, `HttpOnly`, `SameSite=Lax`) that the forms of the items typed by hand (the ISBN, the description, the search) offer as the place of the item, in the list that is otherwise “the device’s current location”; it is set when such a form is sent with a place, and on the page of the devices, where the first line forgets it. The server keeps nothing for the browser: the cookie is a suggestion, used only if it names a place that is made (a cookie that names another, or that is not a path, is ignored), and the person sees what is chosen before sending. It does not change the location of the manual entry, which only a scan of a label or the form of the devices sets, nor what the phones do. The module that consults sets and reads no cookie.

- Two people who name the same new place at once: both orders reach the worker, one makes it and the other is told that it was made already; nothing is made twice.

- The pages carry no sign-in (Access): whoever reaches the port acts, and nobody is told who did what but by the device.

9 Inserting texts into messages

A chain of `QString::arg()` calls is a fault when a text from outside comes before another call: a text inserted by one call that holds a `%2` is taken for a place to fill by the next call. The link of a place whose name holds a slash is `...salon%2Fcaisson...`, and the next

`arg()`, which filled the count, turned it into `salon6Fcaisson`: the search by that link found nothing, and the page that said so showed the damaged name. The calls where a text came before another `arg()` are one call with all the texts (the places, the facets of the catalogue, the messages of the worker and of the sources, which carry scanned codes and the error strings of the network); `EntryTest::deepLocation` keeps the case. The rule for new code: a message with several markers is filled by one `arg()` call.

10 When a part is absent

Table 10 says what the scan device shows when a part is missing, what is kept, and how the scan is recovered. The rule is that a scan is either recorded, or said not to be, at once and aloud.

Table 10: What happens when a part is absent

Absent	Shown	Kept	Recovered by
gateway	“failed to connect”	Binary Eye’s history	importing it
broker	“Not recorded”	Binary Eye’s history	scan again, or import
worker	“queued, worker offline”	the broker’s disk	the worker’s return
sources	“queued”	the database	automatic retries
entry module	nothing	the database	its to-do list

“Shown” is what the scan device shows; in the first two cases, it also sounds its error.

The gateway absent the scan device cannot connect. Binary Eye then sounds and vibrates its error (any failure of the request, as for a status outside 200 to 299: `CameraActivity.kt` of its source) and shows the failure. The scan is not lost if Binary Eye’s *Save scan history* is on, which is its default: its history is then imported in the entry module (Importing Binary Eye’s history).

The broker absent the gateway is reached but cannot hand the scan over: it answers 503, “Not recorded, scan again”, which makes the error heard. The same recovery as above.

The worker absent the broker keeps the inputs for it on disk (Delivery), and the gateway, which knows from the worker’s presence that no verdict will come, answers “queued” at once.

The sources absent the input is recorded and waits as `unavailable`, retried by the worker (Statuses).

The entry module absent nothing is lost; the failed scans wait in the database for it.

11 The language of the pages

The pages of the entry module and the one sentence that the gateway gives the phone are in English or in a language that has a catalogue; the language is a key of the configuration file, read by the two programs at start. Nothing else is translated: the log is English so that a report can be read by anyone, the manual pages are documents, the data are in their own languages, and the texts that the worker stores in the database (the detail of a verdict) are the worker's, shown as they are.

11.1 The sentences

The code is written in English, each sentence whole, between `_( )`: the sentence is its own key. The values go where `%1` to `%9` are, `QString::arg` fills them, so that a language can put them where its grammar wants (the French writes "emplacement : %1", with a space before the colon). A count is written for its plural, `_n("%n copy", "%n copies", count)`, and the catalogue gives the forms that the language has; `%n` is replaced by the number. Three macros more: `N_( )` marks a text that is kept to be translated later (a table of labels), `T_( )` a title that a function receives (the English `QString`, which `Html::page` translates to show and keeps in English to find the entry of the navigation that is current), and the empty message is never translated (a catalogue holds its header under it). There are 286 sentences; no sentence is built by gluing words (the old "N cop" and "y" or "ies" are gone). The markup stays in the sentence, which is long where a form is ("<form …>"): the translator keeps the tags and the places as they are, and a test checks it.

11.2 The catalogues

`make pot` writes `sapupra.pot`, the list of the sentences, by `xgettext`; `LL.po` is the catalogue of a language, text, one entry for each sentence, read in diffs and edited by hand or by any gettext editor; `msgfmt` compiles it to `locale/LL/LC_MESSAGES/sapupra.mo` in the build directory, and `make install` puts it in the locale directory of the installation. `make update-po` merges a new list into the catalogues; `make check` runs `check-translation`, which refuses a catalogue that lacks a sentence of the code, that keeps one that the code no longer has, or that has an untranslated or a fuzzy sentence, and `TranslationTest`, which reads `fr.po` and refuses a translation that changes the tags, the places or the line ends of its sentence. The tools are those of the package gettext, needed to build and not to run; `configure` looks for them (`--with-gettext=auto|yes|no`), and without them the catalogues are not made and the pages are in English. The tree does not use `AM_GNU_GETTEXT` and the directory `po` of the convention, which bring a set of macros and rules (and expect `libintl` at run time) to do what some forty lines of `Makefile.am` do here.

11.3 The reading of a catalogue

The programs do not use `libintl`. `Translation` (in the common library) reads the `.mo` file itself: the table of the English sentences and the table of their translations, in either byte order, the plural message being "singular, a zero byte, plural" and its translation the forms one after the other; the header gives the charset, which must be UTF-8, and the rule of the plural forms, a C expression of `n` that a small evaluator reads (the ternary operator, the comparisons, `&&`, `||`, `%`, the arithmetic): the rules of French, English, Russian and Arabic are in the tests. A file that is not a catalogue, is cut, or has a rule that is not understood is refused with the reason, and the pages stay in English.

The reason is measured, not guessed. With the library of the C language, the language of the messages is given by the locale of the process and by `LANGUAGE`, which counts only when the locale is not "C". A service is started without `LANG`, so its locale is "C"; a program may set one with `setlocale`, but only a locale that the machine has generated: on this workspace (glibc 2.39) `C`, `C.utf8` and `POSIX` are the only ones, and with `LANGUAGE=fr` a program that had set `C.UTF-8` was answered in English. The first version of `Translation` was built that way (it chose the language's own locale, else `C.UTF-8`, else `en_US.UTF-8`, and set `LANGUAGE`) and failed its own French test here, as it would on a bare server. A program that reads its own catalogue needs neither the locale, nor the library (on FreeBSD, which has `libintl` in a port, nor that), nor `LANG`. The price is about two hundred lines and the format of `.mo`, which is simple and stable; what is not read is the context of a message (`msgctxt`, not used) and a charset other than UTF-8.

11.4 Choosing the language

`[interface] language=fr` in the configuration file; empty or `en` is the English of the code. The code is two or three lowercase letters, and perhaps a region (`fr`, `pt_BR`): it is a name of a directory, so that nothing else is accepted. A language without catalogue keeps the English, and the log says which file was looked for and which languages are installed; the programs start all the same, since a page in English is better than no page. The `lang` attribute of the pages follows the language. The catalogue directory is that of the installation (`$(localedir)`, defined at build as `SAPUPRA_LOCALEDIR`); a test uses the build directory.

The choice is one for the whole service, since there is one author on a local network. The header `Accept-Language` of the browser is the natural next step (a language for each visitor, a catalogue loaded for each), and is not built.

11.5 The words of the data

A page says two kinds of words: those of the program (a title, a button, a notice), which the catalogue translates, and those that the database holds. If the database holds the word "book", a French page shows "book" under a heading that says "Support", and a catalogue entered with the English pages shows "book" while the one entered with the French pages

would have held “livre”. The rule that keeps one language on the screen is that the database holds *keywords* and never a word of a language, and that the page says the keyword in the language of the page. The statuses of an input (`received`, `queued`, `dismissed`) were already so since 0.27.0 (`statusWords`); 0.28.0 does the same for the rest, in `Vocabulary` (the common library): the medium (`book`, `sound`, `video` and `other`), how an edition was described (`resolved`, `copied` or `by hand`), the role of a contributor (author, translator, illustrator, editor, composer, performer, foreword, preface, photographer, director) and the language of the text. Each is a table of the keyword and its English words, marked `N_( )` and so in the catalogue of the language; a keyword that the table does not know is shown as it is, so that a value that a source brought is never lost nor wrongly translated.

The languages are the one case where the sources do not use keywords: they write `fre` (the code of ISO 639-2, bibliographic or terminologic: `fre` and `fra`), `hun`, or the name, `English`. The table has forty-five of them with their two codes (and the codes for several languages and for an undetermined one); the page says the name in the language of the page (`fre` and `French` both become “français” on a French page), and the search follows: a language is asked for by its code, its English name or its name in the language of the page (“hongrois”), and the catalogue looks for all of them, so that records written in either way are found together. What is not in the table is shown as written. The database is not changed and no record is rewritten; a file made by any earlier version reads as before.

The same rule holds for what 0.31.0 adds: the kind of an input (`Vocabulary::kind`: scan, location, label, place, removal, …) and the reason of a removal (`Vocabulary::reason`: donated, sold, lost, entered by mistake, other). The author asked whether a catalogue begun with English pages must be searched with *book* from French pages. It does not: the medium is a keyword and the search takes the keyword and the words of the page alike; the places where a keyword was still shown raw were the kind of an input, in the lists of inputs and on the page of an input, and they are translated now. What no table can translate are the words the author typed (the names of places, a title, a note): they are his, in his language, as a label on a shelf is.

What stays in the language of its source, on purpose: the titles, subtitles, subjects, publishers, places, series, notes and the extent (“159 p.”), the names of persons. They are the descriptions of the book, not words of the program; a subject translated by the program would be a heading that no thesaurus holds. The catalogue of a library in several languages shows them in the languages of its books, which is also what a reader of the shelf sees. A dynamic translation of them was considered and set aside: it would need an outside service (against “everything on the local network”) or a table of subjects, and would be wrong exactly where a subject is particular. What may be done later, if the mixture of `fre` and `French` in the stored data is judged a nuisance in the exports, is to write the code at the time of the record (the worker), a change of the worker alone.

11.6 The French

The words, for whoever corrects them (in `fr.po` alone): an item, alone or counted, is a *document* (the author’s remark: *exemplaire* is the word of a set, “3 exemplaires de ce

document", and a counter that says "6 exemplaires de 4 éditions" counts documents); a *copy* of an edition, where several are meant, is an *exemplaire* (the copies of the edition on the page of a copy, the confirmation of another copy). The English follows: an item is counted as *items* and *copy* is kept for the set (a message catalogue has no context, so the two senses are two sentences). A scan is a *lecture* and a scan device a *lecteur* (no program is named in the sentences, but for the history that only one program writes), the broker is the *courtier* and the worker the *travailleur*, a location an *emplacement*, the to-do list *la liste des actions*, to dismiss *écarter*, the catalogue *le catalogue*, a search that was made *recherché* and not *cherché*, a field *un champ* and a regular expression *un motif* (the words of the page for the reader who is not a computer scientist, on the author's word), what is typed *entré* or *saisi* and never *tapé*. The punctuation follows the French (a space before a colon), and the quotation marks are the guillemets. The writer is not French, and the sentences were not read by one: the author's reading, which is that of a French speaker, is the test. The header of `fr.po` carries no version of its own: `Project-Id-Version` is `sapupra $Revision: 98 $`, which Subversion expands once the property `svn:keywords` is set to `Revision` on `fr.po` and `sapupra.pot` (a command of the author's, `svn propset svn:keywords Revision`), and `make pot` writes it so.

12 Location labels

`spplabel` prints labels: for each code given, the code drawn in the chosen kind, with its text written in clear under it, all on pages of SVG, A4 by default, ready to print and cut. It is written (0.29.0). The codes are drawn by libzint, the library of Zint (BSD licence; `libzint-dev` on Debian, `graphics/zint` on FreeBSD), and the program is built only where the library is found (`./configure --with-zint=auto|yes|no|DIR`), as the pages are translated only where gettext is found. Where the library is in a place the compiler does not search (FreeBSD's ports put it in `/usr/local`, and Zint's package ships no `.pc` file: the report of a user), `configure` tries the compiler's own places, then `/usr/local` and `/opt/local`, with `-isystem` and `-L` and an rpath; `--with-zint=DIR` names the prefix and tries that alone. Zint 2.15 and 2.16 have the functions and the fields used. It is a command run by hand, at the printer's side: it reads no database, starts no service and needs no broker; of the configuration file it reads the two prefixes, `[locations] prefix=` and `[items] prefix=`, and nothing else. The label of a location holds the path that the author chooses, and the database learns the location at its first scan (Where the locations are). Table 11 gives the options. Those marked "labels" are the ones that the labels of our own add (Labels of our own); the others are the location labels, the ones that the author asked for first.

The layout is the program's own and the encoder is Zint's: `Labels` (in the common library) makes the pages from rectangles, whatever draws them, and `Barcode` (with `spplabel` alone) asks Zint for the rectangles of a code. So `LabelsTest` tries the layout with an encoder of its own, and only `BarcodeTest` needs the library. A label is the code, with a quiet zone of four modules (at least two millimetres) round it, and under it a heading in bold (the

last part of a path; the number of an item; the content of a code given by `-c`) and, smaller, up to three lines of detail (the places that contain the location, separated by a slash; the text of `--text`). A text that is too long for its place is shortened with an ellipsis and the program says so on its standard error; the code, which holds all of it, never is. With no grid asked for, a label is 45 by about 36 millimetres for a code of 20 millimetres, so that an A4 page holds 28 of them, four across and seven down.

Table 11: The options of `spplabel`

Option	Meaning	Default
`-l`, `--location PATH`	a location label: the code holds the location prefix and PATH in its normal form, as `LOC:salon/bookcase 2/shelf 4`; the last part is written large, the places that contain it under it; given once for each label	(none)
`-c`, `--code CONTENT`	a code holding CONTENT as it is, CONTENT in clear; a location label made so must hold the prefix itself	(none)
`--item N` (labels)	the label of item N: `ITM:N-c` (c the check digit), the number in clear	(none)
`--items FROM-TO` (labels)	the labels of a range of items, as `100-140`, at most 5000	(none)
`--text LINE` (labels)	a line in clear under the label given just before it, as a short author and title, so that a label that has come unstuck is still known; not after `--items`	(none)
`-k`, `--kind KIND`	the kind of code: `qr`, `datamatrix`, `aztec` (two-dimensional), `code128` (one-dimensional, for a narrow edge, ASCII and Latin letters only); the four are read by the scan devices and hold letters	`qr`
`-e`, `--ecc LEVEL`	the error correction of a QR or Aztec code, `L`, `M`, `Q` or `H` (Table 12); `Q` for a label that will be rubbed	`M`
`-s`, `--size MM`	the side of a two-dimensional code, or the height of a Code 128, in millimetres, 5 to 150	20
`-p`, `--page SIZE`	`A4` or `Letter`	`A4`
`--columns N`, `--rows N`	the grid of the sheet, to match a sheet of adhesive labels: the labels are the cells of the page inside the margin	as many as fit

Continued on next page

Continued from previous page

Option	Meaning	Default
`--skip N`	leave the first N places blank, for a sheet already partly used; less than the places of a page	0
`--margin MM`	the margin of the page	10
`--no-frame`	no dashed line round each label	a frame
`--plain`	the code alone, black shapes on nothing, one SVG file of exactly the size asked for each label, for a slicer or a laser cutter	off
`-o`, `--output FILE`	the SVG file; with several pages, FILE-1.svg, FILE-2.svg and so on	standard output
`--config FILE`	the configuration file, for the prefixes `[locations] prefix=` and `[items] prefix=`; a file asked for that cannot be read is an error, the usual one that cannot is a notice	the usual
`--location-prefix P`, `--item-prefix P`	the prefixes, which win over the file; they must not begin alike	`LOC:`, `ITM:`
`-h`, `--help`, `--version`	as every program of the package; the version also says that of libzint	

Listing 1 gives examples: two location labels, a location label as a Code 128 for a narrow edge, a label with the labels of our own, a range on a sheet already partly used, and the code of an item alone, for a plate. Two options would read the database, read-only, and are left for later if they are wanted: `--from-database` (the text under an item's label taken from the catalogue instead of `--text`) and `--all-locations` (a label for each location that the catalogue knows).

Listing 1: Examples of `spplabel`

```
$ spplabel -l 'salon/bookcase 2/shelf 4' -l 'salon/bookcase 2/shelf 5' -o shelves.svg
$ spplabel -k code128 -c 'LOC:CAVE 1'
$ spplabel --item 1234 --text 'Kabos: Az érdekes Ujság Décameronja'
$ spplabel --items 100-140 --skip 6 -o items.svg
$ spplabel --item 1234 --plain -s 20 -o plate.svg
```

The SVG is in millimetres, on a white page, with the text in a sans-serif font that the viewer chooses (the widths that decide where a line is broken are estimated, so a viewer with a wider font may crowd a text; the code is never affected). It was tried by decoding what it draws: a sheet of 28 labels rendered at 300 dots to the inch, and the codes of the four kinds, with accents, with ideographs and at the four levels, read back by two decoders (zbar and zxing-cpp), and codes of 10, 12 and 15 millimetres (Making spplabel).

13 Labels of our own

Designed on the author's questions. He chose its first question first: the label holds the prefix and the number of the item (option A, the options); he then took the other three as proposed (the journal): a scan of a label shows the item and puts it at the current location of the device; the items that get a label are those that have no barcode; under the code, the number and an optional short author and title. Two stages are built (0.29.0 and 0.30.0); the inventory pages are the third.

13.1 The problem

Most items have a barcode, and the barcode is that of the edition: the scan makes a copy of it. Some items have none, or none that says anything: an old book, a booklet, a copy whose barcode was cut off. Such an item is entered by hand (a record typed, or an ISBN typed) and gets a number and the location of the scan device, and that is all; it has no mark by which a later scan can find it. So it cannot be put somewhere else by scanning, nor counted in an inventory. And two copies of one edition, even with a barcode, cannot be told apart by it. A label of the catalogue's own, as libraries put on their books, gives the item what the edition's barcode cannot: a code that is the item's alone.

13.2 The code

Chosen (option A of the decision): `ITM:`, the number of the item in decimal, a hyphen and a check digit, as in `ITM:1234-4`. The prefix is the same device as `LOC:` of the locations: a letter and a colon cannot begin an ISBN or a product code, so that a label is never taken for a book, nor a book for a label; and `ITM:` is not the start of `LOC:`, nor the reverse (checked at start). It is configurable (`[items] prefix=`, beside `[locations] prefix=`). The check digit (Luhn, as in EAN; Verhoeff, which finds every swap of two neighbouring digits, is the alternative, a detail of the work) finds a misread or a mistyped number, which the gateway then answers as it answers an ISBN with a wrong digit, "scan again", and which nothing writes. The number is the item's own, `item.id`, so that nothing is added to the database.

The other forms were weighed in the decision: an EAN-13 of the range 20 to 29, which the stores keep for their own use, would be read by any scanner but says nothing by itself and may be a product; a serial printed in advance and bound to an item at its first scan lets one print a sheet at once and stick as one goes, at the price of a binding step and a column `item.label` (a change of the schema, not made). The check digit is Luhn's, with the rightmost digit of the number doubled; `Code::checkItemLabel` says whether a text read is not a label, a label misread (a hyphen where it should not be, a letter, a leading zero, a wrong digit) or good, and it was tried by changing each digit of a number in every other way, which it always finds.

13.3 Two ways to tie a label to an entry

The author put the two ways that there are.

1. *The key.* The item is entered in the database first, by hand, by its ISBN or by a search in the sources; a label is then printed that holds what ties the item to its entry, its number. A scan of the label is a scan of that key.

2. *The content.* A label is printed that holds all that the database needs to create the entry (an author, a title, a year, a publisher); a scan of it carries that content to the worker, directly or through the broker, and the entry is created from it.

The first is the proposal, and the reasons are these. The second needs the same work as the first (the data must be typed somewhere to be printed, and the catalogue is the place where they are checked, completed by a source and found again), and gives it up again: the label holds a copy of the entry, which a correction in the catalogue leaves as it was; the code is large (the Kabos book of the author's shelves, with the publisher, is about 90 bytes, a code of 41 modules at level M, Table 12, against 21 for a number); and, above all, a scan of the label of an item already in the catalogue (to move it, to count it) would create a second entry, since the label does not say which entry it is. To say it, it would have to hold the key as well, and it is then the first way with more on it. Its merits are real: a label can be printed before the database exists, by another machine, and survives the loss of the database. The first way has the second merit too if a line of clear text goes under the code (`--text`): a label that has come unstuck is still a book that can be named, and a lost database is rebuilt from the labels read by eye.

The path of the second way exists, all the same. A scan reaches the gateway as a code, the gateway publishes it, the worker classifies it; a code of a kind that carries a description, handed to the entry module's own kind `record`, would create the entry. It is kept as a possible exchange format (a book passed on with its label, a description copied from another system), to be built if a use comes, and is not the way to label one's own shelves. The decision lists it as option D.

13.4 What a QR code can hold

A QR code (Denso Wave, 1994; ISO/IEC 18004) is a square of black and white *modules*. Its *version*, from 1 to 40, fixes the side: 17 + 4 × version modules (21 for version 1, 177 for version 40), with a quiet margin of four white modules around it. Part of the square is fixed (the three big squares at the corners that tell the reader where the code is and how it is turned, the timing lines, the alignment patterns); the rest holds the data and, with them, the code of Reed and Solomon that lets a reader rebuild what is dirty, torn or hidden. The *level* of that correction is chosen: L restores about 7 % of the code words, M 15 %, Q 25 %, H 30 %; the higher the level, the less room for data. The data are written in one of four *modes*, and the reader chooses, for each part of the text, the densest one that holds it: numeric (digits only, about 3.3 bits a digit), alphanumeric (the digits, the capitals

A to Z, and the nine signs space, `$ % * + - . / :`, 5.5 bits a character), byte (any byte, in practice the UTF-8 of the text, 8 bits; a letter with an accent takes two) and kanji. A single lowercase letter takes the whole text to the byte mode, which is why the prefixes are in capitals. Table 12 gives the capacities, by Denso's table.

Table 12: What a QR code holds: characters of the alphanumeric mode at level M, bytes at the four levels

Version	Modules	Alphanumeric (M)	Bytes L	Bytes M	Bytes Q	Bytes H
1	21	20	17	14	11	7
2	25	38	32	26	20	14
3	29	61	53	42	32	24
5	37	122	106	84	60	44
6	41	154	134	106	74	58
10	57	311	271	213	151	119
40	177	3391	2953	2331	1663	1273

Numeric digits at level L go from 41 (version 1) to 7089 (version 40); in the alphanumeric mode, level L, from 25 to 4296. For the labels: `ITM:1234-4` is ten characters of the alphanumeric set, so it is a version 1 code of 21 modules, even at level H (ten alphanumeric characters are its capacity). `LOC:SALON B3` is also version 1 at levels L to M; with a lowercase letter, `LOC:Salon B3` is twelve bytes, version 1 at level M. A location name of thirty characters is version 3. What limits a label is the size of its module and not the standard: a phone held at the length of a hand reads a module of half a millimetre, so a code of 20 mm is comfortable up to about 40 modules (version 6, about a hundred bytes at level M) and a code of version 1 has a module of a millimetre. The standard allows nearly three thousand bytes; no label of the shelves needs a tenth of that.

13.5 Where the locations are

In the database, not in the configuration file. The table `location` (`id`, `name`, unique, `created`) holds the names. The name is what follows the prefix of the label: the path `salon/bookcase 2/shelf 4` for the label of `LOC:salon/bookcase 2/shelf 4`. Three columns refer to it: `device.location`, the current location of each scan device, which a location label changes; `item.location`, where the item was when it was entered; and `input.location`, the location of the device when each input arrived, which keeps the history. The configuration file holds only the prefix (`[locations] prefix=`, `LOC:` by default), because the gateway and `spplabel` must agree on what a location label looks like. So `spplabel` needs no database for a location label.

Who creates a location? The author, on purpose (option C of the author's decision, the journal). A place exists once a row of `location` is made for it, by an order of kind `place` from the page *Places* of the entry module (or by the import of a history); making

a place makes its ancestors, `salon/bookcase 2` and `salon` for `salon/bookcase 2/shelf 4`. Nothing makes a place implicitly: a scan of a label whose place is not made is refused (`unknown`, and the phone sounds its error), and so is a location typed for a device, so that a misspelt label is found when it is stuck. The same page renames a place (the paths below it change with it, in one transaction) and merges one into another (the items, the devices and the history move, the old row goes); these two are what a misspelling needs. Printing a label creates nothing, and a label cannot be used before its place is made. This is also the bootstrap condition the author named: a device has no current location until one is chosen, in *Location* or by scanning a label of a made place, and an item catalogued before that has none.

Three columns refer to the table: `device.location` is the current location of the device, which the scans use; `item.location` the place of the item; `input.location` the place in force at the time of the input, or the item's own place when it was typed with one. The entry module offers the places made to choose from (`Catalogue::places`) and shows them with their counts.

A location is a path, from the largest place to the smallest, the parts separated by a slash, as the author put it: a room has several bookcases, each many shelves, `LOC:room/bookcase/shelf`. Apart from the prefix the content is free (the only character that has a meaning is the slash). The normal form, which `Code` gives and everything uses, has no empty part, nothing round a part and one space where there were several inside it, so that a label written `LOC:salon / bookcase 2/shelf 4` is the place of `LOC:salon/bookcase 2/shelf 4`. Capitals count: `Salon` and `salon` are two places, as before. The hierarchy is read from the names and stored nowhere: the parent of a path is the path up to its last slash, so that no table and no column is added for the hierarchy, and a place is said to be within another when its path is that other or begins with it and a slash. Items are at the exact place scanned; a search for a place finds the copies there and below it (*salon* finds the shelves of its bookcases, not *salonette*), and the counts of the catalogue list every place that contains one, with the number of copies at or below it. Renaming a room is a change of every name that begins with it, which `Catalogue::movePlace` does in one transaction.

What a path costs on a label: `LOC:salon/bookcase 2/shelf 4` is 28 bytes (a lowercase letter takes the code to the byte mode), a code of 29 modules at level M, so that a module of 20 mm is 0.7 mm (Table 12); the same in capitals is alphanumeric and 25 modules. Three levels of a few words each stay far below the 40 modules that a phone reads comfortably at 20 mm.

13.6 What a scan of a label does

The gateway recognises the prefix as it recognises the location prefix, and publishes an input of kind `label` (Table 4) with the number; it is not sent to the sources. The worker records it as every input, and then:

- the item exists: the verdict is `labelled` and says the item (title, authors, place), the phone shows it; the item takes the current location of the scan device, if it has one,

as an item that is scanned takes it (the verdict says “put at X (was at Y)”, “already at X”, or “no location is current” when the device has none). Scanned again at the same place, it changes nothing; the history of where it was is the inputs, each of which keeps the item and the location of the device, so that no table is added;

- the item does not exist: `unknown`, and the scan device sounds its error (a label stuck on the wrong thing, a database restored from an older copy). The item page lists the places where its label was scanned (`Catalogue::visits`: the time, the device, the place), read from the history.

The sentence that the phone shows is one more of the catalogue of the gateway’s, in the language of the pages.

13.7 Making the labels

`spplabel` draws the codes, with the text in clear (Table 11). `--item`, or `--items` with a range such as `100-140`, draws `ITM:N-c` for a number N, with the number in clear under it; `--text` adds the line that the author wants under the number (a short author and title). The page of an item shows its number, its label code and the command to run, with the author and the title already quoted for the shell. No database is read by `spplabel` and no library is added to the entry module (libzint stays with `spplabel` alone).

The size: a QR code of 20 mm (the default of `spplabel`) is read by a phone at a hand’s length; Code 128 is the choice for a spine or a narrow edge. For an old book, the place of the label is the author’s concern and the library’s practice a guide: not on the paper, which an adhesive harms, but on a slip put in the book, or inside the cover; `spplabel` only draws the sheet. A slip of card is the usual one; the author’s own idea is a thin plate of PETG of 0.6 mm with the code printed on it in two colours by a multicolour printer (a Prusa XL, a CoreOne INDX), which `--plain` serves: the SVG of a code of exactly the size asked, black shapes without a margin, which a slicer extrudes, the quiet zone being the plate’s own (at least four modules of white round the code, so a plate of 28 mm for a code of 20). What the question of acid comes to is in the research journal (PETG in a book).

13.8 The stages

Each is a delivery, in this order, after the decision.

1. Done in 0.29.0: `spplabel` itself, with the options of Table 11 (the location labels, `--item`, `--items` and `--text`); the label code in `Code` (made, read and checked); the page of an item with its number and the command. Nothing changes for the gateway and the worker.

2. Done in 0.30.0: the gateway and the worker: the kind `label`, the verdicts `labelled` and `unknown`, the move; the page of an item shows the places where its label was scanned.

3. The inventory: for a location and a date, the items there that a label scan has seen since and those it has not. An item with a barcode that has no label cannot be seen by an inventory, since its scan would make a copy: so the inventory sees the items with a label, which are those that have no barcode (the decision's second question); an item with a barcode is counted by the catalogue, not by the inventory.

The tests follow the existing ones: `CodeTest` for the code and its check digit (written), `LabelsTest` and `BarcodeTest` for the sheets (written), `GatewayTest` for the kind and the refused misread, `WorkerTest` for the move and `unknown`, `EntryTest` for the pages.

14 Removing

Designed and built in 0.31.0, on the author's question: a book is donated, or was catalogued by mistake, and the catalogue must let go of it; the same for places and perhaps other things.

14.1 Options

- Delete the row. Simple, and wrong here: the inputs that made the copy name it (`input.item`), the history of the scans and of the labels with it, and the label of a donated book may still be stuck on a book that comes back. A deletion cannot be undone, and a mistaken removal (a tap on the wrong copy) is easy.

- Mark the copy removed, with the time and the reason, and leave it where it is (chosen). Everything that counts or lists copies leaves it out; nothing is lost; a copy put back is as it was. The reason is kept because it is the one thing that tells later why a book is not on the shelf: donated, sold, lost, entered by mistake, other (keywords, said in the language of the page).

- A second table of withdrawn copies. The same, with a move of the row and of what points to it: more code for no use.

14.2 What a removal is

An input of kind `remove` with the number of the copy in `code` and the reason, made by the form on the page of the copy; the worker alone writes (`Catalogue::removeItem`). The copy keeps its row, its edition, its place and its history; it is out of the view `catalogue`, of the counts by place and by medium, of the number of copies of an edition (so that `copies=ask` does not ask about a copy that is gone), of the searches and of the CSV file. Its edition stays: when the last copy goes, the edition remains, and a scan of its barcode finds it and catalogues a first copy again without asking a source. A `restore` (the form on the page of the copy, reached from `/removed`) puts it back. The label of a removed copy, scanned, is answered `unknown` with the words "was removed (reason) on date", and moves

nothing. A removal of a copy that is removed already, or a restore of one that is not, is `unrecognized`; a copy that does not exist is `unknown`.

A `correct` replaces the description of the edition of an item (`Catalogue::correctEdition`, one transaction): medium, title, subtitle, publisher, place, year, languages, extent and series; the contributors and subjects are written again (`writeParts`, shared with `insertEdition`, which keeps the sort name of a name that stays and completes it from the record) and the contributors left with no edition are deleted; the row of the full-text index is made again. The ISBN is replaced only if the origin of the description is *by hand*: for a description from a source it is the key by which the book was found, with the product number and the ISSN. The `source_record` rows are not touched. The verdict is `corrected`, with the number of copies, and its input (the latest of the kind) gives the date shown on the page of the item (`Description::mCorrected`). An item that does not exist is `unknown`, a record with no title `unrecognized`. The form is the form of the description by hand, filled (`Entry::recordForm` with a record), so that there is one form for both.

14.3 Places, editions, devices

A place made by mistake is deleted (Locations); one with a history is merged. An edition has no removal: it is a description, and it is out of every list as soon as it has no copy. A scan device cannot be removed, since every one has the inputs it sent; if the list of devices in the forms grows with names typed by mistake, the remedy is to hide the devices with no input for a long time, which is not built (FutureResearch.org). The inputs themselves are never removed: they are the history, and an input set aside is `dismissed`, not deleted.

A purge, the real deletion of a copy that was catalogued by mistake and that no history should keep, is not offered. The reason `mistake` hides the copy from every list; if a real deletion is wanted it can be added for a copy removed with that reason, the inputs of the copy being made anonymous rather than deleted.

The moving of a copy by hand is the same `label` input that a scan of its label makes, with a `place`; the worker records the place in `input.location` and puts the copy there.

15 The database

15.1 The files of the database

A database in WAL mode is a main file, `catalogue.db`, and two files beside it that SQLite makes when a connection opens it: `-wal`, the log to which the writer appends what it commits, and `-shm`, the index of that log in shared memory, which the connections map. Readers read the main file and the log through the index. The log is moved into the main file by a checkpoint: SQLite does it by itself when the log reaches 1000 pages (4 MB), and when the last connection closes, which also deletes the two files. A catalogue of some hundreds of books never reaches 1000 pages, so that everything it holds is in the log until

a connection closes: the main file may be of a single page for weeks, and a copy of it alone is empty.

The worker closes with a checkpoint `Catalogue::close`, for a connection that writes, runs `PRAGMA wal_checkpoint(TRUNCATE)` before closing it: the whole log goes into the main file and is emptied, even while another program holds the database open. After the stop of the worker the main file alone is the catalogue.

What stays on the disk the last connection to close deletes the two files only if it can checkpoint, which a read-only connection (the entry module, `spplist`) cannot. So if the entry module is stopped after the worker, the two files stay (the log empty, the index of 32 KB); they are harmless and the next opener reuses them. Stopping the entry module before the worker leaves the main file alone. A worker killed leaves a log that is full, and the next opener recovers it.

Who reads a connection that is only a reader needs to read the log and the index, and to map the index; if it may not create them (the directory is closed to its account), SQLite can still read them if they are there and can be read, and refuses if they are not (`attempt to write a readonly database`). So a directory that every account may enter (mode 0755), and files that every account may read (0644), let any account run `spplist` while a service has the database open. When no program has it open the two files are not there, and an account that cannot create them cannot read the file: `Catalogue::openReadOnly` then opens the main file alone with the SQLite option `immutable`, which ignores the log and the locks. That is sound because no log file exists (or it is empty): the main file is the whole catalogue. If a log with content exists and cannot be read, the error says how to run the program as the account of the worker. A worker that starts while `spplist` reads cannot spoil the reading: the main file changes only at a checkpoint, which the worker does at 4 MB of log or when it stops.

A copy by an ordinary account `sqlite3 file .backup copy` opens the file for writing, as SQLite does for every connection (the option `-readonly` changes nothing for a database in WAL mode), so that it fails with `attempt to write a readonly database` where `openReadOnly` would use the option `immutable`; the output file is made before the failure, hence an empty copy. The copy of the main file is right only if no log with content exists: `cp` and the option `immutable` both ignore the log, and a log with content (the worker running, or killed) means a catalogue that is old, without any error. The Manual gives the cases (Backing up the catalogue); the safe recipe for every case is the command under the account of the worker. A tool that decides by itself, as `openReadOnly` does, is possible and not made.

The modes the database directory is 0755 and the files 0644 (the unit sets the mode of the state directory to 0755 and the umask to 0022; the rc script inherits the umask 022). The catalogue is not a secret from the local network, since the entry module

shows it to anyone who can reach it; it is not one from the accounts of the machine either. The directory of the database is open to enter, not to write: the account that cannot write in it cannot change the catalogue.

15.2 Principles

One SQLite file, written by the worker alone, read by the entry module and the web front. It runs in WAL mode, where readers do not block the writer; the readers open it read-only. Its version is the `user_version` of SQLite, and the worker upgrades an older file at start. Version 2 (0.17.0) adds the column `input.edition`, version 3 (0.20.0) the table `candidate`, version 4 (0.30.0) the ancestors of the places already stored (a place is made, with its parents), version 5 (0.31.0) the columns `item.removed` and `item.removal` and the filter of the view `catalogue`, version 6 (0.44.0) the column `contributor.sort_name`; a file of an older version is upgraded by the worker, all its steps in one transaction that is undone as a whole if one fails, and the entry module, which opens the file read-only and cannot upgrade it, refuses it and says to start the worker first.

15.3 Tables

Listing 2 is the schema; an *edition* is a description shared by the copies of one publication, an *item* one copy.

Listing 2: The schema of the database

```
CREATE TABLE device (
    id        TEXT PRIMARY KEY,           -- device identifier
    location  INTEGER REFERENCES location(id),
    updated   TEXT NOT NULL);
CREATE TABLE location (
    id        INTEGER PRIMARY KEY,
    name      TEXT NOT NULL UNIQUE,       -- text of the label or entry
    created   TEXT NOT NULL);
CREATE TABLE input (
    id        TEXT PRIMARY KEY,           -- the UUID of the message
    seq       INTEGER NOT NULL UNIQUE,    -- order of arrival
    device    TEXT NOT NULL REFERENCES device(id),
    kind      TEXT NOT NULL,
    origin    TEXT NOT NULL,
    received  TEXT NOT NULL,
    code      TEXT,
    symbology TEXT,
    addon     TEXT,
    device_time TEXT,                     -- the scan device's own time
    class     TEXT,                       -- from the classification
    normalized TEXT,                      -- ISBN-13 or GTIN-13
```

```
    location  INTEGER REFERENCES location(id),
    completes TEXT,                      -- the input it completes
    repeats   TEXT,                      -- the input it repeats
    status    TEXT NOT NULL,
    detail    TEXT,
    attempts  INTEGER NOT NULL DEFAULT 0,
    retry_at  TEXT,                      -- UTC, ISO 8601
    first_try TEXT,                      -- UTC, ISO 8601
    item      INTEGER,
    edition   INTEGER REFERENCES edition(id), -- version 2: the edition of a scan held
    as another copy
    message   TEXT NOT NULL);            -- the input, as received
CREATE TABLE edition (
    id        INTEGER PRIMARY KEY,
    medium    TEXT NOT NULL,             -- book, sound, video, other
    title     TEXT NOT NULL,
    subtitle  TEXT,
    publisher TEXT,
    place     TEXT,
    year      TEXT,
    languages TEXT,
    extent    TEXT,
    series    TEXT,
    isbn13    TEXT,
    gtin      TEXT,
    issn      TEXT,
    origin    TEXT NOT NULL,             -- resolved, copied, by hand
    created   TEXT NOT NULL,
    updated   TEXT NOT NULL);
CREATE TABLE contributor (
    id        INTEGER PRIMARY KEY,
    name      TEXT NOT NULL UNIQUE,
    sort_name TEXT);          -- "Family, Given" where a source said it
CREATE TABLE edition_contributor (
    edition     INTEGER NOT NULL REFERENCES edition(id),
    contributor INTEGER NOT NULL REFERENCES contributor(id),
    role        TEXT,
    position    INTEGER NOT NULL,
    PRIMARY KEY (edition, position));
CREATE TABLE subject (
    edition   INTEGER NOT NULL REFERENCES edition(id),
    subject   TEXT NOT NULL);
CREATE TABLE item (
    id        INTEGER PRIMARY KEY,
    edition   INTEGER NOT NULL REFERENCES edition(id),
```

```
    location  INTEGER REFERENCES location(id),
    input     TEXT REFERENCES input(id),  -- the input that made it
    created   TEXT NOT NULL,
    removed   TEXT,                       -- version 5: when it was removed
    removal   TEXT);                      -- and why: donated, sold, lost, mistake,
    other
CREATE TABLE candidate (                   -- version 3
    input     TEXT NOT NULL REFERENCES input(id),
    position  INTEGER NOT NULL,           -- the order the source gave
    record    TEXT NOT NULL,              -- the Record, as JSON
    PRIMARY KEY (input, position));
CREATE TABLE source_record (
    id        INTEGER PRIMARY KEY,
    input     TEXT NOT NULL REFERENCES input(id),
    edition   INTEGER REFERENCES edition(id),
    source    TEXT NOT NULL,              -- openlibrary, bnf, ...
    request   TEXT NOT NULL,
    fetched   TEXT NOT NULL,
    format    TEXT NOT NULL,              -- json, unimarc, marc21, marc21-xml, dc
    body      TEXT NOT NULL);
```

The times are ISO 8601 text (`received` with its offset; `retry_at` and `first_try` in UTC, compared as text); `device_time` is the text of the scan device, as it wrote it, and an index on `device` and `device_time` finds the scans of a history already recorded. The neighbours of an input are the inputs of the same device just before and after it in `seq`. The indexes are on `(device, device_time)`, `(normalized, code)`, `status` and, for the retries, `retry_at` where the status is `unavailable`; no foreign key joins `completes`, `repeats` and `item`, which name rows the worker writes in the order of the events. The file is opened with foreign keys on, in WAL mode; a file that is not a catalogue of sapupra, or is of a later version, is refused and left as it is. The worker needs Qt's SQLite driver, with FTS5, and TLS for the sources: `configure` tries them.

15.4 Full-text index

An FTS5 table, `edition_fts`, holds for each edition its title, subtitle, contributors, publisher, series and subjects, as one row whose `rowid` is the edition's. The worker writes it in the same transaction as the edition; with the contributors and subjects in other tables, a row written by the program is simpler than triggers. The pages of the web front do not query it (0.26.0: the field that searched it, *Anywhere*, was dropped); it is kept for Datasette, whose full-text search of a table uses it, and for what may need it later.

15.5 A view for the web front

A view, `catalogue`, gives one row per item with its edition, its contributors as one text, its location and its date of entry; it is what the pages of the web front query (and what Datasette shows first, for whoever runs it). It leaves out the copies that were removed (`item.removed` is not null), so that nothing that reads it (the pages, the CSV file, Datasette) counts or lists them. The web front opens the file read-only, but not as immutable: the worker keeps writing it.

16 The web front

The catalogue is consulted with pages of the entry module: routes that only read (Table 9), served from the same port as the rest, with no script (a form that sends by GET, links for the facets and the pages), so that a phone's browser is enough. The choice was the author's, between those pages and Datasette: no program to install beside, one service fewer, the look of the other pages, the lookups of the database in the language of the rest; against that, a few hundred lines to keep, where Datasette had them for nothing. The comparison is in ResearchJournal.pdf.

The pages read the database through the class `Catalogue`, as the other pages do, and the entry module has no SQL of its own: `items` (with the number found, for the page) and `itemsFound`, for a `Catalogue::Query`, by title or the latest first; the facets of locations, media and origins; `copies` and `editionOf` for the page of a copy. A page shows fifty copies; a page beyond the last is the last, a page that is not a number the first. The CSV file is not paged (200000 copies at most): it is the file for a spreadsheet, the fields quoted, the line ends `CR LF`. Every field that comes from the catalogue is escaped for HTML.

A field for each attribute (0.22.0). The search is a `Catalogue::Query`: a pattern for each of the attributes `patternAttributes()` lists (title, subtitle, author, translator, other, publisher, place, language, series, subject, isbn, issn, extent), a location, a medium, an origin, a year from and to, a least number of copies and the days between which a copy was entered. The address of both pages carries every field under its own name (`title`, `author`, …, `year_from`, `year_to`, `copies`, `entered_from`, `entered_to`, `origin`, `medium`, `location`, `page`), so that a search can be a bookmark and the links of the pages keep it whole. The simple form has five inputs (Title, Author, ISBN or barcode, Location, Medium) and the link to the advanced one; the advanced form, `/catalogue/advanced`, has them all in groups. Both are the same handler: a field that only the advanced form has, in the address of the simple one, shows the advanced form, so that nothing typed is hidden. The fields are not corrected or capitalised by the phone (the attributes of the input element).

What a search shows (0.23.0). The author found that after *Search* the page stayed on the fields, that a search with no hit could show the last result or the whole catalogue, and asked for a clear button and the version on every page. So: the form sends to `/catalogue#result`, and the page of a search puts the fields in a `details` element, folded, before the heading *Result* (no script), so that the browser lands on the result; a search that

finds nothing says *No hit* with what was asked and nothing else; a form sent with every field empty says so, and only the bare address is the home of the catalogue (the counts and the latest copies). *Reset* (in French *Réinitialiser*, since 0.35.0; *Effacer* did not say what was erased) is a link to the bare page of the form being shown, not a reset button, which would only restore the values the page was sent with. Every page of the module is sent with `Cache-Control: no-store`, since it shows a database that changes and a browser that keeps the page of the last search would show it for the next. The foot of `Html::page` names the program and its version (`PACKAGE_VERSION`, the one that `sppentry --version` prints).

The *Anywhere* field (0.22.0 to 0.25.0), words searched in the full-text index, is gone (0.26.0). The author did not see its use once each attribute had its own field; the argument for it, one box for someone who does not know where a word is, did not outweigh the second way to search that it asked the author to learn, with its own rules (what a word is, the star, the operators that are not). With it went `Catalogue::ftsQuery`, `Query::mWords` and the parameter `q` of `/catalogue` and `/catalogue.csv` (the `q` of the to-do list is another one). The ISBN field kept the rule that was in the field's way: a whole number is an edition, digits alone a beginning.

The order of the answer (0.34.0). The answer is ordered by the title unless the address says `sort=` and one of `Catalogue::sortKeys()` (title, author, publisher, year, medium, location, entered), and `order=desc` for the other way. The headings of the columns of a result are links that carry the search and the order: a click orders by that column, a second click on the same column the other way, a small triangle and `aria-sort` saying which. The order is made by the program, not by SQL (`NOCASE` puts `Éluard` after `Zola`): the copies that satisfy the query are read in the order of their number, and sorted with `QCollator` in the language of the pages, capitals ignored, a number read as a number (shelf 2 before shelf 10); a year is compared as a number; a value that is empty comes after the others, in either direction; ties are broken by the title, then by the number of the copy, so that a page is the same each time. Only the list of the latest copies, which is by number, is still cut by the database. The CSV file keeps the order of the search. The form of a search carries `sort` and `order` as hidden fields, so that a new search keeps the order chosen.

Seven, plus or minus two (0.34.0). A person keeps about seven things in mind at once (Miller, 1956), and the ergonomics of forms draws the rule that a screen offers no more than five to nine choices at a time, the others being grouped or folded. The pages follow it where it can be done without hiding what is asked for. The menu has six entries. The advanced search, which had some twenty fields in a grid, has four groups of at most nine controls, each in a `fieldset`: the work (title, subtitle, author, translator, other contributors, subject, medium), the edition (publisher, place, language, series, extent, years from and to), the numbers (ISBN or barcode, ISSN) and the copies (entered between two days, least number of copies, location, how it was described). The form of an item entered by hand has two, the work and the edition. A table of results has six columns. The places on the home of the catalogue were a flat list that grows with the shelves (eighty for a room); they are a folded tree, `<details>`, which shows the rooms, and the furniture, the rows and the shelves on demand, each node being a link with its count. What is not changed: the list of the places on *Places* (it is a working list, not a choice), the menu of a phone (six

entries already) and the lists of the to-do page, which are tables to read and not controls. `EntryTest::groupsOfNoMoreThanNine` counts the controls of each group of the pages and fails above nine.

The patterns. SQLite has no such search in its full-text index, nor a `REGEXP` function without a function added to the connection, which the SQL module of Qt does not offer without its driver's handle (and a direct link to SQLite); so the database narrows the copies by the lists, the numbers (the years, the copies, the days: one SQL expression each) and the program tries the patterns on what is left (`Catalogue::matching`): the values of the editions, with the roles of the contributors and the subjects, are read in bulk, each pattern compiled once, and a verdict is kept for each edition so that its copies cost nothing more. The engine is Qt's `QRegularExpression` (PCRE2): for found or not found it answers as `grep -E` does, the leftmost-longest rule of POSIX mattering only for the extent of a match; a backslash inside brackets is an escape, not a character. A pattern is tried on the values of its attribute one by one (each contributor, each subject, each language), so that `^` and `$` stand at the ends of a value and nothing crosses from one to the next; an attribute with no value (an edition with no year) is never found by a pattern, `^$` included. Capitals do not matter, and the accents are removed from the text and from the pattern alike (decomposed, the combining marks dropped). The ISBN field is the one with a rule of its own: digits, hyphens and spaces that make an ISBN or a barcode are the edition's exact number, in either form; digits alone that do not are the beginning of a number; anything else is a pattern as typed, on the ISBN-13 and on the product number. The year is two numbers; a first that is not all digits is a pattern on the year like the other fields and the second is not used (the page says so).

What cannot be asked is reported by `Catalogue::problems`, by field: a pattern that does not compile, a year, a number of copies or a day that is not one. The page then says, naming each field, that nothing was searched (an error 400 for the CSV file). There is no fallback to plain text: a search that quietly meant something else would find something else, and a field that is not valid is the typing of a moment. The catastrophic patterns of backtracking engines are possible; the module is single-threaded and for one person's local network, so the risk is that of a page that waits.

Speed. The test `CatalogueTest::largeCatalogue` writes 30000 editions (40000 copies) through a separate connection in one transaction and times every kind of search; the figures of the run of 0.22.0 are in the research journal. The searches by numbers (the years, the copies) are SQL and take some milliseconds; those by pattern read and test the copies that are left, about a second for all 40000, and the other criteria (the lists, the numbers) narrow that. A page asks for its fifty rows and the count together, so that the patterns are tried once.

The pages were tried on test catalogues (three copies; a hundred and twenty for the pages) and on catalogues of invented books made by `sppgen`, up to 20000 copies (A catalogue to try the programs on): the author's own catalogue is still to come. On 20000 copies a list in a chosen order takes 110 to 240 ms in the database, a pattern on the author 400 ms and on the title 470 ms, the copies of a place 30 ms, the counts by place 13 ms. The sorting by the alphabet of the language is done once on the whole answer, not on a page.

A page of a copy that does not exist, or whose number is not a number, is a 404; a form sent by POST to a catalogue route is a 404 too, so that nothing there writes.

Datasette, which was tried before (0.65.5 on the test fixtures, then on a sample of 33 items, 4 media and 8 locations), remains possible and is outside the build; its metadata is the sample file (`datasette.yml.sample`: the items, the editions with their search box, a stored question *search* and one for the scans to do), the Manual says how to start it (Manual.pdf). It runs under the account `sapupra` (the read-only connection of a database in WAL mode uses the `-shm` file of the worker, The files of the database), has no password and offers a `select` to anyone who can reach it. Nothing in `sapupra` depends on it, and it is not a service.

17 The printed lists

A list on paper, or a file to give, is wanted from time to time: the copies sorted by author, by title and by location. `spplist` is a command like `spplabel` (Location labels): run by hand, it reads the database (read-only, through `Catalogue::listing`, which does not need the service to stop) and writes an Org document, which Org exports to PDF or HTML like the other documents of the tree. Three tables, each of the copies that were not removed, chosen with `-t` (one, two or all): by author (author, title, location), by title (title, author, location), by location (location, author, title).

The authors the contributors whose role is author or is not given; for a work with none (a disc), those who are not translators. In the table by author a copy is under each of its authors, and one with no author comes last; in the others the authors are together.

The names a name is written and sorted as Surname, Forename. Where a source said how the name is filed (`contributor.sort_name`: the MARC records, the BnF and MusicBrainz say it; a body is filed as it is written) that is used: *García Márquez, Gabriel*, *Pink Floyd*. Otherwise the rule `Listing::inverted`: the last word is the surname, with the particles before it (de, van, von, …); a name with a comma is left as it is, and so is one word. That rule is a heuristic, wrong for a name of the Far East or a body that no source filed (Open Library does not). `--as-written` turns both off, and then the sort is by forename. A name that a source filed is not corrected by a later one that did not; one that a source did not file is completed by a later one that does.

The order `QCollator` in the language of the pages, capitals ignored, numbers as numbers; an empty first column comes last. The article that starts a title is set aside for the order only, by the language of the edition (`Listing::filing`): *the, a, an* in English; *le, la, les, l', un, une, des* in French; *der, die, das, ein, eine* in German; *il, lo, la, i, gli, le, l', un, uno, una* in Italian; *el, la, los, las, un, una* in Spanish; *a, az, egy* in

Hungarian. The first language of the edition that the program knows decides; where there is none, or it has no table, nothing is set aside. The title is written in full.

The document a header that Org understands (title, date, language, no table of contents, LuaLaTeX, A4, margins of 18 mm), a heading and a line for each table, a `longtable` so that the tables run over the pages, the widths of the columns fixed to fit the page. The cell is escaped: the bar becomes `\vert{}`, and a marker of emphasis that opens a word is preceded by `\zwnj{}`, an entity that prints nothing and that Org does not take for the start of an emphasis (a title like `*Foo*` stays as typed). The headings are in the language of the configuration file or of `--language` (the sentences are translated like the others, `Listing.cpp` being in `TRANSLATED_FILES`).

The tests `ListingTest` (the names, the cells, the three tables on a small catalogue, the order, one or two tables, the French headings); `GeneratorTest::listed` feeds the listing with a generated catalogue.

18 A catalogue to try the programs on

The searches, the pages and the lists have been tried on catalogues of a few hundred copies; the catalogue of the author will have some thousands. `sppgen` makes one: n copies of invented editions, in a tree of places, entered through the calls of the worker (a record input with its place, `Catalogue::catalogue`), so that the database is the one the programs read. It is a tool of the developer: built, not installed, and it refuses a file that exists, so that it cannot overwrite a real catalogue. It says where it is every hundredth of the copies (not more often than every thousand; `-p N` sets it), with the rate and the time left, as a million copies take hours.

The debug log of the clients. The transport, not each program, writes it, so that the gateway, the entry module and the consultation have the same lines (`HttpServer`, at the level debug): a line "connection from ADDRESS" when a client connects, a line "ADDRESS METHOD PATH -> STATUS" when the answer is sent, `refused` for what the server turns away (too many connections, a request not received in time or malformed) and "the client closed the connection before the answer" when it leaves. The query of the address and the body are not written: they may hold what was typed or scanned. An IPv4 address that came in on a dual-stack socket is written as an IPv4 address.

The data titles in English, French and Hungarian drawn from word lists, with accents (a sort that puts `Éclair` after `zebra` is found at once); authors from a pool a quarter the size of the catalogue, some of them with a particle (`Jean de Valmont`) so that the inversion of the names is exercised; a translator for one book in ten; publishers, places, years from 1850 to 2026, languages, extents, series, subjects; 88 % books, the rest sound, video and other, with performers for the discs.

The numbers an ISBN-13 in the range 979-10 with the right check digit, drawn until it is new, so that none is the ISBN of a book; the barcode of a disc is such a number too.

The places `-r` rooms (up to eight: salon, bureau, chambre, …), each with two to four pieces of furniture, each with two to four rows of three to five shelves; a shelf is written as `salon/caisson gauche/rayon 1/étagère 1`. That makes twelve to eighty shelves for each room, made with their parents; a copy is put on a shelf, some shelves holding more than others, and 3 % are left without a place. 4 % of the copies are another copy of an edition entered before, so that editions with several copies exist.

The seed the same seed, the same catalogue (`-s`); the dates of entry are those of the run.

The tests `GeneratorTest` (the check digit, the catalogue and its numbers, the same seed twice, the listing of it).

The figures of the run on a catalogue of thousands, and what they changed, are in the research journal.

19 Configuration

One file, read by the programs, each its own section and the common ones, in an INI form that the library reads itself (`Config`): a value is taken as it is written, from after the `=` to the end of the line, without the spaces around it, so that a password may hold any character; a comment is a line of its own. (Qt's `QSettings`, first used, changes a value at a semicolon, a backslash, a double quote or a comma: ResearchJournal.pdf.) Each program gives the broker the name of its account as its client identifier, so that the broker's log names the account. Listing 3 is an example. The file is read by every account (owner root, mode 0644), because the commands that people run, `spplist` and `spplabel`, read the database, the language and the prefixes in it. What is secret, the broker's passwords and the tokens of the sources, is not in it but in a second file beside it, with the same syntax and the same sections, `sapupra.secret.ini` (owner root, group `sapupra`, mode 0640), which only the services read and whose values win (The secrets). A program is given another file by `--config`, and its secrets are those of the file of that name with `.secret` before the `.ini`. Every value of the example can be changed there: the addresses and ports, the wait for a verdict, the delay before an input without receipt is published again, the repetition window, the location prefix and the item prefix (shared by the gateway, the entry module and `spplabel`), the level of each program's log (`log_level`: quiet, normal or debug, which `-q` and `-d` of the command line override, so that a service is run at its level and a run by hand asks for the debugging messages), the order of the sources. In this version `sppwork` knows the sources `openlibrary`, `bnf` and `musicbrainz`, and refuses a name it does not know, so that a misspelling is found at start. The example below names `oszk` and `discogs`, which are not written yet: `sppwork` 0.13.0 refuses them.

Listing 3: An example of the configuration file

```
[broker]
host=192.168.1.10
port=1883

[database]
file=/var/db/sapupra/catalogue.db

[gateway]
listen=192.168.1.10:8080
user=gateway
wait=3
receipt_timeout=30

[interface]
language=fr

[locations]
prefix=LOC:

[items]
prefix=ITM:

[worker]
user=worker
repeat_window=10
contact=someone@example.org

[entry]
listen=192.168.1.10:8081
user=entry
receipt_timeout=30
max_upload=4
default_device=entry

[sources]
isbn.978-2=bnf,openlibrary
isbn.978-963=oszk,openlibrary
isbn.978-3=dnb,openlibrary,bnf
isbn.default=openlibrary,bnf
gtin=musicbrainz,discogs,bnf
search=openlibrary,bnf,dnb,oszk
```

Listing 4: The file of the secrets, beside the configuration file

```
[gateway]
```

```
password=...

[worker]
password=...

[entry]
password=...
```

19.1 The secrets

The services (`sppgate`, `sppwork`, `sppentry`) read the file and then the file of the secrets if there is one (Listing 4); `sppconsult` has no secret and reads the first alone; a value of the second wins, and a password left in the first still works, so that an installation of an earlier version goes on without a change. What a person runs (`spplist`, `spplabel`) reads the first file alone. The reason is the access: the first file must be open to every account, and a password must not be. Once the second file exists, a service that finds a password or a token in a file that every account can read says so in its log (`Config::notices`), and so does it for a file of the secrets that is open to all. The pairing of the names is by the name of the file, not by a key, so that a file cannot send a service to read another one. Only passwords and tokens are secrets: a key called `password` or whose name ends in `token`.

20 Code layout

The tree stays flat. The common library, `libsppcommon.a`, built and not installed, holds `Config`, `Log` (standard error or syslog, quiet and debug; the debugging lines of the standard error begin with the time; the normal messages go to syslog at the level `notice`, which FreeBSD's default configuration keeps), `MqttClient` (libmosquitto driven by Qt's event loop through `QSocketNotifier` on its socket), `HttpServer` (see HTTP), `Message` (the inputs, receipts and verdicts, to and from JSON), `Code` (validation and classification, ISBN-10 to ISBN-13), `SignalWatcher` (the stop signals of a service, turned into Qt signals through a pipe), `Record` (the normalized description of an edition, to and from JSON), and what the entry module needs that has no database in it: `HttpForm` (the fields of a form, urlencoded or multipart), `Html` (escaping, the frame of a page), `InputPublisher` (publishes inputs and follows their receipts and verdicts; the gateway and the entry module share it) and `HistoryImport` (the history of Binary Eye, read and planned) and `Translation` (the sentences of the pages in the language of the configuration, The language of the pages) and `Labels` (the pages of labels as SVG, Location labels; `Vocabulary` too, the words of the data in the language of the page). Each program has its classes (`Gateway`; `Worker`, `Catalogue` for the database, `Resolver` with a class per source: `HttpResolver`, the common part of the sources reached by HTTP, and under it `OpenLibraryResolver`, `BnfResolver` and `MusicBrainzResolver`, and `Z3950Resolver`, with `Marc` for the records of the libraries, when YAZ is there; `Entry`), those of the worker in a library of their own, `libsppwork.a`, which

the program and its tests share, and its `main` (`sppgate.cpp` for `sppgate`), which reads its options with `getopt_long(3)`: those of the standard (`--help`, `--version`, `--debug`, `--quiet`), `--config`, and, for a service, `--syslog`. Each has a manual page: section 8 for a service (`sppgate.8`), section 1 for a command. The services go to `sbindir`; their service files are built, for the two programs, from the templates `sapupra.service.in` and `sapupra.rc.in` (the worker's own lines are marked in the unit's, and the lines that shut in `sppconsult` too), and `make check` verifies that none is left with a placeholder. `Entry` is with `sppentry.cpp` (and `sppconsult.cpp`, which sets its mode that only consults) in the program, not in a library: it reads the database through `Catalogue` (opened read-only, `openReadOnly`), which is in `libsppwork.a`. `spplabel` has `Barcode`, the one class that includes libzint's header, and is installed in `bindir`, not `sbindir`; it is the only program that is built conditionally (on libzint), and its page, `spplabel.1`, is shipped in any case. `spplist` is installed in `bindir` too, with its page, `spplist.1`; its work is done by `Listing` (the sorting of the names, the cells of the Org tables and the document), which is in `libsppwork.a` beside `Catalogue`, whose `listing()` it reads, so that a test can feed it without a program. `sppgen` is in no directory of installation (`noinst_PROGRAMS`); it is `Generator`, in `libsppwork.a` as well, behind a `main` that reads four options.

The unit tests use QtTest, which is in `qt6-base-dev`: one program per class, named after it (`CodeTest` for `Code`), run by `make check`; Table 13 says what each covers. The tests that need a broker start a Mosquitto of their own (`TestBroker`), on a free port of the loopback interface, never the machine's broker, and are skipped where there is no `mosquitto` program.

Table 13: The unit tests

Test	Covers
`CodeTest`	the sample, check digits, misreads, ISBN-10 and ISBN-13; location paths; the labels of items, each digit changed
`LabelsTest`	the labels, the pages (28 to an A4), the skip, a grid, the escaping, the wrapping, the plain mode
`BarcodeTest`	libzint's drawings: the sizes that the standard gives, the four kinds, the refusals
`ListingTest`	the names turned about, the cells (the bar, the marks of Org), the three tables, the order, one table, the French headings
`GeneratorTest`	the check digits, a catalogue of the size asked, the same seed, the listing fed with it
`LogTest`	the levels; the time on the debugging lines only
`MessageTest`	topics, identifiers, each kind, each rejected field
`ConfigTest`	values, lists, integers, booleans, listening addresses

Continued on next page

Continued from previous page

Test	Covers
`HttpServerTest`	the parser; a late answer; the time and connection limits
`MqttClientTest`	acknowledgement, retained status, session, reconnection
`SignalWatcherTest`	signals delivered, one after the other; one watcher only
`GatewayTest`	each reply, both clients, locations (unknown places), item labels, receipts, no broker
`InputPublisherTest`	the signals of an input, the worker's status, republication, `abandon`, no broker
`RecordTest`	the record's JSON, its rejected fields, its summary
`CatalogueTest`	the schema and its upgrades, a foreign file, recording, places (made, renamed, merged), copies, retries
`OpenLibraryResolverTest`	answers, failures, the rate; real editions and authors, the authors of a work
`DnbResolverTest`	the real answers: the marks, the online editions, the large print, the search; failures
`BnfResolverTest`	the real answer: record, roles, duplicates, ISBN-10
`MusicBrainzResolverTest`	the real answer: a disc entered twice, the UPC-A
`MarcTest`	ISO 2709 and MARCXML read; MARC 21 (the LC's record included) and UNIMARC mapped
`Z3950ResolverTest`	the protocol and the searches by author and title, against a server of the tests (YAZ's own)
`WorkerTest`	the whole path, from the broker to the database, places and item labels, several users, two workers (below)
`HttpFormTest`	urlencoded and multipart forms, a file, line breaks, errors
`HtmlTest`	escaping, the frame of a page, the refresh and the notice
`HistoryImportTest`	the history of the trial; the plan, the labels, the repeats

Continued on next page

Continued from previous page

Test	Covers
`EntryTest`	each page, each form, the places and their four pages, a place in a hierarchy, what is published, the import, the answer sorted by each column, the groups of controls, the folded tree of places, the pages in French (below)
`TranslationTest`	the languages, the plural rules, a bad catalogue, the French catalogue sound

The tests of the three sources reached by HTTP play the source: a fake Open Library, BnF or MusicBrainz, on the loopback interface, that answers with what the real one answered (`FakeSources.h`).

`GatewayTest` plays the worker itself: an MQTT client that answers each input with the receipt and the verdict that the test chooses. `WorkerTest` plays the gateway: it publishes inputs and reads the receipts and the verdicts, against a Mosquitto of the test, a fake Open Library (`FakeOpenLibrary.h`, shared with the resolver's test) and a database in a temporary directory. It covers the presence, the receipt before the verdict, the locations, the copies, the repetitions, the code set aside and scanned again, the unrecognized codes, the Romanian book completed by its ISBN, the ISBN-10, the records by hand and by choice, the duplicated inputs, the source down then back, giving up, the product numbers, the ambiguous answer, the order of the sources, a book that the first source does not know and the second does (the Simondon, by the BnF), a CD (by MusicBrainz), the history, the search, the recovery of the inputs not settled, the session kept while the worker is away, and the settings refused. The sample of the collection is that of ResearchJournal.pdf.

`EntryTest` plays the worker, as `GatewayTest` does, and the database: it fills a database as the worker would have (a scan not found, a product number, an unrecognized code, one waiting for a source, a catalogued item, and a location), starts the entry module on it and reads the pages over HTTP as a browser does, sends the forms, and checks what the fake worker receives (the kind, the `for`, the device, the order of a location and an item) and what the page of the result shows once the verdict has come. It covers the broker away, the worker offline, the database that is not there yet, a form sent twice, and the import (preview, confirmation, progress, a file imported twice).

The order of writing: the library and its tests (written); the gateway (written; tried against the broker alone, `mosquitto_sub` showing what it publishes); the worker (written; tried by the author with Open Library, and with the BnF and MusicBrainz added in 0.13.0, whose trial is waited for); the entry module, with the import of Binary Eye's history; `spplabel` (written in 0.29.0, with the labels of items; the recognition of a label by the gateway and the worker is the next stage).