Manual 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 User **5**
- 2.1 Scanning with an Android phone: Binary Eye . . . 5
- 2.2 Scanning with an iPhone or an iPad: AllCodeRelay . . . 6
- 2.3 At the shelf: what the phone answers . . . 6
- 2.4 Locations . . . 8
- 2.5 Printing the labels . . . 9
- 2.6 Items that need attention . . . 10
- 2.7 Removing and moving a copy . . . 14
- 2.8 Correcting a description . . . 15
- 2.9 Several people at once . . . 15
- 2.10 Consulting the catalogue . . . 16
  - 2.10.1 What the search fields accept . . . 17
    - 2.10.1.1 The patterns . . . 18
- 2.11 Printed lists of the catalogue . . . 19

3 Admin **21**
- 3.1 The machine . . . 21
- 3.2 The broker: Mosquitto . . . 21
  - 3.2.1 On Debian . . . 21
  - 3.2.2 On FreeBSD . . . 22
    - 3.2.2.1 The pid file . . . 23
  - 3.2.3 Checking the broker . . . 24
- 3.3 The gateway . . . 25
  - 3.3.1 The account and the configuration . . . 25
  - 3.3.2 Trying it by hand . . . 26

3.3.3 As a service on Debian . . . 28
3.3.4 As a service on FreeBSD . . . 29
3.4 The worker . . . 29
3.4.1 What it needs . . . 29
3.4.2 The database and the configuration . . . 30
3.4.3 Trying it by hand . . . 32
3.4.4 What it does with a scan . . . 36
3.4.5 Backing up the catalogue . . . 37
3.4.6 Upgrading to a newer version . . . 38
3.4.7 Running commands as the account sapupra . . . 38
3.4.8 As a service on Debian . . . 40
3.4.9 As a service on FreeBSD . . . 40
3.4.10 The level of the log . . . 42
3.4.11 Starting, stopping and the order . . . 43
3.4.12 The Z39.50 sources . . . 44
3.5 The entry module . . . 44
3.5.1 What it needs . . . 45
3.5.2 The configuration . . . 45
3.5.3 The language of the pages . . . 45
3.5.4 Trying it by hand . . . 47
3.5.5 As a service on Debian . . . 47
3.5.6 As a service on FreeBSD . . . 47
3.6 The web front . . . 48
3.6.1 Reaching the catalogue from outside . . . 48
3.6.1.1 The program sppconsult . . . 48
3.6.1.2 A proxy: Caddy . . . 49
3.6.1.3 A proxy: nginx . . . 50
3.6.1.4 Whatever the proxy . . . 51
3.6.2 Datasette, if you want it . . . 52

4 Developer/Prototyping 52
4.1 Building . . . 52
4.1.1 Building the programs . . . 54
4.1.2 After a reboot . . . 56
4.1.3 Service files . . . 57
4.2 Trying the phone without the gateway . . . 57
4.3 When the phone cannot connect . . . 58
4.4 Trying the broker by hand . . . 59
4.5 Trying the bibliographic sources by hand . . . 59
4.6 A catalogue to try the programs on . . . 60

List of Listings

1 Labels of locations, and of an item . . . 9
2 Writing the printed lists . . . 20
3 Installing Mosquitto and creating the accounts, on Debian . . . 21
4 The broker configuration of sapupra, `sapupra.conf` in `/etc/mosquitto/conf.d/` 22
5 Installing Mosquitto on FreeBSD, with an account of its own . . . 23
6 The broker configuration on FreeBSD, `mosquitto.conf` . . . 23
7 Replacing a broker that the service does not know . . . 24
8 Checking that a scan waits for the worker . . . 24
9 The account sapupra on Debian . . . 25
10 The account sapupra on FreeBSD . . . 25
11 The configuration files, from their examples . . . 25
12 The values of sapupra.ini that the gateway needs . . . 26
13 The values of sapupra.secret.ini that the gateway needs . . . 26
14 Terminal 2: the gateway's account, accepted, then refused . . . 27
15 Terminal 1: the gateway, and what it printed in the trial . . . 27
16 Terminal 2: what the gateway publishes . . . 27
17 Terminal 3: the state of the gateway . . . 28
18 Terminal 3: a scan as Binary Eye sends it, without the phone . . . 28
19 Terminal 1: a broker that does not listen at the address given . . . 28
20 The gateway as a systemd service . . . 28
21 The gateway as an rc service on FreeBSD, in the trial of 0.12.0 . . . 29
22 The directory of the database on FreeBSD . . . 31
23 The values of sapupra.ini that the worker needs . . . 32
24 The values of sapupra.secret.ini that the worker needs . . . 32
25 Terminal 2: the worker, and what it printed in the trial of 0.12.0 . . . 33
26 Terminal 1: what the gateway printed during the same trial . . . 34
27 Terminal 3: the state of the gateway, and the scans as Binary Eye sends them 35
28 Terminal 3: the inputs in the database, with the sqlite3 program (package sqlite3) . . . 35
29 A copy of the catalogue . . . 37
30 A copy of the main file by an ordinary account, when there is no log . . . 37
31 What sudo allows, on FreeBSD . . . 39
32 A command as the account sapupra, with no sudo . . . 39
33 The one command, for an ordinary account (Debian paths) . . . 39
34 The content of /etc/sudoers.d/sapupra . . . 39
35 The copy, made by the ordinary account . . . 39
36 The worker as a systemd service . . . 40
37 The worker as an rc service on FreeBSD, in the trial of 0.12.0 . . . 40
38 The system log of the trial of 0.13.0 (FreeBSD) . . . 41
39 Debugging by hand, the service stopped, then started again . . . 42
40 A log file for the programs of sapupra, FreeBSD . . . 43

41 The values of sapupra.ini that the entry module needs . . . 45
42 The values of sapupra.secret.ini that the entry module needs . . . 45
43 The language of the pages . . . 46
44 The entry module by hand . . . 47
45 The first page from the machine . . . 47
46 The entry module as a systemd service . . . 47
47 The entry module as an rc service on FreeBSD . . . 47
48 Reading the catalogue through an SSH tunnel . . . 48
49 The values of sapupra.ini that sppconsult uses . . . 49
50 Trying sppconsult, and running it as a service . . . 49
51 Debian: sppconsult under the account nobody . . . 49
52 Caddyfile: the catalogue behind a password, in HTTPS . . . 50
53 Caddyfile: without the CSV file . . . 50
54 nginx: the catalogue behind a password, in HTTPS . . . 51
55 Installing Datasette in a virtual environment . . . 52
56 Starting Datasette by hand . . . 52
57 Packages for building the programs, Debian then FreeBSD . . . 53
58 Building the library and running its tests . . . 54
59 The results of the unit tests, from the output of make check . . . 55
60 Why a service is not running after the boot . . . 56
61 A test listener on port 8080, and a request sent to it . . . 57
62 What Binary Eye sends, as the listener printed it . . . 58
63 The packets exchanged with the phone on port 8080 . . . 59
64 What listens on port 8080, on FreeBSD and on GNU/Linux . . . 59
65 The requests of the worker, made by hand . . . 59
66 Timing a request three ways, and a control . . . 60
67 Making a catalogue of invented books and looking at it . . . 60

1 Introduction

This manual is the guide to installing and using `sapupra`, by role:

User the person who catalogues the collection and consults the catalogue: the scanning applications on the phone, the feedback at the shelf, the items that need attention, the consultation.

Admin the person who installs and runs the programs on the machine of the local network: the broker, the gateway, the worker, the entry module, the web front.

Developer/Prototyping the person who builds the programs from their sources, or tries a part before the rest exists.

Within each role the parts follow the path of a scan, from the phone to the catalogue (Concept.pdf). A part not yet written is marked *to come*: the manual grows with the

programs. The reference of each program, its options and its files, is in its manual page (Index.pdf); this manual says how to put the parts together.

Commands shown after a `$` are typed in a terminal; `HOST` stands for the name or the address of the machine that runs `sapupra` on the local network, `PASSWORD` for a password chosen at installation.

2 User

2.1 Scanning with an Android phone: Binary Eye

Binary Eye is a free barcode scanner. Install it from F-Droid or from Google Play, where it is found by its name; both carry the same application.

In its settings, under *Forwarding*:

Forward scans on.

URL to forward to the address of the gateway, given by the admin, with the name of the phone: `http://HOST:8080/scan?device=NAME`, where `HOST` is the machine's address on the local network and `NAME` a short name for this phone, different on each phone (for example `salon` or `phone1`; letters, digits, `-` and `_`, at most 32). It tells the catalogue which phone scanned what, and so where. A test listener can stand in for the gateway (see Trying the phone without the gateway).

Request type `POST application/json`.

Scanner/device ID the same name, if the URL does not give it.

Test URL sends a test request; the answer of the gateway is its state, as "sapupra gateway: broker connected, worker online".

And in the other settings of Binary Eye:

Scan*, *Scan continuously on, so that the camera stays open from one item to the next and no result screen interrupts it, with a *Delay between continuous scans* long enough to move to the next book.

Content*, *Open immediately off. This is what opens the browser: with it on, Binary Eye skips the result screen and executes the scan at once, and for a number that is not a web address it is a web search of that number, in the search engine of *Open unknown data in* (Google by default). The scan has been forwarded to the gateway before that, so the book is catalogued all the same, but the browser takes the screen at every scan.

Content*, *Automated actions no action. An action of this list (open a URL, a custom intent) is run after each scan, and may open a browser too.

These names are those of Binary Eye 1.75's settings and of its source. Tried by the author on 2026-10-27, with Binary Eye 1.75.5 on a Pixel 3a running Android 12: with these three settings the scan reaches the gateway, the camera stays open, and the browser does not open.

On Android 17, Binary Eye 1.75.2 to 1.75.5 cannot reach a machine of the local network: Android 17 blocks the local network to the applications written for it unless they hold a permission that these versions do not ask for, and the phone then reports that it "failed to connect … after 5000ms". Granting *Nearby devices* to Binary Eye in Android's settings does not help. The problem was reported to Binary Eye's author, who fixed it: with Binary Eye 1.75.6 (Google Play), tried by the author on 2026-11-04 on a phone running Android 17, the URL test works and the scanner scans. With an earlier version, update it, or scan with a phone running Android 16 or older.

2.2 Scanning with an iPhone or an iPad: AllCodeRelay

AllCodeRelay is a free application of the App Store for iPhone and iPad (iOS 13 or later) that scans continuously and sends each code to a configured address. It has not been tried with `sapupra` yet; trying it is the first step:

1. Install AllCodeRelay from the App Store.
2. Set its webhook address to the test listener of Trying the phone without the gateway, as `http://HOST:8080/`.
3. When iOS asks whether the application may find devices on the local network, allow it.
4. Scan a book. The listener shows the request if it arrived. iOS may refuse a plain `http://` address for an application that did not declare it; if nothing arrives, that is the first suspect, and the gateway will need HTTPS.

Tell the result to the developer: it decides whether AllCodeRelay is supported.

2.3 At the shelf: what the phone answers

For each scan the phone shows the answer of the gateway (the titles come from the worker):

the title the item is catalogued;

"location: " and the location a location label was scanned: the items that follow are there;

an error, with the place and "not made" the label is of a place that nobody has made: make it in the entry module (*Places*), or correct the label;

the title and "put at ..." a label of an item was scanned: the item is shown and is now at the location of the phone ("already at", if it was there; "no location is current", if no location was set);

an error, "Label misread, scan again" the check digit of an item's label is wrong: scan it again;

"queued" the lookup is still running, or the worker is offline; its result goes to the catalogue, or to the list of items that need attention;

an error, with "not found, set aside" or a reason the item could not be identified: set it aside. Binary Eye vibrates and sounds its error when vibration and sound are on in its settings;

an error, with "Not recorded, scan again" the scan did not reach the broker: scan the item again; if it goes on, tell the admin.

The answers of the gateway, all of them, are in Table 1: the first column is what Binary Eye shows (a good answer is a plain message; an error is shown, and vibrates and sounds, as the settings say). The words after the colon in the first two lines are those of the catalogue, so they change from book to book.

Table 1: What the phone shows after a scan

Message	Meaning	What to do
the title, author, year	catalogued: a new item	nothing
the same, ending with "(again)"	the same code was read again within a few seconds, or the book is catalogued already and the admin has set `copies=no`: nothing is added	nothing
location: NAME	a location label was read	nothing
Place not made: NAME	the place of the label is not made; error 422	*Places*: make it, or correct the label
the title, "put at NAME (was at OLD)"	the label of an item was read; the item moved	nothing
Label misread, scan again	an item's label with a wrong check digit; error 422	scan it again
queued	a source did not answer; the worker retries	nothing; see *Waiting for a source*

Continued on next page

Continued from previous page

Message	Meaning	What to do
Already in the catalogue, not added: title (item N, K copies). Decide in the entry module, To do: add it as copy K+1, or discard this scan.	the book is in the catalogue and the admin has set `copies=ask` (the default); the scan is held, error 422	open *To do* and choose one of the two buttons (Items that need attention)
no source knows ISBN …	no source has it; error 422	*To do*: type another ISBN, search, or enter it by hand
product code …: no source knows it; if this is a book, type its ISBN	a barcode that is not an ISBN	*To do*: type the ISBN printed in the book
N records found: the entry module chooses	the sources give several books for the code	*To do*: choose one
Not recorded, scan again	the scan did not reach the broker	scan it again; if it goes on, tell the admin

A failed item is never lost: it stays in the list of items that need attention, with where and when it was scanned and the items scanned just before and after it.

2.4 Locations

A location is set at the beginning of a batch, and stays the location of the phone for every item it scans until another one is set. It is set by scanning a label stuck on the shelf, whose code holds `LOC:` and the location (`LOC:SALON B3`), or, with the entry module, by choosing it for that scan device among the places made (the page *Scan devices* of *Places*, Items that need attention). The labels are printed by `spplabel` (Printing the labels); the entry module shows the command for each place.

A location can be as precise as you like, as a path from the room to the shelf, the parts separated by a slash: `LOC:salon/bookcase 2/shelf 4`. The text after `LOC:` is free; only the slash means something, and spaces round a part do not count (`LOC:salon / bookcase 2/shelf 4` is the same place). Capitals do count: `Salon` and `salon` are two places. A place is made on purpose, in the page *Make, rename, merge, delete* of *Places* (its parents are made with it: `salon/bookcase 2/shelf 4` makes `salon` and `salon/bookcase 2` too), and nothing makes it otherwise: a label of a place that is not made is refused, and so is a location typed for a phone, so that a misspelt label is found when it is stuck. The same page renames a place (the places below it follow) and merges a place into another (its items move, the old place goes), which is the remedy of a misspelling that went unseen. A place made by mistake is deleted, if nothing depends on it: no place below, no phone there, no copy (even a removed one) and no scan made there; otherwise merge it. Before the first scans a place must be made and chosen as the phone's location, in *Scan devices* or by scanning its label; until then the items have no location. An item entered by hand (*New*

item, *Look up*) can be given a place of its own, which does not become the phone's. In the catalogue, a search for a place finds what is in it and in the places below it: *salon* lists the shelves of its bookcases (not a place that merely begins with the same letters, *salonette*), and the counts list each room, each bookcase and each shelf, with the number of copies at or below it. The path is read as a path everywhere: a link to a place with spaces, accents or slashes in its name finds its items (an earlier version mangled the slashes of such a link and found nothing).

The code of a place can be scanned from the screen, with no label printed: *Places*, *Labels*, has a column *On the screen* whose link *Show the code* opens a page with the code alone, large, and the place under it; *Show it on the whole screen* there makes it the size of the window, for a phone held up to the screen of the computer. The search result of a place shows the same code, folded with its label. It is the code that the label has (`LOC:` and the place), so that scanning it sets the location of the phone in the same way. The entry module draws it with the library of Zint like `spplabel`; built without it, the page says so and shows the command that prints the label.

2.5 Printing the labels

The labels of the shelves, and those of the items that have no barcode, are printed by `spplabel`, a command that you run where the printer is, with no service running and no network: it draws a code with its text under it, as pages of SVG, which a browser, Inkscape or LibreOffice opens and prints. It is installed with the others, if the library of Zint was there when the programs were built (Building the programs); its manual page is `spplabel(1)`.

The entry module shows the command ready, so that it is copied and not typed: the page *Labels* of *Places* has one for each place, and the result of a search for a place has one, folded, under *The label of this place*; the page of an item has its own (*The label of this item*). The command is in a box that scrolls sideways on a small screen: the page itself does not.

Listing 1: Labels of locations, and of an item

```
$ spplabel -l 'salon/bookcase 2/shelf 4' -l 'salon/bookcase 2/shelf 5' -o shelves.svg
$ spplabel --item 1234 --text 'Kabos: Az érdekes Ujság Décameronja' -o item.svg
$ spplabel --items 100-140 --columns 3 --rows 8 --skip 5 -o items.svg
```

a location label `-l` with the path. The code holds `LOC:` and the path as it is written; under the code, the last part is in bold and the places that contain it follow, so that the label can be read at the shelf as well as scanned. Print one label for each shelf, and stick it where the phone will reach it.

an item label for a copy that has no barcode, or whose barcode says nothing: `--item` with its number, which the page of the item shows, with the command ready (*The label of this item*). The code is `ITM:`, the number, a hyphen and a check digit (`ITM:1234-4`):

a label that is misread, or a number mistyped, is found out by that digit. `--text` puts a line under the number, as the author and the title, so that a label that has come unstuck is still known. Scanning such a label shows the item and puts it at the current location of the phone: it is how an item with no barcode is moved from a shelf to another, or seen again. Print them for the items that have no barcode; the page of an item lists the places where its label was scanned.

the sheet A4 by default (`--page Letter`), twenty-eight labels to a page with a code of 20 mm (`--size`). To print on a sheet of adhesive labels, give its grid, `--columns` and `--rows`, and for a sheet that is partly used `--skip` the places gone. When there are several pages, `-o items.svg` writes `items-1.svg`, `items-2.svg` and so on. A text that is too long for its place is shortened, and the program says so; the code is not shortened.

the kinds the code is a QR code, the one that phones read best and that holds accents. `--kind` asks for a Data Matrix, an Aztec code or a Code 128 (a bar code, for a narrow edge; it holds letters without accents). `--ecc` is the error correction: `Q` or `H` for a label that will be rubbed, at the price of a larger code.

the prefixes `LOC:` and `ITM:`, which the gateway and the entry module know too; `spplabel` reads them in `sapupra.ini` (`[locations]` and `[items]`) when it can read the file, which is open to every account (and the secrets are in another file); if it cannot, it says so and uses the usual ones; `--config`, `--location-prefix` and `--item-prefix` say otherwise. Do not change a prefix once labels are printed.

a plate `--plain` writes the code alone, as black shapes of exactly the size asked, one file for each label, for a 3D printer or a cutter: `spplabel --item 1234 --plain -s 20 -o plate.svg`. A plate has no text and no page, so `--text` and the options of the page (`--page`, `--columns`, `--rows`, `--skip`, `--margin`, `--no-frame`) do nothing with it; the program warns on its standard error if one is given. The code needs a white margin of four modules round it, which the plate gives (a plate of 28 mm for a code of 20 mm).

The labels were checked by reading them back with two decoders of codes, from the rendered SVG; check yours with the phone before printing a hundred.

2.6 Items that need attention

The entry module is a set of pages of the local network that any browser opens, the phone's included, at the address that the admin gives (`http://HOST:8081/`, The entry module in the admin's part), which opens on the catalogue; the list of what waits is at `/todo`. The menu on top of each page has six entries, as buttons: *Catalogue* (Consulting the catalogue), *To do*, *New item*, *Look up*, *Places* and *Import*; the foot of *To do* says again what each is for. The device that the pages themselves use, when no phone is concerned,

is called *manual entry* (`entry` in the database and the logs): it is the one preselected in the lists of devices, and the place that it holds is the place of an item entered by hand.

To do shows the scans whose lookup failed, oldest first, each with the phone, the code (and what it is: `isbn`, `gtin`), the location then, the time and why it failed. A scan that waits for a source that did not answer is not there: it has its own list below, *Waiting for a source*, and the worker tries it again by itself (after 1 minute, 10 minutes, 1 hour, then daily, for a week). A code typed in the field on top finds the waiting scan, with its ISBN hyphens or without. Under them, the *Latest inputs* show what the worker did last, which is how to see that all went through.

A click on *settle* (or *decide*, for a book that is in the catalogue already) opens the scan: what was read, the scans of the same phone just before and after it (to recognize the book in the hand), and three ways to settle it, and one to set it aside. When the sources gave several records for the code (the message *N records found: the entry module chooses* on the phone), the page begins with them: *Which one is yours?*, each record with its year, publisher, place, number of pages and ISBN, and a button *Catalogue this one*. Take the one in your hands: the year and the publisher tell the printings apart. The record is catalogued for the scan, at the location it was scanned at, and the edition takes the ISBN of the scan if the record has none, so that the next copy is found by it. The records are kept by the worker, so that the choice can be made the next day or after a restart; a scan of before 0.20.0 has none (the page says so: scan it again). The ways below remain open:

Type the ISBN for a book whose barcode is not its ISBN (the publisher's own number) or that had no barcode, the ISBN printed in the book; hyphens and spaces are fine. The worker looks the ISBN up as it would a scan. The barcode is kept as the edition's product number, so that the next copy, scanned by its barcode, is found without any request;

Enter the description by hand the last resort: the title (required), the medium, the authors one per line (`Name`, or `Name | role`), the publisher, the place, the year, the languages separated by commas, the extent, the series, the subjects one per line, and the ISBN if there is one. The scan becomes an item with that description, at the location it was scanned at.

Look up for a book with neither ISBN nor barcode that a source knows: a part of the title, the author's surname, or both; or, alone, the card number of the Library of Congress (below). The worker asks the sources (Open Library and the BnF by default), and the page of the answer lists the books found; the button *Catalogue this one* makes the item, at the location the scan was taken at. Take the book nearest to the one in hand: the sources list works and editions, and the ISBN shown may be that of another edition of the same work. A book with an ISBN is looked up again by it, as a typed ISBN is, so that the sources describe that edition; one without is catalogued as the list shows it. The same book given by two sources is shown once. The list is kept by the module for a while only: after it was restarted, search again. Seen

against real answers of Open Library and of the BnF (a search for Simondon), not yet live.

A source answers only with the books that have *all* the words typed, as they are spelled: a word wrong (computing for computer) or a word too many finds nothing. When a search finds nothing, the worker asks again by itself, with the two longest words of the title (when it has more than two) and the author as typed, and the verdict says so; if that finds nothing too, try the surname alone. The to-do list, the page of the input and the page of the answer say what was asked, title and author as they were typed, so that the mistake can be seen. At most ten books are offered, whatever the number of sources, and the verdict names the sources that gave them; the Library of Congress is not asked by a search by title and author unless it is named in `[sources] search` (`search=openlibrary,bnf,loc`); it is asked for a book by its ISBN.

The card number. An American book without an ISBN has a card number printed in it (*Library of Congress Card No. 73-1830*). Type it in the field of that name, written as printed (`73-1830`, `2001-012345`, `n78-890351`), *alone*: the page refuses it with a title or an author. The Library of Congress is asked (the key `[sources] lccn` has `loc` by default, and only it can be); the program writes the number the way its index has it (`73001830`). This needs a worker built with YAZ; without it the verdict says that no source can look up a card number. Tried against the real server on 2026-11-04 with `73-1830`: it gave the book.

Dismiss this scan sets aside a scan taken by mistake (the code of a box, of something not to be catalogued): it stays in the database, marked dismissed, and leaves the list.

Either form answers with a page that waits for the worker and shows its verdict: "Catalogued as item 12: Title, Author (Year) — from openlibrary" (the source or sources that answered), or why not (an ISBN that no source knows: try the other way). A form sent twice (a double click, a reload) is one input. A scan that is catalogued is no more in the list, and cannot be settled again.

Already in the catalogue. When a book that the catalogue holds already is scanned again, outside the few seconds of a double read, the worker does what the admin has set (The worker, `copies`). By default (`ask`) it adds nothing and holds the scan in *To do*, whose reason says "Already in the catalogue, not added", with the item and how many copies there are. The phone showed the same words as an error: that is the whole of the message, nothing is lost and nothing is added until you decide. The page of the scan ("decide") asks one thing, with two answers that say what they do:

Add it as copy N for the person who has another copy of this book: a new item is made for this scan, at the location it was scanned at;

Discard this scan, add nothing for the copy that is catalogued, scanned again: the scan is set aside, marked dismissed, and the catalogue does not change.

A person who owns no second copy sets `no` and is never asked: the scan is answered "(again: catalogued already as item N, no other copy is added)"; one who does, `yes`, and the copy is added without a question. When several scans are held, *To do* also offers *Discard these N scans, add nothing*: it is what to do after a history of Binary Eye was imported again (every book was in the catalogue already, so every scan is held); the scans stay in the database, marked dismissed.

Look up is the same search without a scan: for an item in hand that has not been scanned; the phone whose location it takes is chosen on the page. It searches by title and author only and says so, with a pointer to *New item* for an ISBN: the ISBN is typed there. The fields come in that order: title, author, then the scan device and the place, with a line that says they are not searched for: they are what the book chosen among the results is catalogued with. An item found is shown on its own page (the page of a copy, with a line *Catalogued: this is the new item N* on top), not in a short message; an item that cannot be settled yet shows the page of the scan, as before.

New item catalogues an item that has no code to scan: the same two forms, with the phone whose location it takes (the list, or another name typed), and a place for the item, chosen among the places made; left empty, it is where the phone is. The place chosen is the item's alone: it does not become the phone's location.

Places is a set of four pages, joined by a line of links on top of each, so that none is crowded:

The places the places made, with the number of items at and below each (each a link to what is there);

Make, rename, merge, delete makes a place (its parents with it), renames one (the places below it follow), merges one into another (its items move, the old place goes) and deletes an empty one;

Scan devices where each phone is, and the place of a phone chosen among the places made, as a label does (with none made it says to make one first); and the form *Correct the identifier of a scan device*, for a phone that wrote its name wrongly (*puxel* for *pixel*): all its scans, the past ones too, take the new identifier; if the new one is a scan device already, the two become one. The phone goes on sending the old name until it is corrected in its settings (the `?device=` at the end of the address of the gateway): until then a new *puxel* appears, to be corrected again. The manual entry is neither renamed nor merged into;

Labels for each place, the command that prints its label (Printing the labels).

The list of the places is there to be seen: the places that a choice offers (for a new item, for a phone, for moving a copy) are the ones listed. The address `/location` of the earlier versions leads to *Scan devices*.

The device *manual entry* is shared: every browser of the network that uses the pages uses it, so that the place set for it is the place of the next item entered by hand from any

of them. When several people enter items at once, each browser can remember a place of its own: on the page *Scan devices*, the form *The place of this browser* sets it (a cookie, kept for a year, that only this module reads), and the forms of the items typed by hand (*New item*, *Look up*) then offer it first, whatever the location of the manual entry; the first line of that list is the location of the device, to use it instead, and a form sent with a place makes the browser remember that place. The first line of the form of the browser forgets it. A browser that refuses cookies simply has none (Several people at once).

Import recovers the scans that the phone could not send. When the gateway or the broker was away, Binary Eye answered with an error and the scan is in its history (which it keeps by default: its setting *Save scan history*). In Binary Eye: *History*, the menu *Export to file*, *JSON file*, a name for the file. In the phone's browser: *Import*, the file chosen, the phone's name chosen (the file does not say which phone it comes from), and, if you want, the time from which to import (`2026-10-26` or `2026-10-26 14:30`). The page then shows what would be sent, the scans in the order they were taken, and how many are recorded already (the same phone, the same time and the same code); nothing is sent before *Send*. Importing the same file twice is harmless: the second time all is "already recorded". The scans that the sources do not know join the to-do list as any scan; those that the sources could not be asked wait for the worker's retries. The location that each scan had is the one that was current when it was taken, if the history holds the label's scan before it; the phone's own location is put back at the end.

2.7 Removing and moving a copy

A book given away, or catalogued by mistake, leaves the catalogue from the page of its copy (the page that opens from any list of the catalogue). Under the copy, two forms:

Remove this copy with the reason: *donated*, *sold*, *lost*, *entered by mistake* or *other reason*. Only that copy goes; the other copies of the book stay, and the book itself (the edition) is kept, so that a scan of its barcode finds it again. The copy is in no search, no count and no CSV file. Nothing is lost: the page of the copy says when and why it was removed, and *Put it back* returns it as it was, with its place and its label;

Move this copy puts it at another of the places made (the same as scanning its label there); the place where it was is kept in the list *Where it has been put* of the page.

The copies removed are listed on the page *Removed* (the link is on the home of the catalogue, once there is one), the latest first, each with its date and its reason. The label of a removed copy, scanned, is refused and says so, and moves nothing.

What is not offered: a real deletion of a copy (a removal can always be undone, and the reason *entered by mistake* says what it was), and the removal of a phone from the list of the phones (it has sent inputs; they are the history). A place made by mistake is deleted on *Places* (*Make, rename, merge, delete*) if it is empty, or merged into another if it is not (Locations).

2.8 Correcting a description

A mistake in a description (a name misspelled, a wrong year, a word in the title) is corrected after the book was catalogued, whether the description was typed or came from a source. On the page of a copy, the line *Correct the description* opens the form of *Enter the description by hand*, filled with what is stored. Change what is wrong and press *Correct*.

- The correction is to the book, not to the copy: all its copies have the new description, and stay where they are.
- The record that the source sent is kept as it was received, and the page of the copy says when the description was corrected.
- The ISBN of a book that came from a source is not changed: it is what the book was found by. The field is offered only for a description typed by hand.
- Authors that stay keep how they are filed in the lists; a name that no book has any more is forgotten.

A mistake in the source is common enough (an old record, a title cut short), and the correction is meant for it. What is not offered is a history of the corrections: the last one stands.

2.9 Several people at once

Several phones, and several browsers, can be used at the same time. What is protected, and what is not:

the catalogue only the worker writes it, one input after the other, in the order they reach it. Two scans of two codes are independent, and each item takes the location of the phone that scanned it, not of the last phone to scan. A browser that reads the catalogue never blocks the worker, nor the worker a browser;

one code scanned twice from one phone or two, while the first lookup is not over, the second waits for it: one lookup, and then a copy or not as the admin has set `copies`;

two decisions about one waiting scan the first is taken; the second is told that the scan is not waiting any more. A form sent twice (a double click, a reload) is one input;

two workers refused. The second one to start on a database says that another worker is running on it, and stops (The database and the configuration);

the location of a phone it belongs to the phone: two people who use one phone share its location, and the last label scanned wins. The pages of the entry module are one phone for every browser, the *manual entry*: the place set for it is the place of the

next item entered by hand from any browser. So that people who enter items at once do not meet, each browser remembers a place of its own, offered on the forms of the items typed (*Scan devices*, *The place of this browser*), or chooses the place on the form of the item (*New item*), which is the item's alone;

no sign-in the pages do not ask who you are: whoever reaches the port acts. The network of the home is the protection.

2.10 Consulting the catalogue

The catalogue is read in a browser of the local network, the phone's included: the entry button *Catalogue*, the page `http://HOST:8081/catalogue`, which is also where the bare address `http://HOST:8081/` leads (The entry module). Nothing there changes the catalogue. The page has:

the way a search answers pressing *Search* brings the page to the result, which is under a heading *Result* with what was asked in a line; the fields are folded under *Change the search*, to be opened to refine it. When nothing matches, the page says *No hit* and shows nothing else: not the last result, not the catalogue. A form sent with every field empty says that no field is filled. *Reset* (in French *Réinitialiser*), next to *Search*, puts the form back to blank: it empties all the fields (it opens the bare page of the form). The pages are never kept by the browser, so a new search never shows the old one;

the version the foot of every page of the entry module says which program answered and its version (`sppentry (sapupra)` and the version), to tell the module that runs from the one that was installed;

the look the pages follow the light or the dark setting of the browser; on a phone a few columns are left out of the lists (the publisher and the medium in the catalogue, the device in the to-do list), the page of a copy and the page of a scan have them, and the to-do list is a list of cards;

a simple search a form with the fields that most searches need: *Title*, *Author*, *ISBN or barcode*, *Location* and *Medium*. A field left empty asks nothing; the fields filled must all hold. The text fields are regular expressions, each tried on that field of the edition alone (What the search fields accept);

an advanced search the link *Advanced search* under the form (it carries what is typed) opens the form with a field for every detail: the title, the subtitle, the author, the translator, the other contributors, the publisher, the place, the language, the series, the subject, the ISBN or barcode, the ISSN, the extent; a year *from* and *to*; the days between which the copy was entered; the least number of copies of the edition; the location, the medium and how the description was obtained. The link *Simple search* goes back. The fields are in four groups, as many as the mind keeps at a glance: *The*

work, *The edition*, *The numbers* and *The copies*. The addresses of both pages carry the fields, so that a search can be kept as a bookmark;

the counts how many items of how many editions, by location and by medium, each a link: a click on a location lists what is there, a click on a medium lists the books or the discs. The places are a tree that is folded: the rooms are shown, each with its number of items, and a click on the little triangle of a room opens its furniture, then its rows, then its shelves, so that a list never shows more than a handful of lines at once;

the latest with nothing asked, the ten copies entered last;

the answer one row per copy, by title, with its authors, its publisher, its year, its medium and its location; fifty to a page, *Previous* and *Next* under them, which keep all the fields. The heading of a column is a link: a click sorts the whole answer (not only the page) by that column, a second click reverses the order, and a little triangle shows the column and the direction. One can sort by title, author, publisher, year, medium, location and day of entry. The order is the alphabet of the language of the pages (an *É* is not after the *Z*), the years and the numbers of the shelves are read as numbers, and a copy that has nothing in that column comes last, whichever the direction;

a CSV file the link *These as a CSV file* downloads the whole answer (not only the page) for a spreadsheet, for the same fields and in the order that was chosen; *All of it as a CSV file*, the whole catalogue. LibreOffice (version 26.8 tried) opens it as it is, accents included;

the page of a copy a click on the title shows everything known about the edition (its item number, the authors with their role, the publisher, the place, the languages, the extent, the series, the subjects, the numbers), and, under *N copies of this edition*, its copies with their location and the day entered. The first column, *Copy*, appears only when the edition has several copies: it is the number of each copy (its link, and the number of its label), the current one in bold, *(this one)*. With one copy the table has two columns, and the item number is the row *Item number* above.

Which source described an edition is said on the page of a copy, in the row *Source of the description*: the name of the library or service that answered, and the day (*BnF, 2026-10-27*); for an edition written by hand, *by hand*. If a book has no author, look at that row: the source did not give one (Open Library gives the author of some editions only on the work, which the worker now asks for). The pages of scans that need attention are those of *To do* (Items that need attention).

2.10.1 What the search fields accept

The catalogue has a field for each detail of an edition. The fields can be used together: all those filled must hold. Table 2 says what each field takes.

Table 2: What the search fields accept

Field	What it accepts
Catalogue, an attribute	A pattern, described below, tried on that field's values alone.
Catalogue, ISBN or barcode	A whole number, with hyphens or without, in 10 or 13 digits: that edition. The beginning of a number (digits only): the editions whose number begins so. Anything else (a `^`, a `$`, a letter): a regular expression on the ISBN-13 and on the barcode as typed.
Catalogue, year from	A year, `1950`: from that year. If it is not a number it is a regular expression on the year, like the other fields, and *year to* is then not used.
Catalogue, year to	A year: up to that year.
Catalogue, entered from, to	A day, `2026-10-26`, both included.
Catalogue, copies	A number: the editions of which you have at least so many copies.
To do, the code	A code as it was scanned, or an ISBN with hyphens; it finds the scan that waits for a source.
Look up, title and author	Words, as the sources understand them (Open Library, the BnF): a part of the title, the surname. Each source applies its own rules.
Import, from	A date, `2026-10-26`, or a date and a time, `2026-10-26 14:30`.

2.10.1.1 The patterns A pattern is an extended regular expression, as `grep -E` reads it: groups, `|`, `?`, `*`, `+`, `{n,m}`, brackets and their classes (`[[:digit:]]`, `[a-z]`), `.`, `^` and `$`. It is read by the engine of Qt, which is Perl-compatible and accepts all that and a little more (`\d`, `\b`, `*?`). For an edition found or not, which is all that a search asks, it answers as `grep -E` does, with one difference that may matter: inside brackets a backslash is an escape here and a plain character in POSIX.

Field by field The pattern of a field is tried on the values of that field only, and the edition is found if one value matches, anywhere in it; `^` and `$` are the beginning and the end of the value. An edition has several values of some fields: the author field is tried on each contributor in turn (an author is a name, `^umberto eco$`, not *Umberto Eco and others*), the subject field on each subject, the language field on each language (asked for by its code, `hun`, by its English name, or by its name in the language of the pages, *hongrois* on a French page: the three find the same books). The title field does not see the publisher: `regény.*Magvető` finds nothing, whatever field it is typed in, where the subtitle is *regény* and the publisher *Magvető*. To ask for two fields, fill two fields.

Author, translator, other The author field is tried on every contributor of the edition, whatever the role; the translator field on the translators; the field *other contributors* on those who are neither an author nor a translator (editors, illustrators, performers).

Capitals and accents They do not matter, in the pattern as in the text: `kertesz`, `Kertész` and `KERTÉSZ` find the same editions. A letter with an accent in the pattern is read without it (`[é]` is read as `[e]`).

With the other criteria The fields, the location, the medium, the years, the days and the copies narrow each other; all of them must hold.

A field that is not valid A pattern that is not one (`(camus` or `[a`), a year that is not a number, a day that is not a day, a number of copies that is not a number: the page says which field and why, and nothing is searched, not even with the fields that are valid. In the CSV file it is an error 400. A pattern is never taken as plain text instead.

Time The patterns are tried by the program on every edition that the other criteria leave, not by the database. The test suite measures it on 30000 generated editions (the design document gives the figures); a pattern that makes the engine backtrack a long time (a group repeated inside a group) would make the page wait.

Table 3 gives examples.

Table 3: Examples of patterns

Field	Pattern	Finds
Title	`^l[ea]s?[ ]`	The titles that begin with *le*, *la* or *les* followed by a blank.
Year, from	`^19[56][0-9]$`	The editions of 1950 to 1969 (a pattern in *from*; *to* is not used).
ISBN	`^9782`	The ISBNs that begin with 978-2: the French-language editions.
Author	`^umberto eco$`	The name exactly, not *Umberto Eco and others* (`eco$` is the name that ends so).
Title	`[[:digit:]]`	A title with a digit in it.
Publisher	`[éé]cole`	*école* and *ecole* alike: anywhere in a value, accents do not matter.

2.11 Printed lists of the catalogue

The catalogue can be put on paper, or given as a file to someone who has no access to the network, with `spplist`. It reads the database, changes nothing (the services may run) and writes an Org-mode document, which Org exports to a PDF or to a page of HTML. The document has up to three tables, each of all the copies that are not removed:

by author the columns *Author*, *Title* and *Location*, sorted by author;

by title *Title*, *Author* and *Location*, sorted by title;

by location *Location*, *Author* and *Title*, sorted by location, which is the list to carry along the shelves.

By default the three are written; `-t` (or `--table`) chooses: one (`-t location`), two (repeat the option, or separate the names with a comma: `-t author,location`) or all of them (`-t all`). Listing 2 shows the use.

Listing 2: Writing the printed lists

```
$ spplist -o catalogue.org
$ spplist -t location --title "Shelves of the house" -o shelves.org
$ spplist -t author,title -D /var/db/sapupra/catalogue.db -o by-name.org
$ emacs --batch catalogue.org -f org-latex-export-to-pdf
$ emacs --batch catalogue.org -f org-html-export-to-html
```

The database is named by `-D`, or else read in the configuration file (`[database] file`, `-c` to name another). Any account can run it, if the directory of the database is open (mode 0755) and the file readable (0644), which is how the installation makes them (The database and the configuration); if not, the program says which is closed. While no program has the database open, only the main file is there, and `spplist` reads it as it is, which is the whole catalogue. The headings and the order follow the language of the pages (`[interface] language`, or `--language`). The document says on its first line the day it was made; `--no-date` leaves the day out, so that two runs give the same file.

What to know about the lists:

the order it is the alphabet of the language: an accent or a capital does not send a word to the end, a number is a number (shelf 2 before shelf 10). The article that starts a title is set aside for the order, by the language of the edition: *The Name of the Rose* is under *N*, *Le Rouge et le Noir* under *R*, *L'Étranger* under *É*; the title is written in full. An edition whose language is not known (the sources do not always say it) is sorted as it is written;

the names an author is written and sorted as *Surname, Forename*: `Imre Kertész` becomes *Kertész, Imre*. Where the source said how the name is filed (the national libraries and MusicBrainz do: *García Márquez, Gabriel*, *Pink Floyd*) that is used. Where it did not (Open Library does not), the program takes the last word of the name for the surname, with the particles before it (*de*, *van*, *von*, …). It cannot know better: a name of the Far East or a body may be turned the wrong way round; a name that has a comma in it is left as it is, and `--as-written` leaves all of them as they are stored;

several authors a copy is listed under each of its authors in the table by author, and has them all, separated by a semicolon, in the other two; a copy without an author

(a disc, often) comes last of its table. The authors are the contributors whose role is author or is not given; for a work that has none, the contributors who are not translators;

the paper the PDF is of A4 pages, in 10 points, the tables running over as many pages as they need, with the headings repeated. The characters in the titles that Org would take for marks (a star, a slash, an underscore) are protected, so a title comes out as it was typed;

the time Org takes its time. The PDF of some hundreds of copies is made in seconds; the HTML of thousands of copies takes minutes (about a minute and a half for 5000 copies), because it is Org that builds the tables. For a very large collection, export the tables one by one, or the PDF.

3 Admin

3.1 The machine

One machine of the local network, running GNU/Linux (Debian, or a derivative) or FreeBSD, always on while scanning, reachable by the phones. Nothing of `sapupra` needs to be reachable from outside the local network: only the worker goes out, to query the bibliographic sources. A firewall on the machine, if there is one, must let the phones and the browsers of the local network reach the gateway's port, the entry module's and the web front's, and the other programs reach the broker's (1883).

3.2 The broker: Mosquitto

The broker is the centre of `sapupra`: the gateway and the entry module publish to it, the worker reads from it. It keeps what it holds on disk, so that a scan published while the worker is stopped waits for it, even across a restart of the broker.

Each program connects with its own account, and each account has its own password: `gateway`, `worker` and `entry`. A password can then be changed, or an account closed, without touching the other programs, and the rights of each account can later be limited to the topics it needs.

3.2.1 On Debian

Listing 3 installs Mosquitto and its command-line clients, and creates the accounts (the first `mosquitto_passwd` creates the file, with `-c`; the next ones add to it, without). The password file must belong to the user `mosquitto`, which the broker runs as.

Listing 3: Installing Mosquitto and creating the accounts, on Debian

```
$ sudo apt install mosquitto mosquitto-clients
$ cd /etc/mosquitto
```

```
$ sudo mosquitto_passwd -c pwfile gateway
$ sudo mosquitto_passwd pwfile worker
$ sudo mosquitto_passwd pwfile entry
$ sudo chown mosquitto:mosquitto pwfile
$ sudo chmod 0600 pwfile
```

The password file is called `pwfile` as in the sample that Mosquitto provides, and as on FreeBSD (the author's choice, 2026-11-03: the same name and the same content on both systems). Debian's main configuration reads every file of the directory `conf.d` beside it whose name ends in `.conf`. Listing 4 is the file of `sapupra`, saved in that directory as `sapupra.conf`: one listener on the local network address of the machine (here 192.168.1.10; a listener without an address listens on every interface), no anonymous access, the password file. Debian's main file turns persistence on already; the line is repeated so that the file is the one of FreeBSD (On FreeBSD), which differs by the path of the password file alone (and, there, by the directory of the persistence and the file of the process number). `autosave_interval` saves the broker's memory to disk every 60 seconds instead of the default 1800: what a crash or a power cut can lose is what arrived since the last save. `log_dest syslog` adds the system log, so that the system log holds the broker's messages, as on FreeBSD.

Listing 4: The broker configuration of sapupra, `sapupra.conf` in `/etc/mosquitto/conf.d/`

```
listener 1883 192.168.1.10
allow_anonymous false
password_file /etc/mosquitto/pwfile
persistence true
autosave_interval 60
log_dest syslog
```

Then `sudo systemctl restart mosquitto`, and the check of Checking the broker.

3.2.2 On FreeBSD

The port is `net/mosquitto`. Four differences with Debian:

The account of the broker the port creates no account for the broker, which then runs as `nobody`, and its password file would have to be readable by every user. Mosquitto 2.0 already warns against a password file that others can read, and says that future versions will refuse it. So a dedicated account, `mosquitto`, is created: the broker, started as root by the service, changes to that account by itself (its `user` option defaults to `mosquitto`, and falls back to `nobody` only when no such account exists). The variable `mosquitto_user` of the service gives that account the file of the process number, which the service creates empty before starting the broker.

The file of the process number the broker writes its process number only when its configuration names the file, by `pid_file`, and it writes it after changing to its account. Without that line the file stays empty, and the service believes that the broker

is not running: a restart prints "mosquitto not running? (check /var/run/mosquitto.pid)" and tries to start a second broker, which finds the port taken by the first.

The password file the port installs one holding three example accounts. The `-c` option of `mosquitto_passwd` overwrites it, as on Debian, so that only the three accounts of `sapupra` remain.

The configuration file the port installs a commented example of every option; it is replaced by the few lines of `sapupra`, with persistence turned on (Debian's main file does it there) in a directory that belongs to the broker's account.

Listing 5 does the first three; the fourth is a line of the configuration. Listing 6 is the configuration file, `mosquitto.conf`, in the same directory as the password file, with the machine's address on the local network in its first line. `log_dest syslog` sends the broker's messages to the system log, `/var/log/messages`.

Listing 5: Installing Mosquitto on FreeBSD, with an account of its own

```
$ sudo pkg install mosquitto
$ sudo pw useradd mosquitto -c 'Mosquitto MQTT broker' -d /nonexistent -s /usr/sbin/
    nologin
$ sudo mkdir -p /var/db/mosquitto
$ sudo chown mosquitto:mosquitto /var/db/mosquitto
$ cd /usr/local/etc/mosquitto
$ sudo mosquitto_passwd -c pwfile gateway
$ sudo mosquitto_passwd pwfile worker
$ sudo mosquitto_passwd pwfile entry
$ sudo chown mosquitto:mosquitto pwfile
$ sudo chmod 0600 pwfile
$ sudo sysrc mosquitto_enable=YES mosquitto_user=mosquitto
```

Listing 6: The broker configuration on FreeBSD, `mosquitto.conf`

```
listener 1883 192.168.1.10
allow_anonymous false
password_file /usr/local/etc/mosquitto/pwfile
persistence true
persistence_location /var/db/mosquitto/
autosave_interval 60
log_dest syslog
pid_file /var/run/mosquitto.pid
```

Then `sudo service mosquitto start`, and the check of Checking the broker. The dedicated account has been tried: the broker logs "running mosquitto as user: mosquitto".

3.2.2.1 The pid file A broker started before the line `pid_file` was added is unknown to the service, since its pid file stayed empty: `stop` and `restart` cannot reach it, and it

keeps port 1883. Every start after that one fails: the new broker writes its number in the pid file, cannot open the port (the system log shows “Error: Address already in use”), and exits, leaving its number in the file, which it can no longer remove. The status then says “not running” while `pgrep` shows the old broker.

Listing 7 stops the old broker by its number, checks that the port is free, starts the service, and checks that the three numbers agree; `NUMBER` is the one `pgrep` gives. A restart must then print no warning.

Listing 7: Replacing a broker that the service does not know

```
$ pgrep -x mosquitto
$ sudo kill NUMBER
$ sockstat -4 -l -p 1883
$ sudo service mosquitto start
$ service mosquitto status
$ pgrep -x mosquitto
$ cat /var/run/mosquitto.pid
$ sudo service mosquitto restart
```

`sockstat` must list no process on port 1883 before the start; the status, `pgrep` and the file must then give one and the same number. This has not been tried yet.

3.2.3 Checking the broker

Listing 8 checks the accounts and the persistence: the first command registers the worker’s session and stops after one second; a message is then published as the gateway would, while no worker listens; the last command, the worker coming back, receives it. Restarting the broker between the second and the last command must not change the result. An anonymous connection must be refused.

Listing 8: Checking that a scan waits for the worker

```
$ mosquitto_sub -h HOST -u worker -P PASSWORD -i worker -c -q 1 -t 'sapupra/test/#' -W
    1
Timed out
$ mosquitto_pub -h HOST -u gateway -P PASSWORD -q 1 -t sapupra/test/phone1 -m '{"
    content":"9782081231412"}'
$ mosquitto_sub -h HOST -u worker -P PASSWORD -i worker -c -q 1 -t 'sapupra/test/#' -v
    -C 1 -W 5
sapupra/test/phone1 {"content":"9782081231412"}
$ mosquitto_pub -h HOST -q 1 -t sapupra/test -m anonymous
Connection error: Connection Refused: not authorised.
Error: The connection was refused.
```

3.3 The gateway

The gateway, `sppgate`, receives the scans of the phones and hands them to the broker. It runs under a system account of its own, `sapupra`, shared with the other programs, and reads one configuration file, `sapupra.ini`, which holds the broker's password of the gateway. Built and installed as Building the programs says, it is in `/usr/local/sbin/`, the example of the configuration in `/usr/local/etc/sapupra.ini.sample`, and its service files (`sppgate.service` or `sppgate.rc`, those of the system) put in place by `make install` (Service files).

3.3.1 The account and the configuration

The account `sapupra` is a system account, without home or shell, with a group of the same name: Listing 9 creates it on Debian, Listing 10 on FreeBSD (`pw` makes the group with the account).

Listing 9: The account sapupra on Debian

```
$ sudo adduser --system --group --no-create-home --home /nonexistent --comment 'Simple
    Library Management System User' sapupra
```

Listing 10: The account sapupra on FreeBSD

```
$ sudo pw useradd sapupra -c 'Simple Library Management System User' -d /nonexistent -s
    /usr/sbin/nologin
```

Listing 11 makes the configuration from its examples, on both systems. There are two files, because they are read by different people. `sapupra.ini` holds the settings, and is open to every account (mode 0644): the commands that anyone runs on the machine, `spplist` and `spplabel`, read in it where the database is, the language and the prefixes of the labels. `sapupra.secret.ini` holds the secrets, the broker's passwords, and is for root and the services alone (group `sapupra`, mode 0640); it has the same syntax and the same sections, and its values win. The services read both; the commands, the first alone.

Listing 11: The configuration files, from their examples

```
$ cd /usr/local/etc
$ sudo cp sapupra.ini.sample sapupra.ini
$ sudo chown root:sapupra sapupra.ini
$ sudo chmod 0644 sapupra.ini
$ sudo cp sapupra.secret.ini.sample sapupra.secret.ini
$ sudo chown root:sapupra sapupra.secret.ini
$ sudo chmod 0640 sapupra.secret.ini
```

A password written in `sapupra.ini` still works, as in the earlier versions, but then the file would have to be closed again to the other accounts, and the commands could not read it; a service that finds a password or a token in a file that every account can read says so in its log. Move it to `sapupra.secret.ini`, under the same section and the same key.

In `sapupra.ini`, the gateway needs (Listing 12):

- in `[broker]`, `host`: the address that the `listener` line of the broker's configuration names, 192.168.1.10 in The broker: Mosquitto; not `localhost`, where that broker does not listen;
- in `[gateway]`, `listen`: the address and the port that the phones reach. A name that the machine resolves is accepted too (`localhost`, its own name); it is resolved once, at the start, and the first IPv4 address is taken (the log says which, in *listening on*). A name that does not resolve stops the program and says so;
- in `[gateway]`, `user`: the account `gateway` at the broker; and, in `sapupra.secret.ini` (Listing 13), `[gateway] password`: the password that `mosquitto_passwd` gave it.

The other values keep the defaults of the example; `sppgate(8)` describes each. A value is taken as it is written, from after the `=` to the end of the line, without the spaces around it: a password is written as it is, without quotes, whatever its characters (`;`, `#`, `\` and `"` included). A comment is a line of its own, starting with `;` or `#`.

The gateway gives the broker the account's name as its client identifier as well: the broker's log, which names a client by its identifier ("Client gateway … disconnected"), then names the account.

Listing 12: The values of sapupra.ini that the gateway needs

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

[gateway]
listen=192.168.1.10:8080
user=gateway
```

Listing 13: The values of sapupra.secret.ini that the gateway needs

```
[gateway]
password=THE-GATEWAY-PASSWORD
```

3.3.2 Trying it by hand

Before it becomes a service, the gateway is run by hand, with three terminals open on the machine, and the phone. Each step says where it is done, what is typed, and what comes back; the outputs are those of a trial on a test machine, its address replaced by the example's, 192.168.1.10. `PASSWORD` is the gateway's password at the broker.

1. Terminal 2, the account: check that the broker accepts the gateway's account and password, by publishing a test message with them (Listing 14). Nothing printed

means accepted. “not authorised” means that the broker does not know that password for `gateway`: give it again with `mosquitto_passwd`, then restart the broker, which reads its password file when it starts.

2. Terminal 1, the gateway: run it under its account, with its debugging messages (Listing 15). It stays in the foreground and prints what it does; it must say “connected to the broker”. Its second line gives the length of the password that it read from `sapupra.secret.ini`, never the password itself.

3. Terminal 2, what reaches the broker: listen to the topics of `sapupra`, with the same account (Listing 16). It prints nothing until a scan comes.

4. Terminal 3, the state of the gateway: ask for it (Listing 17); the phone’s browser, at the same address, shows the same line. “worker offline” is expected: the worker is not started yet (The worker).

5. The phone: scan a book. The phone answers “queued, worker offline”; terminal 2 prints the input that the gateway made of the scan, and terminal 1 one line for it (Listing 15, the last lines). The scan is at the broker, which keeps it for the worker. Listing 18 makes, from terminal 3, the request that Binary Eye makes, for a trial without the phone.

6. Terminal 1: Ctrl-C stops the gateway.

Listing 14: Terminal 2: the gateway’s account, accepted, then refused

```
$ mosquitto_pub -h 192.168.1.10 -u gateway -P PASSWORD -t sapupra/test -m test
$ mosquitto_pub -h 192.168.1.10 -u gateway -P WRONG-PASSWORD -t sapupra/test -m test
Connection error: Connection Refused: not authorised.
Error: The connection was refused.
```

Listing 15: Terminal 1: the gateway, and what it printed in the trial

```
$ sudo -u sapupra /usr/local/sbin/sppgate --debug
sppgate: info: listening on 192.168.1.10:8080; broker 192.168.1.10:1883
sppgate: debug: connecting to the broker at 192.168.1.10:1883 as gateway (client
    identifier gateway, password of 6 characters)
sppgate: info: sapupra 0.11.0.2 started
sppgate: info: connected to the broker at 192.168.1.10:1883
sppgate: debug: published bd5a199b-14a5-4da8-a0e2-4690cedfeb72 (publication 1, message
    4)
sppgate: info: pixel3a: scan "9782081231412": 202 queued, worker offline
^Csppgate: info: stopping (signal 2)
```

Listing 16: Terminal 2: what the gateway publishes

```
$ mosquitto_sub -h 192.168.1.10 -u gateway -P PASSWORD -t 'sapupra/v1/#' -v
```

```
sapupra/v1/input/pixel3a {"code":"9782081231412","device":"pixel3a","deviceTime
    ":"2026-10-26 18:02:11:417","id":"bd5a199b-14a5-4da8-a0e2-4690cedfeb72","kind":"
    scan","origin":"binaryeye","raw":{"content":"9782081231412","deviceId":"pixel3a","
    format":"EAN_13","timestamp":"2026-10-26 18:02:11:417"},"received":"2026-10-26T19
    :18:40.624+01:00","symbology":"EAN_13","v":1}
```

Listing 17: Terminal 3: the state of the gateway

```
$ curl http://192.168.1.10:8080/
sapupra gateway: broker connected, worker offline, 0 input(s) waiting for their receipt
```

Listing 18: Terminal 3: a scan as Binary Eye sends it, without the phone

```
$ curl -H 'Content-Type: application/json' -d '{"content":"9782081231412","format":"
    EAN_13","timestamp":"2026-10-26 18:02:11:417","deviceId":"pixel3a"}' 'http
    ://192.168.1.10:8080/scan?device=pixel3a'
queued, worker offline
```

When the gateway cannot reach the broker, terminal 1 says why, and what to check; the gateway keeps trying, every 5 seconds, and answers the phone “Not recorded, scan again” meanwhile. Listing 19 is what it printed with `host=localhost` while the broker listened on 192.168.1.10 only: nothing listens on port 1883 of `localhost`. A password that the broker refuses is reported as “not authorised”, with the account to check: step 1 tells whether the broker or `sapupra.ini` is wrong.

Listing 19: Terminal 1: a broker that does not listen at the address given

```
sppgate: warning: cannot connect to the broker at localhost:1883: Connection refused;
    nothing listens on localhost port 1883: is the broker running, and does the
    listener line of its configuration name this address? (sockstat -4 -l -p 1883 on
    FreeBSD, ss -ltn on GNU/Linux, show where it listens; [broker] host must be that
    address); trying again every 5 s
```

3.3.3 As a service on Debian

Listing 20 makes systemd read the unit that `make install` put in place (Service files), starts the gateway now and at every start of the machine, and shows its messages.

Listing 20: The gateway as a systemd service

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable --now sppgate
$ systemctl status sppgate
$ journalctl -u sppgate
```

3.3.4 As a service on FreeBSD

Listing 21 enables the rc script (`make install` puts it in place; the first line of the session, `install`, was needed until 0.16.0), starts the gateway, and checks it; its messages go to the system log, `/var/log/messages`. `daemon(8)` runs it as `sapupra`, writes its own process number in `/var/run/sppgate.pid`, and starts it again if it stops. Tried by the author on FreeBSD 14.5 (the session is Listing 21, as root). The last command printed nothing in 0.12.0: the messages were logged at a level that the default `/etc/syslog.conf` of FreeBSD does not keep in `/var/log/messages`. From 0.13.0 the normal messages are logged at `notice`, which it keeps, so the command shows the lines "started" and "stopping". `service sppgate restart` disturbs neither the broker nor the worker; the three are not interdependent (Starting, stopping and the order).

Listing 21: The gateway as an rc service on FreeBSD, in the trial of 0.12.0

```
# install -m 555 sppgate.rc /usr/local/etc/rc.d/sppgate
# sysrc sppgate_enable=YES
sppgate_enable:  -> YES
# service sppgate start
Starting sppgate.
# service sppgate status
sppgate is running as pid 96466.
# service sppgate stop
Stopping sppgate.
Waiting for PIDS: 96466.
# service sppgate status
sppgate is not running.
# grep sppgate /var/log/messages
```

3.4 The worker

The worker, `sppwork`, is the only program that writes the catalogue. It takes from the broker what the gateway publishes, records it in the database, looks the code up in the bibliographic sources, catalogues what it finds, and answers the gateway with a verdict, which the gateway turns into the answer of the phone. It is the one program of the local network that goes out to the Internet, to the sources; it needs nothing else from it. It runs under the account `sapupra`, like the gateway, and reads the same `sapupra.ini`.

3.4.1 What it needs

Beyond what the gateway needs, the worker needs Qt's SQLite driver (a package of its own on Debian: `libqt6sql6-sqlite`; on FreeBSD `qt6-base_sqldriver-sqlite`), a SQLite with the FTS5 full-text index, and TLS for the https requests to the sources. `configure` tries all three and says what is missing, with what to install (the FreeBSD package for the SQLite driver, `qt6-base_sqldriver-sqlite`, is confirmed). The machine must be able to

reach the sources on port 443: `openlibrary.org`, `catalogue.bnf.fr` and `musicbrainz.org`; and, for the German national library, `services.dnb.de` (port 443, SRU); for the national libraries of Hungary and Italy and the Library of Congress, which are reached by Z39.50 (a plain TCP connection, with no TLS), `tagetes2.oszk.hu` on port 1616, `opac.sbn.it` on port 2100 and `lx2.loc.gov` on port

1. The Z39.50 sources are

built if YAZ is installed (Debian `libyaz-dev`, FreeBSD `yaz`) when `configure` runs, and left out, with a notice, if it is not (`--with-yaz=no` leaves them out on purpose, `--with-yaz=yes` makes YAZ a prerequisite). The rest of the system stays on the local network.

3.4.2 The database and the configuration

The database is one file, `catalogue.db`, in a directory that belongs to the account `sapupra`, which alone writes in it, and that every account can enter (mode 0755), the files being readable by all (0644): the catalogue is no secret on the machine (the pages of the entry module show it to the whole network), and a person who is not `sapupra` must be able to list it with `spplist`. The worker creates the file. On Debian the unit makes `/var/lib/sapupra` by itself (systemd does it, at each start of the unit, by `StateDirectory=`; the worker never makes a directory), with that mode (As a service on Debian); on FreeBSD Listing 22 makes `/var/db/sapupra`. A directory made with the mode 0750 by an earlier version is opened with `chmod 0755` (and on Debian the unit sets it again at each start).

The files. While a program has the database open, SQLite keeps two more files beside it (the database is in *WAL mode*, which lets the entry module read while the worker writes):

`catalogue.db-wal` (*write-ahead log*) where the worker appends what it records, before it is moved into `catalogue.db`. SQLite moves it by itself when it holds 1000 pages (4 MB), which a catalogue of some hundreds of books never reaches: until then `catalogue.db` keeps the size it had when it was made, and the records are in the log;

`catalogue.db-shm` (*shared memory*) the index of the log, 32 KB, which the programs that have the database open share.

A new catalogue is 128 KiB (131072 bytes) before it holds a single book, and the log is empty (0 bytes) when the worker has just started or stopped: this is not a fault. SQLite never puts two tables, or two indexes, in one page of 4096 bytes. The schema has 15 tables (five of them the hidden tables of the full-text index), 14 indexes (seven of them made by SQLite for the keys that must be unique) and a view: 29 pages of their own, and 3 for the list of all of them, 32 pages in all. The file grows from there, a page at a time, as the books come in (`sqlite3 catalogue.db 'pragma page_count'` says how many pages it has).

So a copy of `catalogue.db` alone may be empty or old while the worker runs, and the two files must never be removed by hand then. When the worker stops it moves the whole log into `catalogue.db` and empties it, so that `catalogue.db` alone is the catalogue. The last program to close the database deletes the two files, but only a program that writes can: if

the entry module (which only reads) is stopped after the worker, they stay, `catalogue.db-wal` empty and `catalogue.db-shm` of 32 KB, harmless, reused at the next start. To leave `catalogue.db` alone in the directory, stop the entry module before the worker. Also in the directory, while the worker runs, `catalogue.db.worker.lock` (below) is removed when it stops.

Only one worker may write a database. The worker takes a lock beside it (the name of the database followed by `.worker.lock`), and a second one started on the same file refuses to start and says which process holds it ("another worker is running on this database"). If the first one died without leaving (a power cut), the lock is taken over by itself when the process is gone; if you are sure that none runs and the message stays, remove that file.

Listing 22: The directory of the database on FreeBSD

```
$ sudo install -d -o sapupra -g sapupra -m 0755 /var/db/sapupra
```

In `sapupra.ini`, the worker needs (Listing 23), besides `[broker]` as for the gateway:

- in `[worker]`, `user`: the account `worker` at the broker (the broker's password file, The broker: Mosquitto), and, in `sapupra.secret.ini`, `[worker] password` (Listing 24);
- in `[worker]`, `contact`: an address at which the operators of the sources can reach you (Open Library and MusicBrainz ask for one). The worker sends it with every request, and refuses to start without it;
- in `[database]`, `file`: where the catalogue is. Left out, it is `catalogue.db` in the directory that `configure` chose (`--with-databasedir`): `/var/lib/sapupra` with the systemd units, which make it, `/var/db/sapupra` with the rc scripts, else `sapupra` under the `localstatedir`. Up to 0.39.0 it was always under the `localstatedir` (`/usr/local/var` by default), which is not where the units make the directory: on Debian the worker then failed with *cannot be opened*. Writing the line stays right when the file is elsewhere.
- in `[worker]`, `copies`: what the worker does with the scan of a book that the catalogue holds already (outside the few seconds of a double read): `ask` (the default) holds it in the entry module's list, "already in the catalogue", until the admin adds the copy or discards the scan; `yes` catalogues every such scan as another copy, at once; `no` answers "already catalogued" and adds nothing. Only a scan of a barcode is concerned, not an ISBN typed or a book chosen.

The `[sources]` section keeps its example: Open Library, then the BnF, for every ISBN; MusicBrainz, then the BnF, for the product numbers of CDs and DVDs; no source for the other codes; `search` names the sources that the search by author and title asks (Open Library and the BnF), all of them, their books listed together. The sample shows how to ask the BnF first for French books (`isbn.978-2=bnf,openlibrary`). Built with YAZ, the worker asks by default the OSZK first for Hungarian books (978-963 and 978-615, then Open Library and the BnF) and SBN first for Italian ones (978-88 and 979-12),

in the sources `oszk` and `sbn`; the German national library (`dnb`) first for German books (978-3), then Open Library and the BnF, always; and, built with YAZ, the Library of Congress (`loc`) third for English books (978-0, 978-1 and 979-8: Open Library, the BnF, then the Library of Congress); one line of the file for the same prefix replaces that order, and `oszk.address`, `sbn.address` and `loc.address` (`host:port/database`) move a server (The Z39.50 sources). The same section says how every source is asked: `timeout` (seconds to wait, 60), `attempt` (seconds without any answer after which the request is made again, 8), `http2` and `keepalive` (both no): the defaults make each request open its connection and close it, which avoids a warning of Qt, and are settings in case a source does not answer to them (Trying the bibliographic sources by hand). `sppwork(8)` describes each value.

Listing 23: The values of sapupra.ini that the worker needs

```
[worker]
user=worker
contact=you@example.org

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

Listing 24: The values of sapupra.secret.ini that the worker needs

```
[worker]
password=THE-WORKER-PASSWORD
```

3.4.3 Trying it by hand

The worker is tried beside the gateway, as the gateway was tried beside the broker: two terminals for the programs, a third for the requests, and the phone. The outputs below are those of a trial of the author's, on a FreeBSD machine, with 0.12.0; its addresses are replaced by the example's, and its database was in `/var/db/sapupra` (on Debian: `/var/lib/sapupra`). Two things changed since: from 0.13.0 each debugging line begins with the time, as `10:41:07.512`; and the sample configuration asks MusicBrainz and the BnF for the product numbers, which 0.12.0 did not (it had no such source), so that the CD of the third request is found when the machine reaches `musicbrainz.org`.

1. Terminal 1: run the gateway as before (Listing 26, whose first line is the command). Terminal 2: run the worker, under its account, with its debugging messages (Listing 25, whose first line is the command). It stays in the foreground and prints what it does; it says that it is online. The state of the gateway (Listing 27, terminal

   (a) says so too: "worker online", where it said "worker offline"

   before.

2. Terminal 3, the requests that Binary Eye makes, one by one (Listing 27); or scan the same codes with the phone: a misread barcode, a location label, a product number

(the barcode of a CD), something that is no code at all, then books. The answer of each is on the last line of its request, with the status of the HTTP answer after it.

3. Terminal 2 and terminal 1 show the same scans, the worker's with the verdict, the gateway's with the answer given to the phone (Listing 25, Listing 26). A book that Open Library does not know (the Simondon, 9782081231412) is answered "no source knows ISBN …", then, when it is scanned again, "set aside, already waiting": it is in the list of the items that need a hand.

4. Look at the database (Listing 28): each scan is in the table `input`, with the status that the worker gave it (the text of its verdict is in the column `detail`).

5. Ctrl-C in terminal 2 stops the worker, which publishes that it is offline; terminal 1 says so.

Listing 25: Terminal 2: the worker, and what it printed in the trial of 0.12.0

```
# sudo -u sapupra /usr/local/sbin/sppwork --debug
sppwork: info: database /var/db/sapupra/catalogue.db: 0 input(s), 0 item(s)
sppwork: debug: connecting to the broker at 192.168.1.10:1883 as worker (client
    identifier worker, password of 10 characters)
sppwork: info: sapupra 0.12.0.25 started
sppwork: info: connected to the broker at 192.168.1.10:1883
sppwork: info: the worker is online
sppwork: info: pixel3a: scan "9782081231413": unrecognized: unrecognized code: check
    digit of an EAN-13 does not match: misread?
sppwork: info: pixel3a: location "SALON B3": located: location: SALON B3
sppwork: info: pixel3a: scan "3336550050675": not-isbn: product code 3336550050675: no
    source knows it (no source is configured for this kind of code); if this is a book,
     type its ISBN
sppwork: info: pixel3a: scan "ISBN-OF-A-BOOK": unrecognized: unrecognized code: not an
    ISBN, an EAN-13, an EAN-8 or a UPC-A
sppwork: debug: asking openlibrary about 9782081231412
sppwork: debug: Open Library does not know ISBN 9782081231412
sppwork: info: pixel3a: scan "978-2-0812-3141-2": not-found: no source knows ISBN
    9782081231412
sppwork: info: pixel3a: scan "9782081231412": repeated: set aside, already waiting
sppwork: warning: QIODevice::read (QSslSocket): device not open
sppwork: debug: asking openlibrary about 9780141988313
sppwork: info: pixel3a: scan "9780141988313": resolved (item 1): Revolutionary Spring,
    Christopher Clark (2024)
^Csppwork: info: stopping (signal 2)
```

The warning `QIODevice::read (QSslSocket): device not open` comes from Qt's TLS socket, not from `sapupra`; it was seen once, and the scan after it was answered. 0.13.0 asks for a connection of its own for every request (no HTTP/2, `Connection: close`), and

the line did not come back in the trial of 0.13.0 (the author, 2026-10-27: no warning at all); whether it is also why Open Library timed out in that trial is not known (Trying the bibliographic sources by hand).

Listing 26: Terminal 1: what the gateway printed during the same trial

```
# sudo -u sapupra /usr/local/sbin/sppgate --debug
sppgate: info: listening on 192.168.1.10:8080; broker 192.168.1.10:1883
sppgate: debug: connecting to the broker at 192.168.1.10:1883 as gateway (client
    identifier gateway, password of 10 characters)
sppgate: info: sapupra 0.12.0.25 started
sppgate: info: connected to the broker at 192.168.1.10:1883
sppgate: info: the worker is online
sppgate: debug: published 6656a1f8-3308-429f-b4b8-085f04c02b8a (publication 1, message
    4)
sppgate: info: pixel3a: scan "9782081231413": 422 unrecognized code: check digit of an
    EAN-13 does not match: misread?
sppgate: debug: published db42472d-cded-4389-853c-cb9d53cd8440 (publication 1, message
    5)
sppgate: info: pixel3a: location "SALON B3": 200 location: SALON B3
sppgate: debug: published fed4255e-7f9d-49df-a1f9-5d73ca64469d (publication 1, message
    6)
sppgate: info: pixel3a: scan "3336550050675": 422 product code 3336550050675: no source
     knows it (no source is configured for this kind of code); if this is a book, type
    its ISBN
sppgate: debug: published 0513d254-b829-41d7-afb9-96d8b8eb9636 (publication 1, message
    7)
sppgate: info: pixel3a: scan "ISBN-OF-A-BOOK": 422 unrecognized code: not an ISBN, an
    EAN-13, an EAN-8 or a UPC-A
sppgate: debug: published 2b50d492-58f1-44b6-b537-5cd01f094383 (publication 1, message
    8)
sppgate: info: pixel3a: scan "978-2-0812-3141-2": 422 no source knows ISBN
    9782081231412
sppgate: debug: published 03aa4119-ac88-40d9-b61d-a4699ef07d2d (publication 1, message
    9)
sppgate: info: pixel3a: scan "9782081231412": 422 set aside, already waiting
sppgate: debug: published e53dc149-65a7-43c5-ba55-1f275a5d5b2b (publication 1, message
    10)
sppgate: info: pixel3a: scan "9780141988313": 200 Revolutionary Spring, Christopher
    Clark (2024)
sppgate: info: the worker is offline
^Csppgate: info: stopping (signal 2)
```

The requests of terminal 3 follow, each with `-w`, which adds the status of the answer (Listing 27). In the trial, the last four were made without it, and their answers ran together on the screen (“unrecognized code:curl -H …”); what is shown for them is their answer as

the gateway's log gives it (Listing 26), with the status that `-w` adds. A request that has a status of 422 is answered with an error: Binary Eye sounds its error signal.

Listing 27: Terminal 3: the state of the gateway, and the scans as Binary Eye sends them

```
$ curl http://192.168.1.10:8080/
sapupra gateway: broker connected, worker online, 0 input(s) waiting for their receipt
$ curl -w ' [%{http_code}]\n' -H 'Content-Type: application/json' -d '{"content
    ":"9782081231413","format":"EAN_13","timestamp":"2026-10-26 18:02:11:417","deviceId
    ":"pixel3a"}' 'http://192.168.1.10:8080/scan?device=pixel3a'
unrecognized code: check digit of an EAN-13 does not match: misread? [422]
$ curl -w ' [%{http_code}]\n' -H 'Content-Type: application/json' -d '{"content":"LOC:
    SALON B3","format":"QR_CODE","timestamp":"2026-10-26 18:03:00:000","deviceId":"
    pixel3a"}' 'http://192.168.1.10:8080/scan?device=pixel3a'
location: SALON B3 [200]
$ curl -w ' [%{http_code}]\n' -H 'Content-Type: application/json' -d '{"content
    ":"3336550050675","format":"EAN_13","timestamp":"2026-10-26 18:04:00:000","deviceId
    ":"pixel3a"}' 'http://192.168.1.10:8080/scan?device=pixel3a'
product code 3336550050675: no source knows it (no source is configured for this kind
    of code); if this is a book, type its ISBN [422]
$ curl -w ' [%{http_code}]\n' -H 'Content-Type: application/json' -d '{"content":"ISBN-
    OF-A-BOOK","format":"EAN_13","deviceId":"pixel3a"}' 'http://192.168.1.10:8080/scan?
    device=pixel3a'
unrecognized code: not an ISBN, an EAN-13, an EAN-8 or a UPC-A [422]
$ curl -w ' [%{http_code}]\n' -H 'Content-Type: application/json' -d '{"content
    ":"978-2-0812-3141-2","format":"EAN_13","deviceId":"pixel3a"}' 'http
    ://192.168.1.10:8080/scan?device=pixel3a'
no source knows ISBN 9782081231412 [422]
$ curl -w ' [%{http_code}]\n' -H 'Content-Type: application/json' -d '{"content
    ":"9782081231412","format":"EAN_13","deviceId":"pixel3a"}' 'http
    ://192.168.1.10:8080/scan?device=pixel3a'
set aside, already waiting [422]
$ curl -w ' [%{http_code}]\n' -H 'Content-Type: application/json' -d '{"content
    ":"9780141988313","format":"EAN_13","deviceId":"pixel3a"}' 'http
    ://192.168.1.10:8080/scan?device=pixel3a'
Revolutionary Spring, Christopher Clark (2024) [200]
```

Listing 28: Terminal 3: the inputs in the database, with the sqlite3 program (package sqlite3)

```
$ sudo -u sapupra sqlite3 /var/db/sapupra/catalogue.db 'select seq, device, kind, code,
     class, status from input'
┌─────┬──────────┬──────┬──────────────────────┬──────────┬──────────┐
│ seq │  device  │ kind │         code         │  class   │  status  │
├─────┼──────────┼──────┼──────────────────────┼──────────┼──────────┤
```

```
  1 | pixel3a | scan     | '9782081231413'   | unrecognized | unrecognized ||
  2 | pixel3a | location | NULL              | NULL         | located      ||
  3 | pixel3a | scan     | '3336550050675'   | gtin         | not-isbn     ||
  4 | pixel3a | scan     | ISBN-OF-A-BOOK    | unrecognized | unrecognized ||
  5 | pixel3a | scan     | 978-2-0812-3141-2 | isbn         | not-found    ||
  6 | pixel3a | scan     | '9782081231412'   | isbn         | repeated     ||
  7 | pixel3a | scan     | '9780141988313'   | isbn         | resolved
  |└────┴─────────┴──────────┴──────────────────┴──────────────┴──────────────┘
```

This SQLite shell drew the table in its box form, and wrote between quotes the text values that look like numbers; another version prints one line for each row, its fields separated by a vertical bar.

The first lookups, with the real source: Clark's book was found by Open Library, the author's name read from the author's page (two requests, one a second), and catalogued as item 1; the Simondon was not known there, so it was set aside. The worker then asks the BnF, from 0.13.0, which knows it (found in the trial of 0.13.0); a book that is in no source stays in the list of the items that need attention. With `--debug` the worker says which source it asks.

3.4.4 What it does with a scan

The phone's answer depends on the worker's verdict:

- the title (and the author and the year), when the book is catalogued; with "(again)" when the same code was read twice within a few seconds, or while it was still being looked up: one book, one item;

- another copy of an edition that the catalogue holds is, as the admin has set `copies`, catalogued at once without any request to the sources, held until the admin decides ("Already in the catalogue, not added", the default), or answered "already catalogued" and left out;

- "queued", when a source did not answer: the worker tries again after 1 minute, 10 minutes, 1 hour, then every day, for a week, and nothing is asked of you; the result goes to the catalogue;

- an error, for a code that no source knows (set aside), for a product number that is not an ISBN, for a misread barcode, or for several records that the entry module will let you choose from. A code set aside and scanned again is answered "set aside, already waiting", and adds nothing to the list.

The location that the phone last set is the location of the items that follow, until another label is scanned.

3.4.5 Backing up the catalogue

The catalogue is the work of the collection. Copy it with SQLite's own command, under the account of the worker (Listing 29): it gives a consistent copy whether the services run or not, after a crash or a power cut as well. The directory of the copy must exist and be writable by that account. How an ordinary account is allowed to run it: Running commands as the account sapupra.

Listing 29: A copy of the catalogue

```
$ sudo -u sapupra sqlite3 /var/lib/sapupra/catalogue.db ".backup /var/lib/sapupra/
    catalogue-backup.db"
```

Copying the file alone (`cp`) is right only when the worker has been stopped (it moves its log into the file then, The database and the configuration), never after a crash or a power cut, which leave the log to be read.

As an ordinary account. The same command, run by an account that may read the files but not write in the directory, fails when no program has the database open:

```
Error: attempt to write a readonly database
```

and leaves an empty file where the copy was to be. The reason is that SQLite, to read a database in WAL mode, needs the two files `-wal` and `-shm` beside it (The database and the configuration); when no program has the database open they do not exist, and an account that may not write in the directory cannot make them. Look at the size of the copy: an empty file is not a backup. What works for such an account depends on what the directory holds:

- the two files `-wal` and `-shm` are there and readable (a service, or the entry module that was stopped last, left them): the same command works, and the copy is complete;
- there is no `-wal` file, or it has a size of 0, which is the usual state after a stop of the worker: the main file is the whole catalogue, and a copy of it is right: `cp`, or SQLite told to ignore the log (Listing 30);
- the `-wal` file has content (the worker crashed, or it is running): do not copy the main file, neither with `cp` nor with the option `immutable`, which both ignore the log and give an old catalogue, silently, without an error. Use the command of the account of the worker.

Listing 30: A copy of the main file by an ordinary account, when there is no log

```
$ ls -l /var/lib/sapupra/
$ sqlite3 -readonly "file:/var/lib/sapupra/catalogue.db?immutable=1" ".backup catalogue
    -backup.db"
$ ls -l catalogue-backup.db
```

3.4.6 Upgrading to a newer version

The catalogue carries the number of its version (the `user_version` of SQLite). When the worker starts with a catalogue of an older version, it upgrades it, once, and says so in the log ("the database is upgraded from version N to M"). The steps of every version since the one found are done in a single transaction: if one fails, nothing is changed and the worker does not start, with the reason in the log. To upgrade:

1. Make a copy of the catalogue (Backing up the catalogue). It is the only way back: a catalogue upgraded is not read by an older version, which refuses it ("made by a later sapupra").

2. Stop the services, install the new version, and read its entry in the file `NEWS`, which says what changed.

3. Start the worker first: it upgrades the catalogue. The entry module and the web front open the catalogue read-only, cannot upgrade it, refuse an older one and say to start the worker first.

4. Start the others.

Nothing in the upgrade removes data: a version adds a column, a table or an index, or completes what an older version stored. The list of the versions is in `Design.org`, section "The database".

3.4.7 Running commands as the account sapupra

The manual asks for `sudo -u sapupra` in three places: the copy of the catalogue (Backing up the catalogue), the trials by hand of the services, and the reading of a database that a stopped worker left (The database and the configuration). Who is to be allowed, and to do what, is a question of `sudoers`; the answer that asks for the least:

- On Debian the administrator needs nothing: the group `sudo` has every right in the `sudoers` of the system, and `sudo -u sapupra` is one of them.

- On FreeBSD a group gives no right to `sudo` by itself. Being in the group `wheel` allows `su` to root; `sudo` is a package, and what it allows is what its `sudoers` says, in which the line for the group `wheel` is a comment in the sample, as far as the author knows (see the commands below). The administrator is allowed by that line, or goes without `sudo`.

- A person who is not an administrator and has to make the copy gets the one command, in a file of its own, with its whole command line and no wildcard (Listing 33, Listing 34).

- Nobody is put in the group `sapupra`, which can read `sapupra.secret.ini`, nor given `sqlite3` or any program of `sapupra` with any argument: `sqlite3` runs a shell (`.shell`, `.system`) and writes where it is told, and the account `sapupra` reads the secrets, so that this would be to give them.

On FreeBSD, who is allowed? Ask `sudo` itself, as the person concerned, and look at the line for `wheel` (the file is `/usr/local/etc/sudoers`). The membership that `pw group show wheel` lists is right; the group of the person's own name, listed without members, is his primary group and says nothing against it.

Listing 31: What sudo allows, on FreeBSD

```
$ sudo -l
$ grep -n wheel /usr/local/etc/sudoers
```

If the line `# %wheel ALL=(ALL:ALL) ALL` is a comment, the group `wheel` has no right with `sudo`: remove the `#` with `visudo` (which checks the file before it saves it), or give `peter` the one command below. As far as the author knows, `sudo` reads the groups of a person from the account database at each command, not from the login, so that a new session does not matter. This was not tried on FreeBSD; the reasons are in the journal.

Without sudo, as root (a member of `wheel` becomes root with `su -`):

Listing 32: A command as the account sapupra, with no sudo

```
# su -m sapupra -c 'sqlite3 /var/db/sapupra/catalogue.db .tables'
```

The option `-m` keeps the shell of the caller, so that the account `sapupra`, whose shell is `nologin`, runs the command all the same (not tried here).

The file is made with the command `visudo -f`, which refuses a file that does not parse, in the directory `/etc/sudoers.d/` (`/usr/local/etc/sudoers.d/` on FreeBSD), mode 0440, owned by root. The `sudoers` of Debian includes that directory; on FreeBSD, look for the line `includedir` in the file of the package. The directory of the copy is made once, owned by `sapupra`, which is the account that writes in it.

Listing 33: The one command, for an ordinary account (Debian paths)

```
# install -d -o sapupra -g sapupra -m 0755 /var/backups/sapupra
# visudo -f /etc/sudoers.d/sapupra
```

Listing 34: The content of /etc/sudoers.d/sapupra

```
Cmnd_Alias SAPUPRA_BACKUP = /usr/bin/sqlite3 /var/lib/sapupra/catalogue.db .backup /var
    /backups/sapupra/catalogue.db
peter ALL = (sapupra) NOPASSWD: SAPUPRA_BACKUP
```

Listing 35: The copy, made by the ordinary account

```
$ sudo -u sapupra /usr/bin/sqlite3 /var/lib/sapupra/catalogue.db \
    ".backup /var/backups/sapupra/catalogue.db"
```

Then `peter` makes the copy with the command of Listing 35, whole, and with no other: another destination, `.shell` or any other argument is refused (tried here with `visudo -c` and a test account: the command was accepted and made a complete copy, and a copy to another file, a `.shell` and `sqlite3` with any other argument were refused). On FreeBSD the program is `/usr/local/bin/sqlite3`, the database `/var/db/sapupra/catalogue.db` and the directory of the copy `/var/backups/sapupra`. `NOPASSWD` is for a copy by a script; leave it out if the person is to type his password. The copy has a fixed name, and the next one replaces it: take it away (`rsync`, `scp`) to keep several.

A copy at fixed hours does not need `sudo`: the table of `cron` of the account `sapupra` itself (`crontab -u sapupra -e`, as root) runs the same command line, with no `sudo` (not tried here).

3.4.8 As a service on Debian

Listing 36 makes systemd read the unit that `make install` put in place, which makes `/var/lib/sapupra` for the account, starts the worker now and at every start of the machine, and shows its messages. The unit orders the start after the broker's; it does not require it (Starting, stopping and the order). Not tried yet on Debian.

Listing 36: The worker as a systemd service

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable --now sppwork
$ systemctl status sppwork
$ journalctl -u sppwork
```

3.4.9 As a service on FreeBSD

Listing 37 enables the rc script (`make install` puts it in place; the first line of the session, `install`, was needed until 0.16.0), starts the worker, and checks it; as for the gateway, `daemon(8)` runs it as `sapupra` and starts it again if it stops, and its messages go to `/var/log/messages`. The directory of the database is made first (Listing 22). Tried by the author on FreeBSD 14.5 (the session is Listing 37, as root); as for the gateway, the last command printed nothing in 0.12.0 and shows the worker's lines from 0.13.0 (Listing 38, the trial of 0.13.0).

Listing 37: The worker as an rc service on FreeBSD, in the trial of 0.12.0

```
# install -m 555 sppwork.rc /usr/local/etc/rc.d/sppwork
# sysrc sppwork_enable=YES
sppwork_enable:  -> YES
# service sppwork start
Starting sppwork.
# service sppwork status
sppwork is running as pid 96538.
# service sppwork stop
```

```
Stopping sppwork.
Waiting for PIDS: 96538.
# service sppwork status
sppwork is not running.
# grep sppwork /var/log/messages
```

The system log of the trial of 0.13.0 on the same machine (Listing 38) shows the whole path of three scans. The worker came back to its session when it was restarted (the broker's line "session taken over"). The Simondon (ISBN

1. was asked of Open Library, which did not answer in

15 seconds, and was found by the BnF; the gateway had answered "queued" after its 3 seconds. The CD (3336550050675) was found by MusicBrainz in less than a second, whatever Open Library did, each source having its queue. The Penguin edition (ISBN 9780141988313), which Open Library had answered in the trial of 0.12.0, was not answered, twice, and waits for its next attempt.

Listing 38: The system log of the trial of 0.13.0 (FreeBSD)

```
Oct 27 13:32:37 bhavana sppgate[411]: listening on 192.168.111.10:8080; broker
    192.168.111.10:1883
Oct 27 13:32:37 bhavana sppgate[411]: sapupra 0.13.0.27 started
Oct 27 13:32:37 bhavana sppgate[411]: connected to the broker at 192.168.111.10:1883
Oct 27 13:32:37 bhavana sppwork[432]: database /var/db/sapupra/catalogue.db: 0 input(s)
    , 0 item(s)
Oct 27 13:32:37 bhavana sppwork[432]: sapupra 0.13.0.27 started
Oct 27 13:32:37 bhavana mosquitto[96292]: 1793104357: Client worker
    [192.168.111.10:16402] disconnected: session taken over.
Oct 27 13:32:37 bhavana mosquitto[96292]: 1793104357: New client connected from
    192.168.111.10:47266 as worker (p4, c0, k30, u'worker').
Oct 27 13:32:37 bhavana sppwork[432]: connected to the broker at 192.168.111.10:1883
Oct 27 13:32:37 bhavana sppwork[432]: the worker is online
Oct 27 13:32:37 bhavana sppgate[411]: the worker is online
Oct 27 13:50:19 bhavana sppgate[411]: pixel3a: scan "9782081231412": 202 queued
Oct 27 13:50:24 bhavana sppwork[432]: pixel3a: scan "3336550050675": resolved (item 1):
     Concertos pour flûte, Wolfgang Amadeus Mozart (1990)
Oct 27 13:50:24 bhavana sppgate[411]: pixel3a: scan "3336550050675": 200 Concertos pour
     flûte, Wolfgang Amadeus Mozart (1990)
Oct 27 13:50:30 bhavana sppwork[432]: Open Library does not answer: Operation timed out
Oct 27 13:50:30 bhavana sppwork[432]: pixel3a: scan "9782081231412": resolved (item 2):
     Du mode d'existence des objets techniques (Nouvelle éd. 2024 augmentée), Gilbert
    Simondon et al. (2024)
Oct 27 13:50:35 bhavana sppgate[411]: pixel3a: scan "9780141988313": 202 queued
Oct 27 13:50:49 bhavana sppwork[432]: Open Library does not answer: Operation timed out
Oct 27 13:50:49 bhavana sppwork[432]: pixel3a: scan "9780141988313": unavailable (
    attempt 1): Open Library does not answer: Operation timed out
```

```
Oct 27 13:52:15 bhavana sppwork[432]: Open Library does not answer: Operation timed out
Oct 27 13:52:15 bhavana sppwork[432]: pixel3a: scan "9780141988313": unavailable (
    attempt 2): Open Library does not answer: Operation timed out
Oct 27 13:55:21 bhavana sppwork[432]: stopping (signal 15)
Oct 27 13:55:21 bhavana sppgate[411]: the worker is offline
Oct 27 13:55:21 bhavana mosquitto[96292]: 1793105721: Client worker
    [192.168.111.10:47266] disconnected.
Oct 27 13:55:21 bhavana sppgate[411]: stopping (signal 15)
```

The worker's lines need no `--debug`. Open Library's timeouts are the subject of Trying the bibliographic sources by hand; from 0.14.0 the line says how long the worker waited, and the debug lines give each request's duration and protocol.

3.4.10 The level of the log

Each program has a level for its log, in its section of `sapupra.ini` (`log_level` in `[gateway]`, in `[worker]` and in `[entry]`, with the same words and the same options):

- `quiet`: the warnings and the errors;
- `normal` (the default): and what the program does (a scan, a connection, the worker's verdicts);
- `debug`: and the details (each publication of the gateway, each request of the worker to a source, with its status, its duration and the protocol, and, for a search or a code, the sources in the order they are asked, each skipped one with the reason, and what each answered; each connection of a client to the gateway, the entry module and the consultation, then its address, its method, its path and the status of the answer, or that the client left before it: never the query of the address nor the body, which may hold what was typed).

The options `-q` and `-d` (`--quiet`, `--debug`) of the command line win over the file. So a service can be left at `normal` in the file while a run by hand asks for `-d` (Listing 39). Stop the service first: a second worker with the same account takes the first one's session at the broker ("session taken over" in the broker's log), and both would write the catalogue.

Listing 39: Debugging by hand, the service stopped, then started again

```
# service sppwork stop
# sudo -u sapupra /usr/local/sbin/sppwork --debug
...
# service sppwork start
```

To have the debugging messages from the service itself, write `log_level=debug` in the section and restart it. Where they land is the system's choice, which the programs do not change; none of the following has been tried:

- Debian: the unit does not use `--syslog`; the lines go from standard error to the journal, which keeps them all: `journalctl -u sppwork`.

- FreeBSD: the rc script uses `--syslog`, which sends the debugging messages to syslog at the level *debug*, and the default `/etc/syslog.conf` keeps only *notice* and above in `/var/log/messages`. A file of their own, for the programs, is made in four steps: the lines of Listing 40, in the file `/etc/syslog.d/sapupra.conf`; then the commands `touch /var/log/sapupra.log`, `chmod 640 /var/log/sapupra.log` and `service syslogd reload`. The daemon does not create the file, and `newsyslog(8)` rotates it once told. Put the level back to `normal` when done: a debugging log of a worker that is asked for a thousand books is long.

Listing 40: A log file for the programs of sapupra, FreeBSD

```
!sppgate,sppwork,sppentry,sppconsult
*.*    /var/log/sapupra.log
!*
```

3.4.11 Starting, stopping and the order

The broker, the gateway, the worker and the entry module are four services (a fifth, `sppconsult`, if the catalogue is to be read from outside, Reaching the catalogue from outside), and none needs another to be running at the moment it starts:

- the gateway answers the phone "queued, worker offline" while the worker is away, and "Not recorded, scan again" while the broker is away;

- the worker records what the broker holds for it when it comes back, and answers then;

- the entry module says on its pages that the broker or the worker is away, and sends nothing meanwhile (the broker) or what the broker keeps for the worker (the worker);

- the programs reconnect to the broker every 5 seconds, wherever it is (`[broker] host` can name another machine).

So the service files only *order* the start, so that on a machine that boots, the broker is up first (Debian: `After=` in the units; FreeBSD: `REQUIRE:` in the rc scripts). They do not require it: a restart of the broker, of the gateway or of the worker stops nothing else. To restart one is to restart that one. The units, and the rc scripts, are made by `make` from one template for each system (`sapupra.service.in`, `sapupra.rc.in`): they differ by the name of the program and, in the unit of the worker, by the directory of the database.

3.4.12 The Z39.50 sources

The national libraries of Hungary and Italy and the Library of Congress answer by Z39.50, a protocol older than the web, in which a client opens a connection, searches an index (the ISBN is the use attribute 7 of the Bib-1 attribute set) and asks for the records, here as MARC, the librarians' format.

- `oszk`: the National Széchényi Library of Hungary, `tagetes2.oszk.hu:1616/B1`, MARC 21. It finds a book by its ISBN-10 and not by its ISBN-13 (the author's run of 2026-10-27), so a 978 ISBN-13 is asked as its ISBN-10 first;
- `sbn`: the Italian union catalogue (the Servizio Bibliotecario Nazionale), `opac.sbn.it`, port 2100, base `nopac`, UNIMARC. It finds a book by its ISBN-13; the ISBN-10 is asked if that fails. It may hold several editions under one ISBN (three of the Eco's *Il pendolo di Foucault*, from 2011, 2010 and 2006): the first record is the book, and all are kept as the sources of the item;
- `loc`: the Library of Congress, `lx2.loc.gov:210/LCDB`, MARC 21 (syntax `usmarc`), for the American users. It finds a book by its ISBN-13; a book it does not know is a source that has nothing, not an error.

The OSZK can also be searched by author and title: name `oszk` in `[sources] search`, as `openlibrary, bnf, oszk`. The author is asked on the name index, which matches each word of the name anywhere in the record and ignores accents; the title index matches the first words of a title only, so the title is used alone, and, when there is an author too, to choose among the books of that author. The first ten books are offered. The titles are cleaned of the punctuation of the cataloguers (a space and a slash, a space and a colon, a final full stop), the names written "Family, Given" are turned into "Given Family", and the first five records of an answer are read. A server that does not answer is a source "unavailable": the lookup goes on with the next source and, if none finds the book, is tried again later as for any source. They can be tried one by one with `yaz-client` (Trying the bibliographic sources by hand). Tried on 2026-10-28 with a Hungarian book (Kertész, *Felszámolás*) and an Italian one (Eco, *Il pendolo di Foucault*): both libraries answered, the books were catalogued with accents right. Where SBN has several editions under one ISBN, the first is taken (a choice among them is provisioned, to be asked for: FutureResearch.pdf). The OSZK serves from 03:30 to 23:00 local time; outside, the source is "unavailable" and the lookup goes on to the next.

3.5 The entry module

The entry module, `sppentry`, serves the pages with which what the worker could not settle is settled by hand (Items that need attention). It publishes inputs on the broker, as the gateway does, and reads the database, which only the worker writes. Built and installed with the others (Building the programs), it is in `/usr/local/sbin/`, its manual page is `sppentry(8)`, and its service file is put in place by `make install` (Service files).

3.5.1 What it needs

- the account `entry` at the broker, made with the others (The broker: Mosquitto);
- the database, which the worker makes at its first start; until then the pages say so, and open it when it is there;
- to run as `sapupra`, the account of the worker, as it opens the database read-only and the services share the files that SQLite keeps beside it (The database and the configuration);
- optionally, the library of Zint, to draw the code of a place on the screen (Locations); without it everything else works;
- a port: 8081 by default, to be reached by the phone and the browsers of the network. There is no password on these pages: anyone on the local network can settle a scan and enter an item, as anyone can open the catalogue. The network is the protection (Design.pdf).

3.5.2 The configuration

In `sapupra.ini`, beside the values that the gateway and the worker already use (Listing 41): in `[entry]`, `listen` and `user`, and in `sapupra.secret.ini` (Listing 42) `password`; the database and the location prefix are those of the worker, in `[database]` and `[locations]`, and the prefix of the labels of items is `prefix` of `[items]`. `max_upload` is the largest file that the import takes, in MiB (4 by default, some ten thousand scans); `default_device` is the phone offered for an item entered by hand when no other is chosen; `log_level` is that of the other programs (The level of the log). The language of the pages is the key `language` of `[interface]`, shared with the gateway (The language of the pages).

Listing 41: The values of sapupra.ini that the entry module needs

```
[entry]
listen=192.168.1.10:8081
user=entry
```

Listing 42: The values of sapupra.secret.ini that the entry module needs

```
[entry]
password=THE-ENTRY-PASSWORD
```

3.5.3 The language of the pages

The pages are in English. To have them in French, put in `sapupra.ini`, for the entry module and the gateway (the gateway has one sentence for the phone, “Not recorded, scan again”):

Listing 43: The language of the pages

```
[interface]
language=fr
```

and restart the services. The language is a catalogue, the file `sapupra.mo`. When the catalogues were made, `make install` puts it in `fr/LC_MESSAGES`, under the directory `share/locale` of the installation (under `/usr/local` by default); another language has its code in place of `fr`. They are made at the build by the tools of the package `gettext` (Debian: `apt install gettext`; FreeBSD: `pkg install gettext`, then `./configure` finds the tools in `/usr/local/bin`); `configure` says at its end whether it found them. Without them, or with `--without-gettext`, the programs are built all the same and the pages are in English. The machine that runs the programs needs nothing of gettext and no locale: the programs read the catalogue themselves, whatever `LANG` is (a service has none). With `language=en`, or no key, the English is the text of the code; with a language that has no catalogue, the pages stay in English and the log says which file was looked for and which languages are installed.

What is translated: the pages of the entry module (the navigation, the titles, the forms, the notices) and the gateway's reply to the phone. What is not: the log, the manual pages, the header of the CSV file, the descriptions of the books (titles, authors, publishers, subjects, places, locations), and the texts that the worker stores in the database (they are the worker's, in English, and are shown as they are). The route names and the names of the fields of the forms do not change, so that a bookmark or a script works in any language.

The words that the database holds (the medium, how an edition was described, the role of a contributor, the language of the text, the kind of an input, the reason of a removal) are keywords and codes that the pages say in the language of the pages: a catalogue entered with the English pages shows *livre*, not *book*, on a French page, and the languages that the sources write as `fre` or as `French` are both *français*. Searching "book" or "livre" is the same thing: the search takes the keyword and the word of the page. The names of places, and what you typed in a description, are yours and stay as you typed them. A word that the program does not know is shown as it was written. The titles and the subjects stay in the language of their book.

Adding a language, say Hungarian (`hu`): `make pot` writes the list of the sentences, `sapupra.pot`; copy it to `hu.po`, give its header (`msginit -l hu -i sapupra.pot -o hu.po` does it), translate every sentence, keeping the markup (the tags, the places `%1` to `%9` and the line ends) as they are and giving the forms of the plurals that the language needs; add `hu` to `LINGUAS` and `hu.po` to `EXTRA_DIST` in `Makefile.am` (the catalogue is then built, shipped and installed), run `./bootstrap` and `make`. `make check` refuses a catalogue that lacks a sentence of the code or keeps one that the code no longer has (`make update-po` brings the catalogues in step), that has an untranslated or a fuzzy sentence, or that changes the markup of a sentence.

3.5.4 Trying it by hand

With the broker and the worker running (The worker), the entry module is run in the foreground, as the account of the worker, with the debugging messages (Listing 44). It says that it listens, that it is connected to the broker, that the worker is online and that the database is open. The pages are then opened in a browser; `curl` shows the same as the browser (Listing 45). A scan that the worker cannot recognize (the code `12345`, sent with the gateway) is a good first case: it stays in the list, the form for the description by hand settles it, and the worker's log shows the line "record: resolved".

Listing 44: The entry module by hand

```
$ sudo -u sapupra /usr/local/sbin/sppentry --debug --config /usr/local/etc/sapupra.ini
```

Listing 45: The first page from the machine

```
$ curl -s http://192.168.1.10:8081/ | grep -c settle
```

The pages say at their top what is wrong: when the broker cannot be reached, nothing can be sent and the pages show what the database holds; when the worker is offline, what is sent is kept by the broker and recorded when the worker is back.

3.5.5 As a service on Debian

Listing 46 makes systemd read the unit that `make install` put in place, starts the entry module now and at every start of the machine, and shows its messages. The unit orders the start after the broker's and requires nothing (Starting, stopping and the order). Not tried yet on Debian.

Listing 46: The entry module as a systemd service

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable --now sppentry
$ systemctl status sppentry
$ journalctl -u sppentry
```

3.5.6 As a service on FreeBSD

Listing 47 enables the rc script that `make install` put in place and starts the entry module; as for the others, `daemon(8)` runs it as `sapupra` and its messages go to the system log. Run as a service on FreeBSD by the author (2026-10-27): it started, listened and opened the database.

Listing 47: The entry module as an rc service on FreeBSD

```
# sysrc sppentry_enable=YES
# service sppentry start
# service sppentry status
```

```
# grep sppentry /var/log/messages
```

3.6 The web front

There is none to install: the pages of the catalogue are those of the entry module (The entry module) and it serves them, from the same address as the pages that settle the scans (`/catalogue`). They open the database for reading only; the worker keeps writing it. The view `catalogue` is what they query (Design.pdf). Nothing to configure: if the entry module runs, the catalogue can be read.

3.6.1 Reaching the catalogue from outside

The pages of the entry module are made for the local network and defend nothing else: no program asks a visitor who he is, the pages are plain HTTP, and the pages that settle scans, enter items and move copies are the same as the pages that consult (Design.pdf gives the reasons). To read the catalogue from elsewhere, from the simplest and safest:

1. a virtual private network into the local network (WireGuard on the router or the machine). Only its port is opened to the internet; from outside, the pages are at the same address as at home, the phone's scans included. Nothing to add to `sapupra`;

2. an SSH tunnel, for a visit from one machine (Listing 48), then the address `localhost`, port 8081, on that machine;

3. `sppconsult`, the program that consults and does nothing else, behind a proxy that gives the encryption and the password: Caddy, which is the shorter to set up and keep (A proxy: Caddy), or nginx for whoever already runs it (A proxy: nginx).

Listing 48: Reading the catalogue through an SSH tunnel

```
$ ssh -N -L 8081:192.168.1.10:8081 you@your-home-address
```

Whatever the way, never open the broker (port 1883), the gateway (8080) or the entry module (8081) themselves to the internet, with a forwarding of the router or otherwise: the broker's passwords are the only protection of the topics, and the pages of the entry module have none and write.

3.6.1.1 The program sppconsult

`sppconsult` is installed with the others, and has a manual page (`sppconsult(8)`). It serves the pages of the search, the page of an item and the CSV file, and no other: every other address answers 404 and every method but GET (and HEAD) answers 405. It does not connect to the broker, has no password, does not read `sapupra.secret.ini`, and opens the database to read it. The pages show no form, no menu but the catalogue, none of the commands that print labels, no removed copy, and not the devices and the times at which an item was scanned; the footer gives no version. A

field longer than 200 characters is refused, and a search that lasts more than 3 seconds is cut and says so (the patterns are tried by the program: a pattern made to be slow costs the 3 seconds and no more). It listens on this machine alone, port 8082 by default (Listing 49); told to listen on another address (a name is resolved, as for the gateway) it starts, and the log says that the pages are then open to whoever reaches the address.

Listing 49: The values of sapupra.ini that sppconsult uses

```
[consult]
listen=127.0.0.1:8082
max_pattern=200
search_seconds=3
```

By hand, and as a service, like the others (Listing 50); the unit and the rc script are made with the others and installed by `make install`, not enabled. The unit also shuts the program in (no new privilege, no home directory, a private `/tmp`, `/usr` and `/etc` read-only). It was not run under systemd here, nor the rc script under FreeBSD: they were written from the others and checked for syntax.

Listing 50: Trying sppconsult, and running it as a service

```
$ sppconsult --debug --config ./sapupra.ini
$ curl -s http://127.0.0.1:8082/catalogue | head
$ sudo systemctl enable --now sppconsult       # Debian
# sysrc sppconsult_enable=YES; service sppconsult start     # FreeBSD
```

The account. The service runs as `sapupra`, like the others. It needs no more than the right to read the database, which every account has since 0.35.0 (the directory 0755, the files 0644, The database and the configuration): a program open to strangers does better under an account that can do nothing else, so that a fault in it gives no write on the catalogue and no access to the secrets. On FreeBSD, `sysrc sppconsult_account=nobody`; on Debian, `sudo systemctl edit sppconsult` and the two lines of Listing 51. Tried here, as `nobody`, with a writer holding the database open and its log full: the program read what the log held.

Listing 51: Debian: sppconsult under the account nobody

```
[Service]
User=nobody
Group=nogroup
```

3.6.1.2 A proxy: Caddy Caddy asks for a name and a certificate by itself, renews it, sends the visitors of the port 80 to the port 443, and checks a password. It is the whole job of the proxy, because `sppconsult` refuses by itself everything but the consultation. What it needs:

- a name of domain that leads to your router (a dynamic DNS service if the address changes), and the ports 80 and 443 of the router forwarded to the machine (the port

80 is for the proof, that Let's Encrypt asks, that the name is yours);

- `sppconsult` running (The program sppconsult);
- Caddy: the package `caddy` (Debian, FreeBSD), which brings its service; the file of its configuration is the `Caddyfile` that the package names (`/etc/caddy/Caddyfile` on Debian, and `/usr/local/etc/caddy/Caddyfile` on FreeBSD, by the package).

The hash of the password is made by `caddy hash-password`, which asks for the password twice and prints a bcrypt hash; the hash goes in the file, the password nowhere. The whole file is Listing 52.

Listing 52: Caddyfile: the catalogue behind a password, in HTTPS

```
library.example.org {
    basicauth {
        peter $2a$14$THE-HASH-MADE-BY-caddy-hash-password
    }
    reverse_proxy 127.0.0.1:8082
}
```

Then `caddy validate --config /etc/caddy/Caddyfile` and the restart of the service of Caddy. To give the whole catalogue as a CSV file only to oneself, and not to whoever has the password, add the two lines of Listing 53 before `reverse_proxy`. More than one person: one line `name hash` for each.

Listing 53: Caddyfile: without the CSV file

```
@csv path /catalogue.csv
respond @csv 404
```

What was tried (Caddy 2.6.2, the package of Ubuntu 24.04, on a name of this machine with the certificate of Caddy's own authority, so that no certificate of Let's Encrypt was asked for): without a password, or with a wrong one, 401; with the right one, the catalogue, 200, with the headers of `sppconsult` unchanged; `/todo` and the other pages 404; a POST 405; the CSV file 404 with the two lines above. Not tried: the certificate of Let's Encrypt (it needs a name that is yours), the package of FreeBSD, and its path of the `Caddyfile`. The word `basicauth` is the one of Caddy 2.6; Caddy 2.8 renamed it `basic_auth`: if `caddy validate` does not accept the one, use the other. Caddy has no limit of the rate of requests of its own (a plugin has one): the password is what stops a stranger, who never reaches `sppconsult` without it, and a visitor who has it costs at most the 3 seconds of a search. A request with the method HEAD (`curl -I`, the health check of a proxy) is answered by `sppconsult` as the GET is, without the body (until 0.37.0 it was answered 501, which a health check takes for a program that is down).

3.6.1.3 A proxy: nginx For whoever knows nginx better than Caddy, or runs it already. Listing 54 is the same job: the certificate (made by `certbot`, whose package gives

the paths below), the password (a file made by `htpasswd -c` of the package `apache2-utils`, or `openssl passwd -apr1`), the limit of the rate, and the methods; the pages that write are refused by `sppconsult`, as behind Caddy. `limit_except` goes in a `location`, not in the `server` (the version of this example in 0.35.0 had it in the `server`, and nginx refuses to load it).

Listing 54: nginx: the catalogue behind a password, in HTTPS

```
limit_req_zone $binary_remote_addr zone=sapupra:1m rate=2r/s;
limit_req_status 429;
server {
    listen 443 ssl;
    server_name library.example.org;
    ssl_certificate     /etc/letsencrypt/live/library.example.org/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/library.example.org/privkey.pem;
    ssl_protocols TLSv1.2 TLSv1.3;
    server_tokens off;
    auth_basic "library";
    auth_basic_user_file /etc/nginx/sapupra.htpasswd;
    limit_req zone=sapupra burst=10 nodelay;
    location / {
        limit_except GET { deny all; }
        proxy_pass http://127.0.0.1:8082;
    }
}
```

Tried (nginx 1.24, Ubuntu 24.04, with a certificate of its own made for the test): `nginx -t` accepts it; without a password or with a wrong one, 401; with the right one, 200; `/todo` 404; a POST 403; thirty requests in a row, 11 answered and 19 refused with 429 (the burst of 10, then 2 a second). Not tried: the certificate of `certbot`.

3.6.1.4 Whatever the proxy

- The password is the whole access control: one for each person, long, and changed when someone should not come back.
- Nothing but the proxy is opened on the router: the ports 80 and 443. Not 8082, which listens on this machine alone, and not 1883, 8080 or 8081.
- A visitor who has the password sees the whole catalogue, with the place of every copy: if that is more than you want to show, use the VPN.
- The proxy keeps the log of who came and when, `sppconsult` only at the level *debug*.
- The pages of `sppconsult` give no version; `sppconsult --version` does, to whoever can run it.

- The headers: `sppconsult` sends a Content-Security-Policy that allows nothing but its own style, so that no script of the pages, if one were put in them, would run.

3.6.2 Datasette, if you want it

A free program (Apache 2.0) written in Python, Datasette serves a SQLite database as a web site, with a search, facets and an interface to write `select` queries. It is not part of `sapupra`: the build does not need it, nothing checks that it works, and nothing in `sapupra` depends on it. It is kept here for the person who likes it better than the pages above. It was tried (0.65.5, Python 3.13, a virtual environment, Ubuntu 24.04; the packages of Debian and FreeBSD were not tried) and `datasette.yml.sample`, installed beside `sapupra.ini.sample`, is its metadata.

- FreeBSD: the package `py312-datasette` (`pkg search datasette` gives the name that matches the Python of the machine).
- Debian: the package `datasette`; it was marked on 2026-10-27 for removal from testing, because of broken dependencies.
- Anywhere: a virtual environment, Listing 55.

Listing 55: Installing Datasette in a virtual environment

```
$ sudo python3 -m venv /opt/datasette
$ sudo /opt/datasette/bin/pip install datasette
$ /opt/datasette/bin/datasette --version
```

Copy the metadata to `datasette.yml` beside `sapupra.ini` and start it under the account `sapupra` (Listing 56): SQLite reads a database in WAL mode through the `-shm` file that the worker made (The database and the configuration), and the simplest way to be allowed to is to be the same account. Its pages have no password and its `/catalogue` page lets anyone run a `select`; use another port than 8081, the local network only.

Listing 56: Starting Datasette by hand

```
$ sudo -u sapupra /opt/datasette/bin/datasette serve \
    /var/lib/sapupra/catalogue.db \
    --metadata /usr/local/etc/datasette.yml -h 192.168.1.10 -p 8001
```

There is no service file for Datasette.

4 Developer/Prototyping

4.1 Building

Building the project, its documents included, is described in the README. The programs are C++17 with Qt 6 (the modules Core, Network and Sql), libmosquitto and,

optionally, YAZ. Table 4 names what each needs, as a package of Debian and of FreeBSD; Listing 57 installs them all. `configure` checks them and names the missing ones.

Table 4: What the programs need to be built, and the packages

What	For	Debian	FreeBSD
A C++17 compiler	the programs	g++	(the base system)
pkg-config	finding the libraries	pkg-config	pkgconf
autoconf, automake	./bootstrap	autoconf automake	autoconf automake gmake
Qt 6.4: Core, Network, Test	all the programs	qt6-base-dev	qt6-base
Qt's SQLite driver	the worker, the entry module	libqt6sql6-sqlite	qt6-base_sqldriver-sqlite
libmosquitto 2.0	MQTT	libmosquitto-dev	mosquitto
The broker, mosquitto	the tests, the service	mosquitto	mosquitto
YAZ 5 (optional)	the Z39.50 sources oszk and sbn	libyaz-dev	yaz
yaz-client (optional)	trying the Z39.50 by hand	yaz	yaz
libzint (optional)	spplabel, the printer of labels	libzint-dev	zint
gettext tools (optional)	the catalogues of the translation	gettext	gettext

Listing 57: Packages for building the programs, Debian then FreeBSD

```
$ sudo apt install g++ pkg-config autoconf automake qt6-base-dev libqt6sql6-sqlite
    libmosquitto-dev mosquitto libyaz-dev libzint-dev gettext
$ sudo pkg install pkgconf autoconf automake gmake qt6-base qt6-base_sqldriver-sqlite
    mosquitto yaz zint gettext
```

YAZ is split on Debian. What `configure` looks for is the development package, `libyaz-dev`: the headers and the pkg-config files `yaz.pc` and `yaz-server.pc` (the second is for the Z39.50 server of the tests). It depends on the library, `libyaz5`, and on the development packages of what YAZ is linked with (`libxslt1-dev`, `libgnutls28-dev`, `libicu-dev`, `libwrap0-dev`), so that one `apt install libyaz-dev` is enough. The package `yaz` holds only the command-line programs (`yaz-client`) and does not help to build. On a machine that only runs programs built elsewhere, `libyaz5` is the only YAZ package needed. `pkg-config --modversion yaz` prints the version (5.x) when `configure` will find it. Without YAZ the programs are built all the same, without the two Z39.50 sources: `configure` says so at its

end.

Zint, which draws the codes of the labels, has no pkg-config file on Debian: `configure` looks for the header `zint.h` and for the library itself, and builds `spplabel` when it finds both (`--with-zint=yes` makes them a prerequisite, `--without-zint` leaves the program out). The development package is `libzint-dev` (the library of 2.13 is used; the port of FreeBSD is `graphics/zint`, version 2.16 works). On FreeBSD the package puts `zint.h` and `libzint.so` in `/usr/local`, which the compiler does not search, and ships no pkg-config file: `configure` looks in `/usr/local` and `/opt/local` by itself, and `--with-zint=DIR` names another prefix (the directory that holds `include/zint.h` and `lib/libzint.so`); with a DIR only that one is tried, and a wrong one is an error. Without it all the rest is built, and `configure` says so at its end.

4.1.1 Building the programs

Written so far: the common library, `libsppcommon.a`, the gateway, `sppgate`, the worker, `sppwork` (with its own library, `libsppwork.a`), the entry module, `sppentry`, the printer of labels, `spplabel` and the lister, `spplist` (both installed in `/usr/local/bin`), the generator of a catalogue to try, `sppgen` (not installed), and their unit tests (Design.pdf). Listing 58 builds them, runs the tests with the documents' own check, builds the documents and installs the gateway and the worker, their manual pages, the example of the configuration and the reader documents under `/usr/local`; on FreeBSD, `make` is `gmake`. It finds `moc` in Qt's own directory, which pkg-config names; `./configure MOC=PATH` gives another one.

`configure` checks every prerequisite of what is built, the documents' included: Emacs and Org, Graphviz, groff (and its HTML device: on Debian the package `groff`, the package `groff-base` lacks it), LuaLaTeX with each LaTeX package and font of the documents (it compiles their preamble to be sure), the compiler, Qt, libmosquitto and `mosquitto`. It lists all that is missing at once, each with what to install on Debian and on FreeBSD, and stops; nothing is discovered later by `make`. What is not wanted is left out: `--disable-pdf`, `--disable-html`, `--disable-programs`.

A warning of the compiler is an error (`-Werror`). The headers of Qt and libmosquitto are read as system headers, so that a warning that a newer compiler finds in them does not stop the build; a warning in the code of `sapupra` does. Should a compiler newer than the code find one, `./configure --disable-werror` builds anyway, and the warning is to be reported.

Listing 58: Building the library and running its tests

```
$ ./bootstrap
$ ./configure
$ make
$ make check
$ make pdf
$ sudo make install
```

The reader documents are installed in their PDF version (under `share/doc/sapupra`), not as Org-mode sources, which the archive keeps. `make install` does not build them: it is run by another account than the build, where Emacs and LaTeX have no place. It stops, before it installs anything, if a PDF is missing, and says to run `make pdf`. `./configure --disable-pdf` installs no document but the plain files (the ChangeLog, the README).

A plain `make`, `make check` and `make install` also say if a file that an older release left is still in the directory (`README`, `stale-files --help`), as a warning that does not stop them.

The lines of the output of `make check` that matter are Listing 59 (Ubuntu 24.04), between which automake prints a few lines of its own. The tests of the MQTT client, of the gateway and of the worker (the last also plays a fake Open Library, a fake BnF and a fake MusicBrainz on the loopback interface, which answer with what the real ones answered, and makes its database in a temporary directory) start a Mosquitto of their own on a free port of the loopback interface; they do not use the machine's broker, and are reported `SKIP` where there is no `mosquitto` program. A failed test leaves its output in `NAME.log`, as `MqttClientTest.log`.

Listing 59: The results of the unit tests, from the output of make check

```
PASS: MessageTest
PASS: LogTest
PASS: CodeTest
PASS: ConfigTest
PASS: SignalWatcherTest
PASS: RecordTest
PASS: LabelsTest
PASS: MusicBrainzResolverTest
PASS: MarcTest
PASS: HttpServerTest
PASS: HttpFormTest
PASS: HtmlTest
PASS: HistoryImportTest
PASS: TranslationTest
PASS: VocabularyTest
PASS: BnfResolverTest
PASS: BarcodeTest
PASS: Z3950ResolverTest
PASS: MqttClientTest
PASS: OpenLibraryResolverTest
PASS: GatewayTest
PASS: EntryTest
PASS: WorkerTest
PASS: CatalogueTest
# TOTAL: 24
# PASS:  24
# SKIP:  0
# XFAIL: 0
```

```
# FAIL:  0
# XPASS: 0
# ERROR: 0
```

4.1.2 After a reboot

A service that comes back by itself after the machine is restarted is *enabled*. `systemctl start` runs a unit now and not at the next boot; `systemctl enable` makes the next boot start it, and `enable --now` does both. The units of `sapupra` are installed and not enabled (Service files). A service can fail to come back for three reasons (the second and the third were met on a Debian machine on 2026-11-03), in the order to look at them. The commands are in Listing 60, run soon after a boot (give it a few minutes, see the second reason).

Listing 60: Why a service is not running after the boot

```
$ systemctl is-enabled mosquitto sppgate sppwork sppentry
$ systemctl is-active mosquitto sppgate sppwork sppentry
$ systemd-analyze blame | head -5
$ findmnt /usr/local
$ journalctl -b -u sppwork
```

1. It is not enabled. The first command says *disabled*; `sudo systemctl enable sppgate sppwork sppentry` cures it.

2. The network took two minutes. `mosquitto` waits for `network-online.target`, and the units of `sapupra` for `mosquitto`, so the three come up when the network is declared up. On a machine with NetworkManager, the service `systemd-networkd-wait-online` may be enabled too: it waits for links that `systemd-networkd` manages, finds none, and gives up after 2 minutes (the second command shows the services *inactive* until then, the third shows `systemd-networkd-wait-online.service` at 2 min). On such a machine `sudo systemctl disable systemd-networkd-wait-online` removes the wait; NetworkManager's own wait service stays, and a link that `systemd-networkd` would manage is the only thing to think of if one is added later.

3. The unit is on a filesystem that is not mounted yet. systemd reads its units at the start of the boot; a unit in `/usr/local/lib/systemd/system` is not found when `/usr/local` is a filesystem of its own (a partition, a ZFS dataset: the fourth command shows a line for it). The unit is then silently left out: the service is *enabled*, *inactive*, and has no entry in the journal of the boot (the fifth command). Started by hand later it works, which hides the fault. Since 0.42.0 the units go to `/etc/systemd/system`, on the root filesystem; with an earlier version, copy them there and `sudo systemctl daemon-reload`.

A program that fails to start is started again after 5 seconds, so a late address or broker does not stop it for good. On FreeBSD the check of the first reason is `sysrc -a | grep _enable` for `mosquitto`, `sppgate`, `sppwork` and `sppentry`; the rc scripts are under `/usr/local/etc/rc.d`, which the system reads when it runs them, after the mounts.

4.1.3 Service files

`make install` installs the files that start the four programs at the boot, those of the system it is built on: the rc scripts on FreeBSD (`sppgate`, `sppwork`, `sppentry` and `sppconsult`, mode 555, in `/usr/local/etc/rc.d` with the default prefix) or the systemd units on GNU/Linux (`sppgate.service` and the others, in `/etc/systemd/system`: on the root filesystem, which systemd reads at the start of the boot, After a reboot). They are installed, not enabled: enabling is the admin's act (`sysrc` for the rc scripts, `systemctl enable` for the units), once the configuration is made. `./configure` takes the system from `uname`; `--with-service-files=rc`, `systemd` or `none` says otherwise, and `--with-rcdir=DIR` and `--with-systemdunitdir=DIR` name the directories (a Debian package wants `/lib/systemd/system`). `make uninstall` removes them. Before 0.42.0 the units went to `/usr/local/lib/systemd/system`; a machine that has them there removes them after installing the new ones, as the old ones would shadow nothing but would stay behind.

4.2 Trying the phone without the gateway

Before the gateway exists, a listener shows what a phone application sends and answers it, so that the phone's side can be set up and checked. Listing 61 answers each request with the text "received", which Binary Eye shows, and prints the request; here as it printed a request made by `curl` in the place of a phone. `nc` answers one connection and stops: the loop starts it again for the next scan (Binary Eye's *Test URL* is a connection too). `-4` makes it listen on IPv4, which the phones use on a local network. Control-C stops it.

Listing 61: A test listener on port 8080, and a request sent to it

```
$ while true; do printf 'HTTP/1.0 200 OK\r\nContent-Type: text/plain\r\nContent-Length:
    9\r\n\r\nreceived\n' | nc -4 -l 8080; echo; done
POST /scan HTTP/1.1
Host: 127.0.0.1:8080
User-Agent: curl/8.5.0
Accept: */*
Content-Type: application/json
Content-Length: 65

{"content":"9789631423334","format":"EAN_13","deviceId":"phone1"}
```

Listing 62 is what the listener printed for the *Test URL* of Binary Eye 1.75.5, on an Android 12 phone whose device identifier is `pixel3a` (the listener without the loop): the JSON object that the gateway will receive for each scan.

Listing 62: What Binary Eye sends, as the listener printed it

```
$ printf 'HTTP/1.0 200 OK\r\nContent-Type: text/plain\r\nContent-Length: 9\r\n\r\
    nreceived\n' | nc -l 8080
POST / HTTP/1.1
Content-Type: application/json
User-Agent: Dalvik/2.1.0 (Linux; U; Android 12; Pixel 3a Build/SP2A.220505.008)
Host: 192.168.111.10:8080
Connection: Keep-Alive
Accept-Encoding: gzip
Content-Length: 150

{"content":"test","format":"None","sequenceSize":"-1","sequenceIndex":"-1","sequenceId
    ":"","timestamp":"2026-10-23 14:59:12:237","deviceId":"pixel3a"}
```

4.3 When the phone cannot connect

In this order:

1. Read the whole message. A scan's message is cut short on the screen; the *Test URL* button of Binary Eye's forwarding settings shows it whole, in place of its label. A message ending in "after 5000ms" means that nothing answered within five seconds; "Connection refused" means that the machine answered, but that nothing listens on that port.

2. Watch the machine. Listing 63 prints every packet exchanged with the phone on port 8080, while the phone tries (`IFACE` is the machine's interface on the local network, as `ifconfig` or `ip address` name it; `PHONE` the phone's address, in its Wi-Fi details):

   no packet at all the phone does not send, or the network does not carry: see 3;

   packets marked `[S]`, unanswered a firewall of the machine drops them: the port must be open to the local network (and, later, the ports of the gateway and of the broker);

   packets marked `[S]` answered by `[R]` nothing listens on the port: Listing 64 shows what listens (an IPv4 line must be there); the listener of Listing 61 stops after each connection unless looped; with the traditional netcat of some GNU/Linux systems, the port is given by `-p`: `nc -l -p 8080`.

3. Open `http://HOST:8080/` in the phone's browser, the listener running. If the browser gets "received" and the scanning application cannot connect, the phone forbids the application the local network: on Android 17 with a Binary Eye older than 1.75.6, see Scanning with an Android phone: Binary Eye; on iOS, the application must be allowed the local network. If the browser cannot connect either, the Wi-Fi separates the phone from the machine: a guest network, or the client isolation of the access point.

Listing 63: The packets exchanged with the phone on port 8080

```
$ sudo tcpdump -n -i IFACE host PHONE and tcp port 8080
```

Listing 64: What listens on port 8080, on FreeBSD and on GNU/Linux

```
$ sockstat -4 -6 -l | grep 8080
$ ss -ltn | grep 8080
```

4.4 Trying the broker by hand

The commands of Listing 8 are also the way to watch what the programs publish: `mosquitto_sub` with the topic `sapupra/#` and `-v` prints every message with its topic.

4.5 Trying the bibliographic sources by hand

The requests that the worker makes can be tried one by one with `curl` (Listing 65) and, for the catalogues served by Z39.50, with `yaz-client`, which has to be installed first: it is in the package `yaz` (Debian: `apt install yaz`; FreeBSD: `pkg install yaz`, as root); `yaz-client -V` tells that it is there. Say who you are in the `User-Agent` (`CONTACT`: your address), as the worker does. The answers of the sources as they came, on a sample of the collection, are in ResearchJournal.pdf; the requests still to make are in FutureResearch.pdf.

Listing 65: The requests of the worker, made by hand

```
$ UA='sapupra-probe/0.14 (CONTACT)'
$ curl -L -A "$UA" 'https://openlibrary.org/isbn/9789631423334.json'
$ curl -A "$UA" 'https://catalogue.bnf.fr/api/SRU?version=1.2&operation=searchRetrieve&
    query=bib.isbn%20adj%20%229782081231412%22&recordSchema=dublincore'
$ curl -A "$UA" 'https://musicbrainz.org/ws/2/release/?query=barcode:3336550050675&fmt=
    json'
$ curl -A "$UA" 'https://services.dnb.de/sru/dnb?version=1.1&operation=searchRetrieve&
    query=isbn%3D9783328115373&recordSchema=MARC21-xml&maximumRecords=3'
$ yaz-client -V
$ printf 'find @attr 1=7 9631423336\nshow 1\nquit\n' | yaz-client tagetes2.oszk.hu
    :1616/B1
$ printf 'format usmarc\nfind @attr 1=7 9780805053081\nshow 1\nquit\n' | yaz-client lx2
    .loc.gov:210/LCDB
```

When the worker's log says that a source does not answer ("Open Library does not answer: Operation timed out (after 14455 ms)"), the same request made by hand tells whether the source is slow or the worker's way of asking is at fault. Listing 66 times it three ways: as `curl` asks by default (HTTP/2 when the server offers it), in HTTP/1.1 only, and in HTTP/1.1 with `Connection: close`, which is how the worker asks by default; and the BnF, as a control. It prints the protocol, the status, and the seconds at which the

connection was made, the TLS handshake was done, the first byte came, and the answer ended.

Listing 66: Timing a request three ways, and a control

```
$ U='https://openlibrary.org/isbn/9780141988313.json'
$ W='%{http_version} %{response_code} connect %{time_connect}s tls %{time_appconnect}s
    first byte %{time_starttransfer}s total %{time_total}s\n'
$ curl -sS -L -A "$UA" -o /dev/null -w "$W" "$U"
$ curl -sS -L -A "$UA" --http1.1 -o /dev/null -w "$W" "$U"
$ curl -sS -L -A "$UA" --http1.1 -H 'Connection: close' -o /dev/null -w "$W" "$U"
```

The author's run of 2026-10-27, from FreeBSD (the times are cumulative, in seconds):

Table 5: Open Library, timed by curl, and the BnF as a control

Request	HTTP	Connected	TLS done	First byte	Total
default	2	3.62	6.04	6.63	6.63
`--http1.1`	1.1	1.15	3.47	4.11	4.11
`--http1.1`, close	1.1	2.31	16.27	16.76	16.76
the BnF (control)	1.1	0.02	0.04	0.13	0.13

What it says: the time goes into setting up the connection (a second or more to be connected, then 2 to 14 seconds for TLS), before the request has even been sent, so neither HTTP/2 nor `Connection: close` can explain the wait (the header is sent after); the BnF, from the same machine, answers in a tenth of a second, so it is not the machine's network; and the three runs differ among themselves by more than the one difference that the options make. Open Library is slow to accept a connection, in a way that varies. The worker's default timeout was 15 seconds, which this exceeds; it is now 60 (`timeout=60`). The worker with `--debug` logs each request with its status, its duration and the protocol it used, to compare with.

4.6 A catalogue to try the programs on

To try the pages and the lists on a collection of thousands of books before having scanned them, the developer makes an invented one with `sppgen`, which is built but not installed (it is a tool of the build tree). It makes a new database file, never touches an existing one, and fills it as the worker would: the copies are entered one by one, with an author or several, a publisher, a year, a language, subjects, an ISBN-13 with a good check digit (in a range that no publisher uses), a medium and a place on a shelf, a few copies having no place. The names and the titles are made up from syllables, in French, English and Hungarian, accents included.

Listing 67: Making a catalogue of invented books and looking at it

```
$ ./sppgen -o /tmp/try.db -n 5000
$ ./sppgen -o /tmp/big.db -n 1000000 -r 8 -p 5000
$ spplist -D /tmp/try.db -t location --no-date -o try.org
$ printf '[database]\nfile = /tmp/try.db\n[entry]\nport = 8089\n' > try.ini
$ ./sppentry -c try.ini
```

`-n` is the number of copies (1 to 1000000, 1000 by default), `-s` the seed (the same seed makes the same catalogue, so that two measures can be compared), `-r` the rooms of the house (1 to 8, 6 by default), each with furniture, rows and shelves, twelve to eighty shelves for a room. On the author's machine about 550 copies are made in a second, a little less as the file grows (40000 copies in 75 seconds): the file of 5000 copies is ready in ten seconds, that of 20000 copies in forty. A million copies took much longer on another machine: the journal has the figures.

`-p N` (`--progress`) sets how often sppgen says where it is: a line every N copies, with the share done, the copies per second and the time left (by default every hundredth of the copies, but not more often than every thousand; `-p 0` says nothing until the end). The rate shows whether the making slows as the file grows.

What it showed (a laptop, 20000 copies, the pages of the entry module; the figures are the time of the database alone):

- all the copies in a given order, 110 to 240 ms;
- a regular expression on the author, 400 ms; on the title, 470 ms (the pattern is tried on every row; the full text index does not serve a regular expression);
- the copies of one place, 30 ms; the ten last entered, under 1 ms; the counts by place, 13 ms; the three tables of `spplist`, 70 ms.

The pages stay quick enough to be used without a break at that size, and nothing in them depends on the number of copies but the sorting of the answer, which is done on the whole answer once, then cut into pages.