Subscription

Administration and how-it-works manual

MLAdvisory © 2026

Contents

Introduction 11
How this manual is organised 11
The console's menu 11

What's new 13
November 2026 13

Getting started with the console 15
Who can sign in 15
The screen layout 15
The top bar 15
The left menu 16
The home page 16
Lists and forms 16
The profile 17
The settings 17
The Services tab 17
The Master key tab 17
Live updates 17
Language 18
The built-in help 18

Key concepts 19
Overview 19
Client 19
Realm 19
Tenant 19
User 20
Access (a user in a tenant) 20
Role 20
Service 20
Feature 21
Plan 21
Contract 21
The relations in tables 21

I **Tenant management**

Clients and tenants . . . 25

The clients and tenants tree . . . 25
Search and filters . . . 25
The ··· menus . . . 25
Contracts ending soon . . . 26
Create or edit a client . . . 26
A client's page . . . 27
A client's logo . . . 27
Delete a client . . . 27
Create or edit a tenant . . . 27
Login credentials . . . 28
A tenant's page . . . 28
A tenant's status . . . 28
A tenant's validity . . . 29
Suspend, delete, restore . . . 29
A tenant's members . . . 30
ALM storage . . . 30
Who may do what . . . 30

Users . . . 33

The user list . . . 33
Filters . . . 33
Actions . . . 33
Create or edit a user . . . 33
A user's page . . . 34
Guest clients . . . 35
Disable and enable . . . 35
Keycloak account changes . . . 35
Delete a user . . . 35
Import users . . . 36
Access to tenants . . . 36
Role and plans . . . 36
Give or change an access . . . 36
An access page . . . 37
Suspend access to a tenant . . . 37
Who may do what . . . 37

Access history . . . 39

The timeline . . . 39
Filters . . . 39
Event types . . . 40
Who made the change . . . 40
The access review . . . 40

Retention . . . 41
Who may see it . . . 41

Purge . . . 43
Deleted tenants . . . 43
Check a tenant . . . 43
Orphan data . . . 44
Purge orphan data . . . 44
Default tenants of services . . . 45
Who may do it . . . 45

Support console . . . 47
Who can use the console . . . 47
Open the console . . . 47
Choose a tenant . . . 48
The tenant's members . . . 48
Request access . . . 49
View as user . . . 49
Policy and active authorizations . . . 49
The support journal . . . 50
End a running consultation . . . 50
Live updates . . . 51

II Reglab Platform

Services . . . 55
The list of services . . . 55
Creating a service . . . 55
Editing a service . . . 55
A service's page . . . 55
Deleting a service . . . 56

Plans . . . 57
The list of plans . . . 57
Creating a plan . . . 57
A plan's page . . . 57
Editing a plan . . . 58
Deleting a plan . . . 58

Features . . . 59
The list of features . . . 59
Creating a feature . . . 59
The base feature . . . 59

A feature's page . . . 60
Editing a feature . . . 60
Deleting a feature . . . 60

III **Services Configuration**

Crawler . . . 63
Header and last-run strip . . . 63
Publications . . . 63
Send to the library again . . . 64
Summaries for the daily mail . . . 64
Sources . . . 64
Runs & schedule . . . 64
Schedule . . . 64
How the crawler crawls . . . 65
Last runs . . . 65
Feeds . . . 65
Add or edit a feed . . . 65
Delete a feed . . . 66
Recipients . . . 66
Sent digests . . . 66
Docling . . . 67

Notifications . . . 69
The list of mails . . . 69
The detail of a mail . . . 69
Delivery . . . 70
Resend a mail . . . 70
The mails subscription sends . . . 70
Contract-end reminders . . . 71

Library . . . 73
A document's path . . . 73
Documents . . . 73
Embeddings for the chat . . . 73
The list of documents . . . 74
Actions on a document . . . 74
Upload . . . 74
Upload files . . . 74
Index from the web . . . 75
Send . . . 75
Search index . . . 75

Reference data . . . 77
Cubes . . . 77
Import a cube . . . 77
The list of cubes . . . 77

Sources . . . 77
The list of sources . . . 77
Follow or change a cube . . . 78

IV **Supervision**

DPM Kafka work . . . 81
Live reading . . . 81
Totals . . . 81
Listeners . . . 81
Partitions . . . 81

V **How it works**

Identities and realms . . . 85
Realms . . . 85
Who opens the console . . . 85
What subscription holds, what Keycloak holds . . . 85
A user: a realm and a username . . . 86
A client's realm . . . 86
Creation and import . . . 86
Deactivation . . . 86
Deletion . . . 87
Last login . . . 87
The services' membership lookups . . . 87
Settings . . . 87

From plans to rights . . . 89
The principle . . . 89
What the lookup answers . . . 89
Who may ask subscription . . . 90
The X-TENANT-ID header and tenant-web-library . . . 90
The membership cache . . . 90
CIRCÉ (irrc) . . . 91
The assistant . . . 91
The UPDATE_PLAN event . . . 91
When subscription sends UPDATE_PLAN . . . 92
What subscription does not announce . . . 92
The delays in short . . . 93
The feature names that are read . . . 93

A tenant's lifecycle . . . 95
Statuses . . . 95
Validity . . . 96

What gives access to a tenant . . . 96
Creation . . . 96
Suspension and reactivation . . . 96
Deletion and waiting period . . . 96
The purge . . . 97
The TENANT_DELETED event . . . 97
What each service purges . . . 97
UPDATE_PLAN messages . . . 98
Mails to members . . . 98
End-of-contract reminders . . . 98
Test tenants . . . 99
Settings . . . 99

Support access . . . 101
Overview . . . 101
Who may view whom . . . 101
The tenant's policy . . . 102
Access requests . . . 102
Authorizations . . . 103
The access decision . . . 103
The consultation session . . . 104
How the services recognise a consultation . . . 105
End reasons . . . 105
Ending a session . . . 106
Events, journal and audit . . . 106
Live updates and e-mails . . . 107
API endpoints . . . 107
subscription settings . . . 108

Security . . . 109
Tokens accepted . . . 109
Who is who . . . 109
What the API requires . . . 110
Calls from the other services . . . 110
Tenant credentials . . . 111
Master keys . . . 112
Rotating the master key . . . 112
Public endpoints and logos . . . 113
Vault and Consul . . . 113

Events and calls . . . 115
Overview . . . 115
Kafka topics . . . 115

Messages **116**
`UPDATE_PLAN` 116
`TENANT_DELETED` 116
`SEND_MAIL` 116
`NEW_PUBLICATION` and `NEW_COMMON_FILE` 116
`SUPPORT_REQUEST_CHANGED` 116
Envelopes **116**
Delivery guarantees **117**
The console's live updates **118**
Where the events come from 118
The stream 118
On the console's side 118
Synchronous calls **119**

Glossary **121**
Terms **121**

Introduction

Subscription is the Reglab platform's administration console. It is where clients and their tenants, users and their access, the platform's services, their features and the plans that open them are declared. The service of the same name answers the other services (CIRCÉ, HERMES, the library, the assistant) when they check that a user may work in a tenant.

This manual describes Subscription as of 24 November 2026. Recent changes are listed in What's new.

The console lets you:

- create clients and their tenants, follow their contract and status, up to their deletion and purge;
- manage users, their role and plans in each tenant, import them and deactivate them;
- declare the platform's services, their features and the plans that group them;
- follow the access history and open, once authorised, a support consultation in a tenant;
- configure the crawler, the notifications, the library's common documents and the reference data;
- supervise the Kafka activity of the DPM jobs.

> **Note**
>
> The console is for platform administrators: an account holding the `ROLE_ADMIN` role in a platform realm (`mla` by default). The **Support console** is also open to MLA support members and to a tenant's administrators. See Who can sign in.

How this manual is organised

This manual has two audiences:

- Platform administrators, who use the console. Getting started and the key concepts come first, then one page per entry of the left menu.
- Developers and operators, who need to know what happens behind the screens. The **How it works** part describes identities and realms, how a plan becomes a right in the other services, a tenant's lifecycle, support access, security and the events exchanged.

The glossary lists the terms of the console and of this manual.

The console's menu

Group	Entries
Tenant management	Client and Tenants, Users, Access history, Purge, Support console
Reglab Platform	Services, Plans, Features
Services Configuration	Crawler, Notifications, Library, Reference Data
Supervision	DPM Kafka work

A typical order of work to open the platform to a new client:

1. Check that the service and the plan being sold exist, with the right features.
2. Create the client and its tenant, with its contract dates.
3. Create or import its users, then give each one a role and one plan per service in the tenant.
4. The services read these rights at the next sign-in: From plans to rights explains how, and how soon.

What's new

This page lists, month by month and newest first, what changed in the Subscription console and in this manual. Each item links the section that describes it.

November 2026

- **First edition of the manual.** The console's screens, one page per entry of the left menu, and how they work: identities and realms, the rights plans open in the other services, a tenant's lifecycle, support access, security and events. See Getting started and the key concepts.

Getting started with the console

Who can sign in

The subscription console is reserved for platform administrators: accounts that hold the realm role `ROLE_ADMIN` in a platform realm (`mla` by default). Signing in goes through the gateway, which sends you to the sign-in page of the `mla` realm; Keycloak reuses the session already open in the browser when there is one.

Once you are signed in, the console checks your account:

Situation	What you see	What you can do
Your account does not hold `ROLE_ADMIN`.	The **Access restricted** page: "Subscription is reserved for MLAdvisory administrators, and your account does not have that role."	**Log out**, or contact your support team.
The sign-in was interrupted before it finished (after signing out, or by leaving the sign-in page).	The **Sign-in did not complete** page.	**Sign in again**.
The platform did not answer when your session was checked.	An error page: "The service is unavailable right now. Try again in a moment."	Reload the page a little later.

The **Access restricted** page only offers the user menu with **Sign out**.

> **Note**
>
> The **Support console** has its own access rules: users who are not platform administrators can open it. See Support console.

Every other page of the console, profile and settings included, requires the `ROLE_ADMIN` role. The subscription service applies the same rule to its own APIs: a platform administrator is a token issued by a platform realm that carries the `ROLE_ADMIN` role, spelt exactly so. A `ROLE_ADMIN` role given in a client's realm is not enough.

The screen layout

The screen is made of:

- on the left, the logo and the name **Subscription**, which lead back to the home page, then the left menu;
- at the top, the top bar;
- in the middle, the open page;
- at the bottom, a footer with the logo, "MLA Solutions", the copyright and the console's version.

The top bar

From left to right, the top bar holds:

- the chevron button (**Show or hide the menu**) that collapses or expands the left menu;
- the **breadcrumb**: the open page (**Home** on the home page), then the current action (**New**, **Detail**, **Edit**…). Its earlier items are links;
- the help menu (question mark), with **Online documentation** (see The built-in help);

- the language choice, which shows the code of the current language (`FR` or `EN`) (see Language);
- the live updates indicator;
- your account's photo, or an icon, and your name (your e-mail when there is no name). They open the **Profile**, **Settings** and **Sign out** menu.

The left menu

The menu has four groups:

Group	Entries
Tenant management	**Client and Tenants**, **Users**, **Access history**, **Support console**, **Purge**
Reglab Platform	**Services**, **Plans**, **Features**
Services Configuration	**Crawler**, **Notifications**, **Library**, **Reference Data**
Supervision	**DPM Kafka work**

Each entry has its own page in this manual:

- **Tenant management**: Client and Tenants, Users, Access history, Support console, Purge;
- **Reglab Platform**: Services, Plans, Features;
- **Services Configuration**: Crawler, Notifications, Library, Reference Data;
- **Supervision**: DPM Kafka work.

The home page

The home page is titled **Contract and Subscription Management System**, with the subtitle "Empowering User Roles and Permissions".

The header offers two shortcuts:

- **New user** opens the user creation form (see Users);
- **New tenant** opens the tenant creation form (see Client and Tenants).

The page then shows:

- the **Tenant management** part, with four tiles: **Tenants**, **Users**, **Services** and **Plans**. Each tile shows how many items there are and opens the matching list. The tenant count leaves out deleted tenants and temporary ones (the test tenants created with a lifetime). A tile stays without a count when its list could not be read;
- the **Crawler** part: the status of the last run, the number of **Publications**, **New at the last run**, the **Next run** (or **Off**) and the date of the last run. **Go to the Crawler** opens the Crawler page;
- the **Library** part, with the **Common documents** tile and their count (see Library);
- the **Reference data** part, with the **Cubes** tile (see Reference Data).

Lists and forms

The list pages of the console all work the same way:

- the **Global search…** box filters the list as you type;
- the column headers marked with an arrow sort the list;
- the paginator, above the list, shows "Showing *x* to *y* of *n* entries" and lets you choose the number of rows per page;
- the ⋮ menu of a row offers **View**, **Edit** and **Delete**;
- the check boxes select rows; the toolbar's **Delete** button deletes the selection, after a confirmation.

A bulk delete sends one request per item. Once every answer is in, a message says "*n* deleted, *m* refused" and names the refused items ("Refused: …"). The list is then read again.

In the forms:

- a required field left empty shows "This field is required.";
- **Save** stays greyed out until the form is valid;

- **Back** leaves the form without saving;
- after a successful save, a confirmation message shows at the bottom of the screen;
- leaving a form that holds unsaved changes opens the **Unsaved changes** dialog: “Your changes are not saved. Leave this page and lose them?”, with **Leave without saving** and **Stay on the page**. Reloading or closing the tab triggers the browser’s warning.

The **Description** fields use a rich text editor: heading, bold, italic, underline, strike, lists, link, quote, code block and **Remove formatting**.

When the server refuses an action, the page shows an error banner that gives the reason.

The profile

Profile, in the menu under your name, opens your account’s page.

- The header gives your name, your login and the note “account managed by the sign-in directory”: the account is changed in Keycloak, not in the console. The **Log out** button ends the session.
- The **Identity** card shows **First name**, **Last name**, **Mail** and **Authorities** (your account’s roles, `ROLE_ADMIN` included).
- The **Preferences** card offers the **Interface language** (Français or English): “Applied at once and kept for your next sessions.”

The settings

Settings, in the menu under your name, opens a page with three tabs: **User settings**, **Services** and **Master key**.

The Services tab

The **Services** tab shows whether the platform’s services answer, and which version runs.

- The summary says “All *n* services are up” or “*m* of *n* services unreachable”. Each card reads the service’s `/actuator/info` through the gateway. **Refresh** reads every card again; the time of the last read is shown (“Read at …”).
- The cards are grouped:
  - **Platform**: **Console** (this application), **Gateway**, **Subscription**, **Admin**;
  - **Regulatory reporting**: **IRRC**, **DPM**, **Reporting engine**;
  - **Library**: **Library**, **Crawler**;
  - **Shared services**: **Notification**, **Referential**.
- Each card shows the status **Up**, **Unreachable** or **Reading**, then the version, the build date (**Built**) and the **Commit**, with its branch and a button to copy the commit id. An **Unreachable** card explains: “No answer through the gateway: the service is down, restarting, or not deployed here.”

The Master key tab

The **Master key** tab is used during a rotation of subscription’s master key, the key that encrypts the tenants’ credentials. It is read only: the keys themselves are never shown.

- A banner sums up the situation, for example “Every tenant credential is on key …”, “Key … no longer encrypts anything” or “Key … still encrypts *n* value(s)”, with what to do.
- The table lists each **Key**, its **Status** (**Current**, **Still in use**, **Can be removed**, **Missing from Vault**, or **Pre-v2 format (key 0)**) and the number of **Values** it encrypts.
- **Refresh** reads the state again.

Rotating and removing a key are done in Vault, following the runbook `rotation-cle-maitre-subscription` in the documentation repository.

Live updates

While you are signed in, the console keeps an event stream open with the subscription service. The Crawler, Library and Support console pages update themselves when an event concerns them.

An indicator in the top bar shows the state of the stream; its tooltip spells it out:

Indicator	Tooltip	Meaning
Green	“Live updates: connected”	The stream is open.
Orange	“Live updates: reconnecting”	The stream has been down for more than 5 seconds.
Red	“Live updates interrupted — data refreshes every 30 s”	The stream has been down for more than a minute.

A shorter outage does not change the indicator. It does not show until the first connection is made.

Language

The console exists in French and English. Change the language:

- with the language choice in the top bar (**Français** or **English**, the current language is ticked);
- or with **Interface language** on the Profile page.

The change applies at once and subscription keeps it for your next sessions. With no saved choice and no language on your account, the console shows in English.

The built-in help

Online documentation, in the help menu (question mark), opens a panel on the right with a short guide to the console, in the current language (in English otherwise). When the guide has several sections, a **Contents** list leads to them. If the guide cannot be loaded, the panel says so: “The documentation could not be loaded.”

Key concepts

This page presents the notions the console handles and how they fit together. The screens that manage them each have their own page: Client and Tenants, Users, Services, Plans and Features.

Overview

In one sentence: a **user** of a **client** receives, in a **tenant** of that client, a **role** and **plans**; each plan belongs to a **service** and opens a selection of that service's **features**. The platform's services ask subscription, on each access, what the user holds in the tenant (see From plans to rights).

Two families of objects stand apart:

- the **catalogue**, shared by every client: the services, their features and the plans (**Reglab Platform** menu);
- the **subscribers**: the clients, their tenants, their users and those users' access to the tenants (**Tenant management** menu).

Client

A **client** is a customer organisation of the platform. It has a name (unique), a description and, once an administrator has set them:

- a **Subdomain**: the label on which the client's front answers;
- a **Realm**: the Keycloak realm the client's users sign in to (see Realm).

A client groups its tenants and its users.

Realm

A **realm** is a space of accounts in Keycloak. The realms the platform accepts are set by configuration; among them, the **platform realms** (`mla` and `shared` by default) are those of the platform administrators and of the service accounts.

A user name is unique only inside its realm: to subscription, a user is the pair (realm, login). The services that ask subscription therefore send the realm that issued the user's token; with no realm, subscription answers for the platform's default realm.

Tenant

A **tenant** is an isolated RegLab environment that belongs to a client. A client may have several tenants. A tenant has a name (unique), a description and a public id (`tenantId`): this is what the services receive to know which tenant a user works in.

A tenant has a **status**:

Status	Meaning
`ACTIVE` (active)	The tenant grants access, within its validity window.
`SUSPENDED` (suspended)	Suspended by a platform administrator: everything is kept, no access, until it is reactivated.
`PENDING_DELETION` (deletion scheduled)	No access; can be restored until its purge date, then purged.

Status	Meaning
`DELETED` (deleted)	Purged: the access rights are removed, the tenant's row is kept and its name stays reserved. This status is final.

Only an active tenant, within its validity window, grants access to the services. The lifecycle is described in Client and Tenants and The tenant lifecycle.

User

A **user** is an account of the platform: a login in a realm, an e-mail, a first name and a last name. It belongs to one client, and one only.

A user may join:

- the tenants of their own client;
- the tenants of **guest clients**, if an administrator has made them **Multi-client** and declared them a guest of those clients (for a consultant or an auditor, for example).

A user can be disabled: they then have access to no tenant at all. See Users.

Access (a user in a tenant)

An **access**, or membership, is a user's place in a tenant. It is made of:

- one **role** in the tenant, whatever their plans (see Role);
- a list of **plans**: at most one plan per service. Without a plan of a service, that service stays closed to the user in that tenant.

A member holds at least one plan: removing their last plan takes them out of the tenant.

An access can be **suspended**: the role and plans stay, but no service lets the user into the tenant until the access is reactivated.

Access is given from Client and Tenants or from the tenant access list; every change is recorded in the Access history. When their rights change, the user receives an e-mail, except in a temporary tenant.

Role

Two roles must not be confused:

Role	Where it is given	What it allows
Platform administrator (`ROLE_ADMIN`)	In Keycloak, on a platform realm	Using the subscription console and its APIs.
Role in a tenant: **Member** or **Admin**	In the user's access to the tenant	**Member**: "Uses the features of their plans." **Admin**: "Also manages the other users' access to the tenant."

A role in a tenant counts for that tenant only: a user can be the admin of one tenant and a member of another. The services read it in subscription's answer; CIRCÉ, for example, makes a tenant's admin the administrator of that tenant in CIRCÉ.

Service

A **service** is a RegLab product (for example `irrc`, CIRCÉ's service). It has a name (unique) and a description. Each service has its own features and plans.

The service's name is the one the applications ask subscription for: CIRCÉ asks for the plans of the `irrc` service, the assistant for those of the `irrc`, `stress` and `library` services. See Services.

Feature

A **feature** is a capability of a service, named (uniquely within its service) and described by a description. The applications open a screen or an action when the user's plan carries the matching feature: `anacredit` or `pilar3` in CIRCÉ, `assistant` for the assistant, for example.

Each service has a `base` feature, created with it: holding it makes the user a member of the service. See Features.

Plan

A **plan** is an offer on a service: it belongs to a service and opens a selection of its features. A plan always carries its service's `base` feature, and two plans of the same service cannot open exactly the same features. A plan's name is unique across the whole platform. See Plans.

Contract

A tenant's **contract** is its validity window: a start date (**Valid from**) and an end of access (**End of access (excluded)**), each optional. Outside that window, the tenant's users have no access. A tenant with no end date is open-ended: access never stops on its own.

Before a contract ends, subscription sends reminders by e-mail to MLA and to the tenant's admins. The Client and Tenants page shows the contracts ending soon and the reminders sent.

The relations in tables

Client and tenant

A client	A tenant
has zero, one or several tenants	belongs to a client

Client and user

A client	A user
has zero, one or several users	belongs to one client only (their home client)
can receive guest users from other clients	can be a guest of other clients if **Multi-client**

Client and realm

A client	A realm
has at most one realm, where all its users sign in	can serve several clients

User and tenant (the access)

A user, in a tenant	Constraint
has one role	**Member** or **Admin**
holds at least one plan	at most one plan per service
can be suspended	role and plans kept, no access

Service, plan and feature

Object	Relation	Constraint
Service ⟶ plan	a service has zero, one or several plans; a plan belongs to one service	a service that has plans cannot be deleted
Service ⟶ feature	a service has its features, `base` among them; a feature belongs to a service	a feature's name is unique within its service

Object	Relation	Constraint
Plan ↔ feature	a plan opens several features; a feature can be opened by several plans	a plan opens only features of its own service, `base` included; two plans of a service do not open the same ones
Plan ↔ access	a plan can be held by users in several tenants	a plan that is held cannot be deleted

I

Tenant management

Clients and tenants

Clients and their tenants are in the **Tenant management › Client and Tenants** menu.

A **client** is a customer organisation. A **tenant** is an isolated RegLab environment that belongs to a client; a client may have several. Users get a role and plans on a tenant: see Users. A tenant's lifecycle (statuses, validity, deletion, purge) is explained in detail in A tenant's lifecycle.

The clients and tenants tree

The page is titled **Tenants**. Its header offers **New client** and **New tenant**.

Under the header come the **Contracts ending soon** panel and the line about tenants without an end date (see Contracts ending soon), then the tree. It has three levels:

- one **client** per top-level row, always expanded, with its logo if it has one, its name (link to the client's page), how many tenants it has, and its **Subdomain** and **Realm** when set. A client without a tenant shows too. Tenants without a client are grouped under a last row, **No client**;
- under each client, its **tenants**, in alphabetical order: name (link to the tenant's page), number of members, state tags (see A tenant's status), description, **Created** and **Deleted** dates;
- under a tenant, its **members**: username (link to the user's page), a **Suspended** tag when their access to the tenant is suspended, their role (**Member** or **Admin**) and their plans (a link to each plan).

A tenant's members are read when you expand it. A tenant shows 20 of them at most; beyond that, a last row **See all N accesses** opens the access list filtered on that tenant (see Access to tenants).

The **Deleted** column gives the purge date of a deleted tenant. For a tenant whose deletion is scheduled, it shows in grey, after a clock, the date the purge is planned for (tooltip "Purge scheduled on this date, restorable until then").

Paging counts clients ("Showing 1 to 10 of N clients").

Search and filters

The bar above the tree offers:

- **Global search…**: searches the names, the descriptions and the plan names. While a search runs, every tenant is expanded and shows all its members, so that no match stays hidden;
- **Select a client**: shows only the clients picked;
- **Show deleted tenants (N)**: deleted (purged) tenants are hidden by default; ticked, the box shows them, greyed out;
- **Without end date (N)**: shows only the active tenants without a validity end date, and the clients that have such tenants. Each one then carries a **Set an end date** button, which opens the validity dialog.

The ··· menus

A **client**'s ··· menu offers **View**, **Edit** and **Delete**. **Delete** is greyed out while the client has tenants (deleted ones included) or users; its tooltip says so: "Cannot be deleted while the client has N tenant(s) and N user(s)."

A **tenant**'s ··· menu offers, depending on its status:

Entry	Offered when the tenant is
View	always
ALM storage	always
Edit	not deleted
Add user	active or suspended

Entry	Offered when the tenant is
Suspend	active
Reactivate	suspended
Validity	not deleted
Schedule deletion	active or suspended
Restore	pending deletion

Add user opens the access form with that tenant and its client already chosen (see Give or change an access).

A **member**'s ··· menu offers **View**, **Edit**, **Suspend access** (or **Reactivate access**) and **Delete**. **Delete** takes the user out of the tenant after the confirmation "Remove X from T? Their role and plans in that tenant are deleted, and X will receive an e-mail about it."

Contracts ending soon

The **Contracts ending soon** panel lists the active tenants whose access ends within 60 days, and those ended within the last 30, the soonest first. It shows only when it has at least one row.

Each row shows:

- the time left: "in 12 days", "in 1 day", "last day", then "ended today", "ended yesterday", "ended N days ago". The colour turns amber under 30 days, red under 7 days, grey once ended;
- the tenant's name (link to its page) and its client;
- the last day of access, "until DD MMM YYYY (included)". The tooltip gives the exact moment access stops ("Access stops on …, excluded");
- a gauge of what is left of the 60 days;
- the reminders already sent for that end date: **60 d**, **30 d**, **7 d**, **ended**. The tooltip gives the date sent and the number of addresses; a faded reminder, with a dashed border and a dash instead of the envelope, was skipped: a more urgent reminder went out instead. With none: "No reminder sent yet";
- the **Extend** button, which opens the tenant's validity dialog (see A tenant's validity).

Under the panel, a line counts the active tenants without an end date: "N active tenant(s) have no end date: no reminder will ever warn about them." **Show them** ticks the **Without end date** filter; **Show all tenants** unticks it.

The reminders themselves (who gets them, and when) are described in End-of-contract reminders.

Create or edit a client

New client (tree header) opens the **Create a new client** form. **Edit** (··· menu or client page) opens **Edit a client**.

Field	Required	Checks
Client name	yes	Unique: "A client with this name or subdomain already exists."
Subdomain	no	One lower-case label (letters, digits, hyphens), 63 characters at most, not starting or ending with a hyphen. What is typed is lower-cased. Unique across clients: "Another client already uses this subdomain."
Realm	no	**No realm** or one of the realms the platform accepts.
Description	no	Formatted text.

Field	Required	Checks
Logo	no	Only once the client is created (“A logo can be added once the client is created.”).

While the subdomain is valid, the form shows the **Front address:** the client’s front answers on (for instance `https://<subdomain>.irrc-int.…` on the integration environment). Otherwise it shows the hint: “Optional: one lower-case label (letters, digits, hyphens) on which the client’s front answers.”

The **Realm** is the Keycloak realm the client’s users sign in to; all of its users must already be in it. The server refuses a realm while some of the client’s users are in another one: “This realm cannot be set: some of the client’s users are in another realm. Move them first.” See A client’s realm.

Save creates the client and opens its page, or saves the change and goes back to the previous page. **Back** leaves without saving; if values were typed, the **Unsaved changes** dialog asks **Leave without saving** or **Stay on the page**.

A client’s page

A client’s page opens from its name in the tree or from **View**. Its title is the client’s name, after its logo. The buttons are **Back**, **New tenant for this client** (the tenant form opens with this client chosen) and **Edit**.

The page shows:

- **Client name** and **Description**;
- **Subdomain**: the address of the client’s front, as a link, or “No subdomain”;
- **Realm**: the realm, or “No realm”;
- **Logo**: the logo and its buttons (see below);
- **Tenants**: the name (link to the tenant) and description of each, sortable columns; or “This client has no tenant.”;
- **Users**: **Username** (link to the user), **Last name**, **First name**, **Mail**; or “This client has no user yet.”

A client’s logo

The logo is managed from the client’s page or its edit form:

1. **Choose a logo** (or **Replace** when there is one) opens the file picker: PNG, JPEG or WebP, 512 KB at most. Another type or a file too heavy is refused at once: “The logo must be a PNG, JPEG or WebP image.”, “The logo weighs N KB, more than the 512 KB allowed.”
2. A preview shows. **Upload** saves it; **Cancel** drops it.
3. **Remove** takes the logo away, after the confirmation “Remove the logo of X?”

The logo shows in the tenants tree and on the client’s page. The applications show it too: the services and a tenant’s members read the logo of the tenant’s client, and the login page of a front answering on the client’s subdomain shows it before sign-in.

Delete a client

Delete (client’s ··· menu) asks “Delete the client X?”. The server refuses while the client has tenants, deleted ones included, or users. The entry is greyed out in that case anyway.

Create or edit a tenant

New tenant (tree header) or **New tenant for this client** (a client’s page) opens the **Create a new tenant** form. **Edit** opens **Edit a tenant**.

The form has three sections.

Identity: “The name users and services know the tenant by, and the client it belongs to.”

- **Name**: required and unique, deleted tenants included: “A tenant with this name already exists.”
- **Client**: required (**Select a client**).

Login credentials, on creation only: see Login credentials.

Description: formatted text, “Shown in the tenants tree.”

Save creates the tenant and opens its page, or saves the change and goes back to the tree. **Back** goes back to the tree; unsaved values bring up the **Unsaved changes** dialog.

Rules when editing:

- A tenant with members cannot change client: the **Client** field is locked, with the hint "This tenant has N member(s): remove them before moving it to another client." The server, for its part, refuses to move a tenant to a client some of its members neither belong to nor are guests of.
- A deleted tenant can no longer be changed: the form is greyed out, with the message "This tenant is deleted: it can no longer be changed."

A new tenant is active, with no validity limit.

Login credentials

The **Login credentials** section shows on creation only: "Chosen when the tenant is created: this form does not change them afterwards."

- **Type**: **Password**, the only type offered.
- **Password**: filled with a password the server generates. **Generate** draws another one. You may also type one; the server refuses an empty or too short password.
- The hint says: "Shown only now: copy it before saving."

> ⚠ Warning
>
> This password is the tenant's encryption secret. No screen shows it again after saving, and none can change it. It is stored encrypted; only the authorised services read it (see A tenant's credentials).

A tenant's page

A tenant's page opens from its name in the tree, from **View** or from the contracts panel.

The header shows the **All tenants** link, the tenant's name, the **Client X** tag (link to the client) and the state tags. The buttons are, depending on the status: **Suspend**, **Reactivate**, **Validity**, **Schedule deletion**, **Restore** (see Suspend, delete, restore), then always **ALM storage** and **Access history** (the history filtered on this tenant, see Access history), and **Edit** while the tenant is not deleted.

The **Definition** card gives:

- **Tenant name**;
- **Reference**: the tenant's public id, the one the services know;
- **Client name** (link) and **Description**;
- **Valid from** and **End of access (excluded)**, or "No limit";
- **Purged on**, for a tenant whose deletion is scheduled;
- **Deleted on**, for a deleted tenant.

The **Contract** card gives:

- the end date, "until DD MMM YYYY (included)", and the time left; or "No end date", with the reminder "Open-ended: access never stops on its own, and no end-of-contract reminder is sent.";
- **Reminders sent**: each reminder for the current end date, with its date and number of recipients, or "skipped"; or "No reminder sent yet";
- **Earlier end dates**: the reminders sent for the end dates the tenant had before it was extended.

The **Access** card lists the tenant's members: see A tenant's members.

A tenant's status

A tenant is in one of these four statuses. Only an active tenant, inside its validity window, gives its users access.

Status	Tag	Tooltip
Active	none	

Status	Tag	Tooltip
Suspended	**Suspended** (amber)	"Suspended since …: its users have no access."
Pending deletion	**Deletion on … (N days left)**, then **Deletion on … (last day)** (red)	"No access; restorable until …, then purged."
Deleted	**Deleted** (red)	"Purged on …: read only."

An active or suspended tenant also carries, depending on its validity:

- **Expired**: "Validity ended on …: its users have no access.";
- **Not yet valid**: "Validity starts on …: no access before."

A tenant's validity

Validity (··· menu, tenant page), **Extend** (contracts panel) and **Set an end date** (**Without end date** filter) open the **Validity of X** dialog: "Its users have access only within this window. Leave a date empty for no limit."

- **Valid from**: date and time, or empty ("No limit").
- **End of access (excluded)**: date and time, or empty. Access stops at that moment; the hint names the last day of access it gives: "Access stops at this moment: the last day of access is DD/MM/YYYY, included."

If the end does not come after the start, the message "The end must come after the start." shows and **Save** stays greyed out. Once saved: "Validity of the tenant X updated". The contracts panel and the count of tenants without an end date update.

The validity can be changed in any status but deleted. A new end date starts the reminders afresh: those already sent stay under **Earlier end dates**.

Suspend, delete, restore

Each action asks for a confirmation naming the tenant, then shows a message.

Action	Confirmation	Effect
Suspend	"Suspend the tenant X? Its users lose access at once. Everything is kept until you reactivate it."	Suspended. The members get an e-mail.
Reactivate	"Reactivate the tenant X? Its users get their access back at once."	Active. The members get an e-mail, except those whose own access is suspended.
Schedule deletion	"Schedule the deletion of the tenant X? Access stops now for all its users. Its data is purged after 60 days; you can restore it until …"	Pending deletion, with the purge date. The members get an e-mail.
Restore	"Restore the tenant X? It gets back the status it had before its deletion was scheduled."	Active, or suspended if it was before. The members get an e-mail.

Deletion can be scheduled whether the tenant has members or not. During the 60 days of waiting, everything is kept and the tenant can be restored. Then the purge removes its members, marks it deleted and asks each service to erase its data: see The purge. The Purge page shows what each service still holds.

> **Note**
>
> A suspended tenant keeps the **Add user** entry, but the server gives and changes no access in it: “This tenant is not active: no access can be given or changed in it.” Removing a user is still possible.

A tenant’s members

The **Access** card of a tenant’s page lists one row per user holding a role or plans there:

- **User**: the username (link to the user), with the **Disabled** tag if their account is disabled, and **Suspended** if their access to this tenant is suspended;
- **Role**: **Member** or **Admin**;
- **Plans**: one link per plan.

See in the access list opens the access list filtered on this tenant; **Give an access** opens the access form with this tenant chosen. Roles, plans and the suspension of an access are described in Access to tenants.

ALM storage

ALM storage (··· menu or tenant page) opens the **ALM storage** page: “What ALM keeps in S3 for this tenant, by kind. Open a kind to see its files; the ones no longer in use are flagged.” The page only reads: it deletes nothing.

The top of the page gives the total size “in N files under” the tenant’s prefix, the number of files **not in use** and of **orphans**, and a bar of the size by kind. The table has one row per kind: **Kind**, **Files**, **Size**, **Oldest**, **Not in use**.

Kind	Content
Run inputs	What a run was given, kept to replay it
Run results	The engine’s answer, deleted once saved: should be close to none
Cash-flow stores	One DuckDB file per run, kept as long as the run
Notebook Parquet	Built on demand for notebooks, kept 3 months
Audit trails	The reporting’s audit trails
Imported files	The files imported into ALM
Other	Anything else under the tenant’s prefix

Opening a kind (**Show the files**) lists its files: **Key**, **Status**, **Size**, **Last modified**, **Run**. The filters are **All**, **Not in use**, **In use**, **Leftover** and **Orphan**, each with its count. The statuses:

- **In use**: something in ALM points at it;
- **Leftover**: its run exists but should no longer have it;
- **Orphan**: nothing in ALM points at it.

If ALM does not answer, the page shows “ALM’s storage could not be read” with the reason (an ALM without this listing, a refusal, or no answer) and **Retry**.

Who may do what

The whole console is reserved to platform administrators (the `ROLE_ADMIN` role in a platform realm): any other account sees the **Access restricted** page. The server applies the same rule to every call of these screens. See Rights and Who opens the console.

The server also refuses, whatever the screen:

- any change to a deleted tenant;
- suspending or reactivating a tenant pending deletion: restore it first (“This tenant is scheduled for deletion: restore it first.”);

- restoring a tenant that is not pending deletion.

Users

Users are in the **Tenant management › Users** menu.

A **user** is a person's account. It belongs to one client, its **home client**, and may be a guest of other clients. It has access to a tenant only through an **access**: a role on that tenant and at least one plan, one plan at most per service. What belongs to Keycloak (sign-in, password, sessions) is explained in Identities.

The user list

The page is titled **Users**: "The accounts of each client. Their access to tenants is given from the tenants tree or the tenant access list." The header offers **Import** (see Import users) and **New**.

Above the list, the Keycloak account changes pending or failing may show: see Keycloak account changes.

For each user, the list shows:

- **Username**: link to their page, with the "Disabled since …" tag if they are disabled;
- **First Name**, **Last Name**, **Email**;
- **Client**: their home client, followed by "+N" if they are a guest of N other clients (tooltip "Guest of: …");
- **Last login**: how long ago, the exact date and time in a tooltip; or "Never", in grey, when no login was seen ("No login seen since logins started being recorded").

The columns sort.

Filters

- **Global search…**: username, first name, last name, e-mail.
- **Select a client**: the users of the clients picked.
- **Inactive for**: **Any activity**, **No login for 30 days**, **… 90 days**, **… 180 days**, **Never signed in**. Each choice shows how many users it keeps. A filter in days keeps only the users with a login seen, older than that.
- **Show disabled users (N)**: disabled users are hidden by default; the switch shows them, greyed out.

With a filter in days, the **Select these N** button ticks, on every page, the active users it keeps, ready for **Disable**. It is not offered for **Never signed in**: having no login seen may only mean none happened since logins started being recorded.

Actions

A user's ··· menu offers **View**, **Edit**, **Disable** (or **Enable**) and **Delete**. **Delete** is greyed out while the user has access to a tenant; the tooltip says so: "Cannot be deleted while the user has access to N tenant(s)."

As soon as a user is ticked, the bar's **Disable** button acts on the selection (see Disable and enable). Its arrow offers **Delete** for the selection (see Delete a user).

Create or edit a user

New opens **Create a new User**; **Edit** opens **Edit a User**.

The **Account** section: "The login of the account and the client it belongs to."

Field	Required	Checks and hint
Realm	no	The realms the platform accepts; on creation, the default realm is already chosen. Hint: "The Keycloak realm the account logs in with; leave the de-

Field	Required	Checks and hint
		fault unless the user belongs to another realm." When editing: "Change the realm only to correct an account registered under the wrong realm."
Username	yes	Unique within its realm, checked while typing: "Another user of this realm already has this username."
Client name	yes	The home client. It cannot be changed after creation.
Multi-client	no	"Can be made a guest of other clients' tenants. Leave unchecked except for a consultant or an auditor."
Mail language	no	**Français**, **English**, or empty ("Not set"): "The language of the mails sent to this user. Not set: the mails carry both French and English."

The **Identity** section: "Shown in the lists and on the user's page."

Field	Required	Checks
First Name	yes	
Last Name	yes	
Email	yes	E-mail format; unique across the whole platform, all realms together: "Another user already has this e-mail."

When the client has a realm, the account must be in that realm: the server refuses another one. A user still a guest of other clients cannot lose **Multi-client**: the box is locked, with "Still a guest of N other client(s): remove them on the user's page before unchecking Multi-client."

Save creates or changes the user, then goes back to the previous page. **Back** leaves without saving, with the **Unsaved changes** dialog if values were typed.

> **i** Note
>
> Creating a user here creates their record in subscription, not their Keycloak account. See What subscription holds, what Keycloak holds.

A user's page

The header shows the **All users** link, the initials, the first and last name (or the username), then the username and the home client (link). The buttons are **Access history** (the history filtered on this user, see Access history) and **Edit**.

The page has four cards:

- **Account**: **Active** or "Disabled since …", a sentence on what that means, the **Disable** or **Enable** button, and the **Last login**;
- **Identity**: **Username**, **First Name**, **Last Name**, **Email** (mail link), **Client**, **Multi-client** (**Yes** or **No**), **Mail language**;
- **Guest clients**: see Guest clients;

- **Access to tenants**: one row per tenant where the user holds a role or plans, with **Tenant** (link), **Role**, **Plans**, and the **Suspended** tag if their access there is suspended. **See in the access list** opens the access list filtered on this user; **Give an access** opens the access form with them chosen.

Guest clients

A user normally joins only the tenants of their home client. A **Multi-client** user may be made a **guest** of other clients, one client at a time, and then join their tenants like one of their own users.

The **Guest clients** card of the user's page:

- without **Multi-client**, says "Only a multi-client user can be a guest of other clients: tick Multi-client in the edit form first.";
- with **Multi-client**, lists the clients they are a guest of (or "Not a guest of any other client yet."), each with **Remove**, then offers **Pick a client** and **Add**. The choice leaves out the home client and the clients already granted.

Remove asks: "Remove X as a guest of C? X is taken out of every tenant of C and will receive an e-mail about it."

Disable and enable

Disabling a user cuts all their accesses at once, on every tenant of every client, without losing them.

Disable (··· menu, **Account** card, or the list's bar for the selection) opens a dialog that spells out:

- **Stops now**: "Every access is cut, on every tenant of every client"; "The Keycloak account is disabled within a minute and its open sessions end; it is not deleted".
- **Kept**: "Plans and roles on each tenant"; "Enable gives every access back as it was".

Once confirmed: "X disabled: the Keycloak account follows within a minute". For a selection, it applies to the active users ticked, one call each; a message says how many were disabled and how many could not be.

Enable asks "Enable X again? Their accesses come back on each tenant, and their Keycloak account is enabled within a minute."

A disabled user stays in the list, marked as disabled. The screens show it with the **Disabled** tag (a tenant's accesses, the access list). How it works is described in Deactivation.

Keycloak account changes

Disabling, enabling or deleting a user asks for a change of their Keycloak account, made in the background within the minute. While some are left, the top of the user list shows them, read again every 30 seconds:

- pending: "N Keycloak account change(s) pending:" followed by each account's username (tooltip: its realm);
- failing: "N Keycloak account change(s) failing", "Tried again automatically; the Keycloak account stays as it was until a try succeeds." Each row gives the action (**Disable** or **Enable**), the user and their realm, the number of failed attempts, the date of the next try and the last error.

Nothing shows when there is none.

Delete a user

Delete (··· menu) asks "Delete the user X?". The server refuses while the user has access to a tenant: remove their accesses first.

For a selection (the **Disable** button's arrow, **Delete**), the users who still have accesses are skipped: "Delete N of these M users? K still have accesses and will be skipped." The final message gives the number deleted, skipped and failed.

Deleting removes the user's record and disables their Keycloak account, without deleting it.

Import users

Import opens the **Import users** page: “Create many accounts at once from an Excel workbook.” It has three steps.

1. **Prepare the file**: “Start from the template: one row per user, the columns in the template’s order.” **Download the template (.xlsx)** downloads `userTemplate.xlsx`. Its columns are `Username`, `Email`, `FirstName`, `LastName`, `ClientName` (the exact name of a client) and `Language` (`fr`, `en` or empty).
2. **Send the file**: choose the **Realm** the accounts are created in (the default realm is already chosen), then **Choose a file**. A `.xls` or `.xlsx` file of 50 MB at most; the import starts as soon as the file is chosen.
3. **Check the result**: “N user(s) created. N refused.”, with the time taken. The refused rows are listed: **Row** (the row’s number in the workbook), **Username**, **Reason**.

A row is refused when:

- a required column is empty (all but `Language`): “The X column is empty.”;
- the e-mail is not valid, or is already used;
- the language is neither `fr` nor `en`;
- no client has that name;
- the realm chosen is not the client’s, when the client has one;
- the username is already taken in that realm.

The other rows are created; blank rows are skipped. “Fix these rows in the file and import it again: the accounts already created are left out.” **Import another file** goes back to step 2; **Back to the users** goes back to the list.

Access to tenants

The **Tenant access** list has no menu entry. It opens from **See all N accesses** in the tenants tree, **See in the access list** on a tenant’s or a user’s page (filtered on it), or the **All accesses** link of an access page. “One row per user and tenant: a role, then the plans the user is entitled to.”

The header offers **New** (see Give or change an access). Each row shows:

- an eye that opens the access page;
- **Tenant name** (link to the tenant);
- **Username** (link to the user), with the **Disabled** and **Suspended** tags when they apply. When neither the account nor the access is cut, the **View as user** link opens the support console on that tenant and user;
- **Plan name**: their plans on that tenant;
- **Role**.

The bar offers the global search, three filters (**Search by tenant…**, **Search by plan…**, **Search by username…**) and **Delete**, active as soon as a row is ticked: “Remove these N accesses? Each user loses their role and plans in that tenant and will receive an e-mail about it.” A row’s ··· menu offers **View**, **Edit**, **Suspend access** (or **Reactivate access**) and **Delete**.

Role and plans

On a tenant, a user has:

- a **role**: **Member**, who “Uses the features of their plans.”, or **Admin**, who “Also manages the other users’ access to the tenant.”;
- at least one **plan**, one plan at most per service. A service they hold no plan of stays closed to them. Plans are described in Plans.

Every access change is recorded in the Access history, and the user gets an e-mail describing their new rights.

Give or change an access

The access form, titled **Create a new UserTenant**, opens from **New** (access list), **Add user** (a tenant’s menu, the tenant already chosen) or **Give an access** (a tenant’s or a user’s page, the one or the other already chosen). **Edit** opens **Edit a UserTenant**.

Who and where: "A user of the client, on one of its tenants."

- **Client** (creation only, required): choosing the client fills the next two lists.
- **Tenant name**: a tenant of the client.
- **Username**: a user of the client, or a guest of the client.

As soon as the tenant and the user are chosen, the form reads what the user already holds on that tenant and shows:

- **Role**: **Member** or **Admin**;
- **Plans, one per service**: one list per service, with **No plan** as the empty value. "At most one plan per service; no plan leaves the service closed."

Save is active only with at least one plan. For a user already a member, taking every plan away shows "A member holds at least one plan. To take the user out of the tenant, use Remove from the tenant." If saving removes plans, a confirmation names them: "Remove the user's plans X from this tenant?" The role and all the plans are saved at once.

When editing, the client, the tenant and the user are locked. For a user already a member, the **Remove from the tenant** button removes their role and all their plans, after the confirmation "Remove X from T? …".

The server refuses:

- a tenant that is not active: "This tenant is not active: no access can be given or changed in it.";
- a user who neither belongs to nor is a guest of the tenant's client;
- two plans of the same service.

The tenant's validity is not checked: an access can be prepared before the contract starts.

An access page

An access page opens from the eye or **View**. The header shows the **All accesses** link, the user's username and, when it applies, the **Suspended** tag. The buttons are **Suspend access** (or **Reactivate access**), **Remove from the tenant** and **Edit**.

The page shows **Tenant name** (link), **Username** (link), **Role**, **Plans** (or "No plan") and **Access**: **Active**, or "Access suspended since …". After **Remove from the tenant**, the page goes back to the access list.

Suspend access to a tenant

Suspend access cuts a user's access to one tenant only: "Suspend X's access to T? Their role and plans stay, but no service lets them in that tenant until the access is reactivated."

Reactivate access asks "Give X back their access to T, with their role and plans?". The server reactivates an access only in an active tenant.

Not to be confused:

	Applies to	Cuts
Suspend access	one user on one tenant	that access only
Disable the user	one user	all their accesses, and their Keycloak account
Suspend the tenant	one tenant	the access of all its users

Who may do what

These screens are reserved to platform administrators, like the whole console; the server applies the same rule. See Rights. A tenant's **Admin** role, given here, gives no access to the console.

Access history

The access history is in the **Tenant management › Access history** menu.

"Every change of a user's access rights, latest first: who gave or took which role or plan, to whom, on which tenant. The export lists who holds what today." The page changes nothing: it reads the history and exports the access review.

It also opens, already filtered, from the **Access history** button of a tenant's or a user's page.

The timeline

Events are grouped by day, the latest first. Each day carries its name ("Today", "Yesterday"), its date and its number of events. Above them, the total: "N event(s) · page X of Y".

Each event shows:

- the time, and how long ago; the tooltip gives the date and time to the second;
- its **type**, in a tag coloured by its nature: a right given, a right taken, a right put on hold or refused, a right changed (see Event types);
- the user concerned (link to their page) and their realm;
- the tenant, "on T" (link to the tenant), or "on every tenant" for an account being disabled or enabled; then the client;
- for a plan change, the service's name; for a role change, "role"; then the before and the after, "none" for an empty side (for instance **Standard → Premium**, or none → **Member**);
- the cause, when the change followed from another one: "because the tenant was purged", "because the guest access to the client was revoked", "because the user was deleted";
- who made the change (see Who made the change);
- two buttons: **Only this user's history** and **Only this tenant's history**, which add that filter.

The page shows 50 events at a time; **Newer** and **Older** change page. **Refresh** reads the history again.

With no event: "No access-rights change recorded yet.", or, with filters, "No access-rights change matches these filters." and the **Clear the filters** button.

Filters

Filter	Empty value	Note
Client	**All clients**	Picking a client narrows the tenant list to its tenants, and clears a chosen tenant of another client.
Tenant	**All tenants**	Each tenant shows its client.
User	**All users**	Each user shows their realm.
Dates	**Any date**	A range from the first to the last day, both included. It applies once both days are picked.
Types	**All event types**	Several types allowed: "N event type(s)".

The filters combine. **Clear the filters** removes them all. They are written into the page's address: a copied link opens the same filtered history again.

Event types

Type	Nature	What happened
Member added	given	The user got a role on the tenant.
Member removed	taken	The user lost their role on the tenant.
Role changed	changed	Their role on the tenant changed.
Plan granted	given	They got a plan of a service.
Plan changed	changed	Their plan of a service was replaced by another.
Plan removed	taken	They lost their plan of a service.
Access suspended	on hold	Their access to the tenant was suspended.
Access reactivated	given	Their access to the tenant was reactivated.
Guest of client	given	They became a guest of a client.
Guest access revoked	taken	They are no longer a guest of a client.
Account disabled	on hold	Their account was disabled, on every tenant.
Account enabled	given	Their account was enabled again.
Support policy changed, **Support access requested**, **Support access granted**, **Support access revoked**, **Support access declined**, **Support session opened**, **Support session closed**, **Support session refused**	depends	MLA support's access to a tenant: see Support access.

One change may give several events: giving an access with a role and two plans records **Member added** and two **Plan granted**.

Who made the change

On the right of each event, who made it:

- **a person**: their initials, their username and their realm;
- **a service**: the id of the service that called;
- **a scheduled job**: "System". That is the case of a tenant's purge, which removes its members at the end of the waiting period.

The access review

Export access review (xlsx) downloads `access-review.xlsx`: who holds which plan on which tenant today. The button shows its scope: the client picked in the filter, or "all clients". The other filters do not apply to the export.

The workbook has one sheet, `Access review`, with one row per plan held, and a row without a service for a member who holds no plan. Deleted tenants are left out. The columns:

Column	Content
`Client`, `Tenant`, `Tenant status`	The tenant and its status.
`Realm`, `Username`, `First name`, `Last name`, `Email`	The user.
`Home client`, `Guest`	Their home client, and whether they are a guest of the tenant's client (`yes` or `no`).
`Role`, `Suspended since`	Their role on the tenant, and when their access was suspended.
`Service`, `Plan`, `Service held since`	The plan held, and since when the user has held a plan of that service on that tenant.
`Last login`	Their last login seen.

The header row is frozen and filterable.

Retention

Each change is recorded when it is made, in the same operation: a change rolled back leaves no trace. The history cannot be edited; every night it is purged of its events older than 90 days.

Changes on a temporary test tenant are not recorded (see Test tenants).

Who may see it

The page is reserved to platform administrators, like the whole console. See Rights.

Purge

The purge page is in the **Tenant management** › **Purge** menu.

"The deleted tenants, and what each service still holds of them: a tenant is purged when MyReg, the library and ALM all hold nothing."

When a deleted tenant's waiting period is over, subscription purges it and asks each service to erase its data (see The purge). This page is there to check that each service did, and to erase the data no purge reaches.

The top of the page picks the view: **Deleted tenants** or **Orphan data**.

Deleted tenants

This view lists the deleted tenants, the latest first. It only reads: it neither deletes nor restores anything.

At the top, tiles count the tenants shown, then those **Purged**, **Data left**, **Incomplete** and **Not checked**.

For each tenant, the table shows:

- **Tenant**: its name, the **test** tag for a test tenant, and its reference (the id the services know);
- **Client**;
- **Deleted**: how long ago (the exact date in a tooltip);
- **Services**: one tag per service, **MyReg**, **Library** and **ALM**, showing its last result, with the number of rows left if any;
- **Status**: the verdict (see Check a tenant);
- a magnifier, **Check now**.

The bar offers a search ("Name, tenant id or client") and three scopes, each with its count: **All**, **Test tenants**, **Client tenants**. A test tenant is a tenant whose name contains `playwright`: the temporary tenants the automated tests create.

The header offers **Refresh**, which reads the list again, and **Check the N shown**, which checks every tenant shown, three at a time.

Check a tenant

Check now asks MyReg, the library and ALM how many rows and stored objects they still hold for that tenant. Each answer shows as soon as it arrives.

Status	Meaning
Not checked	Nothing asked yet.
Checking…	At least one service has not answered yet.
Purged	All three services answer that nothing is left.
Data left	At least one service still holds something.
Incomplete	No service holds anything, but at least one could not answer.

Once checked, the row's arrow opens the **Details per service**: for each service, the time of the check and either "Nothing left.", or the list of what is left, table by table (the **store** mention tells an object store from a table), the largest first, or the reason it could not answer:

- "This service cannot count a tenant's data yet (no such endpoint)."
- "This service refused the request: platform administrators only."
- "This service did not answer."

> **Note**
>
> A tenant showing **Data left** right after its deletion may simply be waiting for the service to handle the purge request. If the data stays, look at the service concerned: a service's failed purge is retried, then set aside (see What each service purges).

Orphan data

"Data the services hold for a tenant id subscription does not know at all, whatever its status. No purge reaches it: subscription never deleted such a tenant."

The view asks each service for the list of tenants it holds data for, and compares it with the tenants subscription knows, deleted ones included. An id only services know is an orphan.

At the top, the number of orphan tenant ids and the total: "N rows and stored objects in all". **Refresh** reads everything again.

The table has one row per orphan id, the largest first: **Tenant id** (with "unknown to subscription"), one column per service with its rows (and its stored objects), a **Total**, and the **Purge** button. In a service's column, a shield marks the id that service holds as its default tenant ("default tenant of this service: protected, never purged here").

If a service could not give its list, a warning names it with the reason: "Some services could not list their tenants, the list may miss their orphans:" With no orphan: "No orphan data: every tenant id the services hold is known to subscription."

Purge orphan data

> **Warning**
>
> This purge is final: "Every row and stored object below is deleted for good, in each service. There is no undo and no backup is taken."

Purge opens the **Purge orphan data** dialog. It lists, per service, what will be deleted, and the services skipped because the id is their default tenant.

1. Type the exact tenant id in **Type the tenant id to confirm**.
2. **Purge for good** first asks subscription whether it knows this tenant now. If it does, nothing is sent: "subscription knows this tenant now: nothing was sent, nothing was deleted."
3. Otherwise each service holding something of that id purges it, one after the other. Its answer shows under its name:

Answer	Meaning
"deleted: N rows, N stored objects"	Done.
"refused: subscription knows this tenant (409)"	The service checked with subscription, which knows this tenant.
"refused: this is the default tenant of the service (409)"	The service protects that id.
"refused: the service could not ask subscription (503)"	The service could not check; nothing is deleted.
"not available: no purge endpoint yet (404)"	That service cannot purge yet.
"refused: platform administrators only"	Authorisation refused.
"failed"	Any other error.

Once each service has answered: "Each service answered. The list is read again on close." **Close** reads the list again.

Default tenants of services

“A service running with subscription disabled keeps all its data under its default tenant, which subscription does not know. That service’s data of it is never purged from here; another service holding data of the same id may list it among the orphans.”

The **Default tenants of services** box, under the table, lists those ids: which service they are the default tenant of, and what each service holds of them. They are never purged in the service that names them so.

Who may do it

The page is reserved to platform administrators, like the whole console. Each service also checks, on its side, that the call comes from a platform administrator. See Rights.

Support console

The support console is in the **Tenant management › Support console** menu. It lets MLA support and administrators open CIRCÉ or ALM "as" a member of a tenant, read only, ask that member for authorization when it is missing, set the tenant's access policy and follow the journal of these accesses.

How it all works (policies, authorizations, sessions, journal) is described in Support access. What the target user sees is described in the CIRCÉ manual, "Support access" page.

Who can use the console

Account	Tenants shown	What it can do
Platform administrator (`ROLE_ADMIN` role of the MLA realm)	every tenant, except the end-to-end tests' temporary tenants	view, request access, set the policy, read the journal, **End** a consultation
Support member (`SUPPORT` role of the MLA realm)	the same	view, request access, read the journal; neither the policy nor **End**
Tenant administrator	only the tenants they are an active administrator of	view the members of their tenants, request access, set the policy, read their tenants' journal, **End** a consultation in their tenants

An account in none of these cases sees **No administered tenant is available.**

> **Note**
>
> **A tenant administrator has no access to the rest of the console.** The support console is the only page open to them: it sits outside the administration layout. They open it with the page's address followed by `?realm=<their realm>`: the sign-in then happens in their organization's realm instead of the MLA realm. Without this parameter, the sign-in happens in the MLA realm.

Open the console

Three paths lead to the console:

- the **Tenant management › Support console** menu;
- in the **UserTenant** access list, the **View as user** link next to each active member (neither disabled nor suspended): the console opens on that member's tenant;
- the link in the e-mail sent to the requester when the user accepts or declines their request: the console opens on the tenant concerned (when the console's address is configured; otherwise the link leads to the product's page).

The console takes the whole page. It has three columns:

- on the left, the list of tenants;
- in the middle, the chosen tenant: its members, then, for whoever administers it, its policy and active authorizations;
- on the right, the tenant's **Support journal**.

At the bottom of the left column are your login, the language choice (**FR** / **EN**) and the **Back** link, which returns to the home page.

Choose a tenant

The **Tenants** list is sorted by name. Each tenant shows its policy (**On request**, **Always** or **Never**) with a coloured dot. Beyond five tenants, the **Filter tenants…** field filters the list by name. The first tenant is selected on opening; the chosen tenant is kept in the page's address.

The header of the chosen tenant shows:

- its name and its client's subdomain;
- the longest authorization allowed ("authorizations up to … h");
- its policy and, below the header, what it allows:

Policy	Text shown
Never	Support cannot open any member's screens. Requests are refused.
On request	Each user authorizes support themselves, for a limited time, and can revoke it.
Always	Authorized MLA support may view members' screens without asking first.

The **Refresh** button (circular arrow) reads the tenant again.

The tenant's members

The **Members** table lists the tenant's members, sorted by login, with their realm, their **Product** (`CIRCE` for CIRCÉ, `ALM` for HERMES ALM, or both) and their **Status** as seen from your account. The **Find a user…** field filters by login or realm.

Above the table, four counters also act as filters (one click filters, a second click removes the filter):

Counter	Members counted
can be viewed now	you can open their session right away
need the user's consent	the user's authorization is required and you have not asked for it yet
requests pending	you have already sent a request, still unanswered
cannot be viewed	no action possible: your own account, a member with no product, or a refusal for another reason

The **Status** column shows **Request pending** for a request you sent, otherwise the reason subscription gives:

Status	Meaning
Authorized	You can view. With a personal authorization, its end and the time left are shown ("Until …").
Consent required	The policy is **On request** and the user has not authorized you.
Policy: never	The tenant's policy forbids consultations.
Your own account	You do not view yourself.
Operator not authorized	Your account cannot view this tenant (account disabled, or a tenant administrator who does not administer this tenant).
Inactive member	The user is not an active member of the tenant, or the tenant is not active.

Status	Meaning
Protected or disabled account	The target account is disabled or holds a privileged role (platform administrator, support, identity-provider administration).
MLA staff	MLA staff accounts are never viewed.
Identity provider unavailable	Keycloak did not answer: try again later.

When a member has both products, a `CIRCE` / `ALM` selector chooses the one to open.

Request access

For a member whose status is **Consent required**, click **Request access**. A form opens under their row:

1. **Suggested duration**: **1 hour**, **4 hours** (the default), **1 day** or **7 days**. It is a suggestion: the user chooses the duration they grant.
2. **Reason**: what the user should know, for example the screen, the report or the error you need to see (2,000 characters at most).
3. Click **Send request**, or **Cancel**.

The console shows **Request recorded. The user must authorize access themselves.** The user receives a "Support access request" e-mail with a link to the chosen product's **Support access** page, and a banner in the application. The member's button becomes **Requested**, and their status **Request pending**.

A request stays valid for seven days. Sending a request again to the same user while the previous one is pending creates no duplicate.

> **Note**
>
> A request gives no access. Only the user can accept it, and the authorization they give in answer applies to you alone: another support member must make their own request.

View as user

For an **Authorized** member, click **View as user →**. A new tab opens on the chosen product, signed in with the member's identity, in this tenant only and read only. Your session in the console stays open.

If the browser blocks the tab, the console shows **Allow the new tab to open, then retry.**

During the consultation, a banner reads "You are viewing … as … (read only)"; **Exit** ends it. A write action shows **This action is unavailable during a support consultation.** The consultation also ends on its own: end of the authorization, revocation, maximum duration, inactivity; see the end reasons.

Policy and active authorizations

This part is shown to platform administrators and to the tenant's administrators only.

Support access policy shows three cards: **Never**, **On request** and **Always**. The policy in force is marked **Current**. Choosing another card shows **Save policy** and **Cancel**. Once saved, the console shows **Policy saved.** and the change goes into the journal.

> **Warning**
>
> Authorized support can view the screens in read-only mode within this tenant. This access bypasses the user's corporate identity provider and its MFA or conditional-access controls. **Always** therefore grants this capability permanently: decide it with the client.

Active authorizations lists the authorizations running in the tenant: the user who gave it, its end date ("Until …") and the time left. With no authorization, the list shows **No authorization is running.**

The support journal

The **Support journal** lists the tenant's support access events, latest first and grouped by day, 50 per page (**Previous**, **Next**):

Event	Wording
consultation opened	"… opened … as …"
consultation ended	"…'s session as … ended", with its reason
opening refused	"Refused: … as …", with the reason
access request	"… asked … for access"
authorization	"… authorized support"
revocation	"… revoked …'s authorization"
request declined by the user	"… declined a support request"
policy change	"… changed the policy: … → …"

An event's second line gives, as the case may be, the end reason, the reason for a refusal, the expiry ("→ date") and the application opened. The end of a consultation also shows the actions the gateway blocked during the session, for example "1 action blocked (read-only)"; the kinds are **read-only**, **another tenant** and **nested consultation**.

Filters opens the form:

- **Operator**: the support member's login. This filter keeps that operator's consultation openings, ends and refusals;
- **User**: a member of the tenant;
- **From** and **To**: the period.

Apply runs the search, **Clear** resets the filters. The number of active filters shows on the **Filters** button.

Export downloads the filtered journal as CSV (`support-access.csv`), every page included.

A tenant administrator sees only their tenants' journal. Events are kept 90 days by default (see Support access).

End a running consultation

In the journal, each running consultation (an opening with no recorded end) carries an **End** button. It is shown to platform administrators and to the tenant's administrators only.

1. Click **End**.
2. Confirm: "End …'s consultation as … now?", then **End** (or **Cancel**).

The operator's session stops at once; their next request shows the **Support consultation ended** page. The journal reloads and the end reads "ended by" followed by your login.

Messages on failure:

Message	Cause
You may not end this consultation.	subscription refused: you are neither a platform administrator nor an active administrator of this tenant, or the session is already closed, or Keycloak did not answer.
This consultation has already ended.	The gateway no longer finds the open session.
The consultation could not be ended. Try again.	Any other error, for example subscription unreachable: nothing was ended.

A support member without these rights ends their own consultation with **Exit**, in the banner of the viewed tab.

Live updates

The console reloads on its own when a request is created, accepted or declined, or an authorization is given or revoked in the tenant shown: members, requests, authorizations and journal are read again without clearing the page or closing the form in progress.

When one of your requests is answered, or an authorization you held is revoked, a notice shows at the bottom of the page for ten seconds:

- “… accepted your request (duration)”;
- “… declined your request”;
- “Access to …‘s account was revoked“.

If an update is missed, **Refresh** reads the tenant again.

II Reglab Platform

Services

Services are in the **Reglab Platform › Services** menu. A service is a RegLab product: it carries its features, which plans open to users (see Key concepts).

Like the whole console, this page is reserved for platform administrators (role `ROLE_ADMIN`).

The list of services

The **Services** page opens on the list of services: "The RegLab products. Each one carries its features, which plans open to users."

For each service, the list shows:

- the **Service name**, which opens its page;
- its **Description**;
- its **Plans**: a "*n* plan(s)" tag, which opens the Plans list, or "No plan".

The **Service name** and **Description** headers sort the list. The **Global search…** box searches the name and the description.

The ⋮ menu of a service offers **View**, **Edit** and **Delete**. Lists, selection and bulk delete work as everywhere in the console (see Lists and forms).

Creating a service

Click **New**. The **Create a new Service** form opens: "This form is dedicated to the creation of a Service."

Fill in:

- the **Service name**: required, and unique. A name already taken is refused: "A service with this name already exists.";
- the **Description**: optional, rich text.

Click **Save**. The message "You successfully created a Service" shows and the list of services comes back.

When it is created, the service automatically receives its `base` feature (description "Access to the service"): it is what makes a user a member of the service, and every plan of the service carries it (see The base feature).

> ⚠ Warning
>
> **The service's name is the one the applications ask for.** To know what a user holds, an application asks subscription for their plan of a service named by its name (compared ignoring case): CIRCÉ asks for `irrc`, the assistant asks for `irrc`, `stress` and `library`. Renaming a service therefore closes it to the users of the applications that ask for it under its former name. See From plans to rights.

Editing a service

Use **Edit** in the ⋮ menu of the list, or the **Edit** button on the service's page. The **Edit a service** form offers the same fields. **Save** shows "You successfully updated a Service" and leads back to the list.

A service's page

A service's page has its name as title and shows:

- the **Service name** and its **Description**;
- its **Features**, in alphabetical order, each linked to its page, or "No feature";
- its **Plans**, in alphabetical order, each linked to its page, or "No plan".

The **Back** and **Edit** buttons are at the top right.

Deleting a service

Use **Delete** in the ⋮ menu, or tick services and click **Delete** in the toolbar. A confirmation is asked:

- for one service: “Delete the service *name*? The server refuses while a plan belongs to it.”;
- for a selection: “Delete these *n* services? The server refuses the ones a plan belongs to.”

A service that still has plans cannot be deleted: “This service cannot be deleted: plans use it.” Delete or move its plans first (see Plans).

Deleting a service also deletes all its features, `base` included.

Plans

Plans are in the **Reglab Platform › Plans** menu. A plan belongs to a service and opens a selection of its features; a user receives plans in a tenant, at most one per service (see Key concepts).

Like the whole console, this page is reserved for platform administrators (role `ROLE_ADMIN`).

The list of plans

The **Plans** page opens on the list of plans: "A plan belongs to a service and opens a selection of its features."

For each plan, the list shows:

- the **Plan name**, which opens its page;
- its **Service name**, which opens the service's page;
- its **Features**: "*n* feature(s)".

The **Plan name** and **Service name** headers sort the list. The **Global search…** box searches the plan's name and its service's name.

The ⋮ menu of a plan offers **View**, **Edit** and **Delete**. Lists, selection and bulk delete work as everywhere in the console (see Lists and forms).

Creating a plan

Click **New**. The **Create a new Plan** form opens: "This form is dedicated to the creation of a Plan."

Fill in:

- the **Plan name**: required. It is unique across the whole platform, all services together;
- the **Service name** ("Select a service"): required;
- the **Features** ("Select the features it opens"): at least one. The list stays greyed out until a service is chosen, then offers only that service's features. Choosing another service keeps only the ticked features that belong to the new service.

Click **Save**. The message "You successfully created a Plan" shows and the previous page comes back.

The server checks the plan:

- the service's `base` feature is added to the plan if it was not ticked: a plan always carries it;
- a name already taken is refused: "A plan with this name, or with the same features, already exists.";
- a plan that would open exactly the same features as another plan of the same service is refused: "Another plan of this service already opens the same features.";
- a plan opens only features of its own service.

A plan's page

A plan's page has its name as title. The **All plans** link leads back to the list; the **Delete** and **Edit** buttons are at the top right.

The page shows three cards:

- **Definition**: the **Plan name** and its **Service name**, linked to the service's page;
- **Features**: the number of features, then each one with its name (linked to its page) and its description, in alphabetical order. Beyond five features, the **Filter the features** box filters the list on the name. A plan without features shows "This plan opens no feature yet.";
- **Used by**: the tenants whose users hold the plan, each linked to its page with "*n* user(s)", or "No tenant uses this plan."

Editing a plan

Use **Edit** in the ⋮ menu of the list, or the **Edit** button on the plan's page. The **Edit a Plan** form ("This form is dedicated to the edition of a Plan.") offers the same fields and the same checks as at creation. **Save** shows "You successfully updated a Plan" and leads back to the previous page.

When a plan's features change, every user who holds it receives an e-mail with the features they gain or lose.

> ⚠ Warning
>
> **The applications see the change after a delay.** Changing a plan that is already held (its features, its service or its name) does not tell the applications: they keep applying what they remember until it expires, up to 5 minutes for CIRCÉ and one hour by default for the assistant when `assistant` is taken off. See What subscription does not announce.

Deleting a plan

Use **Delete** in the ⋮ menu or on the plan's page, or tick plans and click **Delete** in the toolbar. A confirmation is asked:

- for one plan: "Delete the plan *name*? The server refuses while users hold it.";
- for a selection: "Delete these *n* plans? The server refuses the ones users still hold."

A plan held by at least one user, in any tenant, cannot be deleted: "This plan cannot be deleted: users still hold it." The **Used by** card of its page shows the tenants concerned; take the plan away from the users first (see Client and Tenants).

Features

Features are in the **Reglab Platform › Features** menu. A feature is a capability of a service; the service's plans open a selection of them to users (see Key concepts).

Like the whole console, this page is reserved for platform administrators (role `ROLE_ADMIN`).

The list of features

The **Features** page opens on the catalogue: "The catalogue of features plans are built from."

The features are grouped by service, the services in alphabetical order; a feature attached to no service shows under "No service". For each feature, the list shows:

- the **Feature name**, which opens its page;
- its **Description**.

The **Feature name** and **Description** headers sort the list. The **Global search…** box searches the feature's name and its service's name.

The ⋮ menu of a feature offers **View**, **Edit** and **Delete**. Lists, selection and bulk delete work as everywhere in the console (see Lists and forms).

Creating a feature

Click **New**. The **Create a new Feature** form opens: "This form is dedicated to the creation of a Feature."

Fill in:

- the **Feature name**: required, and unique within its service. A name already taken in the service is refused: "A feature with this name already exists.";
- the **Service** ("Select a service"): required;
- the **Description**: optional, rich text.

Click **Save**. The message "You successfully created a Feature" shows and the list comes back. The new feature is open to nobody until a plan carries it: add it to the plans you want next (see Plans).

> ⚠ Warning
>
> **The name is a code.** The applications open a screen or an action when the user's plan carries a feature with a precise name, for example `anacredit`, `pilar3` or `mcp` in CIRCÉ, `assistant` for the assistant. A feature that no application reads opens nothing; renaming a feature that is read closes what it opened. The list of the names read is in From plans to rights.

The description is for people: when a user's rights change, the e-mail they receive names each feature by its description, or by its name when it has none.

The base feature

Each service has a `base` feature, created with the service (description "Access to the service"). It makes a user a member of the service: CIRCÉ, for example, counts as a member only a user whose plan carries `base`.

It is protected:

- it can be neither deleted, nor renamed, nor attached to another service: "A service's base feature cannot be deleted, renamed or moved.";
- every plan of the service carries it: the server adds it to a plan that lacks it.

It goes away only with its service.

A feature's page

A feature's page has its name as title and shows its **Feature name**, its **Service** and its **Description**. The **Back** and **Edit** buttons are at the top right.

Editing a feature

Use **Edit** in the ⋮ menu of the list, or the **Edit** button on the feature's page. The **Edit a Feature** form offers the same fields. **Save** shows "You successfully updated a Feature" and leads back to the list.

The server refuses:

- to attach to another service a feature that a plan opens: "This feature cannot move to another service: plans use it.";
- to rename or move the `base` feature (see The base feature).

Deleting a feature

Use **Delete** in the ⋮ menu, or tick features and click **Delete** in the toolbar. A confirmation is asked:

- for one feature: "Delete the feature *name*? The server refuses while a plan opens it.";
- for a selection: "Delete these *n* features? The server refuses the ones a plan opens."

A feature that a plan opens cannot be deleted: "This feature cannot be deleted: plans use it." Take it off the plans first (see Plans). A service's `base` feature cannot be deleted.

III Services Configuration

Crawler

The crawler is in the **Services Configuration › Crawler** menu.

The crawler (the `crawler` service) reads the regulators' sites and RSS feeds, finds the regulatory publications there and hands them to the common library, the one every tenant reads (see Library). Every day at 13:00 it also sends a summary mail, the digest, to a list of recipients.

The page is titled **Crawler**: "The regulatory publications found on the regulators' sites and feeds, and handed to the common library of every tenant."

Like the whole console, it is for platform administrators only. The crawler itself checks that the caller is one (see Rights).

> **Note**
>
> When the crawler does not answer, each list says so: "The crawler did not answer. Is it running?".

Header and last-run strip

The **Run now** button starts a crawl of every source switched on. While the crawl runs, it reads **Crawling…** and stays disabled. The message "Crawl started. This page follows it until it ends." confirms the start; when a crawl is already running, the page answers "A crawl is already running.".

Under the header, a strip sums up the last run:

- its status (see Last runs);
- "Last run on …" and its duration;
- "… new publication(s)";
- "Next run on …", or "Not scheduled".

The page follows the runs live: the crawler reports each change of a run, and the console reads the list again at once (see Events and calls). While a run is going, the page also reads the runs every 30 seconds, in case the live updates are down. When a run ends, the publications and the sources are read again.

The page has seven tabs: **Publications**, **Sources**, **Runs & schedule**, **Feeds**, **Recipients**, **Sent digests** and **Docling**.

Publications

The **Publications (n)** tab lists the publications the crawler found, in pages of 25, 50 or 100.

The filters are:

- the search (**Title, publisher, reference…**);
- where the publications stand with the library: **All**, **Sent**, **Skipped**, **Not sent**, **Indexed by the library**, **Failed in the library**, **Awaiting the library**.

The columns are:

- **Publication**: the title (or the reference), linking to the publication, then the publisher, the type and the reference;
- **Published**;
- **Detected**: when the crawler found it;
- **Sent to the library**: the date it was sent and what the library made of it, **Indexed** (the date in a tooltip), **Failed** (the reason in a tooltip) or **Pending** ("The library has not said yet whether it indexed it."); a skipped publication shows **Skipped** with the reason, one never sent **Not sent**;
- **Summary**: see Summaries for the daily mail;
- a **Send to the library again** button.

With no publication, the list reads "No publication found yet: run a crawl, or add feeds.".

Send to the library again

A publication, sent, skipped or not sent, can be handed to the library again:

- the **Send to the library again** button of its row sends it at once;
- after ticking rows, **Send to the library again (n)** first asks: “Send n publication(s) to the library again? The library downloads each one and indexes it if it does not hold it yet.”.

The outcome shows above the list: “n publication(s) sent to the library.”, or, when some could not be handed over to the message broker, how many failed, to try again.

Summaries for the daily mail

The library can summarise, with a language model, the publications it indexed. The summaries go into the 13:00 digest.

A summary can be asked for a publication that is indexed and was never summarised, or whose summary failed. Its box is ticked in the **Summary** column; the bar above the list offers:

- **Tick all (n)**: every publication of the page that can be summarised;
- **Untick**;
- **Start the summaries (n) · ~x tokens**, which asks: “Have the library summarise n publication(s)? About x input tokens, plus those of the answer.”.

The **Summary** column shows where each summary stands: **Not asked**, **In progress**, **Summarised** (the date and the model in a tooltip) or **Failed** (the reason in a tooltip). It also shows the tokens estimated before the request, then the input and output tokens used, and the cost in dollars when the library gives it. The bar adds up the summaries of the page.

Read the summary opens the summary as the digest carries it.

While a summary is in progress, the page reads the summaries again every 10 seconds.

Sources

The **Sources (n)** tab lists every source the crawler reads: the regulators’ sites (**Site**) and the RSS feeds (**Feed**). “A source switched off is left out of the runs; « Run » crawls it alone, switched off or not.”

The columns are:

- **Source**: its name, linking to its site, and its kind;
- **Last run**: the date, then how many publications it listed, or **Failed** with the error; “Not read yet” for a source never read;
- **Last success**;
- **Crawled**: the switch that turns the source on or off from the next run;
- **Run**: crawls this source alone; the button is disabled while a crawl runs.

Refresh reads the list again.

Runs & schedule

The **Runs & schedule** tab holds the schedule, the crawl settings and the log of runs.

Schedule

- **When to run (cron)**: six fields, “second minute hour day-of-month month day-of-week”, read in the time zone shown. For example `0 0 1 * * *` runs every day at 01:00.
- Four shortcuts fill the field: **Every day at 01:00** (`0 0 1 * * *`), **Weekdays at 06:30** (`0 30 6 * * MON-FRI`), **Every 6 hours** (`0 0 */6 * * *`), **Every hour** (`0 0 * * * *`).
- **Run on this schedule**: unticked, the crawler does not run on its own; use **Run now** instead.

Under the form are the next run, and who last changed the schedule and when. **Save** is only active when the schedule changed and the cron has six fields. It applies at once.

How the crawler crawls

Field	Values	Effect
Look back (months)	1 to 120	How far back a crawl may go; a source stops sooner, at a page of publications already known.
Pause between requests (ms)	0 to 10,000, in steps of 100	Between two requests to the same site. Raise it if a site starts refusing the crawler.
Timeout (s)	1 to 300	For connecting to a site and for each read; a site that does not answer in time fails its source.

The three values are whole numbers. **Save** is only active when a value changed and all are within their bounds. The settings apply from the next run.

Last runs

The table shows the last 30 runs; **Refresh** reads it again.

- **Started**;
- **Started by**: **Crawler start**, **Schedule** or **Run now**, with the user; a run limited to one source shows "only « name »";
- **Status**: **Running**, **Succeeded**, **Partly failed**, **Failed** or **Interrupted**;
- **Duration**;
- **New publications**.

A row unfolds to show:

- **Sources that failed**, each with its error;
- **What each source cost and gave**: per source, **Requests**, **Time**, **Listed**, **New**, then what the library made of them, **Indexed**, **Failed**, **Pending**. A source that made at least 20 requests and found no new publication shows **Nothing for many requests**. For older runs, which did not record these figures, the page only shows the **Publications listed by each source**;
- **Publications found**: the run's new publications, as links.

Feeds

The **Feeds (n)** tab lists the RSS feeds the crawler reads at each run, besides the regulators' sites. A feed is switched off in the Sources tab; it then shows **Off**. An icon marks a feed with an include or exclude pattern.

The columns are **Name**, **Publisher**, **Format** and **URL**, then the edit and delete buttons.

Add or edit a feed

Add a feed opens the form:

- **Name** (required);
- **Publisher** (required), e.g. `ECB`, `BIS`;
- **URL** (required), starting with `http://` or `https://`;
- **Keep only the items matching** (optional);
- **Leave out the items matching** (optional).

The two patterns are regular expressions, case-insensitive, of at most 1,000 characters. Each is looked for in the title, in each category and in the description. An item of the feed is kept when the first is found in one of them (or is empty) and the second is found in none. A pattern the crawler refuses is flagged under its field.

Test reads the feed once, without saving it, and shows how many items it lists, how many the patterns keep, and how many of the first ones are documents the library can index. Then come the **Kept** items,

with the reason for those the library would skip, and the items **Dropped by the patterns**, each with its reason.

Save is only active when the three required fields are filled and the URL is valid. A new feed is read from the next run.

Delete a feed

Deleting asks: “Delete the feed « name »? Publications already found stay in the library.”.

Recipients

The **Recipients (n)** tab manages who receives the digest. “The digest goes out every day at 13:00 to the enabled recipients, one mail each.”

> ⚠ Warning
>
> The mails go through the notification service, which accepts certain domains only. An address outside those domains is refused: send it a test mail to check it.

The header shows how many recipients are enabled out of the total, and offers **Refresh** and **Add a recipient**.

The columns are:

- **Email address**;
- **Name**;
- **Regulators**: those whose publications the recipient receives, or **All regulators**;
- **Receives the digest**: the switch that turns the recipient on or off;
- **Last change**: the date and who made it;
- the **Edit**, **Send a test mail** and **Delete** buttons.

Deleting asks: “Remove … from the digest recipients?”.

The add or edit form holds:

- **Email address** (required): a valid address, of at most 254 characters, not already in the list (“This address is already in the list”);
- **Name** (optional), at most 100 characters;
- **Regulators**: **All regulators**, “those the crawler adds later included”, or **Some only**, picked in **Choose the regulators** (at least one). In that second case, the recipient gets only these regulators’ publications, and no mail on a day without any.

Sent digests

The **Sent digests** tab lists, day by day and newest first, every digest the crawler sent, in pages of 20, 50 or 100. The columns are **Day**, **Mails**, **Publications** (and how many are indexed), **Recipients** and **Sent at**.

For each day:

- **Open the digest** shows the mails as they went out; when the digest went out in several mails, a switcher moves from one to the other;
- **Resend to an address** sends the mails again as they were sent, to this address alone, which need not be a recipient; the recipients’ list is left as it is. When the day has several mails, **Mails** chooses **Every mail of the day (n)** or a single one. An address outside the domains the notification service allows receives nothing;
- **Copy the public link** (when the browser allows it) and **Open the public page** give access to the digest’s public page; “No public page” otherwise.

Preview the next digest shows the digest as it would go out now: when it goes out, how many publications and recipients, or “The digest is off” with how many publications are waiting. The library may still index or summarise publications before then; **Refresh** reads the preview again.

Docling

The **Docling** tab follows the library's conversion queue: docling-serve converts the library's documents to Markdown, a few at a time, and the others wait their turn. The conversion covers both the common documents and the clients' documents.

The tab shows:

- whether docling-serve is up: **docling-serve answers**, with the most conversions it runs at a time, or **docling-serve does not answer**: conversions are then on hold and the queued documents keep their place;
- **Queued**, split between common and client documents;
- **Converted** and **Failed (no Markdown)** in the last hour;
- **Average time** per document, over the last hour;
- the **Estimated end** of the queue;
- the **Converting now (n of max)** table: **Document**, **File**, **Library** (**Common** or **Client**), **Started**, **Since**.

The tab updates live, and reads the queue every 10 seconds. When the library does not answer, it keeps the figures of the last read and says so. A library that does not serve its queue yet shows "Not available yet".

The documents themselves (validating the Markdown, relaunching docling) are managed in Library.

Notifications

Notifications are in the **Services Configuration › Notifications** menu.

The platform's services do not send their mails themselves: they hand them to the notification service (`notification`), which sends them through Azure Communication Services and keeps their history. The **Sent mails** page shows that history: "The mails the platform sent, newest first. They are kept 60 days.".

It therefore holds the mails of every service that goes through the notification service, for example the crawler's digest and subscription's own mails described below.

Like the whole console, the page is for platform administrators only. The notification service itself checks who reads the history and who resends a mail: only platform administrators may resend one.

The list of mails

The list comes in pages of 25, 50 or 100 mails.

The filters are:

- the search (**Subject or recipient…**);
- the status: **All**, **Sent**, **Pending**, **Failed**, **Rejected**.

The columns are:

- **Date**: when the notification service received the mail;
- **Status**: the sending status and, once a delivery report has arrived, a second badge (see Delivery);
- **Subject**; a mail that sends another one again shows the **Resent** link, which opens the original;
- **Recipients**: the first three, then "+n"; the tooltip gives them all;
- the number of attachments (paperclip icon);
- **Attempts**.

Status	Meaning
Sent	Handed over to the mail service.
Pending	Being sent, or waiting for another attempt.
Failed	Given up after the retries.
Rejected	Refused unsent: a recipient is not allowed.

> **Note**
>
> The notification service sends only to addresses in the domains it allows. A single recipient outside those domains has the whole mail refused: it then shows as **Rejected**.

When the notification service does not answer, the list reads "The notification service did not answer. Is it running?".

The detail of a mail

A click on a row opens the mail:

- **Status**, with its meaning;
- **Received** and **Sent**;
- **Attempts**;
- **Recipients**, all of them;
- **Resent from**, for a mail that sends another one again: a link opens the original;
- **Last error:**, when sending met one;
- **Delivery** (see below);

- **Attachments**: each one can be downloaded;
- **Content**: the body of the mail, shown as it was sent.

A mail older than 60 days can no longer be read: “The mail could not be read. It may be older than 60 days, or the notification service did not answer.”.

Delivery

Sent means the mail service (Azure Communication Services) accepted the mail. **Delivered** means the recipient’s mail server took it, as reported a few minutes later, recipient by recipient.

The **Delivery** table gives, for each recipient, the **Status**, the date it was **Attempted** and the server’s **Message**.

Delivery status	Meaning
Delivered	The recipient’s mail server took the mail.
Expanded	Handed to a distribution list, which delivers to its members.
Unknown	The recipient’s mail server gave no clear answer.
Suppressed	Not sent: the address is on the mail service’s suppression list (an earlier bounce or unsubscribe).
Filtered as spam	Filtered as spam on the way.
Quarantined	Held in quarantine by the recipient’s mail system.
Bounced	Refused by the recipient’s mail server (unknown address, full mailbox…).
Not delivered	The mail service could not deliver it.

In the list, the delivery badge sums up all recipients, or the worst one.

Without a report, the detail says why: for a **Sent** mail, the report can take a few minutes, and mails sent before delivery reports were introduced have none; for any other mail, the mail service never accepted it.

Resend a mail

The **Resend** button, at the bottom of the detail, sends the mail again to the same recipients. It first asks: “Send “subject” again to the same n recipient(s)? A new mail is created; this one stays in the history.”.

The new mail then opens and the outcome shows: “Mail resent:” followed by its status.

A resend is refused:

- for a **Pending** mail: “This mail is still being sent: it can be resent once it is sent, failed or rejected.”;
- for a user who is not a platform administrator: “You are not allowed to resend mails.”;
- for a mail that no longer exists: “This mail no longer exists: it may be older than 60 days.”.

The mails subscription sends

subscription itself sends four kinds of mail, through the notification service. They leave once the change is saved: a change rolled back sends nothing, and a failed send is logged by the service without undoing the change.

Mail	When	Recipients	Subject (in English)
Rights change	A user’s role in a tenant, or their services and features, changed	The user	“Your rights have changed – workspace”
Tenant status	The tenant is suspended, scheduled for deletion, opened again,	The tenant’s members with an e-mail address; when it opens again,	“Access suspended”, “Workspace being deleted”, “Access re-

Mail	When	Recipients	Subject (in English)
	or its deletion is cancelled	not those whose own access stays suspended	stored", "Deletion cancelled"
Contract end	The tenant's validity date is near or past	MLA, and the tenant's administrators	"Contract ending in n days", "Contract ending today", "Contract ended"
Support access	A support access request is made, authorized or declined	The user concerned for a request; the requester for an authorization or a decline	"Support access request", "Support access authorized", "Support access request declined"

The rights-change, status and contract-end mails are in the language the user chose, or in French then English when they chose none; MLA gets its own in the language configured for it. The support access mails are in French or English, depending on the recipient's language. A temporary tenant sends no rights-change mail.

Contract-end reminders

Every day (at 08:00 Paris time by default), subscription looks for the active tenants whose validity date is near or has just passed, and sends:

- to MLA, a single mail to all its addresses, 60, 30 and 7 days before the end (by default);
- to each administrator of the tenant who still has access and an e-mail address, 30 and 7 days before the end (by default);
- to both, a mail once the date has passed, within the following 7 days (by default).

A reminder already sent is never sent again; a reminder whose sending fails is tried again at the next run. A tenant without a validity date is never concerned.

> **Note**
>
> These reminders are not set in the console. The time, the thresholds, MLA's addresses and the language of MLA's mails are configuration properties of the service (`application.contract-reminders.*`). The contracts nearing their end, and the reminders sent for each one, are shown from the tenants (see Tenants).

> **Warning**
>
> Without subscription's dedicated Kafka credentials (`application.notification.kafka-username` and `kafka-password`), none of these mails goes out; the service says so once at startup. See Events and calls.

Library

The library is in the **Services Configuration › Library** menu.

The **Common documents** page manages the common library of the `library` service: “Regulations and guidelines of the common library, which every tenant reads and the chat searches.”. The tenants’ own documents are not listed there.

Documents reach the common library in two ways: those the crawler finds at the regulators, and those an administrator adds here.

Like the whole console, the page is for platform administrators only; the library and subscription themselves check that the caller is one.

The page has three tabs: **Documents**, **Upload** and **Search index**.

A document’s path

A library document goes through these steps, which the **Status** column of the **Documents** tab shows:

Status	What happens
Queued	The document waits its turn in docling’s queue.
Converting	docling converts it to Markdown.
Markdown to validate	docling produced a Markdown, which an administrator must read and validate.
No Markdown	The conversion produced no Markdown.
Embedding	The library cuts the document into passages and embeds them for the chat.
Embedded	The chat can find the document by meaning.
Embedding failed	The embedding failed.

A document is embedded for the chat only once its Markdown is validated, or, when docling produced none, once the embedding of its plain text is asked for. docling’s queue itself is followed in the crawler’s Docling tab.

Documents

The **Documents** tab shows the chat’s embeddings at the top, then the list of common documents.

Embeddings for the chat

“Each document is cut into passages and embedded, so the chat can find them by meaning. Library embeds only on request, each embedding costing model tokens.”

The block counts the documents embedded out of the total, in progress, failed, not embedded yet, and those cut with an older passage size. It lists the failures, with their error.

Three buttons start an embedding in the background, each after a confirmation that recalls the token cost:

- **Embed n documents**: those never embedded or whose embedding failed;
- **Re-embed the n outdated documents** (shown when there are any): those cut with an older passage size, cut again at the current size;
- **Re-embed all common documents**.

These buttons are only active once the library has said no embedding is in progress, and when embeddings are on in the environment; otherwise the block reads “Embeddings are off in this environment.”. During an embedding, a progress bar shows.

The list of documents

The filters are:

- **Filter by title or file name**;
- the status (**Any status** by default);
- **To validate (n)**: the documents whose Markdown awaits a validation;
- **Older passages (n)** (shown when there are any): the documents embedded with an older passage size.

The columns are **Title**, **File**, **Published**, **Status** and **Chat**; the first four sort the list. The **Chat** column shows **Embedded**, **Not embedded**, or **Embedded · older passages**, the tooltip then giving the original passage size.

After ticking rows:

- **Delete selected (n)**;
- **Validate selected (n)**: only the documents whose Markdown awaits a validation are sent; the others are left out, and the message says so.

Relaunch docling on documents without Markdown (n) queues every **No Markdown** document again, after a confirmation: docling converts them one after the other, which can take a long time, and files it cannot read are left out.

The list updates live when a common document changes, and is read again every 30 seconds while documents are queued, converting or embedding.

Actions on a document

Each row offers, depending on the document’s status:

- **Preview**: opens the Markdown docling produced, rendered or as source (**Show the Markdown source** / **Show the rendering**). If the conversion is still running, or produced nothing, the preview says so. The bottom of the preview offers **Relaunch docling**, **Embed the plain text** and **Validate**;
- **Validate** (status **Markdown to validate**): the library embeds the document from its Markdown;
- **Embed the plain text** (status **No Markdown**): the library embeds the file’s plain text;
- **Relaunch**, a menu of three actions:
  - **Relaunch docling**: not possible while the document is queued or converting. The current Markdown is replaced, and the new one must be validated; a confirmation says so, except for a document without Markdown;
  - **Relaunch indexing**: the library reads the file again and puts it back in the search index; not possible while the document is queued, converting or embedding;
  - **Relaunch embedding**: for an **Embedded** or **Embedding failed** document;
- **Delete**.

> ⚠ Warning
>
> Deleting cannot be undone and affects every tenant: “Delete « name » from the common library? Every tenant loses it, and it cannot be undone.”. The document leaves the library and the chat.

Upload

The **Upload** tab adds documents to the common library: “Library indexes them in the background.”.

Upload files

Drag files into the area, or click **browse your computer**. Accepted: `.pdf`, `.doc`, `.docx`, `.odt`, `.txt`, up to 50 MB each, each name once.

Each file shows with its **Format** and its **Status**: **Ready**, **Unsupported format**, **Too large** or **Duplicate**. Refused files are not sent; a bin removes a file from the list.

Index from the web

The **Document URLs, one per line** field takes the addresses of pages or PDFs; the library downloads each one and indexes it. Blank lines are ignored; a line that is not an `http(s)` URL (**Not an http(s) URL**) or repeats another (**Duplicate**) is left out.

Send

The bottom bar counts the files and URLs ready; **Send to the common library (n)** sends them. A progress bar follows each file.

Then the page says how many documents went, and offers **Add more documents** and **Go to the crawler**. Failed sends stay listed, with their reason, and **Retry** sends them again; the others are already sent.

> **i** Note
>
> Files go straight to the library: a file marked **Sent** is in its hands. URLs go through subscription, which hands them to the library over Kafka: a URL sent is in the message broker's hands, and the library downloads it afterwards. See Events and calls.

Search index

"If library searches miss documents the library holds, rebuild its search index from the database. It covers the common and the custom documents."

Rebuild the search index asks for a confirmation: "Library re-reads every common and custom document. It can take several minutes, and searches may miss documents until it finishes.".

During the rebuild, the page shows "Rebuilding: counting the documents…", then how many documents are done out of the total, with a progress bar. Once finished, it gives the date of the last rebuild and how many documents were indexed, or why it failed. The library refuses a second rebuild while one runs.

Reference data

Reference data is in the **Services Configuration › Reference Data** menu.

The **Cubes** page manages the reference data of the `referential` service: “Reference data every tenant reads in RegLab: benchmark statistics from the ECB, the EBA and other sources.”. This data is held in cubes: a cube is an SDMX dataflow (agency, id, version), with its structure (its dimensions), its code lists and its observations.

Like the whole console, the page is for platform administrators only.

The page has two tabs: **Cubes** and **Sources**.

Cubes

Import a cube

Drop a bundle on the **Import a cube** area, or click **choose a zip**.

A bundle is a `.zip` file that holds:

- `structure.json`: the code lists, the data structure and the dataflow;
- `data.csv`: the observations, in SDMX-CSV.

The bundle is imported in one go: all of it, or nothing. A file that is not a zip, or is larger than 512 MB (the most the referential reads), is refused before it is sent.

Once the import succeeds, the page gives the cube it fed, the number of series and observations, the code lists created and extended, and “New cube.” when the cube did not exist. When it fails, nothing changed: the page gives the reason and the faulty lines of the files (the first 50, then how many more).

The list of cubes

Cubes in the referential (n) lists the cubes, sorted by name; **Refresh** reads the list again. The search (**Search a cube, an id or a dimension**) filters the list.

The columns are:

- **Cube**: its name and its id `agency:id(version)`;
- **Dimensions**: those of its structure, in series-key order;
- **Agency**.

Each row offers:

- **Re-import**: pick a new bundle; the page asks “Replace cube by file? Its data, structure and code lists become the bundle’s (code lists another cube shares only gain codes). All of it, or nothing.”, then **Replace**;
- **Delete**: the page asks “Delete cube and all its data? Its structure and code lists go too, unless another cube shares them. A loader source feeding it reloads it in full at its next update.”, then **Delete**.

Sources

The **Sources** tab shows the cubes the referential’s loader keeps up to date on its own: “the ECB every day; the EBA Risk Dashboard, Pillar 3 Data Hub, transparency exercise and stress test, the ECB list of MFIs and Banque de France Webstat every week.”.

The list of sources

The header offers **Refresh**, **Follow a cube** and **Update all now** (disabled while an update runs).

The columns are:

- **Cube**: the source, the id followed, **Paused** for an entry switched off, and what the entry loads (series key and start period for the ECB, template, file or dataset for the others);

- **Last update**: its status (**Running**, **Succeeded**, **Failed**), its date, its duration and its trigger (**scheduled** or **by hand**), the error message of a failure, and the date of the last success when the last update did not succeed; "Not updated yet" otherwise;
- **Loaded**: the number of series and observations of the last successful update, and the file loaded.

Source	What an entry loads
ECB	A cube of the ECB Data Portal, by its series key.
EBA annex	The EBA Risk Dashboard: one cube per sheet of the annex, or the credit risk parameters.
EBA Pillar 3	One Pillar 3 template; paused until the licence of the institutions' disclosures is cleared.
EBA transparency exercise	One file of the EU-wide transparency exercise.
EBA stress test	One file of the EU-wide stress test.
ECB list of MFIs	The whole list of monetary financial institutions.
Banque de France Webstat	One Webstat dataset; paused until a Webstat API key is set.

Each row offers:

- **Past updates**: unfolds the entry's update history;
- **Load an annex file (xlsx)**: for **EBA annex** entries only; an `.xlsx` workbook of at most 64 MB;
- **Change** (see below);
- **Stop following**: for an **ECB** entry only, after a confirmation: "Stop updating cube? The data already loaded stays in the referential.";
- **Update now**: starts the update of this entry alone.

The load, stop and update buttons are disabled while the entry is being updated. While an update runs, the list is read again every 3 seconds.

Follow or change a cube

Follow a cube opens **Follow an ECB cube**. Only ECB entries are created here; the others ship with the referential.

Field	Required	Rule
Agency	yes	`ECB` by default; an SDMX agency id.
Dataflow	yes	The dataflow id, e.g. `SUP`.
Version	no	Empty: the latest. Otherwise numbers separated by dots, e.g. `1.0`.
Series key	yes	Codes joined by `+`, dimensions by a dot, an empty part for any code, e.g. `Q.FR+DE.W0.....`
From period (optional)	no	E.g. `2015-Q1`.
Updated every day	—	Switched off, the entry is paused.

An invalid field is highlighted ("Check the highlighted fields.") before anything is sent.

Change opens the same form. For an ECB entry, every field can change; for the others, only the update switch (**Updated every week**) can, and the form explains what the entry loads.

IV Supervision

DPM Kafka work

DPM Kafka work is in the **Supervision › DPM Kafka work** menu.

The `dpm` service gets its work over Kafka: each of its listeners reads a topic with one or more consumer threads. The page shows, live, "what each of dpm's Kafka consumer threads is doing right now, for which tenant, and the messages waiting behind it". It shows whether one tenant keeps another waiting, or whether a message is stuck.

The page changes nothing: it reads a snapshot dpm serves (see Events and calls). Like the whole console, it is for platform administrators only, and dpm checks it itself: when it refuses access, the page shows **Not allowed** ("dpm shows its Kafka work to platform admins only.").

Live reading

The page reads dpm every 5 seconds while the browser tab is visible. The header shows:

- **Live · every 5 s**;
- **Paused while the tab is hidden**: no read while the tab is in the background;
- **Stale**: the last read failed; the page then shows the last snapshot received, with the time of the failure;
- **Stopped**: dpm refused access; reads stop until **Refresh** is clicked.

The snapshot's time ("dpm snapshot at …") shows next to it; **Refresh** reads at once.

When dpm does not answer, the page says why: "This dpm does not serve its Kafka work yet.", "dpm could not be reached." or "dpm answered with an error (…).".

Totals

Three tiles sum up the snapshot:

- **Busy threads**: the busy threads out of the total, across every listener;
- **Tenants served**: the tenants a thread is working for right now;
- **Waiting messages**: the messages waiting, and on how many partitions.

Listeners

Each listener has its block, titled with its topic and its id, with its number of threads, busy threads and waiting messages. A coloured strip shows its threads: each tenant has its own hue, the same for its threads and for the partitions it holds. When dpm could not read a listener's lag, the block shows "Lag unavailable: …".

The threads table has these columns:

- **Thread**;
- **State**: **Busy** or **Idle**;
- **Tenant**: the tenant's name, read from subscription's list of tenants, or its id when it is not there;
- **Message**: the type of the message being handled;
- **Task**: the start of the task id; a button copies the whole id;
- **Stage**: where the handling stands;
- **Elapsed**: how long the thread has been on this message; it is flagged past 1 minute, then as stuck past 5 minutes;
- **Partition**: the partition and offset of the message.

An idle thread shows "Waiting for a message" and the partitions assigned to it.

Partitions

Under the table, each partition of the topic shows:

- the number of messages waiting;
- the committed offset and the end offset;
- who holds it: “queued behind tenant on thread” when messages are waiting, “thread on tenant” otherwise, or “No message being handled”.

> **Tip**
>
> The messages waiting on a partition are queued behind the message the thread holding it is handling. A partition with waiting messages, held by a thread whose **Elapsed** is flagged, therefore shows a long message holding others back, possibly other tenants’ ones.

V How it works

Identities and realms

This page explains, for technical readers, how subscription and Keycloak share identities: the realms, what each holds, and how a user is created, disabled and deleted. The screens are described in Users.

Realms

A Keycloak **realm** is a directory of accounts, with its own sign-in rules. The platform accepts several:

- `security.jwt.allowed-realms`: the realms whose tokens the services accept. They are the only realms a user or a client can be given; `GET /api/v1/realms` gives them to the screens, with the default realm.
- `application.identity.default-realm` (`mla` by default): the realm of a user created without one, and of a membership lookup that names none. It must be one of the accepted realms, or subscription does not start.
- `security.jwt.platform-realms` (`[mla, shared]` by default, a security-library setting): the realms whose tokens may be a platform administrator's or a service's. Administrators come from `mla`, service accounts from `shared`.

Who opens the console

The console signs in through the gateway, on the `mla` realm by default. It reads the signed-in account from the gateway (`/api/account`) and opens its pages only to an account carrying the `ROLE_ADMIN` authority; any other account sees the **Access restricted** page ("Subscription is reserved for MLAdvisory administrators, and your account does not have that role."), which offers only **Log out**.

The server makes its own check: a platform administrator is a token issued by a platform realm and carrying exactly the `ROLE_ADMIN` realm role (`ReglabAuthorities.isPlatformAdmin`). Every subscription endpoint is reserved to them, except those listed in Rights and Security: the services' lookups, the public read of a client by its subdomain, the signed-in user's language, and so on.

The **Admin** role a user holds on a tenant is subscription data, not a Keycloak role: it gives no access to the console.

What subscription holds, what Keycloak holds

subscription holds (`reglab_user` and the accesses)	Keycloak holds (the realm's account)
The account's realm and username	The account itself, its password and how it signs in
First name, last name, e-mail	Whether the account is enabled
The home client, **Multi-client**, the guest clients	The open sessions
The mail language (the `language` property)	The realm roles (`ROLE_ADMIN`, `SUPPORT`…)
When the user was disabled (`disabledSince`)	The login events
The last login seen (`lastLoginAt`), copied from Keycloak	
The accesses: role and plans on each tenant, suspension	

subscription never creates nor deletes a Keycloak account, and touches neither its password nor its roles. It does one thing to an account: disable it (closing its sessions) or enable it again (see Deactivation). It also reads the realms' logins (see Last login).

A user: a realm and a username

A username is unique only inside its realm: a user is the pair (realm, username). The `reglab_user` table has a uniqueness constraint on that pair. The e-mail, for its part, is unique across the whole platform.

A token is tied to its user by its `preferred_username` and by its issuer's realm (`iss`, `…/realms/<realm>`). The services pass that realm to subscription in their membership lookups; a service that passes none gets the default realm.

A client's realm

A client may have a realm (`client.realm`) and a subdomain (`client.subdomain`). Together they say where its users sign in:

- A client's front answers on `<subdomain>.irrc-<environment>.…`. Before anyone signs in, it reads `GET /api/v1/public/clients/by-subdomain/{subdomain}`, open to all, which gives the realm to sign in to, the client's name and whether it has a logo. That read answers only for a client with a subdomain and a realm still accepted (404 otherwise); caches may keep it 5 minutes.
- A client's realm has no default: empty, it stays unset.
- The accounts of the users of a client with a realm must be in that realm. subscription refuses to create, correct or import a user in another realm, and refuses to give a client a realm while some of its users are elsewhere (`error.client.usersOutsideRealm`). Guest clients do not count: only the home client decides.
- A realm is checked only when written. Taking a realm out of `security.jwt.allowed-realms` leaves its clients pointing at it: at startup, subscription logs the clients whose realm is no longer accepted (`ClientRealmCheck`). The public read no longer finds them, and their users cannot sign in until the realm is accepted again or the client gets another one.

Creation and import

Creating a user (`POST /api/v1/reglab-users`) or importing users (`POST /api/v1/reglab-users/import`) creates their record in subscription, and nothing in Keycloak.

The user's realm is, in that order: the one the request names, else their client's, else the default realm. It must be accepted (400 otherwise) and, when the client has one, be the client's. The creation screen and the import offer the default realm already chosen: for a client with its own realm, that realm must be chosen instead.

When editing, the realm can be corrected, for an account registered under the wrong realm, but never to a refused realm. The screen does not let the home client change after creation.

Deactivation

`PUT /api/v1/reglab-users/{id}/deactivation` disables a user; `DELETE` on the same address enables them again. In one transaction, disabling:

1. sets `disabledSince`: from then on, the membership lookups find the user in no tenant;
2. keeps their roles and plans, which enabling gives back as they were;
3. sends irrc `UPDATE_PLAN` for each of their tenants;
4. records the **Account disabled** event in the access history;
5. adds a Keycloak account task `DISABLE` (`ENABLE` when enabling) to `keycloak_account_task`.

Disabling a disabled user, or enabling an active one, does nothing.

The task is carried out by `KeycloakAccountJob`, every minute:

- An account has one task at most: a new one replaces the pending one. Disabling then enabling before the task runs leaves only the enabling.
- The task looks for the account in the user's realm by username, else by e-mail.
- `DISABLE` disables the account and closes its open sessions; `ENABLE` enables it again. The account is never deleted.
- With no matching account (the user never signed in), the task counts as done.
- Two matching accounts make the task fail: it waits for a person to sort out the duplicate.
- A failed task keeps its error and is retried after a wait that doubles at each failure, from one minute up to `application.keycloak-accounts.max-retry-delay` (6 hours).

- Each task is handled in its own transaction, under a `FOR UPDATE SKIP LOCKED` lock: a single replica handles it.

The pending and failing tasks show at the top of the user list (`GET /api/v1/reglab-users/account-tasks`, see Keycloak account changes).

subscription calls Keycloak's admin API with, in each realm, a confidential client `subscription-provisioning` (`application.keycloak-accounts.client-id`) whose service account holds `realm-management`'s `view-users` and `manage-users` roles. Its secret, per realm, comes from Vault (`application.keycloak-accounts.client-secrets`). A realm without a secret leaves its tasks failing. Without `application.keycloak-accounts.server-url`, the job does nothing (one warning at startup) and the tasks wait.

Deletion

`DELETE /api/v1/reglab-users/{id}` is refused (403, `error.user.hasAccesses`) while the user has an access to a tenant. Otherwise, in one transaction, the deletion:

- adds a `DISABLE` task for their Keycloak account, which is disabled but not deleted;
- records **Account disabled**, with the cause "because the user was deleted", unless they were disabled already;
- deletes their record.

Last login

`LastLoginJob` reads, every 15 minutes (`application.keycloak-accounts.login-cron`), the `LOGIN` events of each realm with a configured secret, since its last read. A realm's first read goes back `login-history` (90 days), within what Keycloak keeps.

- The user is found by realm and username, else by e-mail, ignoring case.
- A login opened by a support impersonation is no login of the user's: it is skipped.
- `lastLoginAt` only ever moves forward.

The realm must save its events (Realm settings › Events › Save events) and the provisioning client's service account must hold `view-events`. "Never" in the screens therefore means "no login seen since recording started", not necessarily "never signed in".

The services' membership lookups

The services ask subscription for a user's accesses:

- `GET /api/v1/user-tenants/{service}/{username}?realm=`: their accesses for a service. An end user reads only their own, in their token's realm; the services of the `user-tenants` group and platform administrators read anyone's, in the realm asked for or the default realm.
- `GET /api/v1/user-tenants/{tenantId}/{service}/{username}?realm=`: their access to one tenant for a service, which the services check before serving the user.

These lookups return only the accesses in force: see What gives access to a tenant. The rights the plans carry are described in Rights.

Every user, whoever they are, also reads and changes their own mail language (`GET` and `PUT /api/v1/reglab-users/me/language`, `fr` or `en`).

Settings

Property	Default	Role
`security.jwt.allowed-realms`		Accepted realms.
`application.identity.default-realm`	`mla`	Default realm.

Property	Default	Role
`application.keycloak-accounts.server-url`		Keycloak's address; without it, no task is carried out.
`application.keycloak-accounts.client-id`	`subscription-provisioning`	Provisioning client, the same in every realm.
`application.keycloak-accounts.client-secrets`		The client's secret, per realm (Vault).
`application.keycloak-accounts.cron`	every minute	The account tasks' runs; `-` disables them.
`application.keycloak-accounts.max-retry-delay`	6 hours	Longest wait between two attempts of a task.
`application.keycloak-accounts.timeout`	10 seconds	Longest a call to Keycloak may take.
`application.keycloak-accounts.login-cron`	every 15 minutes	Reading the logins; `-` disables it.
`application.keycloak-accounts.login-history`	90 days	How far back a realm's first read of logins goes.

From plans to rights

This page explains, for technical readers, how the plans a user holds in a tenant become rights in the platform's other services: the question those services ask subscription, the token they ask it with, how they remember the answer, and the `UPDATE_PLAN` event that tells them to forget it.

The principle

subscription does not push rights to the services: each service asks for them.

1. A user calls a service (CIRCÉ, the assistant…) with their Keycloak token and names the tenant in the `X-TENANT-ID` header.
2. The service asks subscription for this user's access, in this tenant, for this service: `GET /api/v1/user-tenants/{tenantId}/{serviceName}/{userName}?realm=…`, with its own service token.
3. subscription answers the plan the user holds for this service, with its features, and their role in the tenant; or `404` if they hold none that grants access.
4. The service decides: the user is a member if the plan carries the expected feature (`base` for CIRCÉ, `assistant` for the assistant); each screen or action then checks its own feature.
5. The service remembers the answer for a few minutes. When a right changes, subscription publishes `UPDATE_PLAN` so that the services forget it.

What the lookup answers

The `GET /api/v1/user-tenants/{tenantId}/{serviceName}/{userName}` endpoint looks up the user's access to the tenant for the service:

- `{tenantId}` is the tenant's public id (`tenantId`), the same as in the `X-TENANT-ID` header;
- `{serviceName}` is the service's name, compared ignoring case (`irrc`, `stress`, `library`…);
- `{userName}` is the user's login, and the `realm` parameter the realm that issued their token. Without `realm`, subscription takes the default realm (`application.identity.default-realm`); a realm subscription does not accept is a `400` error.

subscription answers an access only when all these conditions hold:

Condition	Otherwise
The user holds a plan of the service in the tenant.	`404`
Their access to the tenant is not suspended.	`404`
Their account is not disabled.	`404`
The tenant is `ACTIVE`.	`404`
Now is within the tenant's validity window (start included, end excluded).	`404`

The validity is compared at each lookup: a contract that reaches its end closes access with no action and no message.

The answer holds the access's id, the user, the tenant (without its status or dates), the **plan** with its **features**, and the user's **role** in the tenant (`MEMBER` or `ADMIN`). Since a user holds at most one plan per service in a tenant, there is at most one answer.

A second endpoint, `GET /api/v1/user-tenants/{serviceName}/{userName}?realm=…`, returns the user's accesses to the service in every tenant that grants access, under the same conditions (`404` when there is none). CIRCÉ uses it to list the tenants a user may name.

Who may ask subscription

The services ask subscription with their own token: a *client credentials* token issued by a platform realm (`mla` or `shared` by default, `security.jwt.platform-realms`). A token issued by a client's realm is never a service token.

Endpoint	Callers admitted
`GET /api/v1/user-tenants/{tenantId}/{serviceName}/{userName}`	platform administrators; the services of the `user-tenants` group
`GET /api/v1/user-tenants/{serviceName}/{userName}`	the same, and the user themself, for their own name and in their own realm

A service belongs to the `user-tenants` group when the Keycloak client of its token is listed in `security.service-callers.groups.user-tenants`. While `security.service-callers.legacy-shared-client` is `true` (the default), the shared `internal_service_client` is admitted too.

On the services' side:

- CIRCÉ (`irrc`) calls subscription through a Feign client named `subscription`, found through service discovery;
- the assistant calls subscription with security-library's `ServiceTokenClient` (`application.s2s.*`), whose credentials come from Vault's shared application secret.

The X-TENANT-ID header and tenant-web-library

A call to a multi-tenant service names its tenant in the `X-TENANT-ID` header, which carries the tenant's public id. The shared tenant-web-library provides the filter that reads it and checks the caller's membership; the assistant uses it. CIRCÉ has its own filter, which follows the same rules (see CIRCÉ).

tenant-web-library's filter (`TenantBindingFilter`) runs after the Spring Security chain, on the covered paths (`/api/**` by default, minus the excluded paths):

1. It reads `X-TENANT-ID`. A missing or blank header is refused with a `400` (`error.tenant.missing`), and so is a value that is not a canonical UUID (`error.tenant.invalid`). In single-tenant mode, the header is ignored and the configured tenant is used.
2. It asks the service whether the caller is a member of the tenant (`TenantMembershipVerifier`, which each service provides). An anonymous caller or a non-member gets a `403` (`error.tenant.forbidden`), the same for an unknown tenant and a forbidden one. A lookup that fails gives a `500` (`error.tenant.lookup-failed`) and lets nothing through. A caller the service declares trusted (`TrustedCallerPolicy`) goes through unchecked; by default, nobody.
3. It runs the request in the tenant, which it also puts in the logging context.

On a path declared with an optional tenant (`optional-tenant-paths`), a request without the header runs with no tenant.

The membership cache

The filter keeps the answers in a cache (`TenantMembershipCache`), by tenant, realm and user name:

Property	Default	Purpose
`reglab.tenant.membership-cache.positive-ttl`	`5m`	How long a membership is trusted: what bounds the access a removed user keeps if the eviction is lost.
`reglab.tenant.membership-cache.negative-ttl`	`30s`	How long a refusal is trusted: what a user who has just been given access may have to wait.
`reglab.tenant.membership-cache.maximum-size`	`10000`	The maximum number of answers kept.

Concurrent calls for the same key wait for a single lookup; an error is never kept. The service empties the cache with `evict(tenant)`, `evict(tenant, callerName)` or `evictAll()`, in particular when it receives `UPDATE_PLAN`.

CIRCÉ (irrc)

CIRCÉ asks for the accesses of the `irrc` service.

- **Membership.** A user is a member of a tenant if subscription answers them an `irrc` plan that carries the `base` feature.
- **Tenant admin.** The `ADMIN` role of the answer makes the user the administrator of that tenant in CIRCÉ, whatever their role in their other tenants.
- **Features.** The screens tied to an offer also ask for a feature of the plan; a member whose plan lacks it gets a `403` (see The feature names that are read).
- **Cache.** CIRCÉ keeps each answer 5 minutes, and a refusal 30 seconds; these durations are set in its code (`CacheConfiguration`). Each instance has its own cache.
- **Service accounts.** The services of the `internal` group (on the `/api/v1/internal/**` routes only) go through without a lookup.

The assistant

The assistant opens to a user in a tenant if their plan, in that tenant, of one of the services of `application.subscription.plan-services` (`irrc`, `stress` and `library` by default) carries the `assistant` feature (compared ignoring case). It offers an application's tools only if that application's plan opens the assistant.

- The assistant asks for each of these services, with the realm of the user's token; a token that names no realm makes no member.
- It keeps the features of each plan, by tenant, realm, user and service: for `membership-cache-ttl` (one hour by default) when they open the assistant, for tenant-web-library's `negative-ttl` (30 seconds by default) otherwise.
- tenant-web-library's filter keeps its own answer too: a membership for `membership-cache-ttl`, a refusal for the `negative-ttl`.
- No caller is trusted: the assistant always acts for a user.

The UPDATE_PLAN event

When a right changes, subscription publishes a message on the Kafka topic `subscription_irrc_topic`:

```
{"tenantId": "<public id of the tenant>", "messageType": "UPDATE_PLAN",
"schemaVersion": 1}
```

- The message names a tenant, never a user; its Kafka key is the `tenantId`, which keeps one tenant's messages in order. The fields are read by name: they are part of the contract between services.
- It leaves inside the change's transaction, once the change is written, and subscription waits for the broker's acknowledgement (timeout `application.kafka.send-timeout`). A failed send rolls the change back and fails the request: a right taken away is never recorded without the services being told.
- The message leaves just before the transaction commits: a lookup made in those few milliseconds still reads the old right and keeps it until it expires.

On receipt:

- **CIRCÉ** (`MembershipConsumer`) forgets the memberships and refusals kept for that tenant; a message without a readable tenant empties the whole cache. Each instance reads every message, in a consumer group of its own (`irrc-membership-<uuid>`).
- **The assistant** (`MembershipConsumer`) forgets the plans kept for that tenant, then the filter's memberships; a message without a readable tenant empties both caches. Each instance has its own group (`assistant-membership-<uuid>`) and starts from the latest message; an unreadable message goes to `subscription_irrc_topic.DLT`.

An instance without a broker, or one that has not joined its group yet, reads nothing: only the caches' expiry then applies.

When subscription sends UPDATE_PLAN

Change	Where it comes from in the console
A user's role and plans in a tenant are replaced (`PUT /api/v1/tenants/{id}/members/{userId}`)	the form of a user's access to a tenant
An access's plan or role changes (`PUT /api/v1/user-tenants/{id}`)	—
A plan is taken away from a user (`DELETE /api/v1/user-tenants/{id}`); the last plan takes the user out of the tenant	—
A user is taken out of a tenant (`DELETE /api/v1/tenants/{id}/members/{userId}`)	**Remove from the tenant**
A user stops being a guest of a client: they are taken out of every tenant of that client	a user's **Guest clients** card
A user's access to a tenant is suspended or reactivated	**Suspend access**, **Reactivate access**
A user is disabled or enabled: one message per tenant where they hold a plan	Users
A tenant's status or validity changes: suspension, reactivation, validity window, scheduled deletion, restore, purge	Client and Tenants

On a purge, subscription also publishes `TENANT_DELETED` on `tenant_lifecycle_topic` (see The tenant lifecycle).

Nothing is sent when the change changes nothing: suspending an access already suspended, reactivating an active access, taking out of a tenant a user who holds nothing there.

What subscription does not announce

In these cases, no `UPDATE_PLAN` leaves: only the expiry of the services' caches applies.

Change	Effect in the services
A plan's definition changes: its features, its service, its name (Plans page)	The services apply the former definition until their cache expires: up to 5 minutes for CIRCÉ; for the assistant, `assistant` taken off a plan stays open up to `membership-cache-ttl` (one hour by default). The plan's holders receive an e-mail.
A feature is renamed (Features page)	As for a changed plan.
An access is created on its own (`POST /api/v1/user-tenants`)	The right gained waits for the kept refusal to expire: up to 30 seconds. The console's access form goes through the replacement, which sends the message.

> **Note**
>
> A right taken away through the console is announced; a right gained that is not announced only waits for a refusal to expire (30 seconds). A change to the catalogue (plans, features) is what can leave a removed right active the longest.

The delays in short

Situation	CIRCÉ	Assistant
Right taken away, `UPDATE_PLAN` received	at the next request	at the next request
Right taken away, `UPDATE_PLAN` lost	5 minutes at most	`membership-cache-ttl` at most (one hour by default)
Right given after a kept refusal, no `UPDATE_PLAN`	30 seconds at most	`negative-ttl` at most (30 seconds by default)
Tenant's contract ended	once the kept answer expires (5 minutes at most)	once the kept answer expires

CIRCÉ's durations are set in its code; the assistant's are properties, served by Consul.

The feature names that are read

A feature opens something only if a service reads its name. The names read today:

Feature	Plan of the service	What it opens
`base`	any service	Membership of the service; in CIRCÉ, access to the tenant.
`anacredit`	`irrc`	CIRCÉ's AnaCredit screens.
`pilar3`	`irrc`	CIRCÉ's Pillar 3 screens.
`ste`	`irrc`	The ECB STE packages (SREP) and their controls.
`stress`	`irrc`	CIRCÉ's stress tests.
`generate-dpm`	`irrc`	Generating XBRL-CSV packages from a workbook.
`mcp`	`irrc`	CIRCÉ's MCP tools for an MCP client.
`assistant`	`irrc`	CIRCÉ's MCP tools when the assistant calls for the user.
`assistant`	`irrc`, `stress` or `library`	The assistant itself, and that application's tools in the assistant.

CIRCÉ compares the names as they are; the assistant compares them ignoring case.

A tenant's lifecycle

This page explains, for technical readers, a tenant's life in subscription, from its creation to its purge, and what each service does about it. The screens are described in Clients and tenants and Purge.

Statuses

A tenant (`Tenant`, table `tenant`) has a status, `TenantStatus`:

Status	Meaning	Access
`ACTIVE`	The normal status.	Yes, inside the validity window.
`SUSPENDED`	Suspended by a platform administrator: everything is kept.	No, until reactivated.
`PENDING_DELETION`	Deletion requested: everything is kept, the tenant can be restored until its purge date.	No.
`DELETED`	Purged: members removed; the row and the login credential are kept, the name and id stay reserved. Final.	No.

The transitions:

From	Action	To	Endpoint
`ACTIVE`	Suspend	`SUSPENDED`	`PUT /api/v1/tenants/{id}/suspension`
`SUSPENDED`	Reactivate	`ACTIVE`	`DELETE /api/v1/tenants/{id}/suspension`
`ACTIVE`, `SUSPENDED`	Schedule deletion	`PENDING_DELETION`	`DELETE /api/v1/tenants/{id}` (202)
`PENDING_DELETION`	Restore	`SUSPENDED` if it was before, else `ACTIVE`	`POST /api/v1/tenants/{id}/restore`
`PENDING_DELETION`	Purge, once the date has passed	`DELETED`	`TenantPurgeJob`

The server answers 409 to an action the status does not allow: suspending or reactivating a `PENDING_DELETION` tenant (`error.tenant.pendingDeletion`), restoring a tenant that is not `PENDING_DELETION` (`error.tenant.notPendingDeletion`), changing anything about a `DELETED` tenant (`error.tenant.deleted`). Suspending a suspended tenant, reactivating an active one, or scheduling the deletion of a tenant already pending changes nothing; in the last case, it keeps its purge date.

The suspension time (`suspendedSince`) survives a deletion request: it is what lets a restore give back `SUSPENDED`.

Validity

Whatever its status, a tenant has a validity window, `validFrom` (included) and `validUntil` (excluded). An empty bound is no limit. `PUT /api/v1/tenants/{id}/validity` replaces the whole window; an end that does not come after the start is refused (400). The validity can be changed in any status but `DELETED`.

The status stays `ACTIVE` when the window is over or not started: it is the access lookup that compares the present moment with the window, at every read. Nothing therefore happens at the moment a contract ends, except that no access is found any more.

The last day of access is the Paris-time day of the instant just before `validUntil`. A `validUntil` on 1 January at 00:00 therefore gives 31 December as the last day.

What gives access to a tenant

The services ask subscription for a user's accesses (see The services' membership lookups). An access is returned only if, at the time of the read:

- the tenant is `ACTIVE`;
- the moment is inside its validity window;
- the user's access to that tenant is not suspended;
- the user is not disabled.

Giving or changing an access is allowed only in an `ACTIVE` tenant (409 `error.tenant.inactive` otherwise); removing an access stays allowed in any status. The validity window is not looked at: an access can be prepared before the contract starts.

Creation

`POST /api/v1/tenants` creates the tenant:

- with a random public id, `tenantId`: the **Reference** of the screens, the one the services store, distinct from subscription's primary key;
- with a unique name, `DELETED` tenants included;
- attached to a client, required;
- `ACTIVE`, with no validity window;
- with its login credential (`TenantCredential`).

The login credential is a password, the tenant's encryption secret. The only type accepted is `PASSWORD`: `KEY` is refused (400), as no service encrypts with a key pair yet. The password may be neither empty nor shorter than `application.credential.minimum-password-length`. subscription adds a random salt to it. The password and the salt are encrypted in the database with the current master key; a restart re-encrypts with the current key the values left on an old one (`TenantCredentialReencryption`).

`GET /api/v1/security/password/generate` provides the generated password the form offers. No screen reads the credential back: only `GET /api/v1/tenant-credentials?tenantId=` serves it, to the services of the `tenant-credentials` group (irrc) and to them alone, never to a user's token, administrators included. Each read is logged with the caller and the tenant, never the secret. It stays readable after the purge.

Suspension and reactivation

Suspension cuts the access of all the tenant's users and removes nothing. Reactivation gives it back. Both tell irrc (see UPDATE_PLAN messages) and mail the members (see Mails to members).

Deletion and waiting period

Scheduling the deletion puts the tenant in `PENDING_DELETION`, whether it has members or not:

- `deletionRequestedAt` gets the moment of the request;
- `deletionScheduledAt` gets the purge date: the request plus the waiting period, `application.tenant-lifecycle.deletion-grace-period`, 60 days by default.

During that period no access is found, everything is kept, and **Restore** gives the tenant back its former status. The restore clears both dates.

> **Note**
>
> The screens say “60 days” as a fixed value, in the confirmation and in the computation of the restore date. If the waiting period is configured otherwise, the date in the confirmation will be wrong; the actual date is the one the tag and the tenant’s page show afterwards, read from the server.

The purge

`TenantPurgeJob` runs every 15 minutes (`application.tenant-lifecycle.purge-cron`). It purges the tenants that are due: `PENDING_DELETION` past their purge date, or test tenants past their lifetime (see Test tenants).

Each tenant is purged in its own transaction, under a `FOR UPDATE SKIP LOCKED` row lock: of several replicas, only one purges a given tenant, and a tenant restored meanwhile is skipped. The purge:

1. deletes all the tenant’s accesses (`user_tenant`) and its members (`tenant_member`);
2. records every right removed in the access history, with the cause “because the tenant was purged” (`TENANT_PURGED`);
3. sets the tenant to `DELETED` with `deletedAt`;
4. sends `UPDATE_PLAN` to irrc;
5. sends `TENANT_DELETED` on `tenant_lifecycle_topic`.

Each send waits for the broker’s acknowledgement. A failed send rolls the tenant’s whole purge back, to be tried again on the next run. The purge mails nobody: the members were told when the deletion was scheduled.

The `DELETED` tenant keeps its row and its login credential; its name and its `tenantId` stay reserved.

The TENANT_DELETED event

`TENANT_DELETED` is published on `tenant_lifecycle_topic`, keyed by `tenantId` so that one tenant’s messages stay in order:

```
{ "tenantId": "12359bb8-b120-11ed-afa1-0242ac120005", "messageType": "TENANT_DELETED",
  "deletedAt": "2026-11-04T03:30:00.123456Z", "schemaVersion": 1 }
```

- `tenantId` is the tenant’s public id, never subscription’s primary key.
- The message is sent once, when the purge commits, but delivered at least once: a consumer must be idempotent.
- A consumer ignores a `messageType` it does not know.
- The message is sent before the purge transaction commits: for a few milliseconds, a service reading the status may still see the tenant not `DELETED`.

A service can check a tenant’s status by its public id with `GET /api/v1/tenant-status?tenantId=`, reserved to the services of the `tenant-status` group and platform administrators. The answer gives `status`, `deletedAt` and `purgeDue` (the tenant would be purged now). An unknown `tenantId` answers 404: the tenant never existed, or it was a test tenant erased after its purge.

What each service purges

Service	What it deletes on `TENANT_DELETED`	Specifics
irrc (MyReg)	The tenant’s objects in S3, then the rows of every table with a `tenant_id` column (read from the catalog, children first), then	One transaction for the rows. A single replica purges; a failure is retried, then put in `tenant_lifecycle_topic.DLT`. No tenant is exempt.

Service	What it deletes on `TENANT_DELETED`	Specifics
	what the replica's caches hold of it.	
library	The tenant's own documents and their files, tasks, summaries, docling conversion markers, its passages in the index, and its users. The common library and the other tenants are untouched.	In batches, each committed on its own. A failure is retried, then `tenant_lifecycle_topic.DLT`.
ALM (stress, risk-app)	Every row and object of the tenant; each replica also drops its local caches of the tenant.	Purges only after checking with subscription that the tenant is `DELETED` (`GET /api/v1/tenant-status`). Never purges its default tenant. Active only when ALM runs with subscription.

Each service purges idempotently: a replayed message finds nothing left. The Purge page asks these three services whether they still hold anything of a deleted tenant, and erases the data of tenants subscription does not know.

UPDATE_PLAN messages

Every status or validity change, like every access change, sends irrc an `UPDATE_PLAN` message on `subscription_irrc_topic`, keyed by `tenantId`. irrc then clears its cache of that tenant's accesses and reads them again from subscription.

The change is written, then the message is sent in the same transaction and its acknowledgement awaited: a failed send rolls the change back and fails the request. A revocation is therefore never committed without irrc being told. An unchanged validity sends nothing.

Mails to members

A tenant's members (a role or a plan, and an e-mail address) get a mail when its status changes:

Event	Mail
Suspension	Access stops.
Deletion scheduled	No access; data purged on the purge date unless restored.
Reactivation, or restore to `ACTIVE`	Access is back. A member whose own access is suspended gets none.
Restore to `SUSPENDED`	The deletion is cancelled; access stays suspended.

A validity change and the purge mail nobody. Each mail is in the user's language, or in French then English when they chose none. The mails leave after the change commits, through the notification service (`SEND_MAIL` on `notification_topic`): a rolled-back change mails nobody, and a failed send is logged without undoing the change.

End-of-contract reminders

`ContractReminderJob` runs every day at 08:00, Paris time (`application.contract-reminders.cron`). It looks only at `ACTIVE` tenants with a `validUntil`: an open-ended tenant is never reminded about.

Recipient	Default thresholds	Setting
MLA (`support.reglab@mladvisory.ca` by default), a single mail to all its addresses, in its language (`fr` by default)	60, 30 and 7 days before the last day	`mla-recipients`, `mla-language`, `mla-thresholds-days`
The tenant's admins: admin role, access not suspended, account not disabled, an e-mail address. Each in their language, or in French then English.	30 and 7 days	`admin-thresholds-days`
Both	End of contract ("ended"), up to 7 days after	`ended-lookback`

A threshold of N days is due from the day N days are left before the last day of access. Of the thresholds due and not handled yet, only the most urgent is sent; the others are recorded as skipped. A late run, or a validity set close to its end, therefore sends a single mail. A contract ended more than 7 days ago is no longer mailed about, so that the first run does not mail about contracts that ended long ago.

Each reminder sent or skipped is recorded (`contract_reminder`: tenant, `validUntil`, threshold, date, number of recipients) and never sent again for the same `validUntil`. Extending a tenant changes its `validUntil`: the reminders start again for the new date, and the old ones stay visible under **Earlier end dates**. Each tenant is handled in its own transaction, under a `FOR UPDATE SKIP LOCKED` lock; a failed mail rolls back the tenant's records, tried again the next day.

The **Contracts ending soon** panel reads `GET /api/v1/tenants/contract-endings?days=60` (the `ACTIVE` tenants whose end falls within 60 days or passed less than 30 days ago); a tenant's **Contract** card reads `GET /api/v1/tenants/{id}/contract-reminders`.

Test tenants

The end-to-end tests create and delete their own tenants through the API. A tenant whose name contains `playwright` (`application.tenant-lifecycle.test-tenant-marker`, ignoring case) is a test tenant, and may get:

- on creation, a lifetime, `expiresAfter` (an ISO-8601 duration such as `PT1H`): the purge purges it when it runs out, with no deletion request. Such a tenant is called temporary;
- on deletion, a notice period shorter than the waiting period, `noticePeriod`.

These durations lie between `minimum-notice-period` (1 hour) and the waiting period; any other tenant asking for one is refused (400).

A temporary tenant leaves no trace: its access changes are neither recorded in the history nor mailed. Two days after its purge (`temporary-tenant-retention`), the purge erases it entirely, row and login credential included. A service receiving its `TENANT_DELETED` later no longer finds it (404); whatever data it may still hold then becomes orphan data, erasable from the Purge page.

Settings

Property	Default	Role
`application.tenant-lifecycle.deletion-grace-period`	60 days	Waiting period before the purge.
`application.tenant-lifecycle.purge-cron`	every 15 minutes	The purge's runs; `-` disables it.
`application.tenant-lifecycle.minimum-notice-period`	1 hour	A test tenant's shortest notice period.

Property	Default	Role
`application.tenant-lifecycle.test-tenant-marker`	`playwright`	What marks a test tenant in its name.
`application.tenant-lifecycle.temporary-tenant-retention`	2 days	Delay before a purged temporary tenant is erased.
`application.contract-reminders.*`	see above	End-of-contract reminders.
`application.kafka.send-timeout`	10 seconds	Longest wait for a Kafka send's acknowledgement.

Support access

This page describes support access to the clients' accounts end to end: who may view whom, how the client authorizes, how a consultation is opened, limited and ended, and what is recorded. It is meant for technical readers. Using the console is described in Support console; the target user's side, in the CIRCÉ manual, "Support access" page.

Overview

A support consultation opens CIRCÉ or HERMES ALM with the application identity of a chosen member, in a single tenant, read only. Support sees the data and can download the exports available to that member; it does not see their browser live.

Five actors take part:

Actor	Role
subscription	holds the policies, the requests and the authorizations; decides whether a consultation is allowed; decides who may end one; holds the journal
the gateway	opens the consultation session, enforces the tenant and read-only mode, re-checks the authorization during the session, reports openings, ends and refusals to subscription
Keycloak	the identities; the gateway gets the target member's session from it; subscription reads the operator's and the target's roles in it
CIRCÉ and ALM	the **Support access** page and the banner where the user answers requests and authorizes
the support console	the subscription-front page from which support and administrators act

The typical path, under the **On request** policy:

1. Support requests access from the console; the user receives an e-mail and a banner.
2. The user authorizes from their own session, for a duration they choose.
3. Support clicks **View as user**; the gateway asks subscription whether it is allowed, then opens a tab signed in with the user's identity.
4. The gateway re-checks the authorization during the session and ends it at the first end reason.
5. Every step is recorded in the support journal.

> ⚠ Warning
>
> **This access bypasses the target user's corporate IdP authentication, their MFA and their conditional-access rules.** Support authenticates with its own account; the gateway then gets a session of the target from Keycloak.

Who may view whom

The operator is one of two people:

- an MLA support member: an account of the `mla` realm, enabled in Keycloak, holding the realm role `SUPPORT` or `ROLE_ADMIN` (effective roles, composites and groups included). They may act in every tenant;

- a tenant administrator: an active member of that tenant with role `ADMIN`. They may act in that tenant only.

subscription reads the operator from Keycloak at every decision, never from the roles the gateway passes on: an account disabled or stripped of its role loses access at the next check.

The target must meet all of these conditions:

- an active member of the tenant: account not disabled, tenant with an active status and within its validity period, a role in the tenant, access not suspended;
- not MLA staff: a user with no client, or whose client has a subdomain listed in `application.support-access.staff-clients` (`mla` by default), is never viewed, whatever their realm;
- in Keycloak: an account found exactly once, enabled, not a service account (`service-account-…`), and holding no privileged role. The list comes from security-library (`ImpersonationTargets`), shared with the gateway: the realm roles `ROLE_ADMIN`, `SUPPORT` and `admin`, and any role of the `realm-management` client. subscription reads the effective roles; the gateway re-checks the tokens issued for the consultation;
- not the operator themself, and not from a consultation already open (no chaining).

The tenant's policy

Each tenant has a support access policy:

Policy	Value	Effect
On request	`ON_REQUEST`	Initial value of every tenant. Each user authorizes support themselves, for a limited time.
Always	`ALWAYS`	No individual authorization is required. The other conditions still apply.
Never	`NEVER`	Every consultation is refused, even when an individual authorization exists.

The policy is changed by a platform administrator or an active administrator of the tenant (`PUT /api/v1/support-access/tenants/{tenantId}/policy`). Each change writes a `SUPPORT_POLICY_CHANGED` event with the policy before and after.

Access requests

A request (`POST /api/v1/support-access/requests`) names the tenant, the target, a suggested duration, an optional reason (2,000 characters at most) and the product to open. subscription accepts it only when the access decision for this operator and this target is `CONSENT_REQUIRED` or `CONSENT_FOR_ANOTHER_REQUESTER`: an authorization is missing. In every other case it answers 403 with the reason (or "Access already authorized").

Rules:

- the suggested duration runs from 15 minutes to `application.support-access.max-grant-duration`;
- the product must have a page in `application.support-access.front-urls`; `{subdomain}` there is replaced by the subdomain of the tenant's client;
- a request lives seven days. A request from the same operator to the same target still pending is returned as it is; an expired one is closed and replaced;
- the operator's e-mail address and language are read from their token, to write to them later.

Once recorded, subscription writes `SUPPORT_ACCESS_REQUESTED` to the journal and, after the transaction commits, sends the target the "Support access request" e-mail, with a link to the product's **Support access** page.

A user sees only the requests addressed to them, and an operator only their own (`GET /api/v1/support-access/requests?tenantId=`).

Declining. The target may decline a request (`POST /api/v1/support-access/requests/{requestId}/decline`, answer 204). Only the target, an active member of the tenant, may (403 otherwise);

an unknown, already resolved or expired request answers 404. The request is closed and marked declined, the journal receives `SUPPORT_ACCESS_DECLINED`, and the operator receives the "Support access request declined" e-mail. They may ask again afterwards.

Authorizations

An authorization is given by the user themself, from their own authenticated session (`POST /api/v1/support-access/tenants/{tenantId}/grant`). A request received, or its e-mail, gives no access.

- **Conditions.** The user is an active member of the tenant, and the policy is **On request** (403 otherwise).
- **Duration.** The user sets the end, between 15 minutes and `application.support-access.max-grant-duration` (30 days by default) from now.
- **Scope.** An authorization given in answer to a request lets in only the operator who made it (their login and realm). An authorization given without a request lets in the whole support.
- **Coexistence.** A user may have several authorizations running: one per operator, plus one for the whole support. A new authorization replaces only the one of the same scope; the replaced one is revoked (`SUPPORT_ACCESS_REVOKED`).
- **Requests resolved.** An authorization given on a request closes that request only; an authorization for the whole support closes every pending request addressed to the user. Each operator whose request is closed receives the "Support access authorized" e-mail, with a link to the console (`application.support-access.console-url`, else the product's page).
- **Revocation. Revoke now** (`DELETE /api/v1/support-access/tenants/{tenantId}/grant`) withdraws all of the user's authorizations in this tenant, with one `SUPPORT_ACCESS_REVOKED` per authorization.

Each authorization writes `SUPPORT_ACCESS_GRANTED`, with its end and the operator it is bound to.

The access decision

The gateway asks subscription whether an operator may view a target (`POST /api/v1/support-access/check`) on opening and during the session. The console and the creation of a request use the same decision. subscription checks in this order and answers with the first reason that applies:

Reason	Cause
`CHAINING`	The call already comes from a consultation.
`INVALID_IDENTITY`	The target or the operator is not named.
`SELF`	The operator is the target.
`TENANT_NOT_FOUND`	Unknown tenant.
`TARGET_NOT_ACTIVE_MEMBER`	The target is not an active member of the tenant.
`TARGET_IS_MLA_STAFF`	The target is MLA staff.
`IDENTITY_PROVIDER_UNAVAILABLE`	Keycloak did not answer, while reading the operator or the target.
`ACTOR_NOT_AUTHORIZED`	The operator is disabled in Keycloak, or is neither an MLA support member nor an active administrator of the tenant.
`TARGET_PRIVILEGED_OR_DISABLED`	The target is disabled, not found, a service account, or holds a privileged role in Keycloak.
`POLICY_NEVER`	The tenant's policy is **Never**.
`ALLOWED`	**Always** policy: allowed, with no end date.
`CONSENT_REQUIRED`	**On request** policy and no authorization running.
`CONSENT_FOR_ANOTHER_REQUESTER`	Authorizations exist, but none for this operator nor for the whole support.

Reason	Cause
`ALLOWED`	An authorization covers the operator: allowed until its end.

The answer carries `allowed`, `until` (the end of the authorization used, empty under **Always**), `reason` and the tenant's name.

The consultation session

Opening. View as user sends the gateway (`POST /api/impersonation`) the tenant, the target and the product's address. The gateway asks subscription for the decision, then returns a single-use entry address (`…/impersonation/enter?ticket=…`); the console opens its tab only on that gateway address. The gateway then opens the target's session with Keycloak, through the `gateway-impersonation` client of the target's realm, and reports the opening to subscription (`IMPERSONATION_STARTED`, with a session reference).

Tokens are kept per gateway web session: a consultation neither replaces nor closes the session the target user opened themself.

Limits during the consultation (gateway):

- the tenant is enforced: a request to another tenant is refused;
- write methods are refused, except an explicit list of viewing or export POST requests: those of `application.impersonation.read-only-requests` and those the services publish themselves in Consul, under `support/read-only-requests/<service>`. `application.impersonation.denied-requests` always wins;
- the authorization is re-checked with subscription at most once every 60 seconds, on a request of the session. This delay cannot be configured: a revocation can therefore take up to 60 seconds to cut an open session;
- the authorization's end, known from each check, is compared on every request; a longer authorization given later does not extend the session;
- when subscription or Keycloak does not answer, each request gets 503 during the grace period; beyond it, the session ends;
- after 5 refused launches in 10 minutes from the same console session, the gateway answers 429 without asking subscription.

The requests refused during the session are counted by kind (read-only, another tenant, nested consultation) and sent with the end.

Gateway settings (Consul):

Property	Default	Role
`application.impersonation.max-session-duration`	`PT4H`	longest consultation
`application.impersonation.unavailable-grace`	`PT5M`	grace period while subscription or Keycloak does not answer
`application.impersonation.sweep-interval`	`PT1M`	interval of the sweep of idle or expired sessions
`application.impersonation.max-refusals`	`5`	refused launches before 429
`application.impersonation.refusal-window`	`PT10M`	window over which refusals are counted
`application.impersonation.published-requests-refresh`	`PT30S`	re-reading of the requests published in Consul
`application.session.valkey.enabled`	`false`	web sessions and consultation state in Valkey, shared between instances

How the services recognise a consultation

The services receive the target's token, obtained by the gateway from Keycloak: they apply the target's permissions and isolation as for the target themself. Nothing in this token grants anything more.

The gateway adds a read-only marker to every request of the consultation, the header `X-SUPPORT-READ-ONLY: true`. This marker only suppresses writes; it grants no access or role:

- CIRCÉ (irrc) creates no profile and does not synchronize the token's information on a read;
- the library likewise disables profile synchronization;
- ALM neither creates nor updates the target's activity record.

Technical logs, audit events and viewing caches may still be fed.

subscription, for its part, refuses any new consultation asked from a consultation: the gateway tells it so in the check (`impersonated`), which gives the reason `CHAINING`.

End reasons

Each session end is an `IMPERSONATION_ENDED` event with its reason:

Reason	Console wording	Meaning
`EXIT`	left by the operator	Support clicked **Exit** in the banner.
`EXPIRED`	authorization expired	The authorization's end has passed.
`REVOKED`	authorization revoked	subscription refused on a re-check: revocation, policy change, target or operator no longer allowed.
`MAX_DURATION`	maximum duration reached	The session reached `max-session-duration`.
`LOGOUT`	signed out	Support signed out, or their own session disappeared.
`IDLE`	session idle	The session went without requests until it expired in the gateway's session store.
`UNAVAILABLE`	service unavailable too long	subscription or Keycloak stayed unreachable beyond the grace period.
`UNKNOWN`	end not reported	The gateway never reported the end; subscription writes it itself (see below).
`TERMINATED`	ended by …	A platform administrator or a tenant administrator ended it from the console.

Orphan sessions. With the in-memory store, a gateway restart loses its sessions, and their end is never reported. Every five minutes (`application.support-access.orphan-session-cron`), subscription writes an end of reason `UNKNOWN`, as `system`, for each opening older than `application.support-access.orphan-session-after` (5 hours by default) left without an end. This end is dated at the opening plus that delay: an upper bound, not the real time. An opening that carries a session reference is closed only by an end with the same reference. The delay must stay longer than the gateway's longest session.

Ending a session

The console's **End** button calls the gateway (`POST /api/impersonation/sessions/{sessionRef}/terminate`). Before closing anything, the gateway asks subscription for permission (`POST /api/v1/support-access/sessions/{sessionRef}/termination`, with the caller's login and realm).

subscription allows it (`ALLOWED`) when the caller is an active administrator of the session's tenant, or an enabled account of the `mla` realm holding `ROLE_ADMIN`, read from Keycloak. It refuses otherwise:

Reason	Cause
`INVALID_IDENTITY`	Caller not named.
`NOT_FOUND`	No opening carries this reference.
`ALREADY_ENDED`	The session already has a recorded end.
`NOT_AUTHORIZED`	The caller does not have this right. A support member may not end another one's session.
`IDENTITY_PROVIDER_UNAVAILABLE`	Keycloak could not tell whether the caller is a platform administrator.

When allowed, the gateway closes the target's Keycloak session, invalidates the consultation's web session and writes the end (`TERMINATED`, with `terminatedBy` and `terminatedByRealm`). The operator's next request gets 401 and their tab shows **Support consultation ended**.

> **i** Note
>
> **Several gateway instances.** With the in-memory session store (the default), a single instance holds a session: an end request received by another instance answers 404. With Valkey (`application.session.valkey.enabled`), every instance sees the same sessions and a single one closes each session.

Events, journal and audit

Support access events are rows of the `access_event` table, the same as the access history:

Event	Written by	When
`SUPPORT_POLICY_CHANGED`	subscription	policy change (before, after)
`SUPPORT_ACCESS_REQUESTED`	subscription	request created
`SUPPORT_ACCESS_GRANTED`	subscription	authorization given
`SUPPORT_ACCESS_REVOKED`	subscription	authorization revoked or replaced
`SUPPORT_ACCESS_DECLINED`	subscription	request declined by the target
`IMPERSONATION_STARTED`	the gateway	consultation opened
`IMPERSONATION_ENDED`	the gateway; subscription for `UNKNOWN`	consultation ended
`IMPERSONATION_REFUSED`	the gateway	opening refused, with subscription's reason

The gateway sends its events to `POST /api/v1/support-access/sessions/events` (`SupportSessionEvent`). subscription checks them before writing them: a type among the three `IMPERSONATION_*`, a known end reason, refused-request counts on an end only, a session reference made of letters, digits, _ and - (64 characters at most), and `terminatedBy` present if and only if the reason is `TERMINATED`. An unknown tenant or target answers 404.

Each row keeps, depending on the event: the date, the real actor and their realm, the target and their realm, the tenant and its client, the operator (on session events only), the end of validity, the application

opened, the end or refusal reason, the refused requests by kind, the session reference, who ended the session, the policies before and after, and the operator an authorization is bound to. Names are copied into the row: changing or deleting an account does not erase them.

- **Refusals deduplicated.** An identical `IMPERSONATION_REFUSED` (same tenant, target, operator and reason) is recorded only once per `application.support-access.refusal-deduplication-window` (5 minutes by default).
- **Append-only.** A database trigger refuses every update and every deletion of `access_event`, except the retention purge.
- **Retention.** Every night (03:40, Paris time, `application.access-event.purge-cron`), the events older than `application.access-event.retention` (90 days by default) are deleted. See also Purge.

The support journal (`GET /api/v1/support-access/journal`) reads only the `SUPPORT_*` and `IMPERSONATION_*` events. For a given tenant it is open to MLA support and to that tenant's administrators; without a tenant, to MLA support alone. It flags as running (`open`) each opening whose session reference has no end. Pages hold 500 rows at most.

The export (`GET /api/v1/support-access/journal/export`) produces `support-access.csv`, UTF-8 with a BOM, with the columns `date`, `event`, `actor`, `actorRealm`, `user`, `userRealm`, `tenant`, `front`, `validUntil`, `endReason`, `refusalReason`, `refusedReadOnly`, `refusedTenant`, `refusedChaining`, `policyBefore`, `policyAfter`, `sessionRef`, `terminatedBy`, `terminatedByRealm`. A cell starting with `=`, `+`, `-`, `@`, a tab or a carriage return is prefixed with an apostrophe, so that a spreadsheet does not run it as a formula.

Live updates and e-mails

After each request created, authorization given, revocation or decline, and once the transaction commits, subscription publishes `SUPPORT_REQUEST_CHANGED` on the Kafka topic `ui_events_topic`, with the tenant's public id, the realm and the login of the target user. A failed send is logged and dropped: the fronts read the state again at regular intervals.

- CIRCÉ (irrc) and ALM (stress) relay the message to their browsers: the user's request banner and badge reload at once.
- subscription also consumes this topic, each instance in a group of its own, and relays it to the console on its `GET /api/v1/events` stream, as `SupportAccessChanged`. Platform administrators receive the tenant's id; the console's other users (support, tenant administrators) receive the event with no field at all, and their console reads the shown tenant again.

The e-mails ("Support access request", "Support access authorized", "Support access request declined") are handed to the notification service as a `SEND_MAIL` message on `notification_topic`, after the transaction commits. An e-mail that cannot be built (product page or client subdomain not configured) is dropped and logged, without undoing the authorization or the decline.

API endpoints

All under `/api/v1/support-access`:

Endpoint	Caller
`GET /tenants`	console user: support sees every tenant, a tenant administrator their own
`GET /tenants/{tenantId}/members`	MLA support, or a tenant administrator
`GET /tenants/{tenantId}`	the tenant's state: MLA support, or an active member
`PUT /tenants/{tenantId}/policy`	platform administrator, or tenant administrator
`GET /tenants/{tenantId}/grants`	MLA support, or tenant administrator
`POST /tenants/{tenantId}/grant`, `DELETE /tenants/{tenantId}/grant`	the user themself, an active member
`POST /requests`, `GET /requests?tenantId=`	operator; target for reading

Endpoint	Caller
`POST /requests/{requestId}/decline`	the request's target
`GET /journal`, `GET /journal/export`	MLA support; a tenant administrator for their tenant
`POST /check`	the gateway alone
`POST /sessions/events`	the gateway alone
`POST /sessions/{sessionRef}/termination`	the gateway alone

"The gateway alone": a token of the `shared` realm, issued to the `svc-gateway` client, a member of the `support-access` caller group (see Security).

subscription settings

Property	Default	Role
`application.support-access.max-grant-duration`	`P30D`	longest authorization and suggested duration
`application.support-access.console-url`	none	the console's address in the e-mails sent to the operator; failing that, the product's page
`application.support-access.front-urls.<product>`	none	each product's **Support access** page, `{subdomain}` replaced by the client's subdomain
`application.support-access.staff-clients`	`[mla]`	subdomains of the clients whose users are MLA staff
`application.support-access.orphan-session-after`	`PT5H`	age of an opening without an end before the `UNKNOWN` end; longer than the longest session
`application.support-access.orphan-session-cron`	`0 */5 * * * *`	schedule of the orphan-session job (`-` disables it)
`application.support-access.refusal-deduplication-window`	`PT5M`	refusal deduplication window
`application.access-event.retention`	`P90D`	how long access events are kept

Security

This page describes how subscription protects what it holds and who may call it: the tokens accepted, what each endpoint requires, the other services' calls, the tenant credentials and their master keys, the public endpoints, and where subscription reads its secrets. It is meant for technical readers (development, operations). No secret value appears here, only property and path names.

Tokens accepted

subscription is an OAuth2 resource server: every call carries a Keycloak access token (JWT), checked by security-library's decoder (`CustomNimbusJwtDecoder`).

- **Accepted realms.** `security.jwt.allowed-realms` lists the realms whose tokens are accepted. The token's issuer (`iss`) must be exactly `<Keycloak URL>/realms/<realm>` for one of them; any other token is refused before Keycloak is contacted. Without this property, the service does not start.
- **Signature.** Only RS256-signed tokens are read; an encrypted or unsigned token gets 401. The public keys come from the realm's JWKS endpoint.
- **Caller's realm.** subscription reads the user's realm from the token's issuer, and their login from the `preferred_username` claim. A user is the pair (realm, login): a login is unique only within its realm.

The browser holds no token: it has a session with the gateway, which attaches the user's token to every request to subscription.

Who is who

subscription tells these callers apart, always from the token:

Caller	Recognised by
Platform administrator	a token from a platform realm (`security.jwt.platform-realms`, `[mla, shared]` by default) holding exactly the role `ROLE_ADMIN` in `access_roles` or `realm_access.roles`
MLA support member	a platform administrator, or a token from the `mla` realm holding the role `SUPPORT` in `access_roles` or `realm_access.roles`
User	their realm and `preferred_username`; their rights in a tenant (active member, administrator) are read from subscription's database
Platform service	a token from a platform realm issued to a Keycloak client listed in a caller group (`security.service-callers.groups.<group>`)
Shared service account	a token from a platform realm issued to the `internal_service_client` client
The gateway, for support access	a token from the `shared` realm issued to the `svc-gateway` client, a member of the `support-access` group

A `ROLE_ADMIN` role held in a client's realm does not make a platform administrator.

What the API requires

The administration console is reserved to platform administrators. subscription applies the first rule matching the request:

Request	Who may call it
`OPTIONS`, any path	anyone
`GET /actuator`, `/actuator/health/**`, `/actuator/info`	anyone (Kubernetes and Consul probes)
the rest of `/actuator/**`	the services of the `actuator` group only (Spring Boot Admin), never a user's token, platform administrators included
`GET /v3/api-docs/**`, `/swagger-ui/**`	anyone; served only where Consul turns them on (locally and on integration)
`GET /api/v1/user-tenants/{service}/{user}`	platform administrator, services of the `user-tenants` group, or the user themself for their own realm
`GET /api/v1/tenant-credentials`	services of the `tenant-credentials` group only, never a user's token, platform administrators included
`GET /api/v1/user-tenants/{tenant}/{service}/{user}`	platform administrator, or services of the `user-tenants` group
`GET /api/v1/tenant-status`	platform administrator, or services of the `tenant-status` group
`GET /api/v1/public/clients/by-subdomain/*` and `…/logo`	anyone, without a token
`GET /api/v1/tenants/{tenantId}/logo`	any authenticated caller, then: service account, platform administrator, or active member of the tenant
`/api/v1/reglab-users/me/language`	any token naming a user; each reads and sets their own language
`GET /api/v1/events` (the live-update stream)	platform administrator, or a user of the support console
`/api/v1/support-access/check`, `/sessions/events`, `/sessions/*/termination`	the gateway alone
`GET /api/v1/access-events`	MLA support member; except for platform administrators, limited to support access events
`/api/v1/support-access/**` (the rest)	any authenticated caller; each operation then checks its rights (see Support access)
everything else, `/api/v1/master-keys` and `/api/v1/security/password/generate` included	platform administrator

The error page of an already authorized request (a 404, say) stays an error and does not turn into a 403.

Calls from the other services

The platform's services call subscription with their own token, obtained from Keycloak through the *client credentials* flow. security-library provides it (`ServiceTokenClient`) from the service's `application.s2s.*` keys: `application.s2s.client-id`, `application.s2s.client-secret`, `application.s2s.token-endpoint` and `application.s2s.grant-type`. The token is renewed before it

expires (`application.s2s.expiration-margin`, 60 seconds by default). A service with its own Keycloak client (`svc-irrc`, `svc-stress`…) keeps its keys in its own Vault path, which wins over the shared client's in `secret/application`.

subscription grants each group of endpoints to the clients listed in its Consul configuration, `security.service-callers.groups.<group>`:

Group	Endpoints	Clients in the local stack
`tenant-credentials`	`GET /api/v1/tenant-credentials`	`svc-irrc`, `svc-stress`, `svc-risk-engine`
`user-tenants`	`GET /api/v1/user-tenants/{service}/{user}` and `/{tenant}/{service}/{user}`	`svc-irrc`, `svc-library`, `svc-stress`, `svc-risk-engine`, `svc-assistant`
`tenant-status`	`GET /api/v1/tenant-status`	`svc-stress`
`support-access`	the gateway's endpoints for support access	`svc-gateway`
`actuator`	`/actuator/**` beyond health and info	`svc-admin`

- While `security.service-callers.legacy-shared-client` is `true` (the default), the shared `internal_service_client` is admitted in every group. The first admission per group is logged, to spot the services still using it. The gateway's support access endpoints also require the `svc-gateway` client of the `shared` realm: the shared client does not get in there.
- A change of these groups in Consul is taken into account without a restart, at the next configuration refresh (about a minute). An invalid value (an empty group) is refused and the value in use is kept; a restart would fail on it, though.
- The groups actually configured in the deployed environments are read in their Consul, `config/subscription/data`.

subscription's outgoing calls. subscription's code does not call the other services over HTTP. It calls:

- Keycloak's admin API, in each realm, with the `subscription-provisioning` client (`application.keycloak-accounts.client-id`) and that realm's secret (`application.keycloak-accounts.client-secrets.<realm>`). It uses it to disable or re-enable accounts, read the sign-ins, and re-read the roles of the operator and the target of support access;
- Kafka, with the user of `spring.kafka.*`; e-mails go out on `notification_topic` with a dedicated user (`application.notification.kafka-username`, `application.notification.kafka-password`), the only one allowed to write that topic. Without both properties, no e-mail is sent.

Tenant credentials

Each tenant has a credential, chosen when it is created (**Login credentials** in the tenant form), which the form no longer changes afterwards. CIRCÉ (irrc) uses it to encrypt the tenant's data.

- **Type.** Only **Password** (`PASSWORD`) is accepted: a **Key** (`KEY`) credential is refused, since no service encrypts with a key pair yet.
- **Password.** It must reach `application.credential.minimum-password-length` characters. **Generate** proposes one (`GET /api/v1/security/password/generate`); it is shown only at that moment. subscription draws a random salt on creation.
- **Storage.** The `password`, `salt`, `private_key` and `public_key` columns of `tenant_credential` are encrypted with the current master key (see below).
- **Reading.** `GET /api/v1/tenant-credentials?tenantId=` returns the credential in clear to the services of the `tenant-credentials` group only. No front reads it, and a platform administrator's token is refused: a stolen administrator session does not yield the keys. Every read is logged with the calling client, its realm, the tenant and the outcome, never the secret (`Tenant credential read: tenant=…, caller=…, realm=…, found=…`).

Master keys

subscription encrypts the tenant credentials with its own master keys (`EncryptionService`):

- each value is encrypted with AES-GCM (the v2 format of `VersionedAesGcmEngine`); it starts with the version byte, then the id of the key that encrypted it (0 to 255);
- each value is bound to the `tenant_credential` table: a value moved to another use of the same secret is rejected;
- each key is derived from its passphrase and salt (PBKDF2, 600,000 iterations), once, on first use.

The keys are properties:

Property	Role
`application.credential.encrypt-password`, `application.credential.encrypt-salt`	key 0; it also reads the values written before the v2 format
`application.credential.keys.<id>.password`, `application.credential.keys.<id>.salt`	the other keys, `<id>` from 1 to 255; a passphrase too weak stops the start-up
`application.credential.current-key-id`	the key of every new write (0 by default); if it is not configured, subscription does not start

Re-encryption at start-up. At every start, subscription re-encrypts with the current key every value not on it (`TenantCredentialReencryption`). A database lock keeps this work to a single instance. A value that fails is logged and left as it is, for the next start. The log then says whether every value is on the current key, or which remain on another key.

The Master key tab. The **Settings** page, **Master key** tab, shows the same state without reading the logs (`GET /api/v1/master-keys`, platform administrators). The keys themselves are never shown, and the screen changes nothing: every action is done in Vault. **Refresh** reads the state again.

The banner gives the verdict, most serious first:

Banner	Meaning	What to do
"… value(s) on key …, which Vault no longer holds"	values name a key no longer configured: they are unreadable	put the key back in Vault under the same id, then restart subscription
"Key … still encrypts … value(s)"	an old key is still in use	keep it in Vault; restart subscription to re-encrypt
"Key … no longer encrypts anything"	the old key is no longer used	remove it from Vault; keep a copy while a database backup older than the rotation exists
"Every tenant credential is on key …"	a single key configured	nothing

The table gives, for each key, its **Status** (**Current**, **Still in use**, **Can be removed**, **Missing from Vault**) and its number of **Values**, plus a **Pre-v2 format (key 0)** row if any remain.

Rotating the master key

Rotation is manual, in Vault; subscription reads its keys only at start-up, so each step ends with a rolling restart of subscription:

1. Add the new key (`application.credential.keys.<id>.password` and `.salt`) without changing `current-key-id`, then restart. Every pod then knows the key before it is used.
2. Make the new key current (`application.credential.current-key-id`), then restart: the values are re-encrypted at start-up. The **Master key** tab must say that the old key no longer encrypts anything.
3. Remove the old key from Vault, then restart.
4. Keep the old key in the team vault while a database backup older than the rotation exists.

The detailed procedure, checks and rollback included, is the `rotation-cle-maitre-subscription` runbook of the documentation repository. The rotation changes the master key only: the tenant credentials do not change, and CIRCÉ's data stays readable.

Public endpoints and logos

Before anyone signs in, a front reads the client its host name names:

- `GET /api/v1/public/clients/by-subdomain/{subdomain}` returns the realm to sign in to, the client's name and whether it has a logo;
- `GET /api/v1/public/clients/by-subdomain/{subdomain}/logo` returns its logo.

These two endpoints are open without a token. They answer only for a client that has a subdomain and an accepted realm, and say nothing more about it; otherwise 404. Their answer may be kept 5 minutes by shared caches.

The logo of a tenant's client (`GET /api/v1/tenants/{tenantId}/logo`) is reserved to the service account, platform administrators and the tenant's active members; any other caller is refused before learning whether the tenant exists. Its answer is private, kept one hour, then revalidated by ETag. Only platform administrators upload, read by client and remove a logo (`/api/v1/clients/{id}/logo`).

Vault and Consul

subscription imports its configuration from Consul, then from Vault (`spring.config.import`: `consul:` then `optional:vault://`). A key in Vault wins over a key of the same name in Consul. Consul is mandatory: an unreachable agent stops the start-up. Vault is marked `optional` only for the case where it is disabled; an unreachable Vault is still governed by the *fail-fast* setting.

Source	Path	Content
Consul	`config/application/data`, then `config/subscription/data` (YAML)	the configuration: `security.jwt.*`, `security.service-callers.*`, `application.support-access.*`, Kafka, etc.
Vault, KV engine `secret`	`secret/application`	keys shared by every service: shared Kafka user, shared service client (`application.s2s.*`), `keycloak.server.url`
Vault	`secret/subscription`	subscription's secrets: database login, Kafka user, master keys (`application.credential.*`), secrets of the `subscription-provisioning` clients (`application.keycloak-accounts.client-secrets.<realm>`)

The connection is set by environment variables: `VAULT_URI`, `VAULT_TOKEN`, `VAULT_KV_BACKEND` (`secret` by default), `VAULT_AUTH_METHOD` (`token` by default; AppRole and Kubernetes are possible with `VAULT_APPROLE_ROLE_ID`, `VAULT_APPROLE_SECRET_ID`, `VAULT_K8S_ROLE`, `VAULT_K8S_AUTH_PATH`), and for Consul `CONSUL_HOST`, `CONSUL_PORT`, `CONSUL_TOKEN`.

Events and calls

This page, for technical readers, describes how subscription and its console talk to the platform's other services:

- the Kafka messages the subscription service writes and reads;
- the live updates subscription pushes to the console;
- the synchronous HTTP calls, the console's and the service's.

It describes the code of the `dev` branches of subscription and subscription-front; the other services' consumers are taken from the Kafka messages architecture page of the platform documentation.

Overview

- subscription **writes** to five topics: `subscription_irrc_topic`, `tenant_lifecycle_topic`, `notification_topic`, `library_topic` and `ui_events_topic`. It **reads** only `ui_events_topic`, to relay changes to the console.
- The console (subscription-front) calls **directly**, from the browser and through the gateway, each service it administers: crawler, library, notification, referential, dpm. subscription is not a go-between, except for the URLs added to the common library.
- The subscription service calls none of these services over HTTP: it reaches library and notification only through Kafka. Its only outgoing synchronous call is Keycloak's admin API.

Kafka topics

Topic	subscription	Message types	Key	Read by
`subscription_irrc_topic`	writes	`UPDATE_PLAN`	`tenantId`	irrc, assistant, risk-app
`tenant_lifecycle_topic`	writes	`TENANT_DELETED`	`tenantId`	irrc, library, assistant, risk-app
`notification_topic`	writes	`SEND_MAIL`	the user, tenant or support access request concerned	notification
`library_topic`	writes	`NEW_PUBLICATION`, `NEW_COMMON_FILE`	`link` or `fileName`	library
`ui_events_topic`	writes and reads	writes `SUPPORT_REQUEST_CHANGED`; reads `CRAWLER_RUN_CHANGED`, `LIBRARY_DOCUMENT_CHANGED`, `LIBRARY_EMBEDDINGS_CHANGED`, `LIBRARY_INDEX_CHANGED`, `DOCLING_QUEUE_CHANGED`, `SUPPORT_REQUEST_CHANGED`	`eventType`	subscription, irrc, risk-app

Other services also write to some of these topics: crawler to `library_topic`, `notification_topic` and `ui_events_topic`, library to `ui_events_topic`, irrc and risk-app to `notification_topic`.

Each topic has its producer in the `bus` package: `Producer` (`subscription_irrc_topic`), `TenantLifecycleProducer`, `NotificationProducer`, `LibraryProducer` and `UiEventProducer`; the reader is `UiEventsConsumer`.

Messages

UPDATE_PLAN

A tenant's memberships changed: a user's plans or role in that tenant, a user taken out, suspended or given back access, a user disabled or enabled, or the tenant's status or validity. The message names the tenant, never the user; it has no `content`.

irrc, assistant and risk-app then forget what they cache for that tenant (every tenant when the message names no readable one); irrc then reads the memberships again from subscription. See Tenant lifecycle and Rights.

TENANT_DELETED

A deleted tenant's grace period is over and the purge has marked it deleted: every service erases what it holds for that tenant. The message is sent once, when the purge commits, and may arrive more than once: each reader must be able to handle it again without harm. `tenantId` is the tenant's public id (the one the services store), never subscription's primary key. See Purge.

SEND_MAIL

A mail ready to go: subscription renders the HTML itself, from the `templates/mail/*.ftlh` templates and the `i18n/mail*.properties` texts. The content holds `subject`, `htmlContent` and `recipients`, with no attachment. The notification service sends it, records it in its history, and refuses the whole mail when a recipient is outside the allowed domains. The mails themselves are described in Notifications.

NEW_PUBLICATION and NEW_COMMON_FILE

Documents for the common library, sent by a platform administrator:

- `NEW_PUBLICATION`: a URL, one message per URL, sent by `POST /api/v1/common-documents/urls` (`CommonDocumentResource`). The content holds `link` only; library downloads the document and indexes it. The crawler sends the same message type for its publications;
- `NEW_COMMON_FILE`: a whole file, base64-encoded in the message (`fileName`, `contentType`, `content`, `metadata`, always empty), sent by `POST /api/v1/common-documents/upload`. The console does not use this entry point: it uploads files straight to library (see Synchronous calls).

SUPPORT_REQUEST_CHANGED

A user's support access request or grant changed: their applications reload their support banner without waiting for their periodic read. The content holds `tenantId`, `realm` and `username`. subscription reads it too, for the support console. See Support access.

Envelopes

Each topic has its JSON envelope. All carry `schemaVersion` at `1`; readers read the fields by name, so an added field does not disturb them.

Field	`subscription_irrc_topic`	`tenant_lifecycle_topic`	`notification_topic`	`library_topic`	`ui_events_topic`
`messageType`	`UPDATE_PLAN`	`TENANT_DELETED`	`SEND_MAIL`	`NEW_PUBLICATION`, `NEW_COMMON_FILE`	the event type
`schemaVersion`	`1`	`1`	`1`	`1`	`1`
`tenantId`	the tenant's public id	the tenant's public id	—	—	— (inside `content` when there is one)
`content`	—	—	the mail	the document	the event
`deletedAt`	—	the instant of the purge	—	—	—

Classes: `GenericProducerMessage`, `TenantLifecycleMessage`, `NotificationMessage` (with `MailMessage`), `LibraryMessage` (with `PublicationContent` or `CommonFileContent`) and `UiEventMessage` (with `SupportRequestChangedContent`), in `bus/message`.


```
{"tenantId":"11111111-1111-1111-1111-111111111111","messageType":"UPDATE_PLAN","schemaVersion":1}
```

```
{"tenantId":"12359bb8-b120-11ed-afa1-0242ac120005","messageType":"TENANT_DELETED",
 "deletedAt":"2026-11-04T03:30:00.123456Z","schemaVersion":1}
```

```
{"messageType":"NEW_PUBLICATION","schemaVersion":1,"content":{"link":"https://www.
example.com/doc.pdf"}}
```

```
{"messageType":"SUPPORT_REQUEST_CHANGED",
                                                    "content":{"tenantId":"12359bb8-
b120-11ed-afa1-0242ac120005","realm":"bpi","username":"s.bassin"},
 "schemaVersion":1}
```


> ⚠ Warning
>
> The message type names and the envelope fields are the contract between the services. A renamed or removed type fails no compilation: it breaks reading in the other services, at runtime. Any change is decided together with the list of readers to ship in the same window.

Delivery guarantees

Each producer waits for the broker's acknowledgement, at most `application.kafka.send-timeout` (10 seconds by default, each environment's value coming from Consul). What happens on a failure depends on the message:

Message	When it is sent	On a failure
`UPDATE_PLAN`	Inside the change's transaction, once the change is written	The change is rolled back and the request fails: a rights removal is never recorded without irrc being told.
`TENANT_DELETED`	Inside the purge's transaction, after its `UPDATE_PLAN`	The purge is rolled back.
`SEND_MAIL`	Once the change is committed	The failure is logged; the change stays. For contract-end reminders, the failure rolls back the reminder's record, and it is tried again at the next run.
`NEW_PUBLICATION`, `NEW_COMMON_FILE`	During the administrator's request	The request fails, with the error `error.messageTooLarge` for a message too large, `error.unknown` otherwise.
`SUPPORT_REQUEST_CHANGED`	Once the change is committed	The failure is logged and the message dropped: the applications catch up at their periodic read.

The broker lets only dedicated Kafka users write `notification_topic`. `NotificationProducer` therefore connects with its own credentials, `application.notification.kafka-username` and `application.notification.kafka-password`; without them, it says so once at startup and no mail goes out. The other topics use the shared Kafka user.

An uploaded file travels whole in one message: file uploads are limited to 50 MB, and the producer's message size to 75,000,000 bytes (a 50 MB file is about 67 MB once base64-encoded).

The console's live updates

The console opens, per browser tab, a server-sent events (SSE) stream on subscription: `GET /api/v1/events` (`text/event-stream`, classes `EventResource` and `SseHub` of the `sse` package). The events are invalidations: they say "this changed, read it again"; the page concerned then reads its data again through the API, which checks the rights.

Where the events come from

`UiEventsConsumer` reads `ui_events_topic` and turns each message it knows into a browser event. Each subscription instance reads the topic in a group of its own (`subscription-ui-events-<uuid>`), from the latest messages: every instance sees every event and relays it to the browsers it serves. Nothing is retried or set aside: an unreadable message is skipped. The `application.kafka.ui-events.auto-startup` property can keep this reader from starting.

Kafka message	Event sent (`nature`)	Data	Screen that reads again
`CRAWLER_RUN_CHANGED`	`CrawlerRunChanged`	`runId`	Crawler, the runs
`LIBRARY_DOCUMENT_CHANGED` (common documents only)	`LibraryDocumentChanged`	`documentId`	Library, the list of documents
`LIBRARY_EMBEDDINGS_CHANGED`	`LibraryEmbeddingsChanged`	—	Library, the embeddings
`LIBRARY_INDEX_CHANGED`	`LibraryIndexChanged`	—	Library, the search index
`DOCLING_QUEUE_CHANGED`	`DoclingQueueChanged`	—	Crawler, the Docling tab
`SUPPORT_REQUEST_CHANGED`	`SupportAccessChanged`	`tenantId` (platform administrators only)	Support console

An id at `null` means several items changed: the page reads the whole list again. Messages of an unknown type, and those about the tenants' own documents, are not relayed.

The stream

- Each frame is a `message` event whose data is a JSON object `{"nature": …, …}`.
- A stream's first frame is `{"nature":"Connected"}`: the console then knows it may have missed events and reads the open pages again.
- A `ping` comment goes out every 20 seconds, well under the ingress' 60-second read timeout.
- A stream is closed after 30 minutes; the browser reconnects.
- The same event received several times within 500 milliseconds goes out once.
- A failed send closes that stream only.

Two audiences receive the stream:

- platform administrators get every event, with its ids;
- the support console's other users (support members, tenant administrators) get only `SupportAccessChanged`, without the tenant, so that a stream never tells them about a tenant they do not administer.

On the console's side

The console broadcasts each event as `event.<nature>`; each page listens to those that concern it and to `event.Connected`. After a cut, it tries to connect again, with at most 60 seconds between two attempts, including after a 401 or 403 refusal.

A dot in the top bar shows the stream's state: green, "Live updates: connected"; orange after 5 seconds down, "Live updates: reconnecting"; red after a minute, "Live updates interrupted — data refreshes every 30 s".

The pages do not rely on the stream alone: they also read again on their own while some work is in progress.

Screen	Periodic read
Crawler, runs	every 30 s while a run is going
Crawler, summaries	every 10 s while a summary is in progress
Crawler, Docling	every 10 s while the tab is open
Library, documents	every 30 s while a document is queued, converting or embedding
Library, embeddings and search index	every 30 s while some work runs
Reference data, sources	every 3 s while an update runs (no stream)
DPM Kafka work	every 5 s while the tab is visible (no stream)

Synchronous calls

The console calls each service from the browser, through the gateway: the address is the gateway's, followed by the service's name and the path, and the call carries the user's token. The service called decides what the user may do: crawler and notification, for example, keep their API for platform administrators and the internal service account, and notification keeps resending a mail for administrators alone.

Caller ⟶ service	Entry points	Screen
console ⟶ crawler	`/api/v1/publications` (list, `POST /send-to-library`); `/api/v1/feeds` (list, create, update, delete, `POST /test`); `/api/v1/crawler/sources` (list, `PUT /{id}`); `/api/v1/crawler/settings` and `/api/v1/crawler/schedule` (read, `PUT`); `/api/v1/crawler/runs` (list, `POST` with an optional `source`, `/{id}`, `/{id}/publications`); `/api/v1/digest-recipients` (list, create, update, delete, `/sources`, `POST /{id}/test`); `/api/v1/digests` (list, `/next`, `/{day}`, `POST /{day}/resend`)	Crawler
console ⟶ library	`/api/v1/crawler-summaries` (read by `documentIds`, `POST`); `/api/v1/documents/markdown/queue`	Crawler, summaries and Docling
console ⟶ library	`/api/v1/documents/common` (list, `DELETE /{id}`, `POST` of a list of ids to delete, `POST /upload`, `GET /{id}/markdown`, `POST /{id}/markdown/retry`, `POST /markdown/retry-missing`, `POST /{id}/embeddings`, `POST /embeddings/selection`,	Library

Caller → service	Entry points	Screen
	`POST /{id}/indexing`); `/api/v1/documents/common/embeddings` (status, `POST`); `/api/v1/documents/index/rebuild` (status, `POST`)	
console → subscription	`POST /api/v1/common-documents/urls`	Library, URLs
console → notification	`/api/v1/mails` (list, `/{id}`, `/{id}/attachments/{attachment}`, `POST /{id}/resend`)	Notifications
console → referential	`/api/v1/structure/dataflow`, `/api/v1/structure/datastructure`; `/api/v1/bundles` (`POST` of a zip, `PUT` and `DELETE /{agency}/{id}/{version}`); `/api/v1/loader/cubes` (list, `POST`, `PUT` and `DELETE /{id}`, `/{id}/runs`, `POST /{id}/refresh`, `POST /{id}/upload`); `POST /api/v1/loader/refresh`	Reference data
console → dpm	`GET /api/v1/admin/kafka-work`	DPM Kafka work
console → subscription	`GET /api/v1/tenants` (tenant names); `GET /api/v1/events` (stream)	DPM Kafka work; every page
subscription → Keycloak	Each realm's admin API, through its service client (`subscription-provisioning` by default): disable or enable an account (disabling ends its sessions), read the logins and impersonations, check an account before a support access	Identities, Support access

> **Note**
>
> The common library's URLs go through subscription, which puts them on `library_topic`; files go straight to library. A URL sent is therefore in the broker's hands, a file sent in library's.

Glossary

The terms of the console and of this manual, in alphabetical order, with the page that describes them.

Terms

Term	Definition	See
Access	A user's place in a tenant: one role and at most one plan per service.	Users
Access history	The record of every change of access rights, kept 90 days.	Access history
Access request	The authorisation request a support member sends a user; it gives no access by itself.	Support console
Access review	The xlsx export of who holds which plan on which tenant today.	Users
Authorisation (consent)	The time-limited permission a user gives from their own session for a support consultation.	Support access
`base` feature	Created with each service, it means membership of the service; it cannot be deleted or renamed.	Features
Client	A customer organisation; it groups tenants and users, and may have a subdomain and a Keycloak realm.	Key concepts
Common document	A document of the common library, added by the crawler or by an administrator.	Library
Common library	The library service's documents that every tenant reads and the assistant searches.	Library
Consultation (support session)	A read-only session the gateway opens with a member's identity, in one tenant.	Support access
Contract	A tenant's validity dates and the reminders sent before its end.	Client and Tenants
Crawler	The service that reads regulators' sites and RSS feeds and hands what it finds to the common library.	Crawler

Term	Definition	See
Crawl run	One pass of the crawler over its sources: scheduled, at start-up or started by hand.	Crawler
Digest	The daily summary of new publications, mailed to each enabled recipient.	Crawler
Disable a user	Cut all their accesses and disable their Keycloak account, keeping their roles and plans.	Users
DPM Kafka work	The live view of what each Kafka consumer of the dpm service is doing, and what waits behind it.	DPM Kafka work
Embedding	Cutting a document into passages and embedding them, so the assistant finds it by meaning.	Library
End-of-contract reminder	A mail sent before or at the end of a tenant's validity.	Notifications
Envelope	The JSON object around a Kafka message (`messageType`, `schemaVersion`, `tenantId`, `content`…).	Events
Feature	A named capability of a service, whose name applications check.	Features
Guest client	Another client whose tenants a multi-client user may join.	Users
Home client	The client a user belongs to.	Users
Live updates	The event stream that tells the console's pages to read their data again.	Getting started
Master key	A key subscription encrypts tenant credentials with; it is rotated in Vault.	Security
Multi-client	A user who may be invited into other clients' tenants.	Users
Operator	The person who opens a consultation: an MLA support member or an administrator of the tenant.	Support access
Plan	An offer on one service, opening a selection of its features (`base` always included).	Plans
Platform administrator	An account of a platform realm (`mla`, `shared`) holding the `ROLE_ADMIN` role; the only one that opens the console.	Getting started

Term	Definition	See
Protected account	An account a support consultation can never use (administrators, support, realm management).	Support access
Purge	The job that turns a tenant past its waiting period to `DELETED`, removes its members and sends `TENANT_DELETED`.	Purge
Realm	A Keycloak directory of accounts; a user is the pair (realm, username).	Identities and realms
Reference data	The reference-data service (SDMX cubes, EBA annexes).	Reference Data
Reference of a tenant	The tenant's public identifier (`tenantId`), the one the services keep.	Client and Tenants
Role in the tenant	Member or Admin, valid in that tenant only; an Admin also manages the others' access.	Key concepts
Service	A product of the platform (`irrc`, `stress`, `library`…), which applications look up by name.	Services
Service token	The client-credentials token a service obtains to call another service.	Security
Support access policy	A tenant's rule for consultations: on request, always or never.	Support console
Support console	The page where MLA support and administrators see a tenant's members, request access, set the policy and read the journal.	Support console
Support journal	The support access events of a tenant, filterable and exportable as CSV.	Support console
Suspended access	An access whose role and plans are kept, but which no service accepts any more.	Users
Tenant	An isolated Reglab environment of a client, known to the services by its `tenantId`.	Key concepts
Tenant credential	A tenant's encryption password, kept encrypted by subscription, chosen at creation and never shown again.	Client and Tenants

Term	Definition	See
Tenant status	`ACTIVE`, `SUSPENDED`, `PENDING_DELETION` or `DELETED`; only `ACTIVE` grants access.	A tenant's lifecycle
`TENANT_DELETED`	The Kafka message, on `tenant_lifecycle_topic`, asking each service to purge a tenant's data.	A tenant's lifecycle
`UPDATE_PLAN`	The Kafka message, on `subscription_irrc_topic`, telling the services to forget the cached memberships of a tenant.	From plans to rights
Validity	The window (start included, end excluded) outside which a tenant grants no access.	Client and Tenants
Waiting period	The period (60 days by default) during which a deleted tenant can be restored, before its purge.	A tenant's lifecycle
`X-TENANT-ID`	The header naming a request's tenant by its public identifier.	From plans to rights