3.9 Release Notes

Releases · https://namazustudios.com/docs/releases/3-9-release-notes/

Warning

Elements 3.9 is still under active development (current version: `3.9.0-SNAPSHOT`) and has not been released. The contents of this page are a draft and may change before the final release.

Overview

Elements 3.9 adds per-application profile limits and automatic primary-profile creation, a new way to attach a user's profile to a session by naming an Application instead of an explicit profile, and account linking for the OIDC browser-redirect login flow.

Highlights

- **Per-application profile limits** — a new `maxProfiles` setting on Application bounds how many profiles a user may create for it.
- **Automatic primary profile creation** — a new `autoCreateProfile` setting on Application, combined with a new `autoCreateProfileApplicationNameOrId` field on the user-create/signup request, lets Elements create a user's primary profile for an Application automatically at signup time. See Creating a User.
- **Session creation by Application, including auto-create** — username/password, OAuth2, and OIDC session requests can now pass `applicationNameOrId` to attach the user's primary profile for that Application, instead of an explicit `profileId` / `profileSelector`. If no primary profile exists yet, it's now created automatically, subject to the Application's own `autoCreateProfile` / `maxProfiles` settings. See Sessions.
- **Authoritative profile pictures and display-name validation** — new `authoritativeProfilePicture` and `displayNameRegex` settings on Application. See Applications.

- **Account linking via the OIDC browser-redirect flow** — starting a login attempt while already holding a session now links the resulting external identity to that user instead of creating a new one, with a new `confirmToken`-gated confirmation step to keep the mutation off the unauthenticated provider callback. See OIDC Login for Thick Clients.
- **Progress API fixes and a new advance-progress endpoint** — `POST /progress` and the superuser `PUT /progress/{id}` path are fixed, and a new `POST /progress/{progressId}/advance` endpoint lets a Mission opt in to client-driven progress advancement. Reported, diagnosed, and prototyped by community contributor @hobolabsdigital -- thank you!
- **System-wide password policy** — a new deploy-time regex config gates every password a client submits (signup, password reset, admin-set password, account linking), paired with a human-readable description shown in the validation error. See below.
- **Stale Datastore/Mapper fix on Element (re)deploy** — a singleton that captured Elements' shared Mongo `Datastore` could go stale on the next Element (re)deploy; see below.
- **CRUD events across nearly every DAO** — create/update/delete events, previously only produced by a handful of DAOs, now cover nearly all of them, making it practical to build audit logging and similar cross-cutting Elements. See Events.
- **Opt-in Guice PRODUCTION stage for injectors** — every Guice injector in the platform (per-Element, jetty-ws itself, the `migrate` / `setup` tools, and more) can now be built with Guice's `Stage.PRODUCTION` instead of the default `Stage.DEVELOPMENT`, via a new system property/environment variable. This pairs with the Datastore/Mapper fix above: capturing the shared `Datastore` in an eager singleton is safe now, so there's no new risk from the earlier construction timing under `PRODUCTION`. See below.

New Features

Per-Application Profile Limits and Auto-Create

`Application` gains two new fields: `maxProfiles` (defaults to `1`) caps how many profiles a user may create for that application, and `autoCreateProfile` (defaults to `true`) governs whether a user's primary profile is created automatically when requested via `autoCreateProfileApplicationNameOrId` on user creation. Lowering `maxProfiles` never affects profiles that already exist -- only new profile creations are gated. Existing applications with no value set for these fields behave as if they were set

to the defaults.

Session Creation by Application

Username/password, OAuth2, and OIDC session requests accept a new `applicationNameOrId` field (an application name or ID). If neither `profileId` nor `profileSelector` is specified, Elements resolves the user's primary profile for that application and attaches it to the session. If no primary profile exists yet, one is now created automatically, subject to the same `autoCreateProfile`/`maxProfiles` gating as signup-time auto-create, via the same `ProfileDao#createSlottedProfile` path; if the application can't be resolved, or auto-create isn't configured for it, the session is simply created without a profile.

Account Linking via the OIDC Browser-Redirect Flow

Starting an OIDC browser-redirect login attempt (`POST /oidc/session`) while already holding a session now links the resulting external identity to that user, the same way the existing Account Linking endpoints do for a possessed `id_token`. No new request field is involved; whether an attempt links or creates a new user is decided purely by whether the caller had a session when the attempt was started.

Because the provider's callback that validates the external identity is always an unauthenticated redirect from the identity provider, with no way to confirm it's the same caller that started the attempt, it no longer performs the account-link mutation itself. A new `confirmToken`, returned only in the original `begin()` response, gates a new `POST /oidc/session/{id}/confirm` step that performs it. See OIDC Login for Thick Clients for the full sequence. This closes a case where a leaked `state` value, which (unlike the poll `id`) necessarily passes through the browser and the identity provider, could otherwise have let an attacker permanently link their own external identity to a victim's account.

Authoritative Profile Pictures and Display-Name Validation

`Application` gains two more new fields: `authoritativeProfilePicture` (defaults to `false`), which when `true` blocks a user from editing their own profile picture for that application via the REST API (it must be set by backend/Element code instead), and `displayNameRegex` (optional), a Java regular expression a profile's display name must match for that application -- profile creates/updates with a non-matching display name are rejected. Leave it blank to skip the check.

Progress API Fixes and Advance-Progress Endpoint

`POST /progress` no longer 400s with `"profile - must not be null"` when a valid profile is supplied, and the superuser `PUT /progress/{id}` path no longer rejects every possible request body. Both were reported with full root-cause analysis by community contributor @hobolabsdigital in #2 and #3 -- thank you for the thorough repros and the `sequence` / `currentStep` data-model deep-dive.

Mission gains a new `authoritative` field (defaults to `true`). A new `POST /progress/{progressId}/advance` endpoint decrements a Progress's remaining actions, advancing Steps and issuing Rewards as needed -- superusers may always call it, and a regular user may only call it for their own Progress on a Mission explicitly marked `authoritative: false`. This is the client-driven progress advancement @hobolabsdigital originally prototyped in #3, now gated per-Mission so authoritative-integrity is preserved by default.

System-Wide Password Policy

Two new deploy-time configuration values, `dev.getelements.elements.password.policy.regex` and `dev.getelements.elements.password.policy.description`, let an operator require passwords to match a regular expression before Elements will accept them. The regex is enforced everywhere a password is accepted or changed by a client: signup, password reset completion, admin-set password, self-service change-password, and email/username-password account linking. It is not applied to server-generated passwords, such as the bootstrap default superuser account or mock/test accounts. See Properties for the exact keys.

The description value is a plain-text, human-readable explanation of the requirement (e.g. "Password must be at least 4 characters."), returned as part of the error message when a submitted password fails the regex, so a client can surface it directly without parsing the regex itself. The default regex is `.{4,}` (minimum length only, matching the length of the built-in default superuser password), which is a change from Elements' previous behavior of accepting any non-blank password. Operators are responsible for keeping the regex and its description in sync -- Elements does not attempt to derive one from the other. This is system-wide only; there is no per-application or per-tenant override.

Guice SPI Loading-Strategy Escape Hatch

The package-level `@GuiceOptions` annotation is now wired into `GuiceSpiModule`, giving third-party Element authors an opt-in escape hatch for the `[Guice/ExposedButNotBound]` crash that can occur when an exported service has no locally-discovered implementation. Elements that don't declare `@GuiceOptions` see no behavior change -- the existing bind/expose scanning remains the default `LEGACY` strategy. Authors can instead declare `GUICE_MODULE_ONLY` to defer every exported service to their own `@GuiceElementModule`(s), or `STRICT` to fail fast at startup with a clear error naming the unbound service instead of Guice's generic crash. See Introduction to Guice and Jakarta in Elements.

CRUD Events Across Nearly Every DAO

Before 3.9, only three DAOs (`ElementDeploymentDao`, `MultiMatchDao`, and `ReceiptDao`) published create/update/delete events. That coverage now extends to nearly every DAO in the system, including profiles, sessions, applications and their configurations, auth schemes, OIDC login attempts and provider configurations, inventory items and item ledger entries, missions and progress, schedules, leaderboards and scores, reward issuances, save data documents, large objects, followers and friends, FCM registrations, smart contracts, vaults and wallets, and more.

Each new event follows the same two-variant pattern already used by the pre-3.9 DAOs and described in Events: a transactional variant carrying a `Transaction` argument, published immediately as part of the write, and a plain variant published only once the enclosing transaction commits (and dropped entirely if it rolls back). A handful of DAOs expose fewer variants where it matches the entity's actual lifecycle -- for example, `ScoreDao` fires a single `SCORE_CREATED_OR_UPDATED` event since scores are always upserted, `ItemLedgerDao` only fires a created event since ledger entries are immutable, and `FriendDao` only fires a deleted event since friendships are formed implicitly through mutual follows rather than a direct create call.

As with all Element events, the authoritative list of event names and their argument types for a given DAO is discoverable at runtime via the CMS's Produced Events screens, or the underlying `GET /elements/system` and `GET /elements/application` endpoints; see Events for details.

Opt-In Eager Singleton Construction (Guice Stage.PRODUCTION)

Every Guice injector in the platform can now be built with `Stage.PRODUCTION` instead of the default `Stage.DEVELOPMENT`, controlled by the `dev.getelements.elements.guice.stage` system property (or the

`ELEMENTS_GUICE_STAGE` environment variable if the property isn't set). Leaving both unset keeps today's `DEVELOPMENT` behavior everywhere, including for Element injectors -- `PRODUCTION` is strictly opt-in. A server deployment that wants the benefits below should set `dev.getelements.elements.guice.stage=PRODUCTION` (or the equivalent environment variable) in its own launch configuration.

When enabled, two things change for Element authors:

- Any service class you mark `@Singleton` is constructed at Element-load time, not lazily on first use. Previously only bindings explicitly marked `.asEagerSingleton()` in a Guice module were built eagerly; a plain `@Singleton` class was left to first use.
- Every binding in your Element's injector is validated up front at load time, so a misconfigured binding fails fast when the Element loads instead of surfacing later at first invocation.

This is safe to rely on together with the Datastore/Mapper fix below: injecting the shared `Datastore` into an eager singleton no longer risks capturing a stale snapshot, since the `Datastore` you receive is a stable proxy regardless of when it's constructed. If your own service binds another shared, mutable dependency directly (not through a similar proxy or a `Provider`), review whether earlier eager construction under `PRODUCTION` stage could now capture a stale reference to it.

Bug Fixes

Stale Datastore/Mapper State on Element Redeploy

Any singleton that captured the injected Morphia `Datastore` directly could go stale the moment an Element (re)deploy rebuilt and swapped the shared `Datastore`/Mapper -- most reliably on every redeploy, since an Element's eager singletons are constructed before its own entities are even registered. The injected `Datastore` is now a stable proxy that always forwards to whichever instance is live at call time, so holding a reference to it -- in Elements' own internal DAOs, or in a downstream Element's -- is safe by construction.