ChangelogDraftsService

September 4, 2026

ChangelogDraftsService

Kind: Service

Source: `atloria-monorepo/apps/api/src/changelog/changelog-drafts.service.ts`

Auto-drafted changelog entries (B5).

`recordChange` is THE seam the four delta/release hooks call. It is fire-and-forget by contract: it never throws (all failures are caught and logged), so a changelog hiccup can never fail a materialize, an upload, an activation, or a webhook. Events batch per project per UTC day into one draft (`windowKey = auto-YYYY-MM-DD` , race-safe via the unique constraint); publishing frees the windowKey so a later same-day event opens a fresh draft instead of mutating a published note.

The AI pass (BullMQ 'changelog' queue, inline fallback) rewrites the deterministic draft into a human release note — but only while the entry is still an untouched draft: owner edits (ownerEditedAt) always win.

`ChangelogDraftsService` manages project changelog entries, including automatic daily drafts generated from release and delta events. Its `recordChange()` method is a fire-and-forget integration seam that never propagates failures, ensuring changelog processing cannot block uploads, materialization, activation, or webhook flows. Drafts are grouped by project and UTC day, then optionally rewritten into release-note language by an AI summarization pass unless an owner has edited the entry.

Methods

Method	Signature	Returns	Description
`onModuleInit`	`onModuleInit()`	`unknown`	
`recordChange`	`recordChange(input: RecordChangeInput)`	`Promise<void>`	Record a raw change-set event into today's draft.

Method	Signature	Returns	Description
`summarizeEntry`	`summarizeEntry(entryId: string)`	`Promise<void>`	Rewrite an untouched draft into a human release note.
`listEntries`	`listEntries(projectId: string, status: string)`	`unknown`	
`getEntry`	`getEntry(projectId: string, entryId: string)`	`unknown`	
`createManual`	`createManual(projectId: string, createdById: string | undefined, input: { title: string; body: string; tags?: string[] })`	`unknown`	
`updateEntry`	`updateEntry(projectId: string, entryId: string, patch: { title?: string; body?: string; tags?: string[] })`	`unknown`	
`publish`	`publish(projectId: string, entryId: string)`	`unknown`	draft → published (+ email fan-out).
`discard`	`discard(projectId: string, entryId: string)`	`unknown`	
`getConfig`	`getConfig(projectId: string)`	`unknown`	
`setConfig`	`setConfig(projectId: string, input: { autoPublish: boolean })`	`unknown`	
`autoPublishSweep`	`autoPublishSweep()`	`Promise<void>`	Hourly: publish drafts whose 24h window has closed, for projects that opted into auto-publish.

Dependencies

- `PrismaService`
- `ChangelogQueue`

- `ChangelogSubscriptionsService`
- `AIProvider` *(optional)*
- `AIUsageService` *(optional)*

Where it refuses work

- `ChangelogDraftsService` stops the work with `NotFoundException` when `!entry` — “Changelog entry not found”.
- `ChangelogDraftsService` stops the work with `NotFoundException` when `!project` — “Project not found”.
- `ChangelogDraftsService` stops the work with `BadRequestException` when `entry.status === 'discarded'` — “Entry is discarded”.
- `ChangelogDraftsService` stops the work with `BadRequestException` when `entry.status !== 'draft'` — “Only drafts can be published”.
- `ChangelogDraftsService` stops the work with `BadRequestException` when `entry.status === 'published'` — “Published entries cannot be discarded”.
- `ChangelogDraftsService` stops the work with an early return when `!entry`, in 2 places.

When something fails

- `ChangelogDraftsService` handles failure in 6 places: it logs it and continues in 4, and turns it into a return value in 2.

Diagram

```
sequenceDiagram
    participant Hook as Delta / Release Hook
    participant Service as ChangelogDraftsService
    participant DB as Database
    participant Queue as BullMQ "changelog" Queue
    participant AI as AI Summarizer
    participant Owner as Project Owner

    Hook->>Service: recordChange(projectId, change)
    Service->>DB: Find/create daily draft<br/>windowKey: auto-YYYY-MM-DD

    alt Existing unpublished daily draft
        DB-->>Service: Append deterministic change details
    else No active draft
        Service->>DB: Create draft with unique windowKey
        Note over Service,DB: Unique constraint handles races
```

```
    end

    Service->>Queue: Enqueue summarizeEntry(entryId)
    Note over Service: Failures are caught and logged;<br/>caller is never blocked

    Queue->>Service: summarizeEntry(entryId)
    Service->>DB: Load entry

    alt Draft has not been owner-edited
        Service->>AI: Rewrite deterministic draft
        AI-->>Service: Human-readable release note
        Service->>DB: Save generated summary
    else ownerEditedAt is set
        Note over Service: Preserve owner-authored content
    end

    Owner->>Service: publish(entryId)
    Service->>DB: Publish entry and release windowKey
    Note over DB: Later same-day changes create a new draft
```

Usage

```
import { ChangelogDraftsService } from './changelog-drafts.service';

@Injectable()
export class ReleaseWebhookHandler {
  constructor(
    private readonly changelogDraftsService: ChangelogDraftsService,
  ) {}

  async handleReleasePublished(projectId: string, release: Release) {
    // Safe to await or intentionally fire-and-forget:
    // recordChange catches and logs its own failures.
    await this.changelogDraftsService.recordChange({
      projectId,
      type: 'release_published',
      title: `Published ${release.version}`,
      details: {
        releaseId: release.id,
        version: release.version,
        artifactCount: release.artifactCount,
      },
    });

    // Continue normal release processing regardless of changelog status.
  }
}

// Owner-facing management flow
const drafts = await changelogDraftsService.listEntries(projectId);
```

```
const entry = await changelogDraftsService.getEntry(drafts[0].id);

await changelogDraftsService.updateEntry(entry.id, {
  title: 'Version 2.4.0',
  body: 'Improved deployment reliability and added release notifications.',
});

// Owner edits are protected from later AI summarization.
await changelogDraftsService.publish(entry.id);
```

AI Coding Instructions

- Treat `recordChange()` as a non-blocking reliability boundary: catch and log all internal failures, and never allow changelog errors to fail hook callers.
- Preserve daily auto-draft batching with the `auto-YYYY-MM-DD` UTC `windowKey`; rely on the database unique constraint for concurrent event safety.
- Never overwrite content after `ownerEditedAt` is set. AI summarization may only update untouched draft entries.
- Keep publishing behavior distinct from drafting: publishing must free the active window key so subsequent same-day events create a new draft.
- Route summarization through the BullMQ `changelog` queue when available, while maintaining the inline fallback for environments without queue processing.

Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `ChangelogQueue`
- DEPENDS_ON → `ChangelogSubscriptionsService`
- DEPENDS_ON → `aiprovider`
- DEPENDS_ON → `AIUsageService`

Referenced By

- `ChangelogController` (DEPENDS_ON)
- `ChangelogModule` (MODULE_PROVIDES)
- `ChangelogModule` (MODULE_EXPORTS)
- `DocVersionService` (DEPENDS_ON)
- `DocsPrService` (DEPENDS_ON)

- `DocAutomationService` (DEPENDS_ON)
- `TechnicalDocsMaterializerService` (DEPENDS_ON)