ATLORIA (SELF DOGFOOD)

DocVersionExportService

September 4, 2026

DocVersionExportService

Kind: Service

Source: `atloria-monorepo/apps/api/src/doc-version/export.service.ts`

`DocVersionExportService` generates exportable representations of a documentation version, including Markdown, static-site content, and PDF artifacts. It also manages cached public manual PDFs, asynchronous rendering state, artifact prerendering, and cleanup for version-specific exports.

Methods

Method	Signature	Returns	Description
`exportMarkdown`	`exportMarkdown(versionId: string)`	`unknown`	Export a doc version as a collection of markdown files (returns JSON structure).
`exportStaticSite`	`exportStaticSite(versionId: string)`	`unknown`	Export as static HTML site structure.
`exportPDFContent`	`exportPDFContent(versionId: string)`	`unknown`	Export as simple PDF-ready HTML (single page).
`exportManualPdfForVersion`	`exportManualPdfForVersion(versionId: string)`	`Promise<{ buffer: Buffer; filename: string }>`	
`exportPDFBuffer`	`exportPDFBuffer(versionId: string)`	`Promise<{ buffer: Buffer;`	Render the doc version to a REAL application/pdf

Method	Signature	Returns	Description
		`filename: string }>`	binary (owner route).
`exportSinglePagePdf`	`exportSinglePagePdf(documentId: string)`	`Promise<{ buffer: Buffer; filename: string }>`	Single published page → PDF (C6 per-page export, all plans).
`getOrRenderPublicManualPdf`	`getOrRenderPublicManualPdf(versionId: string)`	`Promise<{ status: 'ready'; url: string } | { status: 'rendering' }>`	
`prerenderVersionArtifacts`	`prerenderVersionArtifacts(versionId: string)`	`Promise<void>`	Activation pre-render hook (C6): warm the whole-manual PDF for a version's (scope, lang) — a DocVersion has exactly one of each — so the reader's Download bu…
`clearExportArtifacts`	`clearExportArtifacts(docVersionId: string)`	`Promise<void>`	Invalidation (C6): a delta patched the ACTIVE version's pages in place (S3 screenshot deltas / S4 tech-docs deltas / asset swaps), so any stored manual PDF i…

Dependencies

- `PrismaService`

- `ModuleRef` *(optional)*
- `BrandingService` *(optional)*
- `AzureBlobService` *(optional)*
- `PdfExportLockService` *(optional)*
- `SnippetsService` *(optional)*

Where it refuses work

- `DocVersionExportService` stops the work with `PdfWorkerUnavailableError` when `!blob` — “Blob storage unavailable — cannot persist the manual PDF”.
- `DocVersionExportService` stops the work with an early return when `!this.snippetsService`.
- `DocVersionExportService` stops the work with an early return when `this.brandingService`.
- `DocVersionExportService` stops the work with an early return when `this.azureBlob`.
- `DocVersionExportService` stops the work with an early return when `cached?.blobUrl`.
- `DocVersionExportService` stops the work with an early return when `this.exportLock && token === null`.

When something fails

- `DocVersionExportService` handles failure in 4 places: it turns it into a return value in 2, logs it and continues in 1, and lets it reach the caller in 1.

Diagram

```
sequenceDiagram
    participant Client
    participant Service as DocVersionExportService
    participant Renderer as PDF/Static Renderer
    participant Storage as Artifact Storage

    Client->>Service: exportManualPdfForVersion(version)
    Service->>Service: Build version export content
    Service->>Renderer: Render PDF content
    Renderer-->>Service: PDF buffer
    Service->>Storage: Store/export artifact
    Storage-->>Service: Artifact location
    Service-->>Client: { buffer, filename }

    Client->>Service: getOrRenderPublicManualPdf(version)
    Service->>Storage: Check cached public PDF
    alt Artifact exists
        Storage-->>Service: Public URL
        Service-->>Client: { status: "ready", url }
    else Artifact missing
        Service->>Renderer: Queue/render PDF artifact
```

```
        Service-->>Client: { status: "rendering" }
    end
```

Usage

```
import { DocVersionExportService } from './doc-version/export.service';

@Injectable()
export class ManualDownloadService {
  constructor(
    private readonly exportService: DocVersionExportService,
  ) {}

  async downloadVersionPdf(versionId: string) {
    const { buffer, filename } =
      await this.exportService.exportManualPdfForVersion(versionId);

    return {
      filename,
      contentType: 'application/pdf',
      body: buffer,
    };
  }

  async getPublicManualPdf(versionId: string) {
    return this.exportService.getOrRenderPublicManualPdf(versionId);
  }

  async refreshVersionArtifacts(versionId: string) {
    await this.exportService.clearExportArtifacts(versionId);
    await this.exportService.prerenderVersionArtifacts(versionId);
  }
}
```

AI Coding Instructions

- Keep all export generation routed through `DocVersionExportService`; avoid duplicating Markdown, static-site, or PDF rendering logic in controllers.
- Use `getOrRenderPublicManualPdf()` for public PDF access because it handles cached artifacts and returns either a ready URL or rendering state.
- Use buffer-returning methods such as `exportPDFBuffer()` and `exportSinglePagePdf()` when an HTTP response needs to stream or download a PDF directly.
- Clear existing artifacts with `clearExportArtifacts()` before forcing regeneration to prevent stale version content from being served.
- Treat PDF rendering as potentially asynchronous and handle the `{ status: 'rendering' }` response in API consumers or UI polling flows.

Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `moduleref`
- DEPENDS_ON → `BrandingService`
- DEPENDS_ON → `AzureBlobService`
- DEPENDS_ON → `PdfExportLockService`
- DEPENDS_ON → `SnippetsService`

Referenced By

- `DocVersionController` (DEPENDS_ON)
- `DocVersionModule` (MODULE_PROVIDES)
- `DocVersionModule` (MODULE_EXPORTS)
- `DocVersionService` (DEPENDS_ON)
- `DocAutomationService` (DEPENDS_ON)
- `PublicProjectController` (DEPENDS_ON)
- `TechnicalDocsMaterializerService` (DEPENDS_ON)