ParserRegistryService

September 4, 2026

ParserRegistryService

Kind: Service

Source: `atloria-monorepo/apps/config-docs-parser/src/services/parser-registry.service.ts`

`ParserRegistryService` is a NestJS service that manages the configured set of `IParser` implementations used to parse configuration documentation files. It initializes and disposes parser resources during module lifecycle events, discovers parsers that support a file, selects the best match, and delegates parsing to that parser.

Methods

Method	Signature	Returns	Description
`onModuleInit`	`onModuleInit()`	`Promise<void>`	
`onModuleDestroy`	`onModuleDestroy()`	`Promise<void>`	
`configure`	`configure(config: ParserRegistryConfig)`	`Promise<void>`	Configure and initialize the registry with custom settings
`getAllParsers`	`getAllParsers()`	`IParser[]`	Get all registered parsers
`getParser`	`getParser(name: string)`	`IParser	undefined`	
`hasParser`	`hasParser(name: string)`	`boolean`	Check if a parser is registered
`getParserNames`	`getParserNames()`	`string[]`	Get parser names
`findParsersForFile`	`findParsersForFile(filePath: string)`	`IParser[]`	Find all parsers that

Method	Signature	Returns	Description
			can handle a given file
`findBestParser`	`findBestParser(filePath: string)`	`IParser	undefined`	
`parseFile`	`parseFile(filePath: string, content: string)`	`Promise<ParseResult>`	Parse a file using the best available parser
`parseFiles`	`parseFiles(files: ParseFileRequest[])`	`Promise<ParseResult[]>`	Parse multiple files, automatically selecting the correct parser for each
`getAllSupportedPatterns`	`getAllSupportedPatterns()`	`string[]`	Get all supported file patterns across all parsers
`getStats`	`getStats()`	`{ parserCount: number; parsers: Array<{ name: string; version: string; patterns: string[] }>; totalPatterns: number; }`	Get registry statistics
`destroyAll`	`destroyAll()`	`Promise<void>`	Destroy all parsers and clear the registry
`isInitialized`	`isInitialized()`	`boolean`	Check if the registry is initialized

Dependencies

- `MarkdownParser`
- `MdxParser`
- `EnvParser`
- `YamlParser`
- `JsonConfigParser`

Where it refuses work

- `ParserRegistryService` stops the work with `Error` when `!this.initialized` — “ParserRegistry not initialized. Call configure() first.”, in 2 places.
- `ParserRegistryService` stops the work with an early return when `!parser`.

Diagram

```
sequenceDiagram
    participant App as NestJS Module
    participant Registry as ParserRegistryService
    participant Parser as IParser
    participant File as Source File

    App->>Registry: onModuleInit()
    Registry->>Registry: configure()
    Registry->>Parser: initialize/configure parsers

    App->>Registry: parseFile(file)
    Registry->>Registry: findParsersForFile(file)
    Registry->>Registry: findBestParser(parsers)
    Registry->>Parser: parse(file)
    Parser->>File: read and analyze content
    Parser-->>Registry: ParseResult
    Registry-->>App: ParseResult

    App->>Registry: onModuleDestroy()
    Registry->>Parser: dispose/cleanup resources
```

Usage

```
import { Injectable } from '@nestjs/common';
import { ParserRegistryService } from './services/parser-registry.service';

@Injectable()
export class DocumentationImportService {
  constructor(
    private readonly parserRegistry: ParserRegistryService,
  ) {}

  async importFile(filePath: string) {
    const parser = this.parserRegistry.findBestParser(filePath);

    if (!parser) {
      throw new Error(
        `No parser is registered for "${filePath}". Available parsers: ${this.parserRegistry
          .getParserNames()
          .join(', ')}`,
```

```
      );
    }

    const result = await this.parserRegistry.parseFile(filePath);

    return result;
  }
}
```

AI Coding Instructions

- Use `ParserRegistryService` as the single entry point for parser discovery and file parsing; avoid directly selecting parser implementations in consumers.
- Ensure new `IParser` implementations are registered through the service configuration flow so they are available after `onModuleInit()`.
- Call `findBestParser()` or `hasParser()` before parsing when handling optional or user-provided file types.
- Preserve parser selection behavior when adding parsers, especially when multiple parsers may support the same file extension or format.
- Do not manually invoke lifecycle methods; NestJS calls `onModuleInit()` and `onModuleDestroy()` as part of module startup and shutdown.

Relationships

- DEPENDS_ON → `markdownparser`
- DEPENDS_ON → `mdxparser`
- DEPENDS_ON → `envparser`
- DEPENDS_ON → `yamlparser`
- DEPENDS_ON → `jsonconfigparser`

Referenced By

- `AppModule` (MODULE_PROVIDES)
- `AppModule` (MODULE_EXPORTS)