IntegrationService

September 4, 2026

IntegrationService

Kind: Service

Source: `atloria-monorepo/apps/parser-orchestrator/src/integration/integration.service.ts`

`IntegrationService` manages the lifecycle and selection of parser integrations within the parser orchestrator. It registers and initializes parsers, resolves the appropriate parser for a file based on supported patterns, coordinates single or batch parsing, and safely tears down parser instances during shutdown or replacement.

Methods

Method	Signature	Returns	Description
`registerParser`	`registerParser(parser: IParser)`	`Promise<void>`	Register a parser implementation
`initializeParser`	`initializeParser(parserName: string, config: ParserConfig)`	`Promise<void>`	Initialize a registered parser
`getParser`	`getParser(parserName: string)`	`IParser	null`	
`hasParser`	`hasParser(parserName: string)`	`boolean`	Check if a parser is registered
`getRegisteredParsers`	`getRegisteredParsers()`	`Array<{ name: string; version: string; patterns: string[]; initialized: boolean;`	

}> | Get all registered parsers | | findParserForFile | findParserForFile(filePath: string) | string | null | Find parser for a file | | parseFile | parseFile(filePath: string, content: string) | Promise<ParseResult | null> | Parse a file using the appropriate parser | | parseFiles | parseFiles(files: string[]) | Promise<ParseResult[]> | Parse multiple files using appropriate parsers | | destroyParser | destroyParser(parserName: string) | Promise | Destroy a parser and release resources |
destroyAll	destroyAll()	Promise	Destroy all parsers
generateEntityId	generateEntityId(type: string, name: string, filePath: string)	string	Generate unique entity ID
generateRelationshipId	generateRelationshipId(sourceId: string, type: string, targetId: string)	string	Generate unique relationship ID
validateEntity	validateEntity(entity: Partial)	{ valid: boolean; errors: string[] }	Validate an entity structure		validateRelationship	validateRelationship(relationship: Partial)	{ valid: boolean; errors: string[] }`	Validate a relationship structure

Where it refuses work

- `IntegrationService` stops the work with `Error` when `!entry`.
- `IntegrationService` stops the work with an early return when `!entry`, in 2 places.
- `IntegrationService` stops the work with an early return when `entry.parser.canParse(filePath)`.

Diagram

```
sequenceDiagram
  participant Client
  participant Service as IntegrationService
  participant Parser as IParser

  Client->>Service: registerParser(parser)
  Service->>Service: Store parser registration

  Client->>Service: initializeParser(name)
  Service->>Parser: initialize()
  Parser-->>Service: Ready

  Client->>Service: findParserForFile(filePath)
  Service->>Service: Match file path against parser patterns
  Service-->>Client: parser name | null

  Client->>Service: parseFile(filePath)
  Service->>Service: Resolve matching parser
  Service->>Parser: parse(filePath)
```

```
  Parser-->>Service: ParseResult
  Service-->>Client: ParseResult | null

  Client->>Service: destroyAll()
  Service->>Parser: destroy()
  Parser-->>Service: Cleanup complete
```

Usage

```typescript
import { IntegrationService } from './integration/integration.service';

// Typically injected through NestJS dependency injection.
async function parseSourceFile(
  integrationService: IntegrationService,
  filePath: string,
) {
  // Parsers should be registered during application bootstrap.
  await integrationService.registerParser({
    name: 'typescript-parser',
    version: '1.0.0',
    patterns: ['**/*.ts', '**/*.tsx'],
    initialize: async () => {
      // Initialize parser resources.
    },
    parse: async (file) => ({
      file,
      // Parser-specific result fields.
    }),
    destroy: async () => {
      // Release parser resources.
    },
  });

  await integrationService.initializeParser('typescript-parser');

  const parserName = integrationService.findParserForFile(filePath);

  if (!parserName) {
    return null;
  }

  return integrationService.parseFile(filePath);
}

// During application shutdown:
async function shutdown(integrationService: IntegrationService) {
  await integrationService.destroyAll();
}
```

AI Coding Instructions

- Register parsers before attempting initialization or parsing; use `hasParser()` and `getParser()` when validating parser availability.
- Ensure parser implementations expose accurate file-matching patterns, since `findParserForFile()` determines which integration handles a file.
- Handle the `null` result from `findParserForFile()` and `parseFile()` for unsupported files or unavailable parser integrations.
- Prefer `parseFiles()` for batch workloads so parser selection and result collection remain centralized in the service.
- Always call `destroyParser()` or `destroyAll()` during shutdown, parser replacement, or test cleanup to release parser resources.

Referenced By

- `IntegrationModule` (MODULE_PROVIDES)
- `IntegrationModule` (MODULE_EXPORTS)