Namespace NotebookAutomation.Core.Configuration

Classes

AIServiceConfig
: Represents the configuration for AI services, supporting multiple providers such as OpenAI, Azure, and Foundry.

AppConfig
: Represents the application configuration for Notebook Automation.

AzureProviderConfig
: Represents the configuration for the Azure provider.

ConfigurationExtensions
: Provides extension methods for IConfiguration and ConfigurationBuilder.

ConfigurationSetup
: Provides methods to set up application configuration with support for various sources, including JSON files, environment variables, and user secrets.

FailedOperations
: Contains methods and constants for managing and recording failed operations.

FoundryProviderConfig
: Represents the configuration for the Foundry provider.

LoggingService
: Provides centralized logging capabilities for the notebook automation system.

MicrosoftGraphConfig
: Configuration for Microsoft Graph API.

OpenAiProviderConfig
: Represents the configuration for the OpenAI provider.

PathsConfig
: Represents the configuration settings for various file paths used in the application.

ServiceRegistration

UserSecretsHelper
: Provides convenient access to user secrets in the application.

Interfaces

ILoggingService

Defines the contract for a centralized logging service for the notebook automation system.

Class AIServiceConfig

Namespace: NotebookAutomation.Core.Configuration

Assembly: NotebookAutomation.Core.dll

Represents the configuration for AI services, supporting multiple providers such as OpenAI, Azure, and Foundry.

```
public class AIServiceConfig
```

Inheritance

object ← AIServiceConfig

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

API keys are not stored in configuration files. Use environment variables or user-secrets:

- OpenAI: OPENAI_API_KEY
- Azure: AZURE_OPEN_AI_API_KEY
- Foundry: FOUNDRY_API_KEY

Properties

ApiKey

Gets the API key for the active provider.

```
[JsonPropertyName("api_key")]
[JsonIgnore]
public string? ApiKey { get; }
```

Property Value

string

Remarks

This allows direct access via indexer like appConfig["aiservice:api_key"].

Azure

Gets or sets the configuration for Azure provider.

```
[JsonPropertyName("azure")]
public AzureProviderConfig? Azure { get; set; }
```

Property Value

AzureProviderConfig

Foundry

Gets or sets the configuration for Foundry provider.

```
[JsonPropertyName("foundry")]
public FoundryProviderConfig? Foundry { get; set; }
```

Property Value

FoundryProviderConfig

Model

Gets the model for the active provider.

```
[JsonIgnore]
public string? Model { get; }
```

Property Value

string

Remarks

This allows direct access via indexer like appConfig["aiservice:Model"].

OpenAI

Gets or sets the configuration for OpenAI provider.

```
[JsonPropertyName("openai")]
public OpenAiProviderConfig? OpenAI { get; set; }
```

Property Value

OpenAiProviderConfig

Provider

Gets or sets the provider name (e.g., OpenAI, Azure, Foundry).

```
[JsonPropertyName("provider")]
public string? Provider { get; set; }
```

Property Value

string

Methods

GetApiKey()

Returns the API key for the configured AI provider.

```
public string? GetApiKey()
```

Returns

string

The API key string, or null if not set.

Class AppConfig

Namespace: NotebookAutomation.Core.Configuration

Assembly: NotebookAutomation.Core.dll

Represents the application configuration for Notebook Automation.

```
public class AppConfig : IConfiguration
```

Inheritance

object ← AppConfig

Implements

IConfiguration

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

This class handles loading and providing access to centralized configuration settings for the Notebook Automation system, including paths, API settings, and application defaults.

Constructors

AppConfig()

Initializes a new instance of the AppConfig class. Default constructor for manual initialization.

```
public AppConfig()
```

AppConfig(IConfiguration, ILogger<AppConfig>, string?, bool)

Initializes a new instance of the AppConfig class. Constructor with dependency injection for configuration and logging.

```
public AppConfig(IConfiguration configuration, ILogger<AppConfig> logger, string?
configFilePath = null, bool debugEnabled = false)
```

Parameters

`configuration` IConfiguration

The configuration to use.

`logger` ILogger<AppConfig>

The logger to use.

`configFilePath` string

The path to the configuration file.

`debugEnabled` bool

Whether debug mode is enabled.

Properties

AiService

Gets or sets the AI Service configuration section.

```
[JsonPropertyName("aiservice")]
public AIServiceConfig AiService { get; set; }
```

Property Value

AIServiceConfig

ConfigFilePath

Gets or sets the path to the configuration file used to load this AppConfig.

```
public string? ConfigFilePath { get; set; }
```

Property Value

string

DebugEnabled

Gets or sets a value indicating whether debug mode is enabled for this configuration.

```
public bool DebugEnabled { get; set; }
```

Property Value

bool

this[string]

Gets or sets a configuration value for the specified key.

```
public string? this[string key] { get; set; }
```

Parameters

`key` string

The key of the configuration value to get or set.

Property Value

string

The configuration value.

MicrosoftGraph

Gets or sets the Microsoft Graph API configuration section.

```
[JsonPropertyName("microsoft_graph")]
```

```
public MicrosoftGraphConfig MicrosoftGraph { get; set; }
```

Property Value

MicrosoftGraphConfig

Paths

Gets or sets the paths configuration section.

```
[JsonPropertyName("paths")]
public PathsConfig Paths { get; set; }
```

Property Value

PathsConfig

PdfExtensions

Gets or sets the list of PDF file extensions to process.

```
[JsonPropertyName("pdf_extensions")]
public List<string> PdfExtensions { get; set; }
```

Property Value

List <string>

VideoExtensions

Gets or sets the list of video file extensions to process.

```
[JsonPropertyName("video_extensions")]
public List<string> VideoExtensions { get; set; }
```

Property Value

List<string>

Methods

Exists(string)

Gets a value indicating if this configuration contains the specified key.

```
public bool Exists(string key)
```

Parameters

`key` string

The key to check.

Returns

bool

True if the configuration contains the specified key, otherwise false.

FindConfigFile(string)

Attempts to find the configuration file in standard locations.

```
public static string FindConfigFile(string configFileName = "config.json")
```

Parameters

`configFileName` string

Name of the configuration file to find.

Returns

string

Path to the configuration file if found, otherwise null.

GetChildren()

Gets the immediate descendant configuration sub-sections.

```
public IEnumerable<IConfigurationSection> GetChildren()
```

Returns

IEnumerable<IConfigurationSection>

The configuration sub-sections.

GetReloadToken()

Gets a change token that can be used to observe when this configuration is reloaded.

```
public IChangeToken GetReloadToken()
```

Returns

IChangeToken

A change token.

GetSection(string)

Gets a configuration sub-section with the specified key.

```
public IConfigurationSection GetSection(string key)
```

Parameters

`key` string

The key of the configuration section.

Returns

IConfigurationSection

The configuration sub-section.

LoadFromJsonFile(string)

Loads configuration from the specified JSON file.

```
public static AppConfig LoadFromJsonFile(string configPath)
```

Parameters

`configPath` string

Path to the configuration JSON file.

Returns

AppConfig

The loaded AppConfig instance.

SaveToJsonFile(string)

Saves the current configuration to the specified JSON file.

```
public void SaveToJsonFile(string configPath)
```

Parameters

`configPath` string

Path where the configuration should be saved.

Exceptions

IOException

Thrown when the file cannot be written to.

SetPdfExtensions(List<string>)

Sets the PDF file extensions to process.

```
public void SetPdfExtensions(List<string> list)
```

Parameters

`list` List<string>

List of PDF file extensions.

SetVideoExtensions(List<string>)

Sets the video file extensions to process.

```
public void SetVideoExtensions(List<string> list)
```

Parameters

`list` List<string>

List of video file extensions.

Class AzureProviderConfig

Namespace: NotebookAutomation.Core.Configuration

Assembly: NotebookAutomation.Core.dll

Represents the configuration for the Azure provider.

```
public class AzureProviderConfig
```

Inheritance

object ← AzureProviderConfig

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Deployment

Gets or sets the deployment name for the Azure OpenAI API.

```
[JsonPropertyName("deployment")]
public string? Deployment { get; set; }
```

Property Value

string

Endpoint

Gets or sets the endpoint URL for the Azure OpenAI API.

```
[JsonPropertyName("endpoint")]
public string? Endpoint { get; set; }
```

Property Value

string

Model

Gets or sets the model name for the Azure OpenAI API.

```
[JsonPropertyName("model")]
public string? Model { get; set; }
```

Property Value

string

Class ConfigurationExtensions

Namespace: NotebookAutomation.Core.Configuration

Assembly: NotebookAutomation.Core.dll

Provides extension methods for IConfiguration and ConfigurationBuilder.

```
public static class ConfigurationExtensions
```

Inheritance

object ← ConfigurationExtensions

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

This class includes utility methods to simplify the process of adding custom objects as configuration sources and converting objects into key-value pairs for configuration. It is designed to handle both simple and complex objects, supporting nested properties and ensuring compatibility with the IConfiguration interface.

Methods

AddObject(IConfigurationBuilder, object)

Adds an object as a configuration source.

```
public static IConfigurationBuilder AddObject(this IConfigurationBuilder
configurationBuilder, object obj)
```

Parameters

`configurationBuilder` IConfigurationBuilder

The configuration builder to add to.

`obj` object

The object to serialize as configuration values.

Returns

IConfigurationBuilder

The same configuration builder.

Remarks

This method serializes the provided object into key-value pairs and adds them to the configuration builder as an in-memory collection.

Class ConfigurationSetup

Namespace: NotebookAutomation.Core.Configuration

Assembly: NotebookAutomation.Core.dll

Provides methods to set up application configuration with support for various sources, including JSON files, environment variables, and user secrets.

```
public static class ConfigurationSetup
```

Inheritance

object ← ConfigurationSetup

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

This class simplifies the process of building a configuration for the application, ensuring compatibility with different environments (e.g., development, production). It supports optional user secrets and config file paths, making it flexible for various deployment scenarios.

Methods

BuildConfiguration(string, string?, string?)

Creates a standard configuration with support for config files and user secrets.

```
public static IConfiguration BuildConfiguration(string environment = "Development", string?
userSecretsId = null, string? configPath = null)
```

Parameters

`environment` string

The current environment (development, production, etc.)

`userSecretsId` string

Optional user secrets ID. If null, will attempt to use assembly-defined ID.

`configPath` string

Optional path to the config file. If null, will search for config.json in standard locations.

Returns

IConfiguration

A configured IConfiguration instance.

BuildConfiguration<T>(string, string?)

Creates a configuration with user secrets support for the given assembly type.

```
public static IConfiguration BuildConfiguration<T>(string environment = "Development",
string? configPath = null) where T : class
```

Parameters

`environment` string

The current environment (development, production, etc.)

`configPath` string

Optional path to the config file. If null, will search for config.json in standard locations.

Returns

IConfiguration

A configured IConfiguration instance.

Type Parameters

`T`

The type from the assembly that has the UserSecretsId attribute.

Class FailedOperations

Namespace: NotebookAutomation.Core.Configuration

Assembly: NotebookAutomation.Core.dll

Contains methods and constants for managing and recording failed operations.

```
public static class FailedOperations
```

Inheritance

object ← FailedOperations

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Examples

Example of using a failed logger to record a failed operation:

```
try
{
    // Perform operation
    await ProcessFile(filePath);
}
catch (Exception ex)
{
    failedLogger.LogError("Failed to process file: {Path}. Error: {Error}",
filePath, ex.Message);
}
```

Remarks

The FailedOperations class provides a centralized way to handle operations that fail during execution. It works in conjunction with specialized loggers created by the LoggingService to record detailed information about failures in a consistent format.

This class is primarily used as a category name for specialized loggers and as a container for constants and static methods related to failed operations.

When operations fail, they should be logged using a failed logger (created with LoggingService.CreateFailedLogger) to ensure consistent tracking and reporting of failures throughout the application.

Methods

RecordFailedFileOperation(ILogger, string, string, Exception)

Records a failed file operation using the provided logger.

```
public static void RecordFailedFileOperation(ILogger failedLogger, string filePath, string operationName, Exception exception)
```

Parameters

`failedLogger` ILogger

The logger to record the failure with.

`filePath` string

The path to the file that failed to process.

`operationName` string

The name of the operation that failed.

`exception` Exception

The exception that caused the failure.

Remarks

This method provides a standardized way to log failed file operations, ensuring consistent formatting and detail level across the application.

The method logs the failure at the Error level, including the file path, operation name, and exception details (message and stack trace).

RecordFailedFileOperation(ILogger, string, string, string)

Records a failed file operation with a custom error message.

```
public static void RecordFailedFileOperation(ILogger failedLogger, string filePath, string
operationName, string errorMessage)
```

Parameters

`failedLogger` ILogger

The logger to record the failure with.

`filePath` string

The path to the file that failed to process.

`operationName` string

The name of the operation that failed.

`errorMessage` string

A custom error message describing the failure.

Remarks

This overload is useful when you want to provide a custom error message rather than recording an exception's details. This is common when operations fail for logical reasons rather than due to exceptions.

The method logs the failure at the Error level, including the file path, operation name, and the provided error message.

Class FoundryProviderConfig

Namespace: NotebookAutomation.Core.Configuration

Assembly: NotebookAutomation.Core.dll

Represents the configuration for the Foundry provider.

```
public class FoundryProviderConfig
```

Inheritance

object ← FoundryProviderConfig

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Endpoint

Gets or sets the endpoint URL for the Foundry API.

```
[JsonPropertyName("endpoint")]
public string? Endpoint { get; set; }
```

Property Value

string

Model

Gets or sets the model name for the Foundry API.

```
[JsonPropertyName("model")]
public string? Model { get; set; }
```

Property Value

string

Interface ILoggingService

Namespace: NotebookAutomation.Core.Configuration

Assembly: NotebookAutomation.Core.dll

Defines the contract for a centralized logging service for the notebook automation system.

```
public interface ILoggingService
```

Remarks

The ILoggingService interface provides methods for creating appropriately configured ILogger instances for different parts of the application.

Properties

CurrentLogFilePath

Gets the full path to the current log file.

```
string? CurrentLogFilePath { get; }
```

Property Value

string

The absolute path to the current log file, or null if logging is not configured to a file.

FailedLogger

Gets the specialized logger instance used for recording failed operations (Microsoft.Extensions.Logging).

```
ILogger FailedLogger { get; }
```

Property Value

ILogger

Logger

Gets the main logger instance used for general application logging (Microsoft.Extensions.Logging).

```
ILogger Logger { get; }
```

Property Value

ILogger

Methods

ConfigureLogging(ILoggingBuilder)

Configures the logging builder with the appropriate providers.

```
void ConfigureLogging(ILoggingBuilder builder)
```

Parameters

`builder` ILoggingBuilder

The logging builder to configure.

GetLogger<T>()

Gets a typed ILogger instance for the specified type T from this LoggingService instance.

```
ILogger<T> GetLogger<T>()
```

Returns

ILogger<T>

An ILogger{T} configured for the specified type.

Type Parameters

T

The type to create the logger for.

Class LoggingService

Namespace: NotebookAutomation.Core.Configuration

Assembly: NotebookAutomation.Core.dll

Provides centralized logging capabilities for the notebook automation system.

```
public class LoggingService : ILoggingService
```

Inheritance

object ← LoggingService

Implements

ILoggingService

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

The LoggingService class offers a robust logging infrastructure for the application, supporting both console and file-based logging. It provides factory methods for creating loggers tailored to specific application needs, including general-purpose loggers and specialized loggers for failed operations.

Key features include:

- Support for Serilog-based logging with configurable levels
- Thread-safe initialization of logging resources
- Fallback mechanisms for console logging in case of initialization failures
- Integration with Microsoft.Extensions.Logging for typed loggers

Constructors

LoggingService(string, bool)

Provides centralized logging capabilities for the notebook automation system.

```
public LoggingService(string loggingDir, bool debug = false)
```

Parameters

`loggingDir` string

The directory where log files should be stored.

`debug` bool

Whether debug mode is enabled.

Remarks

The LoggingService class offers a robust logging infrastructure for the application, supporting both console and file-based logging. It provides factory methods for creating loggers tailored to specific application needs, including general-purpose loggers and specialized loggers for failed operations.

Key features include:

- Support for Serilog-based logging with configurable levels
- Thread-safe initialization of logging resources
- Fallback mechanisms for console logging in case of initialization failures
- Integration with Microsoft.Extensions.Logging for typed loggers

Properties

CurrentLogFilePath

Gets the full path to the current log file.

```
public string? CurrentLogFilePath { get; }
```

Property Value

string

The absolute path to the current log file, or null if logging is not configured to a file.

FailedLogger

Gets the specialized logger instance used for recording failed operations.

```
public ILogger FailedLogger { get; }
```

Property Value

ILogger

Logger

Gets the main logger instance used for general application logging.

```
public ILogger Logger { get; }
```

Property Value

ILogger

Methods

ConfigureLogging(ILoggingBuilder)

Configures the logging builder with the appropriate providers asynchronously.

```
public void ConfigureLogging(ILoggingBuilder builder)
```

Parameters

`builder` ILoggingBuilder

The logging builder to configure.

GetLogger<T>()

Gets a typed ILogger instance for the specified type T from this LoggingService instance.

```
public virtual ILogger<T> GetLogger<T>()
```

Returns

ILogger<T>

An ILogger{T} configured for the specified type.

Type Parameters

T

The type to create the logger for.

Remarks

This is an instance method for creating a typed logger, which uses the type name as the category name. This is the preferred way to create loggers for classes when you have a LoggingService instance.

InitializeLogging()

Initializes the logging infrastructure.

```
protected virtual void InitializeLogging()
```

Class MicrosoftGraphConfig

Namespace: NotebookAutomation.Core.Configuration

Assembly: NotebookAutomation.Core.dll

Configuration for Microsoft Graph API.

```
public class MicrosoftGraphConfig
```

Inheritance

object ← MicrosoftGraphConfig

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

This class encapsulates the necessary parameters for authenticating and accessing Microsoft Graph API resources, including client credentials, API endpoints, and scopes. It is designed to be serialized and deserialized from JSON configuration files.

Properties

ApiEndpoint

Gets or sets aPI endpoint for Microsoft Graph.

```
[JsonPropertyName("api_endpoint")]
public string ApiEndpoint { get; set; }
```

Property Value

string

Remarks

The API endpoint specifies the base URL for accessing Microsoft Graph services. Typically, this is "https://graph.microsoft.com/v1.0" for production environments.

Authority

Gets or sets authority URL for Microsoft Graph authentication.

```
[JsonPropertyName("authority")]
public string Authority { get; set; }
```

Property Value

string

Remarks

The authority URL is used to direct authentication requests to the appropriate Azure Active Directory. For example, "https://login.microsoftonline.com/common" for multi-tenant applications.

ClientId

Gets or sets client ID for authenticating with Microsoft Graph.

```
[JsonPropertyName("client_id")]
public string ClientId { get; set; }
```

Property Value

string

Remarks

The Client ID is a unique identifier assigned to the application by Azure Active Directory. It is used during the authentication process to identify the application.

Scopes

Gets or sets scopes required for Microsoft Graph API access.

```
[JsonPropertyName("scopes")]
public List<string> Scopes { get; set; }
```

Property Value

List<string>

Remarks

Scopes define the permissions that the application needs to access Microsoft Graph resources. Examples include "User.Read" and "Mail.Send".

TenantId

Gets or sets the tenant ID for Microsoft Graph authentication.

```
public string? TenantId { get; set; }
```

Property Value

string

Remarks

The tenant ID is used to identify the directory in Azure Active Directory that the application belongs to. It can be set to "common" for multi-tenant applications or a specific tenant ID for single-tenant applications.

Class OpenAiProviderConfig

Namespace: NotebookAutomation.Core.Configuration

Assembly: NotebookAutomation.Core.dll

Represents the configuration for the OpenAI provider.

```
public class OpenAiProviderConfig
```

Inheritance

object ← OpenAiProviderConfig

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Endpoint

Gets or sets the endpoint URL for the OpenAI API.

```
[JsonPropertyName("endpoint")]
public string? Endpoint { get; set; }
```

Property Value

string

Model

Gets or sets the model name for the OpenAI API.

```
[JsonPropertyName("model")]
public string? Model { get; set; }
```

Property Value

string

Class PathsConfig

Namespace: NotebookAutomation.Core.Configuration

Assembly: NotebookAutomation.Core.dll

Represents the configuration settings for various file paths used in the application.

```
public class PathsConfig
```

Inheritance

object ← PathsConfig

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

This class encapsulates paths for directories and files that are essential for the application's operation, including paths for OneDrive resources, notebook vaults, metadata files, logging, and prompt templates. It is designed to be serialized and deserialized from JSON configuration files.

Properties

LoggingDir

Gets or sets directory for log files.

```
[JsonPropertyName("logging_dir")]
public string LoggingDir { get; set; }
```

Property Value

string

Remarks

This directory is used to store application log files for debugging and monitoring purposes.

MetadataFile

Gets or sets path to the metadata file.

```
[JsonPropertyName("metadata_file")]
public string MetadataFile { get; set; }
```

Property Value

string

Remarks

The metadata file contains structured information about the application's resources.

NotebookVaultFullpathRoot

Gets or sets full path to the root directory for the notebook vault.

```
[JsonPropertyName("notebook_vault_fullpath_root")]
public string NotebookVaultFullpathRoot { get; set; }
```

Property Value

string

Remarks

This path is used to locate the directory where notebook vault files are stored. /// This property specifies the name of the folder that should be treated as the main program folder when generating index files. This folder will have template-type: main and the index will be named whatever is the folder name.

OnedriveFullpathRoot

Gets or sets full path to the root directory where OneDrive files are stored locally.

```
[JsonPropertyName("onedrive_fullpath_root")]
public string OnedriveFullpathRoot { get; set; }
```

Property Value

string

Remarks

This path is used to locate the local storage directory for OneDrive files.

OnedriveResourcesBasepath

Gets or sets base path for OneDrive resources.

```
[JsonPropertyName("onedrive_resources_basepath")]
public string OnedriveResourcesBasepath { get; set; }
```

Property Value

string

Remarks

This path is used to locate the base directory for OneDrive-related resources.

PromptsPath

Gets or sets directory containing prompt template files.

```
[JsonPropertyName("prompts_path")]
public string PromptsPath { get; set; }
```

Property Value

string

Remarks

This directory is used to store template files for generating prompts in the application.

Class ServiceRegistration

Namespace: NotebookAutomation.Core.Configuration

Assembly: NotebookAutomation.Core.dll

```
public static class ServiceRegistration
```

Inheritance

object ← ServiceRegistration

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

AddNotebookAutomationServices(IServiceCollection, IConfiguration, bool, string?)

Adds core notebook automation services to the dependency injection container.

```
public static IServiceCollection AddNotebookAutomationServices(this IServiceCollection
services, IConfiguration configuration, bool debug = false, string? configFilePath = null)
```

Parameters

`services` IServiceCollection

The service collection to configure.

`configuration` IConfiguration

The application configuration.

`debug` bool

Whether debug mode is enabled.

`configFilePath` string

Returns

IServiceCollection

The configured service collection.

Class UserSecretsHelper

Namespace: NotebookAutomation.Core.Configuration

Assembly: NotebookAutomation.Core.dll

Provides convenient access to user secrets in the application.

```
public class UserSecretsHelper
```

Inheritance

object ← UserSecretsHelper

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

The `UserSecretsHelper` class simplifies the retrieval of sensitive information stored in user secrets, such as API keys and client credentials. It leverages the application's configuration system to access these secrets securely.

Constructors

UserSecretsHelper(IConfiguration)

Provides convenient access to user secrets in the application.

```
public UserSecretsHelper(IConfiguration configuration)
```

Parameters

`configuration` IConfiguration

The configuration to use for accessing user secrets.

Remarks

The `UserSecretsHelper` class simplifies the retrieval of sensitive information stored in user secrets, such as API keys and client credentials. It leverages the application's configuration system to access these secrets securely.

Methods

GetMicrosoftGraphClientId()

Gets a Microsoft Graph client ID from user secrets, if available.

```
public string? GetMicrosoftGraphClientId()
```

Returns

string

The client ID if found in user secrets; otherwise, null.

GetMicrosoftGraphTenantId()

Gets a Microsoft Graph tenant ID from user secrets, if available.

```
public string? GetMicrosoftGraphTenantId()
```

Returns

string

The tenant ID if found in user secrets; otherwise, null.

GetOpenAIApiKey()

Gets an OpenAI API key from user secrets, if available.

```
public string? GetOpenAIApiKey()
```

Returns

string

The API key if found in user secrets; otherwise, null.

GetSecret(string)

Gets any user secret by key.

```
public string? GetSecret(string key)
```

Parameters

`key` string

The key of the user secret to get.

Returns

string

The value if found; otherwise, null.

HasSecret(string)

Determines whether a specific user secret exists.

```
public bool HasSecret(string key)
```

Parameters

`key` string

The key of the user secret to check.

Returns

bool

True if the user secret exists; otherwise, false.

Namespace NotebookAutomation.Core.Models

Classes

DocumentProcessingProgressEventArgs

Represents the progress of document processing, including the current file and status.

QueueChangedEventArgs

Event arguments for queue changes.

QueueItem

Represents a document in the processing queue.

VaultFileInfo

Represents information about a file within a vault for index generation.

VaultIndexOptions

Options for vault index generation.

Enums

DocumentProcessingStatus

Represents the status of a document in the processing queue.

ProcessingStage

Represents the current processing stage of a document.

Class DocumentProcessingProgressEventArgs

Namespace: NotebookAutomation.Core.Models

Assembly: NotebookAutomation.Core.dll

Represents the progress of document processing, including the current file and status.

```
public class DocumentProcessingProgressEventArgs : EventArgs
```

Inheritance

object ← EventArgs ← DocumentProcessingProgressEventArgs

Inherited Members

EventArgs.Empty , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

Initializes a new instance of the DocumentProcessingProgressEventArgs class.

Constructors

DocumentProcessingProgressEventArgs(string, string, int, int)

Represents the progress of document processing, including the current file and status.

```
public DocumentProcessingProgressEventArgs(string filePath, string status, int currentFile,
int totalFiles)
```

Parameters

`filePath` string

The path of the file being processed.

`status` string

The current processing status message.

`currentFile` int

The current file index being processed.

`totalFiles` int

The total number of files to process.

Remarks

Initializes a new instance of the DocumentProcessingProgressEventArgs class.

Properties

CurrentFile

Gets the current file index being processed.

```
public int CurrentFile { get; }
```

Property Value

int

Remarks

This property indicates the index of the file currently being processed in the batch, starting from 1. It is useful for displaying progress to the user.

FilePath

Gets the path of the file being processed.

```
public string FilePath { get; }
```

Property Value

string

Remarks

This property provides the full path to the file currently being processed. It is useful for logging and tracking the progress of document processing tasks.

Status

Gets the current processing status message.

```
public string Status { get; }
```

Property Value

string

Remarks

This property contains a descriptive message about the current status of the document processing operation, such as "Processing" or "Completed".

TotalFiles

Gets the total number of files to process.

```
public int TotalFiles { get; }
```

Property Value

int

Remarks

This property specifies the total number of files in the batch that are being processed. It is useful for calculating the overall progress percentage.

Enum DocumentProcessingStatus

Namespace: NotebookAutomation.Core.Models

Assembly: NotebookAutomation.Core.dll

Represents the status of a document in the processing queue.

```
public enum DocumentProcessingStatus
```

Fields

`Completed = 2`

The document has been successfully processed.

`Failed = 3`

Processing the document failed.

`Processing = 1`

The document is currently being processed.

`Waiting = 0`

The document is waiting to be processed.

Enum ProcessingStage

Namespace: NotebookAutomation.Core.Models

Assembly: NotebookAutomation.Core.dll

Represents the current processing stage of a document.

```
public enum ProcessingStage
```

Fields

`AISummaryGeneration = 2`

Generating an AI summary of the document.

`Completed = 5`

Processing is complete.

`ContentExtraction = 1`

Extracting content from the document (PDF text or video transcription).

`MarkdownCreation = 3`

Creating a markdown note from the document.

`NotStarted = 0`

Processing has not yet started.

`ShareLinkGeneration = 4`

Generating OneDrive share links.

Class QueueChangedEventArgs

Namespace: NotebookAutomation.Core.Models

Assembly: NotebookAutomation.Core.dll

Event arguments for queue changes.

```
public class QueueChangedEventArgs : EventArgs
```

Inheritance

object ← EventArgs ← QueueChangedEventArgs

Inherited Members

EventArgs.Empty , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

The `QueueChangedEventArgs` class provides information about changes to the processing queue, including the current state of the queue and the item that was changed, if applicable.

Constructors

QueueChangedEventArgs(IReadOnlyList<QueueItem>, QueueItem?)

Event arguments for queue changes.

```
public QueueChangedEventArgs(IReadOnlyList<QueueItem> queue, QueueItem? changedItem = null)
```

Parameters

`queue` IReadOnlyList<QueueItem>

`changedItem` QueueItem

Remarks

The `QueueChangedEventArgs` class provides information about changes to the processing queue, including the current state of the queue and the item that was changed, if applicable.

Properties

ChangedItem

Gets the item that changed, if applicable.

```
public QueueItem? ChangedItem { get; }
```

Property Value

QueueItem

Remarks

This property identifies the specific item in the queue that triggered the event. It may be null if the event does not pertain to a specific item.

Queue

Gets the current state of the processing queue.

```
public IReadOnlyList<QueueItem> Queue { get; }
```

Property Value

IReadOnlyList <QueueItem>

Remarks

This property provides a snapshot of the queue's state at the time of the event.

Class QueueItem

Namespace: NotebookAutomation.Core.Models

Assembly: NotebookAutomation.Core.dll

Represents a document in the processing queue.

```
public class QueueItem
```

Inheritance

object ← QueueItem

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

The `QueueItem` class encapsulates information about a document that is queued for processing, including its file path, type, status, stage, and metadata. It provides properties to track the progress and state of the document throughout the processing lifecycle.

Constructors

QueueItem(string, string)

Represents a document in the processing queue.

```
public QueueItem(string filePath, string documentType)
```

Parameters

`filePath` string

The path of the file to be processed.

`documentType` string

The document type (e.g., "PDF", "VIDEO").

Remarks

The `QueueItem` class encapsulates information about a document that is queued for processing, including its file path, type, status, stage, and metadata. It provides properties to track the progress and state of the document throughout the processing lifecycle.

Properties

DocumentType

Gets the document type (e.g., "PDF", "VIDEO").

```
public string DocumentType { get; }
```

Property Value

string

Remarks

This property specifies the type of document being processed, which determines the processing logic to be applied.

FilePath

Gets the path of the file to be processed.

```
public string FilePath { get; }
```

Property Value

string

Remarks

This property specifies the full path to the file that is queued for processing.

Metadata

Gets document-specific metadata.

```
public Dictionary<string, object> Metadata { get; }
```

Property Value

Dictionary<string, object>

Remarks

This property contains additional metadata related to the document, such as extracted content or processing results.

ProcessingEndTime

Gets the time when processing completed for this file.

```
public DateTime? ProcessingEndTime { get; }
```

Property Value

DateTime?

Remarks

This property records the timestamp when processing finished for the file.

ProcessingStartTime

Gets the time when processing started for this file.

```
public DateTime? ProcessingStartTime { get; }
```

Property Value

DateTime?

Remarks

This property records the timestamp when processing began for the file.

Stage

Gets the current processing stage.

```
public ProcessingStage Stage { get; }
```

Property Value

ProcessingStage

Remarks

This property tracks the specific stage of processing, such as "ContentExtraction" or "MarkdownCreation".

Status

Gets the current processing status of the file.

```
public DocumentProcessingStatus Status { get; }
```

Property Value

DocumentProcessingStatus

Remarks

This property indicates the overall status of the file, such as "Waiting" or "Completed".

StatusMessage

Gets additional information about the file's current state.

```
public string StatusMessage { get; }
```

Property Value

string

Remarks

This property provides a descriptive message about the file's current processing state.

Class VaultFileInfo

Namespace: NotebookAutomation.Core.Models

Assembly: NotebookAutomation.Core.dll

Represents information about a file within a vault for index generation.

```
public class VaultFileInfo
```

Inheritance

object ← VaultFileInfo

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

ContentType

Gets or sets the content type (reading, video, transcript, etc.).

```
public string ContentType { get; set; }
```

Property Value

string

Course

Gets or sets the course name.

```
public string? Course { get; set; }
```

Property Value

string

FileName

Gets or sets the file name.

```
public string FileName { get; set; }
```

Property Value

string

FullPath

Gets or sets the full file path.

```
public string FullPath { get; set; }
```

Property Value

string

Module

Gets or sets the module name.

```
public string? Module { get; set; }
```

Property Value

string

RelativePath

Gets or sets the relative path from the vault root.

```
public string RelativePath { get; set; }
```

Property Value

string

Title

Gets or sets the friendly title for display.

```
public string Title { get; set; }
```

Property Value

string

Class VaultIndexOptions

Namespace: NotebookAutomation.Core.Models

Assembly: NotebookAutomation.Core.dll

Options for vault index generation.

```
public class VaultIndexOptions
```

Inheritance

object ← VaultIndexOptions

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Depth

Gets or sets the specific depth level to process (null for all levels).

```
public int? Depth { get; set; }
```

Property Value

int?

DryRun

Gets or sets a value indicating whether gets or sets whether to perform a dry run without creating files.

```
public bool DryRun { get; set; }
```

Property Value

bool

ForceOverwrite

Gets or sets a value indicating whether gets or sets whether to force overwrite existing index files.

```
public bool ForceOverwrite { get; set; }
```

Property Value

bool

Namespace NotebookAutomation.Core.Services

Classes

AISummarizer

Provides AI-powered text summarization using Microsoft.SemanticKernel with Azure OpenAI integration. Implements intelligent chunking strategies for large text processing, optimized for MBA coursework content including video transcripts, PDF documents, and academic materials. Supports variable substitution for metadata augmentation and configurable prompt templates.

OneDriveCliOptions

Options for configuring the behavior of OneDrive operations via the command-line interface.

OneDriveService

Provides methods for authenticating and accessing OneDrive files/folders.

PromptTemplateService

Service for loading prompt templates and performing variable substitution. Handles different template types (e.g., chunk summary, final summary) and supports dynamic prompt file loading.

TextChunkingService

Provides text chunking operations for AI summarization services. Implements intelligent text splitting with overlap to maintain context continuity.

VaultRootContextService

Provides scoped context for vault root path overrides during processing operations.

Interfaces

IAISummarizer

Defines the contract for AI-powered text summarization services. Provides methods for generating summaries with variable substitution and configurable prompt templates.

IOneDriveService

Interface for OneDrive service operations including authentication, file operations, sharing, and path mapping.

IPromptService

Interface for services that manage and process prompt templates.

ITextChunkingService

Defines the contract for text chunking operations used in AI summarization. Provides methods for splitting large texts into manageable chunks with intelligent overlap.

Class AISummarizer

Namespace: NotebookAutomation.Core.Services

Assembly: NotebookAutomation.Core.dll

Provides AI-powered text summarization using Microsoft.SemanticKernel with Azure OpenAI integration. Implements intelligent chunking strategies for large text processing, optimized for MBA coursework content including video transcripts, PDF documents, and academic materials. Supports variable substitution for metadata augmentation and configurable prompt templates.

```
public class AISummarizer : IAISummarizer
```

Inheritance

object ← AISummarizer

Implements

IAISummarizer

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Examples

```
// Basic usage
var summarizer = new AISummarizer(logger, promptService, kernel);
var summary = await summarizer.SummarizeWithVariablesAsync(longText);

// With metadata variables and custom prompt
var variables = new Dictionary<string, string>
{
    ["course"] = "MBA Strategy",
    ["type"] = "video_transcript",
    ["onedrivePath"] = "/courses/strategy/week1"
};
var summary = await summarizer.SummarizeWithVariablesAsync(
    inputText,
    variables,
    "chunk_summary_prompt"
);
```

Remarks

This class provides two main summarization strategies:

- Direct summarization for smaller texts (under ~12,000 characters)
- Chunked summarization with aggregation for larger texts

The chunking strategy splits large texts into overlapping segments, processes each chunk independently, then aggregates the results into a cohesive final summary. This approach ensures comprehensive coverage while respecting token limits of the underlying AI models.

Supports fallback to ITextGenerationService for testing scenarios when SemanticKernel is unavailable.

Constructors

AISummarizer(ILogger<AISummarizer>, IPromptService?, Kernel?, ITextChunkingService?)

Provides AI-powered text summarization using Microsoft.SemanticKernel with Azure OpenAI integration. Implements intelligent chunking strategies for large text processing, optimized for MBA coursework content including video transcripts, PDF documents, and academic materials. Supports variable substitution for metadata augmentation and configurable prompt templates.

```
public AISummarizer(ILogger<AISummarizer> logger, IPromptService? promptService, Kernel?
semanticKernel, ITextChunkingService? chunkingService = null)
```

Parameters

`logger` ILogger<AISummarizer>

The logger instance for tracking operations and debugging.

`promptService` IPromptService

Service for loading and processing prompt templates from the file system.

`semanticKernel` Kernel

Microsoft.SemanticKernel instance configured with Azure OpenAI.

`chunkingService` ITextChunkingService

Optional text chunking service for splitting large texts. If null, creates a default instance.

Examples

```
// Basic usage
var summarizer = new AISummarizer(logger, promptService, kernel);
var summary = await summarizer.SummarizeWithVariablesAsync(longText);

// With metadata variables and custom prompt
var variables = new Dictionary<string, string>
{
    ["course"] = "MBA Strategy",
    ["type"] = "video_transcript",
    ["onedrivePath"] = "/courses/strategy/week1"
};
var summary = await summarizer.SummarizeWithVariablesAsync(
    inputText,
    variables,
    "chunk_summary_prompt"
);
```

Remarks

This class provides two main summarization strategies:

- Direct summarization for smaller texts (under ~12,000 characters)
- Chunked summarization with aggregation for larger texts

The chunking strategy splits large texts into overlapping segments, processes each chunk independently, then aggregates the results into a cohesive final summary. This approach ensures comprehensive coverage while respecting token limits of the underlying AI models.

Supports fallback to ITextGenerationService for testing scenarios when SemanticKernel is unavailable.

Exceptions

ArgumentNullException

Thrown when logger is null.

Methods

LoadChunkPromptAsync()

Loads the chunk prompt template from the prompt service for individual chunk processing.

```
protected virtual Task<string?> LoadChunkPromptAsync()
```

Returns

Task <string>

A task that represents the asynchronous load operation. The task result contains:

- The chunk prompt template content if successfully loaded
- null if the prompt service is unavailable or loading fails.

Remarks

This method attempts to load the "chunk_summary_prompt.md" template file from the prompts directory. The chunk prompt is specifically designed for processing individual text segments before aggregation. Failures are logged as warnings but do not throw exceptions, allowing the system to fall back to default prompts.

Exceptions

ArgumentNullException

Thrown when logger is null.

LoadFinalPromptAsync()

Loads the final prompt template from the prompt service for summary aggregation.

```
protected virtual Task<string?> LoadFinalPromptAsync()
```

Returns

Task <string>

A task that represents the asynchronous load operation. The task result contains:

- The final summary prompt template content if successfully loaded

- null if the prompt service is unavailable or loading fails.

Remarks

This method attempts to load the "final_summary_prompt.md" template file from the prompts directory. The final prompt is specifically designed for aggregating multiple chunk summaries into a cohesive result. Failures are logged as warnings but do not throw exceptions, allowing the system to fall back to default prompts.

Exceptions

ArgumentNullException

Thrown when logger is null.

ProcessPromptTemplateAsync(string, string, string)

Processes and prepares the prompt template for use in summarization operations. Handles loading of prompt templates when not already provided and validates input text.

```
protected virtual Task<(string? processedPrompt, string processedInputText)>
ProcessPromptTemplateAsync(string inputText, string prompt, string promptFileName)
```

Parameters

`inputText` string

The input text to be summarized.

`prompt` string

The prompt string that may need processing or loading.

`promptFileName` string

The prompt template filename to load if prompt is empty.

Returns

Task<(string processedPrompt, string processedInputText)>

A task that represents the asynchronous processing operation. The task result contains:

- A tuple with the processed prompt (or null if loading failed) and the input text
- The processed prompt will be loaded from promptFileName if the prompt parameter is empty.

Remarks

This method serves as a preparation step for both direct and chunked summarization. It attempts to load the specified prompt template file when no prompt is provided directly. Loading failures are logged as warnings but do not prevent the operation from continuing.

Exceptions

ArgumentNullException

Thrown when logger is null.

SummarizeWithChunkingAsync(string, string?, Dictionary<string, string>?, CancellationToken)

```
protected virtual Task<string?> SummarizeWithChunkingAsync(string inputText, string? prompt,
Dictionary<string, string>? variables, CancellationToken cancellationToken)
```

Parameters

`inputText` string

`prompt` string

`variables` Dictionary<string, string>

`cancellationToken` CancellationToken

Returns

Task<string>

Exceptions

ArgumentNullException

Thrown when logger is null.

SummarizeWithSemanticKernelAsync(string, string, CancellationToken)

Summarizes text using Microsoft SemanticKernel with the specified prompt for smaller texts that don't require chunking.

```
protected virtual Task<string?> SummarizeWithSemanticKernelAsync(string inputText, string prompt, CancellationToken cancellationToken)
```

Parameters

`inputText` string

The text content to summarize.

`prompt` string

The prompt to guide the summarization process.

`cancellationToken` CancellationToken

Cancellation token for async operations.

Returns

Task <string>

A task that represents the asynchronous summarization operation. The task result contains:

- The generated summary text if successful
- null if the operation fails or SemanticKernel is unavailable
- A simulated summary string for testing scenarios.

Remarks

This method is used for direct summarization of smaller texts (typically under 12,000 characters). It creates a semantic function from the provided prompt and processes the entire text in a single operation.

The method uses OpenAI prompt execution settings configured for balanced performance:

- MaxTokens: 4000 (allowing comprehensive summaries)
- Temperature: 1.0 (balanced creativity and consistency)

- TopP: 1.0 (full vocabulary consideration)

Falls back to a default summarization prompt if none is provided. All errors are logged and handled gracefully by returning null.

Exceptions

Exception

> Various exceptions may be thrown by the underlying SemanticKernel operations, all of which are caught, logged, and result in a null return value.

SummarizeWithVariablesAsync(string, Dictionary<string, string>?, string?, CancellationToken)

```
public virtual Task<string?> SummarizeWithVariablesAsync(string inputText,
Dictionary<string, string>? variables = null, string? promptFileName = null,
CancellationToken cancellationToken = default)
```

Parameters

`inputText` string

`variables` Dictionary<string, string>

`promptFileName` string

`cancellationToken` CancellationToken

Returns

Task<string>

Exceptions

ArgumentNullException

> Thrown when logger is null.

Interface IAISummarizer

Namespace: NotebookAutomation.Core.Services

Assembly: NotebookAutomation.Core.dll

Defines the contract for AI-powered text summarization services. Provides methods for generating summaries with variable substitution and configurable prompt templates.

```
public interface IAISummarizer
```

Examples

```
var summarizer = serviceProvider.GetService<IAISummarizer>();
var summary = await summarizer.SummarizeWithVariablesAsync(
    "This is a long text that needs summarization.",
    new Dictionary<string, string> { { "course", "AI Basics" }, { "type", "lecture" } },
    "custom_prompt",
    CancellationToken.None);

Console.WriteLine(summary);
```

Remarks

This interface enables dependency injection and mocking for unit testing scenarios. Implementations should support:

- Direct summarization for short texts
- Chunked processing for large texts
- Variable substitution for metadata augmentation
- Customizable prompt templates for flexible summarization behavior

Methods

SummarizeWithVariablesAsync(string, Dictionary<string, string>?, string?, CancellationToken)

Generates an AI-powered summary for the given text using the best available AI framework. Automatically selects between direct summarization and chunked processing based on text length. Supports variable substitution for metadata augmentation and custom prompt templates.

```
Task<string?> SummarizeWithVariablesAsync(string inputText, Dictionary<string, string>?
variables = null, string? promptFileName = null, CancellationToken cancellationToken
= default)
```

Parameters

`inputText` string

The text content to summarize. Cannot be null or empty.

`variables` Dictionary<string, string>

Optional dictionary of variables for prompt template substitution and metadata enhancement. Common variables include:

- course: The course name
- type: The type of content (e.g., lecture, notes)
- onedrivePath: The OneDrive path for related files
- yamlfrontmatter: YAML metadata for the content

`promptFileName` string

Optional prompt template filename (without .md extension) to customize summarization behavior. Defaults to "final_summary_prompt" if not provided.

`cancellationToken` CancellationToken

Optional cancellation token to cancel the asynchronous operation.

Returns

Task<string>

A task that represents the asynchronous summarization operation. The task result contains:

- The generated summary text for successful operations
- An empty string if the operation fails but the service is available
- null if no AI service is available

Exceptions

ArgumentException

Thrown when `inputText` is null or empty.

Interface IOneDriveService

Namespace: NotebookAutomation.Core.Services

Assembly: NotebookAutomation.Core.dll

Interface for OneDrive service operations including authentication, file operations, sharing, and path mapping.

```
public interface IOneDriveService
```

Examples

```
var oneDriveService = serviceProvider.GetService<IOneDriveService>();
await oneDriveService.AuthenticateAsync();

var files = await oneDriveService.ListFilesAsync("/Documents", CancellationToken.None);
foreach (var file in files)
{
    Console.WriteLine(file);
}
```

Remarks

This interface provides methods for interacting with OneDrive, including:

- Authentication and token management
- File operations: upload, download, list, and search
- Sharing: create and retrieve shareable links
- Path mapping between local and OneDrive directories

Implementations should handle Microsoft Graph API integration and provide robust error handling for network operations.

Methods

AuthenticateAsync()

Authenticates with Microsoft Graph using device code flow.

```
Task AuthenticateAsync()
```

Returns

Task

A Task representing the asynchronous operation.

Examples

```
await oneDriveService.AuthenticateAsync();
Console.WriteLine("Authentication successful.");
```

Remarks

This method initiates the device code flow for authentication, allowing users to authenticate by entering a code on a Microsoft login page. It retrieves and caches tokens for subsequent API calls.

If the authentication fails, an exception is thrown with details about the failure.

ConfigureVaultRoots(string, string)

Configures the local and OneDrive vault root directories for path mapping.

```
void ConfigureVaultRoots(string localVaultRoot, string oneDriveVaultRoot)
```

Parameters

`localVaultRoot` string

The local vault root directory path.

`oneDriveVaultRoot` string

The OneDrive vault root directory path.

Examples

```
oneDriveService.ConfigureVaultRoots("C:\Vault", "/Vault");
```

Remarks

This method sets up the root directories for mapping between local and OneDrive paths.

CreateShareLinkAsync(string, string, string, CancellationToken)

Creates a shareable link for a file in OneDrive.

```
Task<string?> CreateShareLinkAsync(string filePath, string linkType = "view", string scope =
"anonymous", CancellationToken cancellationToken = default)
```

Parameters

`filePath` string

The local file path or OneDrive file path. If it's a local path, it will be converted to a OneDrive-relative path.

`linkType` string

The type of sharing link to create. Default is "view".

`scope` string

The scope of the sharing link. Default is "anonymous".

`cancellationToken` CancellationToken

Cancellation token.

Returns

Task <string>

The shareable link URL if successful, null otherwise.

Examples

```
var link = await oneDriveService.CreateShareLinkAsync("/Documents/file.txt", "edit",
"organization", CancellationToken.None);
Console.WriteLine(link);
```

Remarks

This method creates a shareable link for the specified file in OneDrive. The link type and scope can be customized.

DownloadFileAsync(string, string, CancellationToken)

Downloads a file from OneDrive to a local path.

```
Task DownloadFileAsync(string oneDrivePath, string localPath, CancellationToken
cancellationToken = default)
```

Parameters

`oneDrivePath` string

The OneDrive file path.

`localPath` string

The local destination path.

`cancellationToken` CancellationToken

Optional cancellation token.

Returns

Task

Task representing the async download operation.

Examples

```
await oneDriveService.DownloadFileAsync("/Documents/file.txt", "C:\Downloads\file.txt",
CancellationToken.None);
Console.WriteLine("File downloaded.");
```

Remarks

This method downloads a file from OneDrive to the specified local path. If the operation is canceled, the task is marked as canceled.

GetShareLinkAsync(string, bool, CancellationToken)

Gets a share link for a file in OneDrive.

```
Task<string> GetShareLinkAsync(string filePath, bool forceRefresh = false, CancellationToken
cancellationToken = default)
```

Parameters

`filePath` string

The path of the file to get a share link for.

`forceRefresh` bool

Whether to force refresh the share link.

`cancellationToken` CancellationToken

Optional cancellation token.

Returns

Task <string>

The share link information as a JSON string.

Examples

```
var shareLink = await oneDriveService.GetShareLinkAsync("/Documents/file.txt",
true, CancellationToken.None);
Console.WriteLine(shareLink);
```

Remarks

This method retrieves a share link for the specified file in OneDrive. If `forceRefresh` is true, a new link is generated.

ListFilesAsync(string, CancellationToken)

Lists files in a OneDrive folder.

```
Task<List<string>> ListFilesAsync(string oneDriveFolder, CancellationToken cancellationToken
= default)
```

Parameters

`oneDriveFolder` string

The OneDrive folder path.

`cancellationToken` CancellationToken

Optional cancellation token.

Returns

Task<List<string>>

List of file names.

Examples

```
var files = await oneDriveService.ListFilesAsync("/Documents", CancellationToken.None);
foreach (var file in files)
{
    Console.WriteLine(file);
}
```

Remarks

This method retrieves a list of file names in the specified OneDrive folder. If the folder does not exist, an empty list is returned.

MapLocalToOneDrivePath(string)

Maps a local file path to its corresponding OneDrive path.

```
string MapLocalToOneDrivePath(string localPath)
```

Parameters

`localPath` string

The local file path to map.

Returns

string

The corresponding OneDrive path.

Examples

```
var oneDrivePath = oneDriveService.MapLocalToOneDrivePath("C:\Vault\file.txt");
Console.WriteLine(oneDrivePath);
```

Remarks

This method converts a local file path to its corresponding OneDrive path based on the configured vault roots.

MapOneDriveToLocalPath(string)

Maps an OneDrive file path to its corresponding local path.

```
string MapOneDriveToLocalPath(string oneDrivePath)
```

Parameters

`oneDrivePath` string

The OneDrive file path to map.

Returns

string

The corresponding local path.

Examples

```
var localPath = oneDriveService.MapOneDriveToLocalPath("/Vault/file.txt");
Console.WriteLine(localPath);
```

Remarks

This method converts a OneDrive file path to its corresponding local path based on the configured vault roots.

RefreshAuthenticationAsync()

Forces a refresh of the authentication tokens by clearing cache and re-authenticating.

```
Task RefreshAuthenticationAsync()
```

Returns

Task

Task representing the async refresh operation.

Examples

```
await oneDriveService.RefreshAuthenticationAsync();
Console.WriteLine("Tokens refreshed.");
```

Remarks

This method clears cached tokens and initiates a new authentication flow to retrieve fresh tokens.

SearchFilesAsync(string, CancellationToken)

Searches for files or folders in OneDrive by name or pattern.

```
Task<List<Dictionary<string, object>>> SearchFilesAsync(string query, CancellationToken
cancellationToken = default)
```

Parameters

`query` string

The search query string.

`cancellationToken` CancellationToken

Optional cancellation token.

Returns

Task<List<Dictionary<string, object>>>

List of file/folder metadata matching the query.

Examples

```
var results = await oneDriveService.SearchFilesAsync("report", CancellationToken.None);
foreach (var result in results)
{
    Console.WriteLine(result);
}
```

Remarks

This method searches for files or folders in OneDrive that match the specified query string.

SetForceRefresh(bool)

Sets the force refresh flag to bypass cached tokens on next authentication.

```
void SetForceRefresh(bool forceRefresh)
```

Parameters

`forceRefresh` bool

If true, will force refresh authentication tokens ignoring cache.

Examples

```
oneDriveService.SetForceRefresh(true);
```

Remarks

Use this method to ensure fresh tokens are retrieved during the next authentication attempt.

UploadFileAsync(string, string, CancellationToken)

Uploads a local file to OneDrive at the specified path.

```
Task UploadFileAsync(string localPath, string oneDrivePath, CancellationToken
cancellationToken = default)
```

Parameters

`localPath` string

The local file path.

`oneDrivePath` string

The OneDrive destination path (including filename).

`cancellationToken` CancellationToken

Optional cancellation token.

Returns

Task

Task representing the async upload operation.

Examples

```
await oneDriveService.UploadFileAsync("C:\Documents\file.txt", "/Documents/file.txt",
CancellationToken.None);
Console.WriteLine("File uploaded.");
```

Remarks

This method uploads a local file to the specified OneDrive path. If the operation is canceled, the task is marked as canceled.

Interface IPromptService

Namespace: NotebookAutomation.Core.Services

Assembly: NotebookAutomation.Core.dll

Interface for services that manage and process prompt templates.

```
public interface IPromptService
```

Examples

```
var promptService = serviceProvider.GetService<IPromptService>();
var template = await promptService.LoadTemplateAsync("welcome_message");
var prompt = promptService.SubstituteVariables(template, new Dictionary<string, string> { {
"name", "John" } });
Console.WriteLine(prompt);
```

Remarks

This interface provides methods for loading, processing, and substituting variables in prompt templates. Implementations should support:

- Loading templates from a configured directory
- Substituting variables in templates
- Generating prompts with substituted variables
- Asynchronous processing of templates

Methods

GetPromptAsync(string, Dictionary<string, string>?)

Gets a prompt with variables substituted.

```
Task<string> GetPromptAsync(string templateName, Dictionary<string, string>? variables)
```

Parameters

`templateName` string

Name of the template to load, without file extension.

`variables` Dictionary<string, string>

Dictionary of variables to substitute.

Returns

Task<string>

The prompt with variables substituted.

Examples

```
var prompt = await promptService.GetPromptAsync("welcome_message", new Dictionary<string,
string> { { "name", "John" } });
Console.WriteLine(prompt);
```

Remarks

This method combines template loading and variable substitution to generate a complete prompt.

LoadTemplateAsync(string)

Loads a template from the configured prompts directory.

```
Task<string> LoadTemplateAsync(string templateName)
```

Parameters

`templateName` string

Name of the template to load, without file extension.

Returns

Task<string>

The template content as a string.

Examples

```
var template = await promptService.LoadTemplateAsync("welcome_message");
Console.WriteLine(template);
```

Remarks

This method retrieves the content of a template file from the configured prompts directory. If the template does not exist, an exception is thrown.

ProcessTemplateAsync(string, Dictionary<string, string>?)

Processes template with variables asynchronously.

```
Task<string> ProcessTemplateAsync(string template, Dictionary<string, string>? variables)
```

Parameters

`template` string

The template string with placeholders.

`variables` Dictionary<string, string>

Dictionary of variable names and values.

Returns

Task<string>

The template with variables substituted.

Examples

```
var result = await promptService.ProcessTemplateAsync("Hello, {{name}}!", new
Dictionary<string, string> { { "name", "John" } });
Console.WriteLine(result); // Outputs: "Hello, John!"
```

Remarks

This method performs variable substitution in the provided template string asynchronously.

SubstituteVariables(string, Dictionary<string, string>?)

Substitutes variables in a template string.

```
string SubstituteVariables(string template, Dictionary<string, string>? variables)
```

Parameters

`template` string

Template with placeholders in the format {{variable_name}}.

`variables` Dictionary<string, string>

Dictionary of variables to substitute.

Returns

string

The template with variables substituted.

Examples

```
var template = "Hello, {{name}}!";
var result = promptService.SubstituteVariables(template, new Dictionary<string, string> { {
"name", "John" } });
Console.WriteLine(result); // Outputs: "Hello, John!"
```

Remarks

This method replaces placeholders in the template string with values from the provided dictionary. If a placeholder does not have a corresponding value, it remains unchanged.

Interface ITextChunkingService

Namespace: NotebookAutomation.Core.Services

Assembly: NotebookAutomation.Core.dll

Defines the contract for text chunking operations used in AI summarization. Provides methods for splitting large texts into manageable chunks with intelligent overlap.

```
public interface ITextChunkingService
```

Methods

EstimateTokenCount(string)

Estimates the token count for the given text using a character-based heuristic. Uses approximately 4 characters per token as a rough estimate for English text.

```
int EstimateTokenCount(string text)
```

Parameters

`text` string

The text to estimate tokens for.

Returns

int

The estimated token count based on character length.

SplitTextIntoChunks(string, int, int)

Splits text into chunks with overlap for optimal processing. Uses character-based chunking with intelligent boundary detection.

```
List<string> SplitTextIntoChunks(string text, int chunkSize, int overlap)
```

Parameters

`text` string

The text to split.

`chunkSize` int

Maximum size of each chunk in characters.

`overlap` int

Number of characters to overlap between chunks.

Returns

List <string>

List of text chunks.

Class OneDriveCliOptions

Namespace: NotebookAutomation.Core.Services

Assembly: NotebookAutomation.Core.dll

Options for configuring the behavior of OneDrive operations via the command-line interface.

```
public class OneDriveCliOptions
```

Inheritance

object ← OneDriveCliOptions

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Examples

Example of setting up options based on command-line arguments:

```
var options = new OneDriveCliOptions
{
    DryRun = context.ParseResult.GetValueForOption(dryRunOption),
    Verbose = context.ParseResult.GetValueForOption(verboseOption),
    Force = context.ParseResult.GetValueForOption(forceOption),
    Retry = true  // Always retry by default
};
oneDriveService.SetCliOptions(options);
```

Remarks

The OneDriveCliOptions class provides a structured way to configure how OneDrive operations behave when invoked through the command-line interface. These options control behaviors such as:

- Whether operations should run in dry-run mode (simulating changes without actually making them)
- Whether to display verbose output during operations
- Whether to force operations even if they might overwrite existing content
- Whether to retry operations on failure

These options are typically set based on command-line arguments provided by the user, and then passed to the OneDriveService for use during operations.

Properties

DryRun

Gets or sets a value indicating whether gets or sets whether to simulate operations without making actual changes.

```
public bool DryRun { get; set; }
```

Property Value

bool

`true` to run in dry-run mode (simulating but not performing operations); `false` to perform actual operations.

Remarks

When DryRun is enabled, the OneDriveService will log what would have happened but won't actually modify, upload, or download any files. This is useful for verifying what operations would be performed without risking any data changes.

Force

Gets or sets a value indicating whether gets or sets whether to force operations even if they would overwrite existing content.

```
public bool Force { get; set; }
```

Property Value

bool

`true` to force operations; `false` to prompt or skip when conflicts occur.

Remarks

When Force is enabled, the OneDriveService will overwrite existing files without prompting for confirmation. If Force is disabled, the service might skip conflicting operations or prompt for confirmation, depending on the specific implementation.

Retry

Gets or sets a value indicating whether gets or sets whether to retry failed operations.

```
public bool Retry { get; set; }
```

Property Value

bool

`true` to retry failed operations; `false` to fail immediately.

Remarks

When Retry is enabled, the OneDriveService will attempt to retry operations that fail due to transient errors, like network issues or rate limiting. The specific retry strategy (number of retries, delays) is determined by the service implementation.

This is particularly useful for operations that are likely to succeed on retry, such as uploads or downloads that might fail due to temporary network issues.

Verbose

Gets or sets a value indicating whether gets or sets whether to display detailed, verbose output during operations.

```
public bool Verbose { get; set; }
```

Property Value

bool

`true` to display verbose output; `false` for standard output.

Remarks

When Verbose is enabled, the OneDriveService will log additional details about each operation, including file sizes, paths, timestamps, and more. This is useful for debugging or for understanding exactly what the service is doing.

Class OneDriveService

Namespace: NotebookAutomation.Core.Services

Assembly: NotebookAutomation.Core.dll

Provides methods for authenticating and accessing OneDrive files/folders.

```
public class OneDriveService : IOneDriveService
```

Inheritance

object ← OneDriveService

Implements

IOneDriveService

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

OneDriveService(ILogger<OneDriveService>, string, string, string[], IPublicClientApplication?)

```
public OneDriveService(ILogger<OneDriveService> logger, string clientId, string tenantId,
string[] scopes, IPublicClientApplication? msalApp = null)
```

Parameters

`logger` ILogger<OneDriveService>

`clientId` string

`tenantId` string

`scopes` string[]

`msalApp` IPublicClientApplication

Methods

AuthenticateAsync()

Authenticates with Microsoft Graph API using token caching for persistent authentication. Uses the same approach as the Python implementation for consistent behavior.

```
public Task AuthenticateAsync()
```

Returns

Task

Task representing the authentication process.

ConfigureVaultRoots(string, string)

Configures the root directories for local and OneDrive vaults for path mapping.

```
public void ConfigureVaultRoots(string localVaultRoot, string oneDriveVaultRoot)
```

Parameters

`localVaultRoot` string

The root directory of the local vault.

`oneDriveVaultRoot` string

The root directory of the OneDrive vault (relative to OneDrive root).

CreateShareLinkAsync(string, string, string, CancellationToken)

Creates a shareable link for a file in OneDrive.

```
public Task<string?> CreateShareLinkAsync(string filePath, string linkType = "view", string
scope = "anonymous", CancellationToken cancellationToken = default)
```

Parameters

`filePath` string

The local file path or OneDrive file path. If it's a local path, it will be converted to a OneDrive-relative path.

`linkType` string

The type of sharing link to create. Default is "view".

`scope` string

The scope of the sharing link. Default is "anonymous".

`cancellationToken` CancellationToken

Cancellation token.

Returns

Task <string>

The shareable link URL if successful, null otherwise.

DownloadFileAsync(string, string, CancellationToken)

Downloads a file from OneDrive to a local path.

```
public Task DownloadFileAsync(string oneDrivePath, string localPath, CancellationToken
cancellationToken = default)
```

Parameters

`oneDrivePath` string

The OneDrive file path.

`localPath` string

The local destination path.

`cancellationToken` CancellationToken

Returns

Task

A Task representing the asynchronous operation.

GetFileByIdAsync(string, CancellationToken)

Gets file or folder metadata by OneDrive item ID.

```
public Task<Dictionary<string, object>?> GetFileByIdAsync(string itemId, CancellationToken
cancellationToken = default)
```

Parameters

`itemId` string

The OneDrive item ID.

`cancellationToken` CancellationToken

Returns

Task<Dictionary<string, object>>

Dictionary of file/folder metadata, or null if not found.

GetFileByPathAsync(string, CancellationToken)

Gets file or folder metadata by OneDrive path.

```
public Task<Dictionary<string, object>?> GetFileByPathAsync(string oneDrivePath,
CancellationToken cancellationToken = default)
```

Parameters

`oneDrivePath` string

The OneDrive file or folder path.

`cancellationToken` CancellationToken

Returns

Task<Dictionary<string, object>>

Dictionary of metadata, or null if not found.

GetItemByIdAsync(string, CancellationToken)

Gets a file or folder by its OneDrive item ID.

```
public Task<string?> GetItemByIdAsync(string itemId, CancellationToken cancellationToken
= default)
```

Parameters

`itemId` string

The OneDrive item ID.

`cancellationToken` CancellationToken

Returns

Task<string>

The file or folder name, or null if not found.

GetShareLinkAsync(string, bool, CancellationToken)

Gets a share link for a file in OneDrive.

```
public Task<string> GetShareLinkAsync(string filePath, bool forceRefresh,
CancellationToken cancellationToken)
```

Parameters

`filePath` string

The path of the file to get a share link for.

`forceRefresh` bool

Whether to force refresh the share link.

`cancellationToken` CancellationToken

Optional cancellation token.

Returns

Task<string>

The share link information as a JSON string.

ListDrivesAsync(CancellationToken)

Lists all available OneDrive drives for the authenticated user.

```
public Task<List<Dictionary<string, object>>> ListDrivesAsync(CancellationToken
cancellationToken = default)
```

Parameters

`cancellationToken` CancellationToken

Returns

Task<List<Dictionary<string, object>>>

List of drive metadata dictionaries.

ListFilesAsync(string, CancellationToken)

Lists files in a OneDrive folder.

```
public Task<List<string>> ListFilesAsync(string oneDriveFolder, CancellationToken
cancellationToken = default)
```

Parameters

`oneDriveFolder` string

The OneDrive folder path.

`cancellationToken` CancellationToken

Returns

Task<List<string>>

List of file names.

ListFilesRecursiveAsync(string, string?, CancellationToken)

Recursively lists all files and folders under a given OneDrive folder path, with optional filtering.

```
public Task<List<Dictionary<string, object>>> ListFilesRecursiveAsync(string oneDriveFolder,
string? fileExtensionFilter = null, CancellationToken cancellationToken = default)
```

Parameters

`oneDriveFolder` string

The OneDrive folder path to start from.

`fileExtensionFilter` string

Optional file extension filter (e.g., ".pdf").

`cancellationToken` CancellationToken

Returns

Task<List<Dictionary<string, object>>>

List of file/folder metadata dictionaries.

ListFilesWithMetadataAsync(string, CancellationToken)

Lists files in a OneDrive folder, returning full metadata for each item.

```
public Task<List<Dictionary<string, object>>> ListFilesWithMetadataAsync(string
oneDriveFolder, CancellationToken cancellationToken = default)
```

Parameters

`oneDriveFolder` string

The OneDrive folder path.

`cancellationToken` CancellationToken

Returns

Task<List<Dictionary<string, object>>>

List of file/folder metadata dictionaries.

ListRootItemsAsync(CancellationToken)

Lists items in the root folder of the default OneDrive.

```
public Task<List<Dictionary<string, object>>> ListRootItemsAsync(CancellationToken
cancellationToken = default)
```

Parameters

`cancellationToken` CancellationToken

Returns

Task<List<Dictionary<string, object>>>

List of file/folder metadata dictionaries in the root folder.

MapLocalToOneDrivePath(string)

Maps a local path to a OneDrive path, preserving structure.

```
public string MapLocalToOneDrivePath(string localPath)
```

Parameters

`localPath` string

The local file or folder path.

Returns

string

Corresponding OneDrive path.

MapOneDriveToLocalPath(string)

Maps a OneDrive path to a local path, preserving structure.

```
public string MapOneDriveToLocalPath(string oneDrivePath)
```

Parameters

`oneDrivePath` string

The OneDrive file or folder path.

Returns

string

Corresponding local path.

RefreshAuthenticationAsync()

Forces a refresh of the authentication tokens by clearing cache and re-authenticating.

```
public Task RefreshAuthenticationAsync()
```

Returns

Task

Task representing the async refresh operation.

SearchAsync(string, CancellationToken)

Searches for files and folders in OneDrive by name or pattern.

```
public Task<List<string>> SearchAsync(string searchPattern, CancellationToken
cancellationToken = default)
```

Parameters

`searchPattern` string

The search pattern (e.g., file name or wildcard pattern).

`cancellationToken` CancellationToken

Returns

Task <List <string > >

List of matching file/folder names.

SearchFilesAsync(string, CancellationToken)

Searches for files or folders in OneDrive by name or pattern.

```
public Task<List<Dictionary<string, object>>> SearchFilesAsync(string query,
CancellationToken cancellationToken = default)
```

Parameters

`query` string

The search query string.

`cancellationToken` CancellationToken

Returns

Task<List<Dictionary<string, object>>>

List of file/folder metadata matching the query.

SetCliOptions(OneDriveCliOptions)

Sets CLI options for dry-run, retry, force, and verbose/debug output.

```
public void SetCliOptions(OneDriveCliOptions options)
```

Parameters

`options` OneDriveCliOptions

SetForceRefresh(bool)

Sets whether to force refresh authentication tokens ignoring cache.

```
public void SetForceRefresh(bool forceRefresh)
```

Parameters

`forceRefresh` bool

If true, will force refresh authentication tokens ignoring cache.

TryAlternativePathFormatsAsync(string, CancellationToken)

Attempts to resolve a OneDrive path using alternative formats if the initial request fails. If not found, suggests similar files/folders in the parent directory.

```
public Task<string?> TryAlternativePathFormatsAsync(string oneDrivePath, CancellationToken
cancellationToken = default)
```

Parameters

`oneDrivePath` string

The original OneDrive path.

`cancellationToken` CancellationToken

Returns

Task<string>

Resolved path or null if not found. Logs suggestions if not found.

UploadFileAsync(string, string, CancellationToken)

Uploads a local file to OneDrive at the specified path.

```
public Task UploadFileAsync(string localPath, string oneDrivePath, CancellationToken
cancellationToken = default)
```

Parameters

`localPath` string

The local file path.

`oneDrivePath` string

The OneDrive destination path (including filename).

`cancellationToken` CancellationToken

Returns

Task

A Task representing the asynchronous operation.

Class PromptTemplateService

Namespace: NotebookAutomation.Core.Services

Assembly: NotebookAutomation.Core.dll

Service for loading prompt templates and performing variable substitution. Handles different template types (e.g., chunk summary, final summary) and supports dynamic prompt file loading.

```
public class PromptTemplateService : IPromptService
```

Inheritance

object ← PromptTemplateService

Implements

IPromptService

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Examples

```
var promptService = new PromptTemplateService(logger, config);
var template = await promptService.LoadTemplateAsync("welcome_message");
var prompt = promptService.SubstituteVariables(template, new Dictionary<string, string> { {
"name", "John" } });
Console.WriteLine(prompt);
```

Remarks

This service provides functionality for:

- Loading prompt templates from a configured directory
- Substituting variables in templates
- Handling default templates for chunk and final summaries
- Dynamic initialization of the prompts directory

The service integrates with application configuration to determine the prompts directory and provides fallback mechanisms to locate the directory in common project structures.

Constructors

PromptTemplateService(ILogger<PromptTemplateService>, IYamlHelper, AppConfig)

Initializes a new instance of the PromptTemplateService class.

```
public PromptTemplateService(ILogger<PromptTemplateService> logger, IYamlHelper yamlHelper,
AppConfig config)
```

Parameters

`logger` ILogger<PromptTemplateService>

The logger to use for logging.

`yamlHelper` IYamlHelper

The YAML helper for processing frontmatter.

`config` AppConfig

The application configuration.

Examples

```
var promptService = new PromptTemplateService(logger, yamlHelper, config);
```

Remarks

This constructor initializes the service and sets up the prompts directory using the provided configuration.

Properties

DefaultChunkPrompt

```
public static string DefaultChunkPrompt { get; }
```

Property Value

string

DefaultFinalPrompt

```
public static string DefaultFinalPrompt { get; }
```

Property Value

string

PromptsDirectory

Gets the path to the prompts directory.

```
public string PromptsDirectory { get; }
```

Property Value

string

Methods

GetPromptAsync(string, Dictionary<string, string>?)

Gets a prompt with variables substituted.

```
public Task<string> GetPromptAsync(string templateName, Dictionary<string,
string>? substitutionValues)
```

Parameters

`templateName` string

Name of the template to load, without file extension.

`substitutionValues` Dictionary<string, string>

Returns

Task<string>

The prompt with variables substituted.

LoadAndSubstituteAsync(string, Dictionary<string, string>)

Loads a prompt template and substitutes variables.

```
public virtual Task<string> LoadAndSubstituteAsync(string templatePath, Dictionary<string,
string> substitutionValues)
```

Parameters

`templatePath` string

Path to the prompt template file.

`substitutionValues` Dictionary<string, string>

Returns

Task<string>

Prompt string with variables substituted.

LoadTemplateAsync(string)

Loads a template by name from the prompts directory.

```
public virtual Task<string> LoadTemplateAsync(string templateName)
```

Parameters

`templateName` string

Name of the prompt template (e.g., "chunk_summary_prompt").

Returns

Task<string>

The template content, or a default template if not found.

ProcessTemplateAsync(string, Dictionary<string, string>?)

Processes template with variables asynchronously.

```
public Task<string> ProcessTemplateAsync(string template, Dictionary<string,
string>? substitutionValues)
```

Parameters

`template` string

The template string with placeholders.

`substitutionValues` Dictionary<string, string>

Returns

Task<string>

The template with variables substituted.

SubstituteVariables(string, Dictionary<string, string>?)

Substitutes variables in a template string.

```
public string SubstituteVariables(string template, Dictionary<string,
string>? substitutionValues)
```

Parameters

`template` string

The template string with placeholders.

`substitutionValues` Dictionary<string, string>

Returns

string

The template with variables substituted.

Class TextChunkingService

Namespace: NotebookAutomation.Core.Services

Assembly: NotebookAutomation.Core.dll

Provides text chunking operations for AI summarization services. Implements intelligent text splitting with overlap to maintain context continuity.

```
public class TextChunkingService : ITextChunkingService
```

Inheritance

object ← TextChunkingService

Implements

ITextChunkingService

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

EstimateTokenCount(string)

Estimates the token count for the given text using a character-based heuristic. Uses approximately 4 characters per token as a rough estimate for English text.

```
public int EstimateTokenCount(string text)
```

Parameters

`text` string

The text to estimate tokens for.

Returns

int

The estimated token count based on character length, or 0 if the text is null or whitespace.

Remarks

This is a simplified estimation method that provides reasonable approximations for:

- English academic text (typical in MBA coursework)
- Mixed alphanumeric content
- Standard punctuation and formatting

///.
The 4:1 character-to-token ratio is a conservative estimate that works well for OpenAI models. Actual token counts may vary based on text complexity, language, and specific tokenizer implementation.

SplitTextIntoChunks(string, int, int)

Splits text into chunks with overlap for optimal processing. Uses character-based chunking with intelligent boundary detection.

```
public List<string> SplitTextIntoChunks(string text, int chunkSize, int overlap)
```

Parameters

`text` string

The text to split.

`chunkSize` int

Maximum size of each chunk in characters.

`overlap` int

Number of characters to overlap between chunks.

Returns

List<string>

List of text chunks.

Exceptions

ArgumentNullException

Thrown when text is null.

ArgumentOutOfRangeException

Thrown when chunkSize or overlap are invalid.

ArgumentException

Thrown when overlap is greater than or equal to chunkSize.

Class VaultRootContextService

Namespace: NotebookAutomation.Core.Services

Assembly: NotebookAutomation.Core.dll

Provides scoped context for vault root path overrides during processing operations.

```
public class VaultRootContextService
```

Inheritance

object ← VaultRootContextService

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

This service allows the dependency injection container to provide different vault root paths to components that need them, without requiring constructor changes throughout the system. It's particularly useful when processing different vaults than the one configured in AppConfig.

Properties

HasVaultRootOverride

Gets a value indicating whether determines if a vault root override is active.

```
public bool HasVaultRootOverride { get; }
```

Property Value

bool

True if a vault root override is set, false otherwise.

VaultRootOverride

Gets or sets the vault root path override.

```
public string? VaultRootOverride { get; set; }
```

Property Value

string

The vault root path to use instead of the configured path, or null to use the configured path.

Namespace NotebookAutomation.Core.Services.Text

Classes

RecursiveCharacterTextSplitter

A sophisticated text splitter that recursively splits text based on a hierarchy of separators, preserving semantic boundaries and ensuring better context maintenance between chunks.

Class RecursiveCharacterTextSplitter

Namespace: NotebookAutomation.Core.Services.Text

Assembly: NotebookAutomation.Core.dll

A sophisticated text splitter that recursively splits text based on a hierarchy of separators, preserving semantic boundaries and ensuring better context maintenance between chunks.

```
public class RecursiveCharacterTextSplitter
```

Inheritance

object ← RecursiveCharacterTextSplitter

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Examples

```
var splitter = new RecursiveCharacterTextSplitter(logger);
var chunks = splitter.SplitText("This is a sample text with multiple paragraphs.");
foreach (var chunk in chunks)
{
    Console.WriteLine(chunk);
}
```

Remarks

This implementation is inspired by the RecursiveCharacterTextSplitter concept from LangChain but adapted specifically for C# and the NotebookAutomation project. The splitter works by:

- Splitting on the strongest separators first (e.g., triple newlines, headers)
- Progressively moving to weaker separators (e.g., single spaces) if necessary
- Preserving special patterns like markdown headers, code blocks, and lists

The splitter is optimized for handling markdown and code content, ensuring semantic boundaries are maintained while splitting text into manageable chunks.

Constructors

RecursiveCharacterTextSplitter(ILogger, int, int, List<string>?, bool)

Initializes a new instance of the RecursiveCharacterTextSplitter class.

```
public RecursiveCharacterTextSplitter(ILogger logger, int chunkSize = 3000, int chunkOverlap
= 500, List<string>? separators = null, bool keepSeparator = true)
```

Parameters

`logger` ILogger

Logger for diagnostic information.

`chunkSize` int

Maximum size of each chunk in estimated tokens.

`chunkOverlap` int

Number of tokens to overlap between chunks.

`separators` List<string>

Optional list of separators to use for splitting, in order of priority.

`keepSeparator` bool

Whether to keep the separator with the chunk.

Examples

```
var splitter = new RecursiveCharacterTextSplitter(logger, chunkSize: 3000,
chunkOverlap: 500);
```

Remarks

This constructor initializes the splitter with default or custom settings for chunk size, overlap, and separator hierarchy. It validates input parameters and ensures the chunk overlap is smaller than the chunk size.

Methods

CreateForCode(ILogger, int, int)

Creates a recursive text splitter optimized for code content.

```
public static RecursiveCharacterTextSplitter CreateForCode(ILogger logger, int chunkSize =
3000, int chunkOverlap = 500)
```

Parameters

`logger` ILogger

Logger for diagnostic information.

`chunkSize` int

Maximum size of each chunk in estimated tokens.

`chunkOverlap` int

Number of tokens to overlap between chunks.

Returns

RecursiveCharacterTextSplitter

A RecursiveCharacterTextSplitter configured for code content.

CreateForMarkdown(ILogger, int, int)

Creates a recursive text splitter optimized for markdown content.

```
public static RecursiveCharacterTextSplitter CreateForMarkdown(ILogger logger, int chunkSize
= 3000, int chunkOverlap = 500)
```

Parameters

`logger` ILogger

Logger for diagnostic information.

`chunkSize` int

Maximum size of each chunk in estimated tokens.

`chunkOverlap` int

Number of tokens to overlap between chunks.

Returns

RecursiveCharacterTextSplitter

A RecursiveCharacterTextSplitter configured for markdown content.

Examples

```
var markdownSplitter = RecursiveCharacterTextSplitter.CreateForMarkdown(logger);
```

Remarks

This factory method creates a splitter with a separator hierarchy optimized for markdown content, including headers, paragraph breaks, and other markdown-specific patterns.

SplitText(string)

Splits the text into chunks recursively respecting the defined separators.

```
public List<string> SplitText(string text)
```

Parameters

`text` string

The text to split.

Returns

List<string>

A list of text chunks that respect the configured size constraints.

Namespace NotebookAutomation.Core.Tools.MarkdownGeneration

Classes

MarkdownNoteProcessor

Provides functionality for converting HTML, TXT, or EPUB files to markdown notes, with optional AI-generated summaries.

MarkdownParser

Parser for markdown content with frontmatter handling.

Class MarkdownNoteProcessor

Namespace: NotebookAutomation.Core.Tools.MarkdownGeneration

Assembly: NotebookAutomation.Core.dll

Provides functionality for converting HTML, TXT, or EPUB files to markdown notes, with optional AI-generated summaries.

```
public class MarkdownNoteProcessor
```

Inheritance

object ← MarkdownNoteProcessor

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Examples

```
var processor = new MarkdownNoteProcessor(logger, aiSummarizer);
var markdown = await processor.ConvertToMarkdownAsync("input.html", openAiApiKey: "your-api-key", promptFileName: "summary_prompt");
Console.WriteLine(markdown);
```

Remarks

This class integrates with the AI summarizer and markdown note builder to process input files and generate markdown notes. It supports:

- TXT file conversion
- HTML file conversion (basic tag stripping)
- EPUB file parsing and conversion
- Optional AI summarization using OpenAI API

The class logs errors for unsupported file types or failed operations and provides detailed diagnostic information.

Constructors

MarkdownNoteProcessor(ILogger, AISummarizer, MetadataHierarchyDetector, AppConfig?)

Initializes a new instance of the MarkdownNoteProcessor class.

```
public MarkdownNoteProcessor(ILogger logger, AISummarizer aiSummarizer,
MetadataHierarchyDetector hierarchyDetector, AppConfig? appConfig = null)
```

Parameters

`logger` ILogger

The logger instance.

`aiSummarizer` AISummarizer

The AI summarizer instance.

`hierarchyDetector` MetadataHierarchyDetector

The metadata hierarchy detector for extracting metadata from directory structure.

`appConfig` AppConfig

Optional application configuration for advanced hierarchy detection.

Examples

```
var processor = new MarkdownNoteProcessor(logger, aiSummarizer, hierarchyDetector);
```

Remarks

This constructor initializes the markdown note builder and AI summarizer, ensuring all dependencies are valid.

Methods

ConvertToMarkdownAsync(string, string?, string?)

Converts a TXT, HTML, or EPUB file to markdown, with optional AI-generated summary.

```
public Task<string> ConvertToMarkdownAsync(string inputPath, string? openAiApiKey = null,
string? promptFileName = null)
```

Parameters

`inputPath` string

Path to the input file.

`openAiApiKey` string

OpenAI API key (optional).

`promptFileName` string

Prompt file for AI summary (optional).

Returns

Task <string>

Markdown note as a string.

Examples

```
var markdown = await processor.ConvertToMarkdownAsync("input.html", openAiApiKey: "your-api-
key", promptFileName: "summary_prompt");
Console.WriteLine(markdown);
```

Remarks

This method processes the input file based on its extension and converts it to markdown. Supported file types:

- TXT: Reads the file content directly
- HTML: Strips HTML tags to extract text
- EPUB: Parses the EPUB file and extracts text from its reading order

If the OpenAI API key and prompt file name are provided, the method generates an AI summary for the content.

Class MarkdownParser

Namespace: NotebookAutomation.Core.Tools.MarkdownGeneration

Assembly: NotebookAutomation.Core.dll

Parser for markdown content with frontmatter handling.

```
public class MarkdownParser
```

Inheritance

object ← MarkdownParser

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Examples

```
var parser = new MarkdownParser(logger);
var (frontmatter, content) = await parser.ParseFileAsync("example.md");
Console.WriteLine(frontmatter);
Console.WriteLine(content);
```

Remarks

This class provides functionality for parsing and manipulating markdown content, including:

- Frontmatter extraction
- Content formatting
- Structure analysis

The parser integrates with the YAML helper for frontmatter processing and provides detailed logging for diagnostics.

Constructors

MarkdownParser(ILogger)

Parser for markdown content with frontmatter handling.

```
public MarkdownParser(ILogger logger)
```

Parameters

`logger` ILogger

Examples

```
var parser = new MarkdownParser(logger);
var (frontmatter, content) = await parser.ParseFileAsync("example.md");
Console.WriteLine(frontmatter);
Console.WriteLine(content);
```

Remarks

This class provides functionality for parsing and manipulating markdown content, including:

- Frontmatter extraction
- Content formatting
- Structure analysis

The parser integrates with the YAML helper for frontmatter processing and provides detailed logging for diagnostics.

Fields

FrontmatterRegex

Regular expression for detecting YAML frontmatter.

```
public static readonly Regex FrontmatterRegex
```

Field Value

Regex

Remarks

This regex is used to identify and extract YAML frontmatter blocks from markdown files. It matches blocks enclosed within triple dashes ("---") and captures the content between them.

Methods

CombineMarkdown(Dictionary<string, object>, string)

Combines frontmatter and content into a complete markdown document.

```
public string CombineMarkdown(Dictionary<string, object> frontmatter, string content)
```

Parameters

`frontmatter` Dictionary<string, object>

The frontmatter dictionary.

`content` string

The content body.

Returns

string

The complete markdown document.

ExtractHeaders(string)

Extracts all headers from markdown content.

```
public static List<(int Level, string Title, int LineNumber)> ExtractHeaders(string content)
```

Parameters

`content` string

The markdown content to parse.

Returns

List<(int Level, string Title, int LineNumber)>

A list of header level and title tuples.

ParseFileAsync(string)

```
public Task<(Dictionary<string, object> Frontmatter, string Content)>
ParseFileAsync(string filePath)
```

Parameters

`filePath` string

Returns

Task<(Dictionary<string, object> Frontmatter, string Content)>

ParseHeader(string)

Extracts the header level and title from a markdown header line.

```
public static (int Level, string Title) ParseHeader(string headerLine)
```

Parameters

`headerLine` string

The header line to parse.

Returns

(int Level, string Title)

A tuple containing the header level and title.

ParseMarkdown(string)

Parses markdown text into frontmatter and content.

```
public (Dictionary<string, object> Frontmatter, string Content)
ParseMarkdown(string markdownText)
```

Parameters

`markdownText` string

The full markdown text.

Returns

(Dictionary <string, object > Frontmatter, string Content)

A tuple containing the frontmatter dictionary and the content body.

SanitizeForFilename(string)

Sanitizes a string for use in a filename.

```
public static string SanitizeForFilename(string input)
```

Parameters

`input` string

The input string to sanitize.

Returns

string

A sanitized filename-safe string.

WriteFileAsync(string, Dictionary<string, object>, string)

Writes a markdown document to a file.

```
public Task<bool> WriteFileAsync(string filePath, Dictionary<string, object> frontmatter,
string content)
```

Parameters

`filePath` string

The target file path.

`frontmatter` Dictionary<string, object>

The frontmatter dictionary.

`content` string

The content body.

Returns

Task<bool>

True if successful, false otherwise.

Namespace NotebookAutomation.Core.Tools.PdfProcessing

Classes

PdfNoteBatchProcessor

Provides batch processing capabilities for converting multiple PDF files to markdown notes.

PdfNoteProcessor

Provides functionality for extracting text and metadata from PDF files and generating markdown notes.

Class PdfNoteBatchProcessor

Namespace: NotebookAutomation.Core.Tools.PdfProcessing

Assembly: NotebookAutomation.Core.dll

Provides batch processing capabilities for converting multiple PDF files to markdown notes.

```
public class PdfNoteBatchProcessor
```

Inheritance

object ← PdfNoteBatchProcessor

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

The PdfNoteBatchProcessor class coordinates the processing of multiple PDF files, either from a specified directory or a single file path. It leverages the PdfNoteProcessor to handle the details of text extraction and note generation for each PDF.

This batch processor is responsible for:

- Identifying PDF files based on their extensions
- Coordinating the processing of each PDF file
- Managing output directory creation and file writing
- Tracking success and failure counts
- Supporting dry run mode for testing without file writes

The class is designed to be used by both CLI and API interfaces, providing a central point for PDF batch processing operations with appropriate logging and error handling. This implementation delegates all batch processing logic to the generic DocumentNoteBatchProcessor<TProcessor> for maintainability and code reuse.

Constructors

PdfNoteBatchProcessor(DocumentNoteBatchProcessor<PdfNoteProcessor>)

Provides batch processing capabilities for converting multiple PDF files to markdown notes.

```
public PdfNoteBatchProcessor(DocumentNoteBatchProcessor<PdfNoteProcessor> batchProcessor)
```

Parameters

`batchProcessor` DocumentNoteBatchProcessor<PdfNoteProcessor>

The batch processor to use for PDF note processing.

Remarks

The PdfNoteBatchProcessor class coordinates the processing of multiple PDF files, either from a specified directory or a single file path. It leverages the PdfNoteProcessor to handle the details of text extraction and note generation for each PDF.

This batch processor is responsible for:

- Identifying PDF files based on their extensions
- Coordinating the processing of each PDF file
- Managing output directory creation and file writing
- Tracking success and failure counts
- Supporting dry run mode for testing without file writes

The class is designed to be used by both CLI and API interfaces, providing a central point for PDF batch processing operations with appropriate logging and error handling. This implementation delegates all batch processing logic to the generic DocumentNoteBatchProcessor<TProcessor> for maintainability and code reuse.

Methods

ProcessPdfsAsync(string, string?, List<string>?, string?, bool)

Processes one or more PDF files, generating markdown notes for each.

```
public Task<BatchProcessResult> ProcessPdfsAsync(string input, string? output, List<string>?
pdfExtensions = null, string? openAiApiKey = null, bool dryRun = false)
```

Parameters

`input` string

Input file path or directory containing PDF files.

`output` string

Output directory where markdown notes will be saved.

`pdfExtensions` List<string>

List of file extensions to recognize as PDF files (defaults to [".pdf"]).

`openAiApiKey` string

Optional OpenAI API key for generating summaries.

`dryRun` bool

If true, simulates processing without writing output files.

Returns

Task<BatchProcessResult>

A tuple containing the count of successfully processed files and the count of failures.

Examples

```
// Process all PDF files in a directory
var processor = new PdfNoteBatchProcessor(logger);
var result = await processor.ProcessPdfsAsync(
    "path/to/pdfs",
    "path/to/notes",
    new List<string> { ".pdf" },
    "sk-yourapikeyhere");

Console.WriteLine($"Processed: {result.processed}, Failed: {result.failed}");
```

Remarks

This method delegates to the generic DocumentNoteBatchProcessor<TProcessor> for all batch processing operations while maintaining backward compatibility with existing PDF-specific API.

ProcessPdfsAsync(string, string?, List<string>?, string?, bool, bool, bool, bool, int?, string?, AppConfig?)

Processes one or more PDF files, generating markdown notes for each, with extended options.

```
public Task<BatchProcessResult> ProcessPdfsAsync(string input, string? output, List<string>?
pdfExtensions, string? openAiApiKey, bool dryRun = false, bool noSummary = false, bool
forceOverwrite = false, bool retryFailed = false, int? timeoutSeconds = null, string?
resourcesRoot = null, AppConfig? appConfig = null)
```

Parameters

`input` string

Input file path or directory containing PDF files.

`output` string

Output directory where markdown notes will be saved.

`pdfExtensions` List<string>

List of file extensions to recognize as PDF files (defaults to [".pdf"]).

`openAiApiKey` string

Optional OpenAI API key for generating summaries.

`dryRun` bool

If true, simulates processing without writing output files.

`noSummary` bool

If true, disables OpenAI summary generation.

`forceOverwrite` bool

If true, overwrites existing notes.

`retryFailed` bool

If true, retries only failed files from previous run.

`timeoutSeconds` int?

Optional API request timeout in seconds.

`resourcesRoot` string

Optional override for OneDrive fullpath root directory.

`appConfig` AppConfig

The application configuration object.

Returns

Task<BatchProcessResult>

A tuple containing the count of successfully processed files and the count of failures.

Events

ProcessingProgressChanged

Event triggered when processing progress changes.

```
public event EventHandler<DocumentProcessingProgressEventArgs>? ProcessingProgressChanged
```

Event Type

EventHandler<DocumentProcessingProgressEventArgs>

Class PdfNoteProcessor

Namespace: NotebookAutomation.Core.Tools.PdfProcessing

Assembly: NotebookAutomation.Core.dll

Provides functionality for extracting text and metadata from PDF files and generating markdown notes.

```
public class PdfNoteProcessor : DocumentNoteProcessorBase
```

Inheritance

object ← DocumentNoteProcessorBase ← PdfNoteProcessor

Inherited Members

DocumentNoteProcessorBase.Logger , DocumentNoteProcessorBase.Summarizer , DocumentNoteProcessorBase.GenerateAiSummaryAsync(string, Dictionary<string, string>, string) , DocumentNoteProcessorBase.GenerateMarkdownNote(string, Dictionary<string, object>, string, bool, bool) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Examples

```
var processor = new PdfNoteProcessor(logger, aiSummarizer);
var (text, metadata) = await processor.ExtractTextAndMetadataAsync("example.pdf");
Console.WriteLine(text);
Console.WriteLine(metadata);
```

Remarks

This class integrates with the AI summarizer and YAML helper to process PDF files and generate markdown notes. It supports:

- Text extraction from PDF pages
- Metadata extraction (e.g., title, author, keywords)
- Course structure detection (module and lesson information)
- Markdown note generation with YAML frontmatter

The class logs detailed diagnostic information during processing and handles errors gracefully.

Constructors

PdfNoteProcessor(ILogger<PdfNoteProcessor>, AISummarizer, IYamlHelper, MetadataHierarchyDetector, AppConfig?)

Provides functionality for extracting text and metadata from PDF files and generating markdown notes.

```
public PdfNoteProcessor(ILogger<PdfNoteProcessor> logger, AISummarizer aiSummarizer,
IYamlHelper yamlHelper, MetadataHierarchyDetector hierarchyDetector, AppConfig? appConfig
= null)
```

Parameters

`logger` ILogger<PdfNoteProcessor>

Logger for diagnostics.

`aiSummarizer` AISummarizer

The AISummarizer service for generating AI-powered summaries.

`yamlHelper` IYamlHelper

The YAML helper for processing YAML frontmatter.

`hierarchyDetector` MetadataHierarchyDetector

The metadata hierarchy detector for extracting metadata from directory structure.

`appConfig` AppConfig

Optional application configuration for advanced hierarchy detection.

Examples

```
var processor = new PdfNoteProcessor(logger, aiSummarizer);
var (text, metadata) = await processor.ExtractTextAndMetadataAsync("example.pdf");
Console.WriteLine(text);
Console.WriteLine(metadata);
```

Remarks

This class integrates with the AI summarizer and YAML helper to process PDF files and generate markdown notes. It supports:

- Text extraction from PDF pages
- Metadata extraction (e.g., title, author, keywords)
- Course structure detection (module and lesson information)
- Markdown note generation with YAML frontmatter

The class logs detailed diagnostic information during processing and handles errors gracefully.

Methods

ExtractTextAndMetadataAsync(string)

Extracts text and metadata from a PDF file.

```
public override Task<(string Text, Dictionary<string, object> Metadata)>
ExtractTextAndMetadataAsync(string pdfPath)
```

Parameters

`pdfPath` string

Path to the PDF file.

Returns

Task<(string Text, Dictionary<string, object> Metadata)>

Tuple of extracted text and metadata dictionary.

Examples

```
var (text, metadata) = await processor.ExtractTextAndMetadataAsync("example.pdf");
Console.WriteLine(text);
Console.WriteLine(metadata);
```

Remarks

This method reads the PDF file, extracts text from its pages, and collects metadata such as:

- Page count
- Title, author, subject, and keywords
- File size and creation date
- Course structure information (module and lesson)

The extracted text and metadata are returned as a tuple. If the file does not exist or an error occurs, the method logs the issue and returns empty results.

GenerateMarkdownNote(string, Dictionary<string, object>?)

Generates a markdown note from extracted PDF text and metadata.

```
public string GenerateMarkdownNote(string pdfText, Dictionary<string, object>? metadata
= null)
```

Parameters

`pdfText` string

The extracted PDF text.

`metadata` Dictionary<string, object>

Optional metadata for the note.

Returns

string

The generated markdown content.

GeneratePdfSummaryAsync(string, Dictionary<string, object>, string?)

Generates an AI summary for the PDF content with proper variable substitution.

```
public Task<string> GeneratePdfSummaryAsync(string pdfText, Dictionary<string, object>
metadata, string? promptFileName = null)
```

Parameters

`pdfText` string

The extracted PDF text.

`metadata` Dictionary<string, object>

The PDF metadata dictionary.

`promptFileName` string

Optional prompt template file name.

Returns

Task<string>

The AI-generated summary text.

Namespace NotebookAutomation.Core.Tools.Shared

Classes

BatchProcessResult

Represents the result and statistics of a batch document processing operation.

DocumentNoteBatchProcessor<TProcessor>

DocumentNoteProcessorBase

Abstract base class for document note processors (PDF, video, etc.).

Class BatchProcessResult

Namespace: NotebookAutomation.Core.Tools.Shared

Assembly: NotebookAutomation.Core.dll

Represents the result and statistics of a batch document processing operation.

```
public class BatchProcessResult
```

Inheritance

object ← BatchProcessResult

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Examples

```
var result = new BatchProcessResult
{
    Processed = 10,
    Failed = 2,
    TotalBatchTime = TimeSpan.FromMinutes(15),
    TotalSummaryTime = TimeSpan.FromMinutes(5),
    TotalTokens = 5000,
    AverageFileTimeMs = 900,
    AverageSummaryTimeMs = 300,
    AverageTokens = 500,
    Summary = "Processed 10 files with 2 failures."
};
Console.WriteLine(result.Summary);
```

Remarks

This class provides detailed statistics about the batch processing operation, including:

- Number of files successfully processed
- Number of files that failed to process
- Total processing time and summary generation time
- Token usage statistics for AI summaries
- Average processing times and token counts

The class is designed for use in CLI or UI applications to provide user-friendly summaries of batch processing results.

Properties

AverageFileTimeMs

Gets or sets average time per file in milliseconds.

```
public double AverageFileTimeMs { get; set; }
```

Property Value

double

Remarks

This property represents the average time taken to process each file in the batch operation.

AverageSummaryTimeMs

Gets or sets average summary time per file in milliseconds.

```
public double AverageSummaryTimeMs { get; set; }
```

Property Value

double

Remarks

This property represents the average time spent generating summaries for each file in the batch.

AverageTokens

Gets or sets average tokens per summary.

```
public double AverageTokens { get; set; }
```

Property Value

double

Remarks

This property represents the average number of tokens consumed by the AI summarizer for each file in the batch.

Failed

Gets or sets number of files that failed to process.

```
public int Failed { get; set; }
```

Property Value

int

Remarks

This property represents the count of files that encountered errors and could not be processed.

Processed

Gets or sets number of files successfully processed.

```
public int Processed { get; set; }
```

Property Value

int

Remarks

This property represents the count of files that were processed successfully during the batch operation.

Summary

Gets or sets user-friendly summary string for CLI or UI output.

```
public string Summary { get; set; }
```

Property Value

string

Remarks

This property provides a concise summary of the batch processing results, suitable for display in command-line interfaces or user interfaces.

TotalBatchTime

Gets or sets total batch processing time.

```
public TimeSpan TotalBatchTime { get; set; }
```

Property Value

TimeSpan

Remarks

This property represents the total time taken to process all files in the batch operation.

TotalSummaryTime

Gets or sets total time spent generating summaries.

```
public TimeSpan TotalSummaryTime { get; set; }
```

Property Value

TimeSpan

Remarks

This property represents the cumulative time spent generating AI summaries for all files in the batch.

TotalTokens

Gets or sets total tokens used for all summaries.

```
public int TotalTokens { get; set; }
```

Property Value

int

Remarks

This property represents the total number of tokens consumed by the AI summarizer during the batch operation.

Class DocumentNoteBatchProcessor<TProcessor>

Namespace: NotebookAutomation.Core.Tools.Shared

Assembly: NotebookAutomation.Core.dll

```
public class DocumentNoteBatchProcessor<TProcessor> where TProcessor
: DocumentNoteProcessorBase
```

Type Parameters

`TProcessor`

Inheritance

object ← DocumentNoteBatchProcessor<TProcessor>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

DocumentNoteBatchProcessor(ILogger, TProcessor, AISummarizer)

Initializes a new instance of the DocumentNoteBatchProcessor<TProcessor> class.

```
public DocumentNoteBatchProcessor(ILogger logger, TProcessor processor,
AISummarizer aiSummarizer)
```

Parameters

`logger` ILogger

The logger instance.

`processor` TProcessor

The document note processor instance.

`aiSummarizer` AISummarizer

The AI summarizer instance.

Properties

Queue

Gets a read-only view of the current processing queue.

```
public IReadOnlyList<QueueItem> Queue { get; }
```

Property Value

IReadOnlyList<QueueItem>

Examples

```
var queue = processor.Queue;
foreach (var item in queue)
{
    Console.WriteLine(item.FilePath);
}
```

Remarks

This property provides a snapshot of the current processing queue, allowing external components to inspect the queue without modifying it.

Methods

CompileProcessingResults(int, int, TimeSpan, TimeSpan, int)

Compiles processing results and statistics into a BatchProcessResult.

```
protected virtual BatchProcessResult CompileProcessingResults(int processed, int failed,
```

```
TimeSpan batchTime, TimeSpan totalSummaryTime, int totalTokens)
```

Parameters

`processed` int

Number of successfully processed files.

`failed` int

Number of failed files.

`batchTime` TimeSpan

Total batch processing time.

`totalSummaryTime` TimeSpan

Total AI summary time.

`totalTokens` int

Total tokens used.

Returns

BatchProcessResult

Compiled batch processing result.

DiscoverAndFilterFiles(string, List<string>, bool, string, string)

Discovers files to process based on input path and file extensions.

```
protected virtual List<string>? DiscoverAndFilterFiles(string effectiveInput, List<string>
fileExtensions, bool retryFailed, string effectiveOutput, string failedFilesListName)
```

Parameters

`effectiveInput` string

Effective input path.

`fileExtensions` List<string>

List of file extensions to process.

`retryFailed` bool

Whether to retry only failed files.

`effectiveOutput` string

Output directory for failed files list.

`failedFilesListName` string

Name of failed files list.

Returns

List<string>

List of files to process, or null if discovery failed.

ExtractContentAsync(string, QueueItem?, int, int)

Extracts content and metadata from a file with progress tracking.

```
protected virtual Task<(string text, Dictionary<string, object> metadata)>
ExtractContentAsync(string filePath, QueueItem? queueItem, int fileIndex, int totalFiles)
```

Parameters

`filePath` string

Path to the file to extract from.

`queueItem` QueueItem

Queue item to update with progress.

`fileIndex` int

Current file index.

`totalFiles` int

Total number of files.

Returns

Task<(string Text, Dictionary<string, object> Metadata)>

Tuple containing extracted text and metadata.

GenerateAISummaryAsync(string, string, Dictionary<string, object>, QueueItem?, int, int, string?, bool, int?, string?, bool)

Generates AI summary using processor-specific methods.

```
protected virtual Task<(string summaryText, int summaryTokens, TimeSpan summaryTime)>
GenerateAISummaryAsync(string filePath, string text, Dictionary<string, object> metadata,
QueueItem? queueItem, int fileIndex, int totalFiles, string? openAiApiKey, bool noSummary,
int? timeoutSeconds, string? resourcesRoot, bool noShareLinks)
```

Parameters

`filePath` string

Path to the file being processed.

`text` string

Extracted text content.

`metadata` Dictionary<string, object>

File metadata.

`queueItem` QueueItem

Queue item to update with progress.

`fileIndex` int

Current file index.

`totalFiles` int

Total number of files.

`openAiApiKey` string

OpenAI API key.

`noSummary` bool

Whether to skip summary generation.

`timeoutSeconds` int?

API timeout in seconds.

`resourcesRoot` string

Resources root directory.

`noShareLinks` bool

Whether to skip share link generation.

Returns

Task<(string summaryText, int summaryTokens, TimeSpan summaryTime)>

Tuple containing summary text, token count, and summary time.

GenerateAndSaveMarkdownAsync(string, string, Dictionary<string, object>, string, string, QueueItem?, int, int, bool, bool)

Generates markdown content and saves it to file with progress tracking.

```
protected virtual Task GenerateAndSaveMarkdownAsync(string filePath, string summaryText,
Dictionary<string, object> metadata, string noteType, string outputPath, QueueItem?
queueItem, int fileIndex, int totalFiles, bool forceOverwrite, bool dryRun)
```

Parameters

`filePath` string

Source file path.

`summaryText` string

Generated summary text.

`metadata` Dictionary<string, object>

File metadata.

`noteType` string

Type of note to generate.

`outputPath` string

Output file path.

`queueItem` QueueItem

Queue item for progress tracking.

`fileIndex` int

Current file index.

`totalFiles` int

Total number of files.

`forceOverwrite` bool

Whether to overwrite existing files.

`dryRun` bool

Whether this is a dry run.

Returns

Task

A Task representing the asynchronous operation.

GenerateOutputPath(string, string, string?)

Generates the output path for a processed file, handling video-specific naming and directory structure.

```
protected virtual string GenerateOutputPath(string inputFilePath, string outputDir,
string? resourcesRoot)
```

Parameters

`inputFilePath` string

The input file path.

`outputDir` string

The base output directory.

`resourcesRoot` string

The resources root directory for calculating relative paths.

Returns

string

The output file path.

GetDocumentType(string)

Determines the document type based on file extension.

```
protected virtual string GetDocumentType(string filePath)
```

Parameters

`filePath` string

File path to check.

Returns

string

Document type (e.g., "PDF", "VIDEO").

InitializeProcessingQueue(List<string>)

Initializes the processing queue with discovered files.

```
protected virtual void InitializeProcessingQueue(List<string> files)
```

Parameters

`files` List<string>

List of files to add to queue.

OnProcessingProgressChanged(string, string, int, int)

Raises the ProcessingProgressChanged event.

```
protected virtual void OnProcessingProgressChanged(string filePath, string status, int
currentFile, int totalFiles)
```

Parameters

`filePath` string

The path of the file being processed.

`status` string

The current processing status message.

`currentFile` int

The current file index being processed.

`totalFiles` int

The total number of files to process.

OnQueueChanged(QueueItem?)

Raises the QueueChanged event.

```
protected virtual void OnQueueChanged(QueueItem? changedItem = null)
```

Parameters

`changedItem` QueueItem

The specific item that changed, or null if the entire queue changed.

PreserveUserContentAfterNotes(string, string)

Preserves user content that appears after the "## Notes" section in existing video markdown files.

```
protected virtual string PreserveUserContentAfterNotes(string existingFilePath,
string newMarkdown)
```

Parameters

`existingFilePath` string

Path to the existing markdown file.

`newMarkdown` string

The newly generated markdown content.

Returns

string

The new markdown with preserved user content appended.

ProcessDocumentsAsync(string, string?, List<string>, string?, bool, bool, bool, bool, int?, string?, AppConfig?, string, string, bool)

Processes one or more document files, generating markdown notes for each, with extended options. Returns a BatchProcessResult with summary and statistics for CLI/UI output.

```
public virtual Task<BatchProcessResult> ProcessDocumentsAsync(string input, string? output,
List<string> fileExtensions, string? openAiApiKey, bool dryRun = false, bool noSummary =
false, bool forceOverwrite = false, bool retryFailed = false, int? timeoutSeconds = null,
string? resourcesRoot = null, AppConfig? appConfig = null, string noteType = "Document
Note", string failedFilesListName = "failed_files.txt", bool noShareLinks = false)
```

Parameters

`input` string

Input file path or directory containing files.

`output` string

Output directory where markdown notes will be saved.

`fileExtensions` List<string>

List of file extensions to recognize as valid files.

`openAiApiKey` string

Optional OpenAI API key for generating summaries.

`dryRun` bool

If true, simulates processing without writing output files.

`noSummary` bool

If true, disables OpenAI summary generation.

`forceOverwrite` bool

If true, overwrites existing notes.

`retryFailed` bool

If true, retries only failed files from previous run.

`timeoutSeconds` int?

Optional API request timeout in seconds.

`resourcesRoot` string

Optional override for OneDrive fullpath root directory.

`appConfig` AppConfig

The application configuration object.

`noteType` string

Type of note (e.g., "PDF Note", "Video Note").

`failedFilesListName` string

Name of the failed files list file (defaults to "failed_files.txt").

`noShareLinks` bool

If true, skips OneDrive share link creation.

Returns

Task<BatchProcessResult>

A BatchProcessResult containing processing statistics and summary information.

ProcessSingleFileAsync(string, QueueItem?, int, int, string, string?, bool, bool, string?, bool, int?, bool, string)

Processes a single file with complete error handling and progress tracking.

```
protected virtual Task<(bool success, string? errorMessage)> ProcessSingleFileAsync(string
filePath, QueueItem? queueItem, int fileIndex, int totalFiles, string effectiveOutput,
string? resourcesRoot, bool forceOverwrite, bool dryRun, string? openAiApiKey, bool
noSummary, int? timeoutSeconds, bool noShareLinks, string noteType)
```

Parameters

`filePath` string

Path to the file to process.

`queueItem` QueueItem

Queue item for progress tracking.

`fileIndex` int

Current file index.

`totalFiles` int

Total number of files.

`effectiveOutput` string

Output directory.

`resourcesRoot` string

Resources root directory.

`forceOverwrite` bool

Whether to overwrite existing files.

`dryRun` bool

Whether this is a dry run.

`openAiApiKey` string

OpenAI API key.

`noSummary` bool

Whether to skip summary generation.

`timeoutSeconds` int?

API timeout in seconds.

`noShareLinks` bool

Whether to skip share link generation.

`noteType` string

Type of note to generate.

Returns

Task<(bool success, string errorMessage)>

Tuple indicating success and any error message.

ValidateAndSetupProcessing(string, string?, AppConfig?)

Validates input parameters and sets up processing configuration.

```
protected virtual (string effectiveInput, string effectiveOutput)?
ValidateAndSetupProcessing(string input, string? output, AppConfig? appConfig)
```

Parameters

`input` string

Input path.

`output` string

Output path.

`appConfig` AppConfig

Application configuration.

Returns

(string effectiveInput, string effectiveOutput)?

Tuple containing effective input and output paths, or null if validation failed.

Events

ProcessingProgressChanged

Event triggered when processing progress changes.

```
public event EventHandler<DocumentProcessingProgressEventArgs>? ProcessingProgressChanged
```

Event Type

EventHandler <DocumentProcessingProgressEventArgs>

Examples

```
processor.ProcessingProgressChanged += (sender, args) => Console.WriteLine(args.Progress);
```

Remarks

This event is raised whenever the progress of the batch processing operation changes. Subscribers can use this event to update progress indicators or logs.

QueueChanged

Event triggered when the processing queue changes.

```
public event EventHandler<QueueChangedEventArgs>? QueueChanged
```

Event Type

EventHandler <QueueChangedEventArgs>

Examples

```
processor.QueueChanged += (sender, args) => Console.WriteLine(args.Queue);
```

Remarks

This event is raised whenever the processing queue is updated (e.g., items added or removed). Subscribers can use this event to monitor the state of the queue.

Class DocumentNoteProcessorBase

Namespace: NotebookAutomation.Core.Tools.Shared

Assembly: NotebookAutomation.Core.dll

Abstract base class for document note processors (PDF, video, etc.).

```
public abstract class DocumentNoteProcessorBase
```

Inheritance

object ← DocumentNoteProcessorBase

Derived

PdfNoteProcessor, VideoNoteProcessor

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

The `DocumentNoteProcessorBase` class provides shared logic for processing document notes, including AI summary generation, markdown creation, and logging. It serves as a foundation for specialized processors that handle specific document types, such as PDFs and videos.

Constructors

DocumentNoteProcessorBase(ILogger, AISummarizer)

Abstract base class for document note processors (PDF, video, etc.).

```
protected DocumentNoteProcessorBase(ILogger logger, AISummarizer aiSummarizer)
```

Parameters

`logger` ILogger

The logger instance.

`aiSummarizer` AISummarizer

The AISummarizer instance for generating AI-powered summaries.

Remarks

The `DocumentNoteProcessorBase` class provides shared logic for processing document notes, including AI summary generation, markdown creation, and logging. It serves as a foundation for specialized processors that handle specific document types, such as PDFs and videos.

Fields

Logger

```
protected readonly ILogger Logger
```

Field Value

ILogger

Summarizer

```
protected readonly AISummarizer Summarizer
```

Field Value

AISummarizer

Methods

ExtractTextAndMetadataAsync(string)

Extracts the main text/content and metadata from the document.

```
public abstract Task<(string Text, Dictionary<string, object> Metadata)>
ExtractTextAndMetadataAsync(string filePath)
```

Parameters

`filePath` string

Path to the document file.

Returns

Task<(string Text, Dictionary<string, object> Metadata)>

Tuple of extracted text/content and metadata dictionary.

GenerateAiSummaryAsync(string?, Dictionary<string, string>?, string?)

Generates an AI summary for the given text using OpenAI.

```
public virtual Task<string> GenerateAiSummaryAsync(string? text, Dictionary<string, string>?
variables = null, string? promptFileName = null)
```

Parameters

`text` string

The extracted text/content.

`variables` Dictionary<string, string>

Optional variables to substitute in the prompt template.

`promptFileName` string

Optional name of the prompt template file to use.

Returns

Task<string>

The summary text, or a simulated summary if unavailable.

GenerateMarkdownNote(string, Dictionary<string, object>?, string, bool, bool)

Generates a markdown note from extracted text and metadata.

```
public virtual string GenerateMarkdownNote(string bodyText, Dictionary<string, object>?
metadata = null, string noteType = "Document Note", bool suppressBody = false, bool
includeNoteTypeTitle = false)
```

Parameters

`bodyText` string

The extracted text/content.

`metadata` Dictionary<string, object>

Optional metadata for the note.

`noteType` string

Type of note (e.g., "PDF Note", "Video Note").

`suppressBody` bool

Whether to suppress the body text and only include frontmatter.

`includeNoteTypeTitle` bool

Whether to include the note type as a title in the markdown.

Returns

string

The generated markdown content.

Namespace NotebookAutomation.Core.Tools.TagManagement

Classes

TagProcessor

Processes tags in markdown files and handles tag management operations.

Class TagProcessor

Namespace: NotebookAutomation.Core.Tools.TagManagement

Assembly: NotebookAutomation.Core.dll

Processes tags in markdown files and handles tag management operations.

```
public class TagProcessor
```

Inheritance

object ← TagProcessor

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

The `TagProcessor` class provides functionality for managing tags in markdown files, including adding nested tags, restructuring tag hierarchies, clearing tags from index files, and enforcing metadata consistency. It supports dry-run mode for testing changes without modifying files and verbose logging for detailed diagnostics.

Constructors

TagProcessor(ILogger<TagProcessor>, ILogger, IYamlHelper, bool, bool)

Initializes a new instance of the TagProcessor class. Initializes a new instance of the TagProcessor.

```
public TagProcessor(ILogger<TagProcessor> logger, ILogger failedLogger, IYamlHelper
yamlHelper, bool dryRun = false, bool verbose = false)
```

Parameters

`logger` ILogger<TagProcessor>

Logger for general diagnostics.

`failedLogger` ILogger

Logger for recording failed operations.

`yamlHelper` IYamlHelper

Helper for YAML processing.

`dryRun` bool

Whether to perform a dry run without making changes.

`verbose` bool

Whether to provide verbose output.

TagProcessor(ILogger<TagProcessor>, ILogger, IYamlHelper, bool, bool, HashSet<string>?)

Initializes a new instance of the TagProcessor class. Initializes a new instance of the TagProcessor with specified fields to process.

```
public TagProcessor(ILogger<TagProcessor> logger, ILogger failedLogger, IYamlHelper
yamlHelper, bool dryRun = false, bool verbose = false, HashSet<string>? fieldsToProcess
= null)
```

Parameters

`logger` ILogger<TagProcessor>

Logger for general diagnostics.

`failedLogger` ILogger

Logger for recording failed operations.

`yamlHelper` IYamlHelper

Helper for YAML processing.

`dryRun` bool

Whether to perform a dry run without making changes.

`verbose` bool

Whether to provide verbose output.

`fieldsToProcess` HashSet<string>

Specific frontmatter fields to process for tag generation.

Properties

Stats

Gets statistics about the processing performed.

```
public Dictionary<string, int> Stats { get; }
```

Property Value

Dictionary<string, int>

Methods

AddExampleTagsToFileAsync(string)

Adds example nested tags to a markdown file for demonstration/testing.

```
public Task<bool> AddExampleTagsToFileAsync(string filePath)
```

Parameters

`filePath` string

Returns

Task<bool>

A Task representing the asynchronous operation.

AddNestedTagsToFileAsync(string, Dictionary<string, object>, string)

Adds nested tags based on frontmatter fields to a file.

```
public Task<bool> AddNestedTagsToFileAsync(string filePath, Dictionary<string, object>
frontmatter, string content)
```

Parameters

`filePath` string

Path to the markdown file.

`frontmatter` Dictionary<string, object>

The parsed frontmatter dictionary.

`content` string

The full content of the file.

Returns

Task<bool>

True if the file was modified, false otherwise.

CheckAndEnforceMetadataConsistencyAsync(string)

Checks and enforces metadata consistency in all markdown files in a directory.

```
public Task<Dictionary<string, int>> CheckAndEnforceMetadataConsistencyAsync(string
directory)
```

Parameters

`directory` string

Returns

Task<Dictionary<string, int>>

A Task representing the asynchronous operation.

CheckAndEnforceMetadataConsistencyInFileAsync(string)

Checks and enforces metadata consistency in a single markdown file.

```
public Task<bool> CheckAndEnforceMetadataConsistencyInFileAsync(string filePath)
```

Parameters

`filePath` string

Returns

Task<bool>

A Task representing the asynchronous operation.

ClearTagsFromFileAsync(string, Dictionary<string, object>, string)

Clears tags from an index file.

```
public Task<bool> ClearTagsFromFileAsync(string filePath, Dictionary<string, object>
frontmatter, string content)
```

Parameters

`filePath` string

Path to the markdown file.

`frontmatter` Dictionary<string, object>

The parsed frontmatter dictionary.

`content` string

The full content of the file.

Returns

Task<bool>

True if the file was modified, false otherwise.

DiagnoseFrontmatterIssuesAsync(string)

Diagnoses YAML frontmatter issues in markdown files.

```
public Task<List<(string FilePath, string DiagnosticMessage)>>
DiagnoseFrontmatterIssuesAsync(string directory)
```

Parameters

`directory` string

The directory path to process.

Returns

Task<List<(string FilePath, string DiagnosticMessage)>>

A list of problematic files with their diagnostic information.

GenerateNestedTags(Dictionary<string, object>, List<string>)

Generates nested tags based on frontmatter fields.

```
public List<string> GenerateNestedTags(Dictionary<string, object> frontmatter,
List<string> existingTags)
```

Parameters

`frontmatter` Dictionary<string, object>

The frontmatter dictionary.

`existingTags` List<string>

List of existing tags.

Returns

List<string>

A list of new tags to add.

GetExistingTags(Dictionary<string, object>)

Extracts existing tags from the frontmatter.

```
public static List<string> GetExistingTags(Dictionary<string, object> frontmatter)
```

Parameters

`frontmatter` Dictionary<string, object>

The frontmatter dictionary.

Returns

List<string>

A list of existing tags.

GetTagPrefixForField(string)

Gets the appropriate tag prefix for a frontmatter field.

```
public static string GetTagPrefixForField(string fieldName)
```

Parameters

`fieldName` string

The field name.

Returns

string

The tag prefix.

NormalizeTagValue(string)

Normalizes a tag value for consistent formatting.

```
public static string NormalizeTagValue(string value)
```

Parameters

`value` string

The raw tag value.

Returns

string

Normalized tag value.

ProcessDirectoryAsync(string)

Processes a directory recursively to add or update nested tags.

```
public Task<Dictionary<string, int>> ProcessDirectoryAsync(string directory)
```

Parameters

`directory` string

The directory path to process.

Returns

Task<Dictionary<string, int>>

Dictionary with processing statistics.

ProcessFileAsync(string)

Processes a single markdown file to add or update nested tags.

```
public Task<bool> ProcessFileAsync(string filePath)
```

Parameters

`filePath` string

Path to the markdown file.

Returns

Task <bool>

True if the file was modified, false otherwise.

RestructureTagsInDirectoryAsync(string)

Restructures tags in all markdown files in a directory for consistency (lowercase, dashes, etc.).

```
public Task<Dictionary<string, int>> RestructureTagsInDirectoryAsync(string directory)
```

Parameters

`directory` string

Returns

Task <Dictionary <string, int> >

A Task representing the asynchronous operation.

RestructureTagsInFileAsync(string)

Normalizes and restructures tags in a single markdown file.

```
public Task<bool> RestructureTagsInFileAsync(string filePath)
```

Parameters

`filePath` string

Returns

Task<bool>

A Task representing the asynchronous operation.

UpdateFrontmatterKeyAsync(string, string, object)

Updates or adds a specific frontmatter key-value pair in all markdown files within a directory.

```
public Task<Dictionary<string, int>> UpdateFrontmatterKeyAsync(string path, string key,
object value)
```

Parameters

`path` string

Path to a directory or file to process.

`key` string

The frontmatter key to add or update.

`value` object

The value to set for the key.

Returns

Task<Dictionary<string, int>>

Dictionary with processing statistics.

Namespace NotebookAutomation.Core.Tools.Vault

Classes

DictionaryExtensions

Extension methods for Dictionary operations.

MetadataBatchResult

Result of batch metadata processing operation.

MetadataEnsureBatchProcessor

Batch processor for ensuring metadata in multiple markdown files.

MetadataEnsureProcessor

Processor for ensuring metadata in markdown files based on directory structure. Extracts program/course/class/module/lesson metadata and updates YAML frontmatter.

VaultIndexBatchProcessor

Batch processor for generating vault index files in an Obsidian vault.

VaultIndexBatchResult

Result of batch vault index generation operation.

VaultIndexProcessor

Processor for generating individual vault index files.

Class DictionaryExtensions

Namespace: NotebookAutomation.Core.Tools.Vault

Assembly: NotebookAutomation.Core.dll

Extension methods for Dictionary operations.

```
public static class DictionaryExtensions
```

Inheritance

object ← DictionaryExtensions

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

GetValueOrDefault<TKey, TValue>(Dictionary<TKey, TValue>, TKey, TValue?)

Gets a value from dictionary or returns default if key doesn't exist.

```
public static TValue? GetValueOrDefault<TKey, TValue>(this Dictionary<TKey, TValue>
dictionary, TKey key, TValue? defaultValue = default) where TKey : notnull
```

Parameters

`dictionary` Dictionary<TKey, TValue>

`key` TKey

`defaultValue` TValue

Returns

TValue

Type Parameters

`TKey`

`TValue`

Class MetadataBatchResult

Namespace: NotebookAutomation.Core.Tools.Vault

Assembly: NotebookAutomation.Core.dll

Result of batch metadata processing operation.

```
public class MetadataBatchResult
```

Inheritance

object ← MetadataBatchResult

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

ErrorMessage

Gets or sets the error message if the operation failed.

```
public string? ErrorMessage { get; set; }
```

Property Value

string

FailedFiles

Gets or sets the number of files that failed to process.

```
public int FailedFiles { get; set; }
```

Property Value

int

ProcessedFiles

Gets or sets the number of files that were processed (updated).

```
public int ProcessedFiles { get; set; }
```

Property Value

int

SkippedFiles

Gets or sets the number of files that were skipped (no changes needed).

```
public int SkippedFiles { get; set; }
```

Property Value

int

Success

Gets or sets a value indicating whether gets or sets whether the operation was successful.

```
public bool Success { get; set; }
```

Property Value

bool

TotalFiles

Gets or sets the total number of files found.

```
public int TotalFiles { get; set; }
```

Property Value

int

Class MetadataEnsureBatchProcessor

Namespace: NotebookAutomation.Core.Tools.Vault

Assembly: NotebookAutomation.Core.dll

Batch processor for ensuring metadata in multiple markdown files.

```
public class MetadataEnsureBatchProcessor
```

Inheritance

object ← MetadataEnsureBatchProcessor

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

The `MetadataEnsureBatchProcessor` class provides functionality for processing metadata in markdown files within a vault directory. It supports queue management, progress tracking, and error handling, with eventing capabilities for real-time monitoring and UI integration.

Constructors

MetadataEnsureBatchProcessor(ILogger<MetadataEnsureBatchProcessor>, MetadataEnsureProcessor, AppConfig)

Batch processor for ensuring metadata in multiple markdown files.

```
public MetadataEnsureBatchProcessor(ILogger<MetadataEnsureBatchProcessor> logger,
MetadataEnsureProcessor processor, AppConfig appConfig)
```

Parameters

`logger` ILogger<MetadataEnsureBatchProcessor>

`processor` MetadataEnsureProcessor

`appConfig` AppConfig

Remarks

The `MetadataEnsureBatchProcessor` class provides functionality for processing metadata in markdown files within a vault directory. It supports queue management, progress tracking, and error handling, with eventing capabilities for real-time monitoring and UI integration.

Properties

Queue

Gets a read-only view of the current processing queue.

```
public IReadOnlyList<QueueItem> Queue { get; }
```

Property Value

IReadOnlyList <QueueItem>

Methods

EnsureMetadataAsync(string, bool, bool, bool)

Processes markdown files in the specified directory to ensure metadata consistency.

```
public Task<MetadataBatchResult> EnsureMetadataAsync(string vaultPath, bool dryRun = false,
bool forceOverwrite = false, bool retryFailed = false)
```

Parameters

`vaultPath` string

Path to the vault directory containing markdown files.

`dryRun` bool

If true, simulates processing without making actual changes.

`forceOverwrite` bool

If true, updates metadata even if it already exists.

`retryFailed` bool

If true, retries only files that failed in previous runs.

Returns

Task<MetadataBatchResult>

A summary of processing results.

InitializeProcessingQueue(string)

Initializes the processing queue with markdown files from the specified directory.

```
protected virtual void InitializeProcessingQueue(string vaultPath)
```

Parameters

`vaultPath` string

Path to the vault directory containing markdown files.

OnProcessingProgressChanged(string, string, int, int)

Raises the ProcessingProgressChanged event.

```
protected virtual void OnProcessingProgressChanged(string filePath, string status, int
currentFile, int totalFiles)
```

Parameters

`filePath` string

The path of the file being processed.

`status` string

The current processing status message.

`currentFile` int

The current file index being processed.

`totalFiles` int

The total number of files to process.

OnQueueChanged(QueueItem?)

Raises the QueueChanged event.

```
protected virtual void OnQueueChanged(QueueItem? changedItem = null)
```

Parameters

`changedItem` QueueItem

The specific item that changed, or null if the entire queue changed.

UpdateQueueItemStatus(string, DocumentProcessingStatus, ProcessingStage, string, int, int)

Updates the status of a specific queue item and fires events.

```
protected virtual void UpdateQueueItemStatus(string filePath, DocumentProcessingStatus
status, ProcessingStage stage, string statusMessage, int currentFile, int totalFiles)
```

Parameters

`filePath` string

The file path to update.

`status` DocumentProcessingStatus

The new processing status.

`stage` ProcessingStage

The new processing stage.

`statusMessage` string

The new status message.

`currentFile` int

Current file index for progress tracking.

`totalFiles` int

Total files for progress tracking.

Events

ProcessingProgressChanged

Event triggered when processing progress changes.

```
public event EventHandler<DocumentProcessingProgressEventArgs>? ProcessingProgressChanged
```

Event Type

EventHandler<DocumentProcessingProgressEventArgs>

QueueChanged

Event triggered when the processing queue changes.

```
public event EventHandler<QueueChangedEventArgs>? QueueChanged
```

Event Type

EventHandler<QueueChangedEventArgs>

Class MetadataEnsureProcessor

Namespace: NotebookAutomation.Core.Tools.Vault

Assembly: NotebookAutomation.Core.dll

Processor for ensuring metadata in markdown files based on directory structure. Extracts program/course/class/module/lesson metadata and updates YAML frontmatter.

```
public class MetadataEnsureProcessor
```

Inheritance

object ← MetadataEnsureProcessor

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

Initializes a new instance of the MetadataEnsureProcessor class.

Constructors

MetadataEnsureProcessor(ILogger<MetadataEnsureProcessor>, AppConfig, IYamlHelper, MetadataHierarchyDetector, CourseStructureExtractor)

Processor for ensuring metadata in markdown files based on directory structure. Extracts program/course/class/module/lesson metadata and updates YAML frontmatter.

```
public MetadataEnsureProcessor(ILogger<MetadataEnsureProcessor> logger, AppConfig
appConfig, IYamlHelper yamlHelper, MetadataHierarchyDetector metadataDetector,
CourseStructureExtractor structureExtractor)
```

Parameters

`logger` ILogger<MetadataEnsureProcessor>

The logger instance.

`appConfig` AppConfig

The application configuration.

`yamlHelper` IYamlHelper

The YAML helper for frontmatter operations.

`metadataDetector` MetadataHierarchyDetector

The metadata hierarchy detector.

`structureExtractor` CourseStructureExtractor

The course structure extractor.

Remarks

Initializes a new instance of the MetadataEnsureProcessor class.

Methods

EnsureMetadataAsync(string, bool, bool)

Ensures metadata is properly set in a markdown file.

```
public Task<bool> EnsureMetadataAsync(string filePath, bool forceOverwrite = false, bool
dryRun = false)
```

Parameters

`filePath` string

The path to the markdown file to process.

`forceOverwrite` bool

Whether to overwrite existing metadata values.

`dryRun` bool

Whether to simulate the operation without making changes.

Returns

Task<bool>

True if the file was updated (or would be updated in dry run), false otherwise.

Class VaultIndexBatchProcessor

Namespace: NotebookAutomation.Core.Tools.Vault

Assembly: NotebookAutomation.Core.dll

Batch processor for generating vault index files in an Obsidian vault.

```
public class VaultIndexBatchProcessor
```

Inheritance

object ← VaultIndexBatchProcessor

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

The `VaultIndexBatchProcessor` class provides functionality for generating index files for each folder in a vault directory. It supports queue management, progress tracking, hierarchy detection, and error handling, with eventing capabilities for real-time monitoring and UI integration.

Features:

- Generates folder-named index files (e.g., "Module 1.md" for "Module 1" folder)
- Auto-detects hierarchy levels (MBA, Program, Course, Class, Module, Lesson)
- Applies appropriate templates based on detected hierarchy
- Organizes content by type using YAML frontmatter and filename patterns
- Optional Obsidian Bases integration for dynamic content views
- Supports dry-run mode for previewing changes.

Constructors

VaultIndexBatchProcessor(ILogger<VaultIndexBatchProcessor>, VaultIndexProcessor, AppConfig)

Batch processor for generating vault index files in an Obsidian vault.

```
public VaultIndexBatchProcessor(ILogger<VaultIndexBatchProcessor> logger,
VaultIndexProcessor processor, AppConfig appConfig)
```

Parameters

`logger` ILogger<VaultIndexBatchProcessor>

`processor` VaultIndexProcessor

`appConfig` AppConfig

Remarks

The `VaultIndexBatchProcessor` class provides functionality for generating index files for each folder in a vault directory. It supports queue management, progress tracking, hierarchy detection, and error handling, with eventing capabilities for real-time monitoring and UI integration.

Features:

- Generates folder-named index files (e.g., "Module 1.md" for "Module 1" folder)
- Auto-detects hierarchy levels (MBA, Program, Course, Class, Module, Lesson)
- Applies appropriate templates based on detected hierarchy
- Organizes content by type using YAML frontmatter and filename patterns
- Optional Obsidian Bases integration for dynamic content views
- Supports dry-run mode for previewing changes.

Properties

Queue

Gets a read-only view of the current processing queue.

```
public IReadOnlyList<QueueItem> Queue { get; }
```

Property Value

IReadOnlyList<QueueItem>

Methods

GenerateIndexesAsync(string, bool, List<string>?, bool, string?)

Generates vault index files for folders in the specified directory.

```
public Task<VaultIndexBatchResult> GenerateIndexesAsync(string vaultPath, bool dryRun =
false, List<string>? templateTypes = null, bool forceOverwrite = false, string? vaultRoot
= null)
```

Parameters

`vaultPath` string

Path to the vault directory.

`dryRun` bool

If true, simulates processing without making actual changes.

`templateTypes` List<string>

Optional filter for specific template types to generate.

`forceOverwrite` bool

If true, regenerates index files even if they already exist.

`vaultRoot` string

Returns

Task<VaultIndexBatchResult>

A summary of processing results.

InitializeProcessingQueue(string, List<string>?)

Initializes the processing queue with folders from the specified vault directory.

```
protected virtual void InitializeProcessingQueue(string vaultPath, List<string>?
templateTypes = null)
```

Parameters

`vaultPath` string

Path to the vault directory to scan for folders.

`templateTypes` List<string>

OnProcessingProgressChanged(string, string, int, int)

Raises the ProcessingProgressChanged event.

```
protected virtual void OnProcessingProgressChanged(string folderPath, string status, int
currentFolder, int totalFolders)
```

Parameters

`folderPath` string

The path of the folder being processed.

`status` string

The current processing status message.

`currentFolder` int

The current folder index being processed.

`totalFolders` int

The total number of folders to process.

OnQueueChanged(QueueItem?)

Raises the QueueChanged event.

```
protected virtual void OnQueueChanged(QueueItem? changedItem = null)
```

Parameters

`changedItem` QueueItem

The specific item that changed, or null if the entire queue changed.

UpdateQueueItemStatus(string, DocumentProcessingStatus, ProcessingStage, string, int, int)

Updates the status of a specific queue item and fires events.

```
protected virtual void UpdateQueueItemStatus(string folderPath, DocumentProcessingStatus
status, ProcessingStage stage, string statusMessage, int currentFolder, int totalFolders)
```

Parameters

`folderPath` string

The folder path to update.

`status` DocumentProcessingStatus

The new processing status.

`stage` ProcessingStage

The new processing stage.

`statusMessage` string

The new status message.

`currentFolder` int

Current folder index for progress tracking.

`totalFolders` int

Total folders for progress tracking.

Events

ProcessingProgressChanged

Event triggered when processing progress changes.

```
public event EventHandler<DocumentProcessingProgressEventArgs>? ProcessingProgressChanged
```

Event Type

EventHandler <DocumentProcessingProgressEventArgs>

QueueChanged

Event triggered when the processing queue changes.

```
public event EventHandler<QueueChangedEventArgs>? QueueChanged
```

Event Type

EventHandler <QueueChangedEventArgs>

Class VaultIndexBatchResult

Namespace: NotebookAutomation.Core.Tools.Vault

Assembly: NotebookAutomation.Core.dll

Result of batch vault index generation operation.

```
public class VaultIndexBatchResult
```

Inheritance

object ← VaultIndexBatchResult

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

ErrorMessage

Gets or sets the error message if the operation failed.

```
public string? ErrorMessage { get; set; }
```

Property Value

string

FailedFolders

Gets or sets the number of folders that failed to process.

```
public int FailedFolders { get; set; }
```

Property Value

int

ProcessedFolders

Gets or sets the number of folders that were processed (index generated).

```
public int ProcessedFolders { get; set; }
```

Property Value

int

SkippedFolders

Gets or sets the number of folders that were skipped (no changes needed).

```
public int SkippedFolders { get; set; }
```

Property Value

int

Success

Gets or sets a value indicating whether gets or sets whether the operation was successful.

```
public bool Success { get; set; }
```

Property Value

bool

TotalFolders

Gets or sets the total number of folders found.

```
public int TotalFolders { get; set; }
```

Property Value

int

Class VaultIndexProcessor

Namespace: NotebookAutomation.Core.Tools.Vault

Assembly: NotebookAutomation.Core.dll

Processor for generating individual vault index files.

```
public class VaultIndexProcessor
```

Inheritance

object ← VaultIndexProcessor

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

The `VaultIndexProcessor` class handles the generation of index files for individual folders within an Obsidian vault. It detects hierarchy levels, applies appropriate templates, categorizes content by type, and optionally integrates Obsidian Bases for dynamic views.

Constructors

VaultIndexProcessor(ILogger<VaultIndexProcessor>, MetadataTemplateManager, MetadataHierarchyDetector, CourseStructureExtractor, IYamlHelper, MarkdownNoteBuilder, AppConfig, string)

Processor for generating individual vault index files.

```
public VaultIndexProcessor(ILogger<VaultIndexProcessor> logger, MetadataTemplateManager
templateManager, MetadataHierarchyDetector hierarchyDetector, CourseStructureExtractor
structureExtractor, IYamlHelper yamlHelper, MarkdownNoteBuilder noteBuilder, AppConfig
appConfig, string vaultRootPath = "")
```

Parameters

`logger` ILogger<VaultIndexProcessor>

`templateManager` MetadataTemplateManager

`hierarchyDetector` MetadataHierarchyDetector

`structureExtractor` CourseStructureExtractor

`yamlHelper` IYamlHelper

`noteBuilder` MarkdownNoteBuilder

`appConfig` AppConfig

`vaultRootPath` string

Remarks

The `VaultIndexProcessor` class handles the generation of index files for individual folders within an Obsidian vault. It detects hierarchy levels, applies appropriate templates, categorizes content by type, and optionally integrates Obsidian Bases for dynamic views.

Methods

GenerateIndexAsync(string, string, bool, bool)

Generates an index file for the specified folder.

```
public Task<bool> GenerateIndexAsync(string folderPath, string vaultPath, bool
forceOverwrite = false, bool dryRun = false)
```

Parameters

`folderPath` string

Path to the folder to generate an index for.

`vaultPath` string

Path to the vault root directory.

`forceOverwrite` bool

If true, regenerates the index even if it already exists.

`dryRun` bool

If true, simulates the operation without making actual changes.

Returns

Task<bool>

True if an index was generated or would be generated, false otherwise.

Namespace NotebookAutomation.Core.Tools.VideoProcessing

Classes

VideoNoteBatchProcessor

Provides batch processing capabilities for converting multiple video files to markdown notes.

VideoNoteProcessor

Represents a processor for handling video files to extract metadata, generate AI-powered summaries, and create markdown notes for knowledge management systems.

Class VideoNoteBatchProcessor

Namespace: NotebookAutomation.Core.Tools.VideoProcessing

Assembly: NotebookAutomation.Core.dll

Provides batch processing capabilities for converting multiple video files to markdown notes.

```
public class VideoNoteBatchProcessor
```

Inheritance

object ← VideoNoteBatchProcessor

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

The `VideoNoteBatchProcessor` class coordinates the processing of multiple video files, leveraging the VideoNoteProcessor for metadata extraction, transcript loading, and note generation. It supports dry-run mode, output directory management, and eventing for real-time progress tracking.

Constructors

VideoNoteBatchProcessor(DocumentNoteBatchProcessor<VideoNoteProcessor>)

Provides batch processing capabilities for converting multiple video files to markdown notes.

```
public VideoNoteBatchProcessor(DocumentNoteBatchProcessor<VideoNoteProcessor>
batchProcessor)
```

Parameters

`batchProcessor` DocumentNoteBatchProcessor<VideoNoteProcessor>

Remarks

The `VideoNoteBatchProcessor` class coordinates the processing of multiple video files, leveraging the VideoNoteProcessor for metadata extraction, transcript loading, and note generation. It supports dry-run mode, output directory management, and eventing for real-time progress tracking.

Methods

ProcessVideosAsync(string, string?, List<string>, string?, bool, bool, bool, bool, int?, string?, AppConfig?, bool)

Processes one or more video files, generating markdown notes for each.

```
public Task<BatchProcessResult> ProcessVideosAsync(string input, string? output,
List<string> videoExtensions, string? openAiApiKey, bool dryRun = false, bool noSummary =
false, bool forceOverwrite = false, bool retryFailed = false, int? timeoutSeconds = null,
string? resourcesRoot = null, AppConfig? appConfig = null, bool noShareLinks = false)
```

Parameters

`input` string

Input file path or directory containing video files.

`output` string

Output directory where markdown notes will be saved.

`videoExtensions` List<string>

List of file extensions to recognize as video files.

`openAiApiKey` string

Optional OpenAI API key for generating summaries.

`dryRun` bool

If true, simulates processing without writing output files.

`noSummary` bool

If true, disables OpenAI summary generation.

`forceOverwrite` bool

If true, overwrites existing notes.

`retryFailed` bool

If true, retries only failed files from previous run.

`timeoutSeconds` int?

Optional API request timeout in seconds.

`resourcesRoot` string

Optional override for OneDrive fullpath root directory.

`appConfig` AppConfig

The application configuration object.

`noShareLinks` bool

If true, skips OneDrive share link creation.

Returns

Task<BatchProcessResult>

A tuple containing the count of successfully processed files and the count of failures.

Examples

```
// Process all video files in a directory
var processor = new VideoNoteBatchProcessor(logger);
var result = await processor.ProcessVideosAsync(
    "path/to/videos",
    "path/to/notes",
    new List<string> { ".mp4", ".mov", ".avi" },
    "sk-yourapikeyhere");

Console.WriteLine($"Processed: {result.processed}, Failed: {result.failed}");
```

Remarks

This method delegates to the generic DocumentNoteBatchProcessor<TProcessor> for all batch processing operations while maintaining backward compatibility with existing video-specific API.

Events

ProcessingProgressChanged

Event triggered when processing progress changes.

```
public event EventHandler<DocumentProcessingProgressEventArgs>? ProcessingProgressChanged
```

Event Type

EventHandler <DocumentProcessingProgressEventArgs>

QueueChanged

Event triggered when the processing queue changes.

```
public event EventHandler<QueueChangedEventArgs>? QueueChanged
```

Event Type

EventHandler <QueueChangedEventArgs>

Class VideoNoteProcessor

Namespace: NotebookAutomation.Core.Tools.VideoProcessing

Assembly: NotebookAutomation.Core.dll

Represents a processor for handling video files to extract metadata, generate AI-powered summaries, and create markdown notes for knowledge management systems.

```
public class VideoNoteProcessor : DocumentNoteProcessorBase
```

Inheritance

object ← DocumentNoteProcessorBase ← VideoNoteProcessor

Inherited Members

DocumentNoteProcessorBase.Logger , DocumentNoteProcessorBase.Summarizer , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

The `VideoNoteProcessor` class provides functionality for processing video files, including:

- Extracting metadata and transcripts from video files
- Generating AI-powered summaries using external services
- Creating markdown notes with YAML frontmatter for knowledge management
- Integrating with OneDrive for share link generation
- Supporting hierarchical metadata detection based on file paths

This class is designed to work with various video formats and supports extensibility for additional metadata management and logging services.

Constructors

VideoNoteProcessor(ILogger<VideoNoteProcessor>, AISummarizer, IYamlHelper, MetadataHierarchyDetector, IOneDriveService?, AppConfig?, ILoggingService?)

Initializes a new instance of the VideoNoteProcessor class.

```
public VideoNoteProcessor(ILogger<VideoNoteProcessor> logger, AISummarizer aiSummarizer,
IYamlHelper yamlHelper, MetadataHierarchyDetector hierarchyDetector, IOneDriveService?
oneDriveService = null, AppConfig? appConfig = null, ILoggingService? loggingService = null)
```

Parameters

`logger` ILogger<VideoNoteProcessor>

The logger instance for logging diagnostic and error information.

`aiSummarizer` AISummarizer

The AI summarizer service for generating summaries.

`yamlHelper` IYamlHelper

The YAML helper for processing YAML frontmatter in markdown documents.

`hierarchyDetector` MetadataHierarchyDetector

The metadata hierarchy detector for extracting metadata from directory structure.

`oneDriveService` IOneDriveService

Optional service for generating OneDrive share links.

`appConfig` AppConfig

Optional application configuration for metadata management.

`loggingService` ILoggingService

Optional logging service for creating typed loggers.

Remarks

This constructor initializes the video note processor with optional services for metadata management and hierarchical detection.

Methods

ExtractMetadataAsync(string)

Extracts comprehensive metadata from a video file.

```
public Task<Dictionary<string, object>> ExtractMetadataAsync(string videoPath)
```

Parameters

`videoPath` string

The path to the video file.

Returns

Task<Dictionary<string, object>>

A task that represents the asynchronous operation. The task result contains a dictionary with extracted video metadata, including properties such as title, type, status, author, file size, resolution, codec, duration, and upload date.

Examples

```
var processor = new VideoNoteProcessor(logger, aiSummarizer);
var metadata = await processor.ExtractMetadataAsync("path/to/video.mp4");
Console.WriteLine($"Video title: {metadata["title"]}");
```

Remarks

This method uses Xabe.FFmpeg to extract detailed video metadata, such as resolution, codec, and duration. It also retrieves basic file properties like size and creation date.

If metadata extraction fails, the method logs warnings and provides simulated values for certain fields to ensure the operation does not fail completely.

ExtractTextAndMetadataAsync(string)

Extracts the main text/content and metadata from the document.

```
public override Task<(string Text, Dictionary<string, object> Metadata)>
ExtractTextAndMetadataAsync(string filePath)
```

Parameters

`filePath` string

Path to the document file.

Returns

Task<(string Text, Dictionary<string, object> Metadata)>

Tuple of extracted text/content and metadata dictionary.

GenerateAiSummaryAsync(string?, Dictionary<string, string>?, string?)

```
public override Task<string> GenerateAiSummaryAsync(string? text, Dictionary<string,
string>? variables = null, string? promptFileName = null)
```

Parameters

`text` string

`variables` Dictionary<string, string>

`promptFileName` string

Returns

Task<string>

GenerateMarkdownNote(string, Dictionary<string, object>?, string, bool, bool)

Generates a markdown note from video metadata and summary.

```
public override string GenerateMarkdownNote(string bodyText, Dictionary<string, object>?
metadata = null, string noteType = "Document Note", bool suppressBody = false, bool
includeNoteTypeTitle = false)
```

Parameters

`bodyText` string

The main content of the note, typically an AI-generated summary.

`metadata` Dictionary<string, object>

Dictionary of metadata to include in the note's frontmatter.

`noteType` string

The type of note being generated, such as "Document Note" or "Video Note".

`suppressBody` bool

If true, suppresses the body content of the note.

`includeNoteTypeTitle` bool

If true, includes the note type as the title in the markdown note.

Returns

string

A complete markdown note with frontmatter and content.

Examples

```
var processor = new VideoNoteProcessor(logger, aiSummarizer);
var markdownNote = processor.GenerateMarkdownNote("Summary text", metadata, "Video Note");
Console.WriteLine(markdownNote);
```

Remarks

This method creates a properly formatted markdown note that combines the AI-generated summary with the video's metadata. The resulting note includes:

- YAML frontmatter with all extracted metadata
- The summary as the main content body
- A consistent structure with appropriate headers

///.

The note is generated using the MarkdownNoteBuilder utility and follows the structure expected by Obsidian or similar markdown-based knowledge management systems.

GenerateVideoNoteAsync(string, string?, string?, bool, int?, string?, bool)

Generates a video note, using transcript if available, otherwise metadata summary.

```
public Task<string> GenerateVideoNoteAsync(string videoPath, string? openAiApiKey, string?
promptFileName = null, bool noSummary = false, int? timeoutSeconds = null, string?
resourcesRoot = null, bool noShareLinks = false)
```

Parameters

`videoPath` string

Path to the video file to process.

`openAiApiKey` string

OpenAI API key for generating summaries.

`promptFileName` string

Optional prompt file name to use for AI summarization.

`noSummary` bool

If true, disables OpenAI summary generation.

`timeoutSeconds` int?

Optional API request timeout in seconds.

`resourcesRoot` string

Optional override for OneDrive fullpath root directory.

`noShareLinks` bool

If true, skips OneDrive share link creation.

Returns

Task<string>

A complete markdown note as a string.

Examples

```
var processor = new VideoNoteProcessor(logger);
string markdownNote = await processor.GenerateVideoNoteAsync(
    "path/to/lecture.mp4",
    "sk-yourapikeyhere",
    "video_summary_prompt.md");

// Save the generated note
File.WriteAllText("lecture_notes.md", markdownNote);
```

```
var processor = new VideoNoteProcessor(logger, aiSummarizer);
string markdownNote = await processor.GenerateVideoNoteAsync(
    "path/to/lecture.mp4",
    "sk-yourapikeyhere",
    "video_summary_prompt.md");

// Save the generated note
File.WriteAllText("lecture_notes.md", markdownNote);
```

Remarks

This method coordinates the entire video note generation process by:

1. Extracting comprehensive metadata from the video file
2. Attempting to load an associated transcript file
3. Generating an AI summary from either the transcript or metadata
4. Combining everything into a well-formatted markdown note

The method prefers using a transcript when available for more accurate summaries, but will fall back to using just the metadata information when no transcript exists.

This is the primary entry point for generating a complete video note from a single video file.

TryLoadTranscript(string)

Attempts to load a transcript file for the given video.

```
public string? TryLoadTranscript(string videoPath)
```

Parameters

`videoPath` string

The path to the video file.

Returns

string

The transcript text if found; otherwise, null.

Examples

```
var processor = new VideoNoteProcessor(logger, aiSummarizer);
string? transcript = processor.TryLoadTranscript("path/to/video.mp4");
if (transcript != null)
{
    Console.WriteLine("Transcript loaded successfully.");
}
else
{
    Console.WriteLine("No transcript found.");
}
```

Remarks

This method searches for transcript files that match the video filename using a prioritized search strategy. It looks for transcript files in multiple locations and formats, including language-specific and generic transcripts.

The search follows this priority order:

1. Language-specific transcripts in the same directory (e.g., video.en.txt, video.zh-cn.txt).
2. Generic transcript in the same directory (video.txt, video.md).
3. Language-specific transcripts in the "Transcripts" subdirectory.
4. Generic transcript in the "Transcripts" subdirectory.

The method also handles name normalization by checking alternative spellings with hyphens replaced by underscores and vice versa.

Namespace NotebookAutomation.Core.Utils

Classes

BaseBlockGenerator
: Provides functionality to generate Obsidian Base blocks from a YAML template with dynamic parameters.

CourseStructureExtractor
: Utility class for extracting course structure information (modules, lessons) from file paths.

FileSizeFormatter
: Provides utility methods for formatting file sizes into human-readable strings.

FriendlyTitleHelper
: Provides utility methods for generating friendly titles from file names or paths.

LoggerExtensions
: Provides extension methods for ILogger and ILogger<TCategoryName> to simplify and standardize logging of messages with file path formatting, exception handling, and flexible message templates.

MarkdownNoteBuilder
: Provides a reusable, strongly-typed builder for generating markdown notes with YAML frontmatter.

MetadataHierarchyDetector
: Detects and infers hierarchical metadata (program, course, class, module) from file paths in a notebook vault.

MetadataTemplateManager
: Manages loading, parsing, and application of metadata templates from the `metadata.yaml` file.

NoQuotesForBannerVisitor
: Custom YamlDotNet visitor to avoid quotes for the banner field.

PathFormatter
: Utility class for formatting file paths in log messages.

PathUtils
: Provides utility methods for working with file and directory paths.

YamlHelper
: Helper class for working with YAML content in markdown files.

Interfaces

IYamlHelper

Interface for YAML helper functionality used to process YAML frontmatter in markdown files.

Class BaseBlockGenerator

Namespace: NotebookAutomation.Core.Utils

Assembly: NotebookAutomation.Core.dll

Provides functionality to generate Obsidian Base blocks from a YAML template with dynamic parameters.

```
public static class BaseBlockGenerator
```

Inheritance

object ← BaseBlockGenerator

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

GenerateBaseBlock(string, string, string, string, string)

Loads the base block template from the specified path and fills in the placeholders.

```
public static string GenerateBaseBlock(string templatePath, string course, string className,
string module, string type)
```

Parameters

`templatePath` string

Path to the YAML template file.

`course` string

Course name to inject.

`className` string

Class name to inject.

`module` string

Module name to inject.

`type` string

Type to inject.

Returns

string

The filled-in base block as a string.

Exceptions

FileNotFoundException

Thrown if the template file does not exist.

Class CourseStructureExtractor

Namespace: NotebookAutomation.Core.Utils

Assembly: NotebookAutomation.Core.dll

Utility class for extracting course structure information (modules, lessons) from file paths.

```
public class CourseStructureExtractor
```

Inheritance

object ← CourseStructureExtractor

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Examples

```
var logger = serviceProvider.GetService<ILogger<CourseStructureExtractor>>();
var extractor = new CourseStructureExtractor(logger);
var metadata = new Dictionary<string, object>();

extractor.ExtractModuleAndLesson("/courses/01_module-intro/02_lesson-
basics/content.md", metadata);
// metadata now contains: { "module": "Module Intro", "lesson": "Lesson Basics" }
```

Remarks

This class provides functionality to identify module and lesson information from file paths based on directory naming conventions. It uses multiple extraction strategies to handle various course organization patterns commonly found in educational content.

The extractor supports detection of module and lesson information by analyzing:

- Directory names containing "module" or "lesson" keywords
- Numbered directory prefixes (e.g., "01_", "02-")
- Filename patterns with embedded module/lesson information
- Hierarchical structures with parent-child relationships

The class provides clean formatting by removing numbering prefixes and converting names to title case for consistent metadata output.

Constructors

CourseStructureExtractor(ILogger<CourseStructureExtractor>)

Utility class for extracting course structure information (modules, lessons) from file paths.

```
public CourseStructureExtractor(ILogger<CourseStructureExtractor> logger)
```

Parameters

`logger` ILogger<CourseStructureExtractor>

Logger for diagnostic and warning messages during extraction operations.

Examples

```
var logger = serviceProvider.GetService<ILogger<CourseStructureExtractor>>();
var extractor = new CourseStructureExtractor(logger);
var metadata = new Dictionary<string, object>();

extractor.ExtractModuleAndLesson("/courses/01_module-intro/02_lesson-
basics/content.md", metadata);
// metadata now contains: { "module": "Module Intro", "lesson": "Lesson Basics" }
```

Remarks

This class provides functionality to identify module and lesson information from file paths based on directory naming conventions. It uses multiple extraction strategies to handle various course organization patterns commonly found in educational content.

The extractor supports detection of module and lesson information by analyzing:

- Directory names containing "module" or "lesson" keywords
- Numbered directory prefixes (e.g., "01_", "02-")
- Filename patterns with embedded module/lesson information
- Hierarchical structures with parent-child relationships

The class provides clean formatting by removing numbering prefixes and converting names to title case for consistent metadata output.

Exceptions

ArgumentNullException

Thrown when `logger` is null.

Methods

CleanModuleOrLessonName(string)

Cleans up module or lesson folder names by removing numbering prefixes and formatting for consistent display.

```
public static string CleanModuleOrLessonName(string folderName)
```

Parameters

`folderName` string

The raw folder name to clean and format.

Returns

string

A cleaned and formatted folder name in title case with proper spacing.

Examples

```
string result1 = CleanModuleOrLessonName("01_module-introduction");     // Returns
"Module Introduction"
string result2 = CleanModuleOrLessonName("02-lesson_basics");          // Returns
"Lesson Basics"
string result3 = CleanModuleOrLessonName("sessionPlanningDetails");    // Returns "Session
Planning Details"
string result4 = CleanModuleOrLessonName("Week-1-Overview");           // Returns
"Week Overview"
```

Remarks

This method performs several transformations to create user-friendly module and lesson names:

1. Removes numeric prefixes (e.g., "01_", "02-", "03.")

2. Converts camelCase to spaced words
3. Replaces hyphens and underscores with spaces
4. Normalizes multiple spaces to single spaces
5. Converts to title case using current culture

The method is culture-aware and uses the current culture's title case rules for proper formatting.

Exceptions

ArgumentNullException

Thrown when `folderName` is null.

ExtractModuleAndLesson(string, Dictionary<string, object>)

Extracts module and lesson information from a file path and adds it to the provided metadata dictionary.

```
public void ExtractModuleAndLesson(string filePath, Dictionary<string, object> metadata)
```

Parameters

`filePath` string

The full path to the file from which to extract course structure information.

`metadata` Dictionary<string, object>

The metadata dictionary to update with module/lesson information. Keys "module" and "lesson" will be added if corresponding information is found.

Examples

```
var metadata = new Dictionary<string, object>();
extractor.ExtractModuleAndLesson(@"C:\courses\01_intro-module\03_lesson-
basics\notes.md", metadata);

// Result: metadata contains:
// { "module": "Intro Module", "lesson": "Lesson Basics" }
```

Remarks

This method uses a multi-stage extraction process:

1. First attempts to extract from the filename itself
2. Then looks for explicit "module" and "lesson" keywords in directory names
3. Finally attempts to identify patterns from numbered directory structures

The method logs debug information during extraction to help with troubleshooting course structure issues. Warning messages are logged if the file path is empty or if extraction fails due to exceptions.

Exceptions

ArgumentException

Logged as warning if extraction fails due to invalid path structure.

Class FileSizeFormatter

Namespace: NotebookAutomation.Core.Utils

Assembly: NotebookAutomation.Core.dll

Provides utility methods for formatting file sizes into human-readable strings.

```
public static class FileSizeFormatter
```

Inheritance

object ← FileSizeFormatter

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

This class is intended for converting byte counts to strings such as "1.23 MB" or "512.00 B". It is stateless and thread-safe.

Methods

FormatFileSizeToString(long)

Converts a file size in bytes to a human-readable string format.

```
public static string FormatFileSizeToString(long bytes)
```

Parameters

`bytes` long

The file size in bytes.

Returns

string

A string representing the file size in a human-readable format, such as KB, MB, GB, etc.

Examples

```
var readableSize = FileSizeFormatter.FormatFileSizeToString(1048576); // Returns "1 MB"
```

Remarks

The method uses appropriate precision based on the size of the file:

- Two decimal places for sizes less than 10.
- One decimal place for sizes less than 100.
- Rounded values for sizes greater than or equal to 100.

Class FriendlyTitleHelper

Namespace: NotebookAutomation.Core.Utils

Assembly: NotebookAutomation.Core.dll

Provides utility methods for generating friendly titles from file names or paths.

```
public static class FriendlyTitleHelper
```

Inheritance

object ← FriendlyTitleHelper

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

The `FriendlyTitleHelper` class offers static methods to convert file names or paths into human-friendly titles by removing numbers, structural words, and formatting the result in title case. It also handles special cases such as acronyms and Roman numerals.

Methods

GetFriendlyTitleFromFileName(string)

Returns a friendly title from a file name (removes numbers, underscores, extension, trims, and capitalizes).

```
public static string GetFriendlyTitleFromFileName(string fileName)
```

Parameters

`fileName` string

The file name or path to process.

Returns

string

A cleaned, human-friendly title string.

Interface IYamlHelper

Namespace: NotebookAutomation.Core.Utils

Assembly: NotebookAutomation.Core.dll

Interface for YAML helper functionality used to process YAML frontmatter in markdown files.

```
public interface IYamlHelper
```

Methods

DiagnoseYamlFrontmatter(string)

Diagnostics for YAML frontmatter in markdown content.

```
(bool Success, string Message, Dictionary<string, object>? Data)
DiagnoseYamlFrontmatter(string markdown)
```

Parameters

`markdown` string

The markdown content to diagnose.

Returns

(bool Success, string Message, Dictionary<string, object> Data)

A tuple with success status, message, and parsed data if available.

ExtractFrontmatter(string)

Extracts the YAML frontmatter from markdown content.

```
string? ExtractFrontmatter(string markdown)
```

Parameters

`markdown` string

The markdown content to parse.

Returns

string

The YAML frontmatter as a string, or null if not found.

ParseYaml(string)

Parses YAML frontmatter to a dynamic object.

```
object? ParseYaml(string yaml)
```

Parameters

`yaml` string

The YAML content to parse.

Returns

object

The parsed object, or null if parsing failed.

ParseYamlToDictionary(string)

Parses YAML frontmatter to a dictionary.

```
Dictionary<string, object> ParseYamlToDictionary(string yaml)
```

Parameters

`yaml` string

The YAML content to parse.

Returns

Dictionary<string, object>

The parsed dictionary, or an empty dictionary if parsing failed.

RemoveFrontmatter(string)

Removes the YAML frontmatter from markdown content.

```
string RemoveFrontmatter(string markdown)
```

Parameters

`markdown` string

The markdown content.

Returns

string

The markdown content without frontmatter.

ReplaceFrontmatter(string, Dictionary<string, object>)

Replaces the frontmatter in a markdown document with new frontmatter.

```
string ReplaceFrontmatter(string markdown, Dictionary<string, object> newFrontmatter)
```

Parameters

`markdown` string

The markdown content.

`newFrontmatter` Dictionary<string, object>

The new frontmatter as a dictionary.

Returns

string

The updated markdown content.

SerializeToYaml(Dictionary<string, object>)

Serializes a dictionary to YAML format.

```
string SerializeToYaml(Dictionary<string, object> data)
```

Parameters

`data` Dictionary <string, object>

The dictionary to serialize.

Returns

string

The serialized YAML string.

UpdateFrontmatter(string, Dictionary<string, object>)

Updates the YAML frontmatter in markdown content.

```
string UpdateFrontmatter(string markdown, Dictionary<string, object> newFrontmatter)
```

Parameters

`markdown` string

The markdown content.

`newFrontmatter` Dictionary <string, object>

The new frontmatter as a dictionary.

Returns

string

The updated markdown content.

Class LoggerExtensions

Namespace: NotebookAutomation.Core.Utils

Assembly: NotebookAutomation.Core.dll

Provides extension methods for ILogger and ILogger<TCategoryName> to simplify and standardize logging of messages with file path formatting, exception handling, and flexible message templates.

```
public static class LoggerExtensions
```

Inheritance

object ← LoggerExtensions

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

These extensions enable consistent logging patterns for file-related operations, supporting both full and shortened file paths depending on log level, and provide overloads for common log levels (Information, Debug, Warning, Error).

Use `LogWithFormattedPath` and its overloads to automatically format file paths in log messages, and use `LogFormatted` for general-purpose message formatting. All methods support structured logging and exception details.

```
logger.LogInformationWithPath("Processed file: {FilePath}", filePath);
logger.LogErrorWithPath(ex, "Failed to process: {FilePath}", filePath);
logger.LogDebugFormatted("Processing {Count} items", count);
```

Methods

LogDebugFormatted(ILogger, string, params object[])

Logs a debug-level message with general string formatting support.

```
public static void LogDebugFormatted(this ILogger logger, string message, params
```

```
object[] args)
```

Parameters

`logger` ILogger

The logger instance to use for logging.

`message` string

A message template with any number of placeholders.

`args` object[]

Arguments to be formatted into the message template placeholders.

Examples

```
logger.LogDebugFormatted("Debug info: {Detail}", detail);
```

LogDebugWithPath(ILogger, Exception?, string, string, params object[])

Logs a debug message with a formatted file path and exception.

```
public static void LogDebugWithPath(this ILogger logger, Exception? exception, string
message, string filePath, params object[] args)
```

Parameters

`logger` ILogger

The logger.

`exception` Exception

The exception to log.

`message` string

The message template with {FilePath} placeholder.

`filePath` string

The file path to format and log.

`args` object[]

Additional arguments for the message template.

LogDebugWithPath(ILogger, string, string, params object[])

```
public static void LogDebugWithPath(this ILogger logger, string message, string filePath,
params object[] args)
```

Parameters

`logger` ILogger

`message` string

`filePath` string

`args` object[]

LogErrorFormatted(ILogger, Exception, string, params object[])

Logs an error-level message with general string formatting support and an exception.

```
public static void LogErrorFormatted(this ILogger logger, Exception exception, string
message, params object[] args)
```

Parameters

`logger` ILogger

The logger instance to use for logging.

`exception` Exception

The exception to include in the log entry.

`message` string

A message template with any number of placeholders.

`args` object[]

Arguments to be formatted into the message template placeholders.

Examples

```
logger.LogErrorFormatted(ex, "Failed: {Detail}", detail);
```

LogErrorFormatted(ILogger, string, params object[])

Logs an error-level message with general string formatting support.

```
public static void LogErrorFormatted(this ILogger logger, string message, params
object[] args)
```

Parameters

`logger` ILogger

The logger instance to use for logging.

`message` string

A message template with any number of placeholders.

`args` object[]

Arguments to be formatted into the message template placeholders.

Examples

```
logger.LogErrorFormatted("Error: {Detail}", detail);
```

LogErrorWithPath(ILogger, Exception, string, string, params object[])

Logs an error message with a formatted file path.

```
public static void LogErrorWithPath(this ILogger logger, Exception exception, string
message, string filePath, params object[] args)
```

Parameters

`logger` ILogger

The logger.

`exception` Exception

The exception to log.

`message` string

The message template with {FilePath} placeholder.

`filePath` string

The file path to format and log.

`args` object[]

Additional arguments for the message template.

LogErrorWithPath(ILogger, string, string, params object[])

Logs an error message with a formatted file path.

```
public static void LogErrorWithPath(this ILogger logger, string message, string filePath,
params object[] args)
```

Parameters

`logger` ILogger

The logger.

`message` string

The message template with {FilePath} placeholder.

`filePath` string

The file path to format and log.

`args` object[]

Additional arguments for the message template.

LogErrorWithPath<T>(ILogger<T>, Exception, string, string, params object[])

```
public static void LogErrorWithPath<T>(this ILogger<T> logger, Exception exception, string message, string filePath, params object[] args)
```

Parameters

`logger` ILogger<T>

`exception` Exception

`message` string

`filePath` string

`args` object[]

Type Parameters

`T`

LogFormatted(ILogger, LogLevel, EventId, Exception?, string, params object[])

Logs a message with general string formatting support, without any special file path formatting.

```
public static void LogFormatted(this ILogger logger, LogLevel logLevel, EventId eventId, Exception? exception, string message, params object[] args)
```

Parameters

`logger` ILogger

The logger instance to use for logging.

`logLevel` LogLevel

The severity level of the log message.

`eventId` EventId

The event ID for structured logging (optional).

`exception` Exception

An optional exception to include in the log entry.

`message` string

A message template with any number of placeholders.

`args` object[]

Arguments to be formatted into the message template placeholders.

Examples

```
logger.LogFormatted(LogLevel.Warning, 0, null, "Warning: {Detail}", detail);
```

LogInformationFormatted(ILogger, string, params object[])

Logs an information-level message with general string formatting support.

```
public static void LogInformationFormatted(this ILogger logger, string message, params
object[] args)
```

Parameters

`logger` ILogger

The logger instance to use for logging.

`message` string

A message template with any number of placeholders.

`args` object[]

Arguments to be formatted into the message template placeholders.

Examples

```
logger.LogInformationFormatted("Processed {Count} items", count);
```

LogInformationWithPath(ILogger, string, string, params object[])

Logs an information-level message with a formatted file path.

```
public static void LogInformationWithPath(this ILogger logger, string message, string
filePath, params object[] args)
```

Parameters

`logger` ILogger

The logger instance to use for logging.

`message` string

A message template that should include a `{FilePath}` placeholder for the formatted path.

`filePath` string

The file path to be formatted and injected into the log message.

`args` object[]

Additional arguments to be formatted into the message template.

Examples

```
logger.LogInformationWithPath("Imported file: {FilePath}", filePath);
```

LogWarningFormatted(ILogger, string, params object[])

Logs a warning-level message with general string formatting support.

```
public static void LogWarningFormatted(this ILogger logger, string message, params
object[] args)
```

Parameters

`logger` ILogger

The logger instance to use for logging.

`message` string

A message template with any number of placeholders.

`args` object[]

Arguments to be formatted into the message template placeholders.

Examples

```
logger.LogWarningFormatted("Warning: {Detail}", detail);
```

LogWarningWithPath(ILogger, Exception, string, string, params object[])

Logs a warning message with a formatted file path and exception.

```
public static void LogWarningWithPath(this ILogger logger, Exception exception, string
message, string filePath, params object[] args)
```

Parameters

`logger` ILogger

The logger.

`exception` Exception

The exception to log.

`message` string

The message template with {FilePath} placeholder.

`filePath` string

The file path to format and log.

`args` object[]

Additional arguments for the message template.

LogWarningWithPath(ILogger, string, string, params object[])

Logs a warning message with a formatted file path.

```
public static void LogWarningWithPath(this ILogger logger, string message, string filePath,
params object[] args)
```

Parameters

`logger` ILogger

The logger.

`message` string

The message template with {FilePath} placeholder.

`filePath` string

The file path to format and log.

`args` object[]

Additional arguments for the message template.

LogWithFormattedPath(ILogger, LogLevel, EventId, Exception?, string, string, params object[])

Logs a message with a formatted file path, automatically shortening or expanding the path based on log level.

```
public static void LogWithFormattedPath(this ILogger logger, LogLevel logLevel, EventId
eventId, Exception? exception, string message, string filePath, params object[] args)
```

Parameters

`logger` ILogger

The logger instance to use for logging.

`logLevel` LogLevel

The severity level of the log message.

`eventId` EventId

The event ID for structured logging (optional).

`exception` Exception

An optional exception to include in the log entry.

`message` string

A message template that should include a `{FilePath}` placeholder for the formatted path.

`filePath` string

The file path to be formatted and injected into the log message.

`args` object[]

Additional arguments to be formatted into the message template.

Examples

```
logger.LogWithFormattedPath(LogLevel.Information, 0, null, "Processed file:
{FilePath}", filePath);
```

Remarks

For Debug or Trace, the full file path is used. For other levels, a shortened path is used.

Class MarkdownNoteBuilder

Namespace: NotebookAutomation.Core.Utils

Assembly: NotebookAutomation.Core.dll

Provides a reusable, strongly-typed builder for generating markdown notes with YAML frontmatter.

```
public class MarkdownNoteBuilder
```

Inheritance

object ← MarkdownNoteBuilder

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

This class simplifies the creation of markdown notes that require YAML frontmatter for metadata, supporting both frontmatter-only and full note (frontmatter + body) scenarios. It uses YamlHelper for serialization.

```
var builder = new MarkdownNoteBuilder();
var frontmatter = new Dictionary<string, object> { ["title"] = "Sample" };
string note = builder.BuildNote(frontmatter, "# Heading\nContent");
```

Constructors

MarkdownNoteBuilder(ILogger?)

Provides a reusable, strongly-typed builder for generating markdown notes with YAML frontmatter.

```
public MarkdownNoteBuilder(ILogger? logger = null)
```

Parameters

`logger` ILogger

Remarks

This class simplifies the creation of markdown notes that require YAML frontmatter for metadata, supporting both frontmatter-only and full note (frontmatter + body) scenarios. It uses YamlHelper for serialization.

```
var builder = new MarkdownNoteBuilder();
var frontmatter = new Dictionary<string, object> { ["title"] = "Sample" };
string note = builder.BuildNote(frontmatter, "# Heading\nContent");
```

Methods

BuildNote(Dictionary<string, object>, string)

Builds a markdown note with both YAML frontmatter and a markdown content body.

```
public string BuildNote(Dictionary<string, object> frontmatter, string body)
```

Parameters

`frontmatter` Dictionary<string, object>

A dictionary of frontmatter keys and values to serialize as YAML.

`body` string

The markdown content body to append after the frontmatter.

Returns

string

A markdown string containing the YAML frontmatter block followed by the content body.

Examples

///.

```
var frontmatter = new Dictionary<string, object> { ["title"] = "Sample" };
string note = builder.BuildNote(frontmatter, "# Heading\nContent");
```

Remarks

The frontmatter is always placed at the top of the note, followed by the markdown body.

CreateMarkdownWithFrontmatter(Dictionary<string, object>)

Builds a markdown note containing only YAML frontmatter (no content body).

```
public string CreateMarkdownWithFrontmatter(Dictionary<string, object> frontmatter)
```

Parameters

`frontmatter` Dictionary<string, object>

A dictionary of frontmatter keys and values to serialize as YAML.

Returns

string

A markdown string containing only the YAML frontmatter block.

Examples

```
var frontmatter = new Dictionary<string, object> { ["title"] = "Sample" };
string note = builder.CreateMarkdownWithFrontmatter(frontmatter);
```

Remarks

The resulting string will have a YAML frontmatter block delimited by `---` and two trailing newlines.

Class MetadataHierarchyDetector

Namespace: NotebookAutomation.Core.Utils

Assembly: NotebookAutomation.Core.dll

Detects and infers hierarchical metadata (program, course, class, module) from file paths in a notebook vault.

```
public class MetadataHierarchyDetector
```

Inheritance

object ← MetadataHierarchyDetector

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

Implements path-based hierarchy detection based on the directory structure. Determines the appropriate program, course, class, and module metadata based on a file's location, following the conventions used in the notebook vault.

///. **Expected Directory Structure:**

```
Vault Root (main-index) - NO hierarchy metadata
└── Program Folders (program-index) - program only
    └── Course Folders (course-index) - program + course
        └── Class Folders (class-index) - program + course + class
            ├── Case Study Folders (case-study-index) - program + course + class + module
            └── Module Folders (module-index) - program + course + class + module
                ├── Live Session Folder (live-session-index) - program + course + class
+ module
                └── Lesson Folders (lesson-index) - program + course + class + module
                    └── Content Files (readings, videos, transcripts, etc.)
```

Features: - Configurable vault root path (from config or override) - Support for explicit program overrides via parameter - Dynamic hierarchy detection based on folder structure - Dynamic fallback to folder names when index files aren't available - Robust path traversal for hierarchy detection.

```
var detector = new MetadataHierarchyDetector(logger, appConfig);
var info = detector.FindHierarchyInfo(@"C:\\notebook-
vault\\MBA\\Course1\\ClassA\\Lesson1\\file.md");
// info["program"] == "MBA", info["course"] == "Course1", info["class"] == "ClassA"
```

Constructors

MetadataHierarchyDetector(ILogger<MetadataHierarchyDetector>, AppConfig, string?, bool, string?)

Detects and infers hierarchical metadata (program, course, class, module) from file paths in a notebook vault.

```
public MetadataHierarchyDetector(ILogger<MetadataHierarchyDetector> logger, AppConfig
appConfig, string? programOverride = null, bool verbose = false, string? vaultRootOverride
= null)
```

Parameters

`logger` ILogger<MetadataHierarchyDetector>

`appConfig` AppConfig

`programOverride` string

`verbose` bool

`vaultRootOverride` string

Remarks

Implements path-based hierarchy detection based on the directory structure. Determines the appropriate program, course, class, and module metadata based on a file's location, following the conventions used in the notebook vault.

///. **Expected Directory Structure:**

```
Vault Root (main-index) - NO hierarchy metadata
└── Program Folders (program-index) - program only
    └── Course Folders (course-index) - program + course
```

```
        └── Class Folders (class-index) - program + course + class
            ├── Case Study Folders (case-study-index) - program + course + class + module
            └── Module Folders (module-index) - program + course + class + module
                ├── Live Session Folder (live-session-index) - program + course + class
+ module
                └── Lesson Folders (lesson-index) - program + course + class + module
                    └── Content Files (readings, videos, transcripts, etc.)
```

Features: - Configurable vault root path (from config or override) - Support for explicit program overrides via parameter - Dynamic hierarchy detection based on folder structure - Dynamic fallback to folder names when index files aren't available - Robust path traversal for hierarchy detection.

```
var detector = new MetadataHierarchyDetector(logger, appConfig);
var info = detector.FindHierarchyInfo(@"C:\\notebook-
vault\\MBA\\Course1\\ClassA\\Lesson1\\file.md");
// info["program"] == "MBA", info["course"] == "Course1", info["class"] == "ClassA"
```

Properties

VaultRoot

Gets the vault root path being used by this detector.

```
public string VaultRoot { get; }
```

Property Value

string

Methods

FindHierarchyInfo(string)

Finds program, course, class, and module information by analyzing the file path structure relative to the vault root.

```
public Dictionary<string, string> FindHierarchyInfo(string filePath)
```

Parameters

`filePath` string

The path to the file or directory to analyze.

Returns

Dictionary<string, string>

A dictionary with keys `program`, `course`, `class`, and possibly `module` containing the detected hierarchy information.

Examples

```
var info = detector.FindHierarchyInfo(@"C:\\notebook-
vault\\MBA\\Finance\\Accounting\\Week1\\file.md");
// info["program"] == "MBA", info["course"] == "Finance", info["class"] == "Accounting",
info["module"] == "Week1"
```

Remarks

This method uses purely path-based analysis to determine hierarchy levels, with no file system access needed. It assumes a standard folder structure where: - The first folder level below vault root is the program (e.g., MBA) - The second folder level is the course (e.g., Finance) - The third folder level is the class (e.g., Accounting) - The fourth folder level is the module (e.g., Week1).

Priority is given to explicit program overrides if provided in the constructor.

UpdateMetadataWithHierarchy(Dictionary<string, object>, Dictionary<string, string>, string?)

Updates a metadata dictionary with program, course, class, and module information appropriate for a specific hierarchy level.

```
public static Dictionary<string, object> UpdateMetadataWithHierarchy(Dictionary<string,
object> metadata, Dictionary<string, string> hierarchyInfo, string? templateType = null)
```

Parameters

`metadata` Dictionary<string, object>

The existing metadata dictionary to update (will be mutated).

`hierarchyInfo` Dictionary<string, string>

The hierarchy information to apply (should contain keys for hierarchical levels).

`templateType` string

Optional template type to determine which hierarchy levels to include. Defaults to including all detected levels.

Returns

Dictionary<string, object>

The updated metadata dictionary with hierarchy fields set if missing or empty.

Examples

```
// For a program-level index
var updated = MetadataHierarchyDetector.UpdateMetadataWithHierarchy(metadata, info,
"program-index");
// Only includes program metadata

// For a class-level index
var updated = MetadataHierarchyDetector.UpdateMetadataWithHierarchy(metadata, info,
"class-index");
// Includes program, course, and class metadata
```

Remarks

Only updates fields that are missing or empty in the original metadata. The method will look for the following keys in the hierarchyInfo dictionary: - program: The program name (top level of the hierarchy) - included for all index types - course: The course name (second level of the hierarchy) - included for course, class and module index types - class: The class name (third level of the hierarchy) - included for class and module index types - module: The module name (fourth level of the hierarchy) - included only for module index types.

Each level only includes metadata appropriate for its level in the hierarchy. For example, a program-level index will only include program metadata, while a class-level index will include program, course, and class metadata.

Class MetadataTemplateManager

Namespace: NotebookAutomation.Core.Utils

Assembly: NotebookAutomation.Core.dll

Manages loading, parsing, and application of metadata templates from the `metadata.yaml` file.

```
public class MetadataTemplateManager
```

Inheritance

object ← MetadataTemplateManager

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

Responsible for loading and parsing the `metadata.yaml` file, which contains template definitions for various document types used in the notebook vault. Provides methods to retrieve, fill, and apply templates to document metadata.

```
var manager = new MetadataTemplateManager(logger, appConfig, yamlHelper);
var template = manager.GetTemplate("video-reference");
var filled = manager.GetFilledTemplate("video-reference", new Dictionary<string, string> {
["title"] = "Sample" });
```

Constructors

MetadataTemplateManager(ILogger, AppConfig, IYamlHelper)

Initializes a new instance of the MetadataTemplateManager class.

```
public MetadataTemplateManager(ILogger logger, AppConfig appConfig, IYamlHelper yamlHelper)
```

Parameters

`logger` ILogger

The logger to use for diagnostic and error reporting.

`appConfig` AppConfig

The application configuration.

`yamlHelper` IYamlHelper

The YAML helper service for parsing metadata.

Methods

EnhanceMetadataWithTemplate(Dictionary<string, object>, string)

Enhances document metadata with appropriate template fields based on the note type.

```
public Dictionary<string, object> EnhanceMetadataWithTemplate(Dictionary<string, object>
metadata, string noteType)
```

Parameters

`metadata` Dictionary<string, object>

The current document metadata to enhance.

`noteType` string

The type of document being processed (e.g., `Video Note`, `PDF Note`).

Returns

Dictionary<string, object>

Enhanced metadata with template fields added and defaults filled in.

Examples

```
var enhanced = manager.EnhanceMetadataWithTemplate(metadata, "Video Note");
```

Remarks

Fields from the template are added if missing or empty in the original metadata. Special handling is applied for certain fields.

GetFilledTemplate(string, Dictionary<string, string>)

Gets a template by type and fills in provided values for placeholders.

```
public Dictionary<string, object>? GetFilledTemplate(string templateType, Dictionary<string, string> values)
```

Parameters

`templateType` string

The type of template to retrieve (e.g., `video-reference`).

`values` Dictionary<string, string>

A dictionary of values to fill in for template placeholders.

Returns

Dictionary<string, object>

A filled template dictionary, or `null` if the template is not found.

Examples

```
var filled = manager.GetFilledTemplate("video-reference", new Dictionary<string, string> { ["title"] = "Sample" });
```

GetTemplate(string)

Gets a template by its type.

```
public Dictionary<string, object>? GetTemplate(string templateType)
```

Parameters

`templateType` string

The type of template to retrieve (e.g., `video-reference`).

Returns

Dictionary<string, object>

A copy of the template dictionary, or `null` if not found.

Examples

```
var template = manager.GetTemplate("pdf-reference");
```

GetTemplateTypes()

Gets all available template types loaded from `metadata.yaml`.

```
public List<string> GetTemplateTypes()
```

Returns

List<string>

A list of template type names.

LoadTemplates()

Loads all templates from the `metadata.yaml` file into memory.

```
public void LoadTemplates()
```

Remarks

Parses each YAML document in the file and stores templates by their `template-type` key. Logs errors and warnings for missing files or parse failures.

Class NoQuotesForBannerVisitor

Namespace: NotebookAutomation.Core.Utils

Assembly: NotebookAutomation.Core.dll

Custom YamlDotNet visitor to avoid quotes for the banner field.

```
public class NoQuotesForBannerVisitor : ChainedObjectGraphVisitor,
IObjectGraphVisitor<IEmitter>
```

Inheritance

object ← ChainedObjectGraphVisitor ← NoQuotesForBannerVisitor

Implements

IObjectGraphVisitor<IEmitter>

Inherited Members

ChainedObjectGraphVisitor.Enter(IObjectDescriptor, IEmitter) ,
ChainedObjectGraphVisitor.EnterMapping(IObjectDescriptor, IObjectDescriptor, IEmitter) ,
ChainedObjectGraphVisitor.VisitScalar(IObjectDescriptor, IEmitter) ,
ChainedObjectGraphVisitor.VisitMappingStart(IObjectDescriptor, Type, Type, IEmitter) ,
ChainedObjectGraphVisitor.VisitMappingEnd(IObjectDescriptor, IEmitter) ,
ChainedObjectGraphVisitor.VisitSequenceStart(IObjectDescriptor, Type, IEmitter) ,
ChainedObjectGraphVisitor.VisitSequenceEnd(IObjectDescriptor, IEmitter) , object.Equals(object) ,
object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

NoQuotesForBannerVisitor(IObjectGraphVisitor<IEmitter>)

```
public NoQuotesForBannerVisitor(IObjectGraphVisitor<IEmitter> nextVisitor)
```

Parameters

`nextVisitor` IObjectGraphVisitor<IEmitter>

Methods

EnterMapping(IPropertyDescriptor, IObjectDescriptor, IEmitter)

Overrides the standard serialization behavior for the "banner" property to emit it without quotes.

```
public override bool EnterMapping(IPropertyDescriptor key, IObjectDescriptor value,
IEmitter context)
```

Parameters

`key` IPropertyDescriptor

`value` IObjectDescriptor

`context` IEmitter

Returns

bool

Class PathFormatter

Namespace: NotebookAutomation.Core.Utils

Assembly: NotebookAutomation.Core.dll

Utility class for formatting file paths in log messages.

```
public static class PathFormatter
```

Inheritance

object ← PathFormatter

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Format(string, LogLevel)

Formats a file path for logging based on the log level. Uses the full path for Debug or Trace levels, and a shortened path otherwise.

```
public static string Format(string path, LogLevel logLevel)
```

Parameters

`path` string

The file path to format.

`logLevel` LogLevel

The logging level.

Returns

string

The formatted path string.

ShortenPath(string, int)

Shortens a path to not exceed the specified maximum length. Keeps the filename and as much of the path as possible.

```
public static string ShortenPath(string path, int maxLength)
```

Parameters

`path` string

The path to shorten.

`maxLength` int

Maximum allowed length.

Returns

string

A shortened path.

Class PathUtils

Namespace: NotebookAutomation.Core.Utils

Assembly: NotebookAutomation.Core.dll

Provides utility methods for working with file and directory paths.

```
public static class PathUtils
```

Inheritance

object ← PathUtils

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

The PathUtils class contains static methods for common path-related operations needed throughout the notebook automation system, such as:

- Finding paths relative to the application or directory
- Ensuring directories exist
- Normalizing paths across platforms
- Generating safe file paths for new files

This class is designed to centralize path handling logic and ensure consistent behavior across different parts of the application, especially for operations that need to handle cross-platform path differences.

Methods

EnsureDirectoryExists(string)

Ensures a directory exists, creating it if necessary.

```
public static string EnsureDirectoryExists(string directoryPath)
```

Parameters

`directoryPath` string

The directory path to ensure exists.

Returns

string

The normalized directory path.

Examples

```
string outputDir = PathUtils.EnsureDirectoryExists("output/reports/monthly");
string filePath = Path.Combine(outputDir, "report.pdf");
```

Remarks

This method checks if the specified directory exists, and creates it (including any necessary parent directories) if it doesn't. It returns the normalized path to the directory, which is useful for chaining operations.

The method normalizes the path to ensure it works correctly on all platforms.

GenerateUniqueFilePath(string)

Generates a unique file path by appending a number if the file already exists.

```
public static string GenerateUniqueFilePath(string baseFilePath)
```

Parameters

`baseFilePath` string

The base file path (including extension).

Returns

string

A unique file path that doesn't exist yet.

Examples

```
string safePath = PathUtils.GenerateUniqueFilePath("output/report.pdf");
File.WriteAllText(safePath, content);
```

Remarks

This method checks if a file already exists at the specified path, and if so, generates a new path by appending a number before the extension. For example, if "report.pdf" exists, it will try "report (1).pdf", then "report (2).pdf", and so on.

This is useful for generating output file paths that won't overwrite existing files.

GetCommonBasePath(IEnumerable<string>)

Gets the common base directory shared by a collection of paths.

```
public static string GetCommonBasePath(IEnumerable<string> paths)
```

Parameters

`paths` IEnumerable<string>

The collection of paths.

Returns

string

The common base directory, or an empty string if there is no common base.

Examples

```
string[] paths = new[] {
    "C:/Projects/MyProject/src/file1.cs",
    "C:/Projects/MyProject/src/Models/model.cs",
    "C:/Projects/MyProject/tests/test1.cs"
};
string common = PathUtils.GetCommonBasePath(paths);
// Result: "C:/Projects/MyProject/"
```

Remarks

This method finds the longest common path prefix shared by all paths in the collection. This is useful for identifying a common working directory or for organizing files that are related but might be scattered across different subdirectories.

The method returns a directory path (ending with a directory separator), or an empty string if there is no common base directory (for example, if the paths are on different drives).

GetPathRelativeToApp(string)

Gets a path for a file relative to the application's base directory.

```
public static string GetPathRelativeToApp(string relativePath)
```

Parameters

`relativePath` string

The relative path from the application directory.

Returns

string

The full path to the file.

Examples

```
string configPath = PathUtils.GetPathRelativeToApp("config/settings.json");
```

Remarks

This method constructs a path relative to the application's base directory, which is the directory containing the executing assembly. This is useful for finding configuration files and other resources that ship alongside the application.

The method handles normalization to ensure paths work correctly on all platforms.

GetPathRelativeToDirectory(string, string)

Gets a path for a file relative to a specified directory.

```
public static string GetPathRelativeToDirectory(string basePath, string relativePath)
```

Parameters

`basePath` string

The base directory path.

`relativePath` string

The relative path from the base directory.

Returns

string

The full path to the file.

Examples

```
string outputPath = PathUtils.GetPathRelativeToDirectory(projectDir, "output/results.json");
```

Remarks

This method constructs a path relative to the specified base directory. It normalizes both paths to ensure they work correctly on all platforms, and combines them using platform-appropriate path separators.

MakeRelative(string, string)

Makes a path relative to a base directory.

```
public static string MakeRelative(string basePath, string fullPath)
```

Parameters

`basePath` string

The base directory path.

`fullPath` string

The full path to make relative.

Returns

string

The relative path, or the full path if it can't be made relative.

Examples

```
string projectDir = "C:/Projects/MyProject";
string fullPath = "C:/Projects/MyProject/src/file.cs";
string relativePath = PathUtils.MakeRelative(projectDir, fullPath);
// Result: "src/file.cs"
```

Remarks

This method attempts to make a path relative to a specified base directory. This is useful for storing and displaying shorter, more readable paths that are relative to a known location like a project directory.

If the full path doesn't start with the base path (meaning it's not actually within the base directory), the method returns the original full path.

NormalizePath(string)

Normalizes a path by converting slashes to the platform-specific path separator.

```
public static string NormalizePath(string path)
```

Parameters

`path` string

The path to normalize.

Returns

string

The normalized path.

Examples

```
// Works on both Windows and Unix:
string normalizedPath = PathUtils.NormalizePath("reports\\monthly/current");
```

Remarks

This method normalizes a path by replacing all forward and backward slashes with the platform-specific path separator character. This ensures that paths work correctly on all platforms, regardless of how they were originally specified.

The method also trims any leading or trailing whitespace to prevent common errors.

Class YamlHelper

Namespace: NotebookAutomation.Core.Utils

Assembly: NotebookAutomation.Core.dll

Helper class for working with YAML content in markdown files.

```
public class YamlHelper : IYamlHelper
```

Inheritance

object ← YamlHelper

Implements

IYamlHelper

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

Provides functionality for parsing, modifying, and serializing YAML frontmatter found in markdown documents, with special consideration for preserving formatting and handling Obsidian-specific conventions.

```
var helper = new YamlHelper(logger);
var frontmatter = helper.ExtractFrontmatter(markdown);
var dict = helper.ParseYamlToDictionary(frontmatter);
```

Constructors

YamlHelper(ILogger)

Initializes a new instance of the YamlHelper class.

```
public YamlHelper(ILogger logger)
```

Parameters

`logger` ILogger

The logger to use for diagnostic and error reporting.

Methods

CreateMarkdownWithFrontmatter(Dictionary<string, object>)

Creates new markdown content with the specified frontmatter.

```
public static string CreateMarkdownWithFrontmatter(Dictionary<string, object> frontmatter)
```

Parameters

`frontmatter` Dictionary<string, object>

The frontmatter as a dictionary.

Returns

string

The created markdown content.

Remarks

Produces a markdown string with only the YAML frontmatter block.

DiagnoseYamlFrontmatter(string)

Diagnoses YAML parsing issues and returns detailed information about any problems found.

```
public (bool Success, string Message, Dictionary<string, object>? Data)
DiagnoseYamlFrontmatter(string markdown)
```

Parameters

`markdown` string

The markdown content to diagnose.

Returns

(bool Success, string Message, Dictionary<string, object> Data)

A diagnostic result containing information about YAML parsing issues.

Remarks

Returns a tuple indicating success, a message, and the parsed data (if successful).

ExtractFrontmatter(string)

Extracts the YAML frontmatter from markdown content.

```
public string? ExtractFrontmatter(string markdown)
```

Parameters

`markdown` string

The markdown content to parse.

Returns

string

The YAML frontmatter as a string, or null if not found.

Examples

```
var yaml = helper.ExtractFrontmatter(markdown);
```

Remarks

Returns only the YAML block delimited by `---` at the start and end of the file.

ExtractTags(Dictionary<string, object>)

Parses and extracts tags from frontmatter.

```
public static HashSet<string> ExtractTags(Dictionary<string, object> frontmatter)
```

Parameters

`frontmatter` Dictionary<string, object>

The frontmatter as a dictionary.

Returns

HashSet<string>

A set of tags found in the frontmatter.

Remarks

Handles both string and array formats for tags.

IsAutoGeneratedStateReadOnly(Dictionary<string, object>)

Checks if the auto-generated-state in frontmatter indicates the content is read-only.

```
public bool IsAutoGeneratedStateReadOnly(Dictionary<string, object> frontmatter)
```

Parameters

`frontmatter` Dictionary<string, object>

The frontmatter dictionary to check.

Returns

bool

True if the auto-generated-state is "read-only" or "readonly", false otherwise.

IsFileReadOnly(string)

Checks if a markdown file has readonly auto-generated-state in its frontmatter.

```
public bool IsFileReadOnly(string filePath)
```

Parameters

`filePath` string

Path to the markdown file to check.

Returns

bool

True if the file has readonly auto-generated-state, false otherwise.

LoadFrontmatterFromFile(string)

Loads frontmatter from a markdown file.

```
public Dictionary<string, object> LoadFrontmatterFromFile(string filePath)
```

Parameters

`filePath` string

Path to the markdown file.

Returns

Dictionary<string, object>

The parsed frontmatter as a dictionary, or an empty dictionary on error.

ParseYaml(string)

Parses YAML frontmatter to a dynamic object.

```
public object? ParseYaml(string yaml)
```

Parameters

`yaml` string

The YAML content to parse.

Returns

object

The parsed object, or null if parsing failed.

Remarks

Uses YamlDotNet for deserialization.

ParseYamlToDictionary(string)

Parses YAML frontmatter to a dictionary.

```
public Dictionary<string, object> ParseYamlToDictionary(string yaml)
```

Parameters

`yaml` string

The YAML content to parse.

Returns

Dictionary<string, object>

The parsed dictionary, or an empty dictionary if parsing failed.

Remarks

Handles common formatting issues and code block wrappers.

RemoveFrontmatter(string)

Removes YAML frontmatter from markdown content if present.

```
public string RemoveFrontmatter(string markdown)
```

Parameters

`markdown` string

The markdown content to clean.

Returns

string

The content without frontmatter.

ReplaceFrontmatter(string, Dictionary<string, object>)

Replaces the frontmatter in a markdown document with new frontmatter.

```
public string ReplaceFrontmatter(string markdown, Dictionary<string, object> newFrontmatter)
```

Parameters

`markdown` string

The markdown content.

`newFrontmatter` Dictionary<string, object>

The new frontmatter as a dictionary.

Returns

string

The updated markdown content.

SaveMarkdownWithFrontmatter(string, string, Dictionary<string, object>)

Saves markdown with updated frontmatter to a file.

```
public bool SaveMarkdownWithFrontmatter(string filePath, string markdown, Dictionary<string,
object> frontmatter)
```

Parameters

`filePath` string

Path to save the file.

`markdown` string

The markdown content.

`frontmatter` Dictionary<string, object>

The frontmatter to update or add.

Returns

bool

True if successful, false otherwise.

SerializeToYaml(Dictionary<string, object>)

Serializes a dictionary to YAML format.

```
public string SerializeToYaml(Dictionary<string, object> data)
```

Parameters

`data` Dictionary<string, object>

The dictionary to serialize.

Returns

string

The serialized YAML string.

SerializeYaml(Dictionary<string, object>)

Serializes a dictionary to YAML.

```
public static string SerializeYaml(Dictionary<string, object> data)
```

Parameters

`data` Dictionary <string, object>

The dictionary to serialize.

Returns

string

The serialized YAML string.

UpdateFrontmatter(string, Dictionary<string, object>)

Updates existing markdown content with new frontmatter.

```
public string UpdateFrontmatter(string markdown, Dictionary<string, object> frontmatter)
```

Parameters

`markdown` string

The original markdown content.

`frontmatter` Dictionary <string, object>

The new frontmatter as a dictionary.

Returns

string

The updated markdown content.

Remarks

Replaces or inserts the YAML frontmatter block at the top of the markdown file.

UpdateTags(Dictionary<string, object>, HashSet<string>)

Updates frontmatter with new tags.

```
public static Dictionary<string, object> UpdateTags(Dictionary<string, object> frontmatter,
HashSet<string> tags)
```

Parameters

`frontmatter` Dictionary<string, object>

The frontmatter dictionary to update.

`tags` HashSet<string>

The set of tags to set.

Returns

Dictionary<string, object>

The updated frontmatter.