

Technical Report – `INAF.Libraries.Net.CommonEmailSenders` Solution

****Version:**** 1.0

****Date:**** 2026

****Target Framework:**** .NET

****Project Type:**** Library

****Author:**** Francesco Carraro

1. Overview

``INAF.Libraries.Net.CommonEmailSenders`` is a class library focused on sending application emails through configurable contexts. It centralizes email delivery logic, configuration lookup, HTML template handling, and logging, so consuming applications can reuse a common email-sending workflow.

The solution currently contains a single SDK-style .NET project:

- ``INAF.Libraries.Net.CommonEmailSenders``

2. Purpose

The library is designed to:

- send emails through a shared sender service;
- support context-specific email settings from configuration;
- prepare email body content from external template files;
- integrate with shared application models and helper libraries;
- provide an extensible base class for specialized email sender contexts.

3. Project Structure

Main project

- ``INAF.Libraries.Net.CommonEmailSenders.csproj``

Main source files

- ``EmailSender.cs``
- ``EmailSenderContexts/IContextSender.cs``
- ``EmailSenderContexts/BaseContextSender.cs``

4. Technical Characteristics

Project format

The project uses the SDK-style format:

- ``Microsoft.NET.Sdk``
- nullable reference types enabled;
- implicit global usings enabled.

Implemented target

Although this report header uses the generic label ``.NET``, the current project file targets:

- ``.net10.0``

External NuGet dependencies

- ``.Mailjet.Api` 3.0.1``
- ``.MailKit` 4.14.1``

Internal project dependencies

The library references the following internal projects:

- ``.INAF.Libraries.Net.CommonWebAppHelpers``
- ``.INAF.Libraries.Net.CommonWebAppModels``
- ``.INAF.Libraries.Net.Log``

5. Core Components

``.EmailSender``

This class is responsible for the actual email delivery operation.

Key responsibilities:

- reads Mailjet credentials from configuration;
- builds the outgoing email request;
- sends the message through ``.MailjetClient``;
- logs sending attempts and error details.

Main behavior:

1. Reads ``.ApiKey`` and ``.SecretKey`` from ``.AppSettings:MailServer``.
2. Validates that credentials are present.
3. Creates a Mailjet request using sender, subject, recipient, and HTML body.
4. Sends the request asynchronously.
5. Logs API errors if the response is unsuccessful.

``.IContextSender``

This interface defines the asynchronous execution contract for context-specific email senders:

- ``.Task RunAsync(Cancellation_token stoppingToken)``

It allows implementations to encapsulate background or workflow-driven email sending operations while exposing a common entry point.

``.BaseContextSender``

This class provides shared logic for context-driven email preparation and dispatch.

Injected dependencies:

- `IServiceProvider``
- `IConfiguration``
- `LogHelper``

Main responsibilities:

- retrieve sender settings from configuration;
- load the HTML email body from a file path;
- resolve shared services such as `EncryptionHelper``;
- update placeholders in the email body;
- invoke `EmailSender`` to send messages;
- delegate post-send state updates to derived classes.

Important extension points:

- `RunAsync(...)`` is virtual and intended to be implemented by derived context senders.
- `setEmailAsSentAsync(...)`` is virtual and intended to persist successful delivery state in a specialized implementation.

6. Configuration Model

The library expects configuration values under `AppSettings``.

Mail server configuration

Used by `EmailSender``:

- `AppSettings:MailServer:ApiKey``
- `AppSettings:MailServer:SecretKey``

Context email configuration

Used by `BaseContextSender.retrieveSettings(...)``:

- `AppSettings:{context}:EmailSettings:SenderName``
- `AppSettings:{context}:EmailSettings:SenderEmail``
- `AppSettings:{context}:EmailSettings:EmailSubject``
- `AppSettings:{context}:EmailSettings:EmailBodyFilepath``

7. Email Sending Flow

The common sending flow implemented by the library is:

1. A context sender starts its execution through `RunAsync(...)``.
2. The sender retrieves context-specific email settings.
3. The HTML template is loaded from the configured file path.
4. A login validation identifier is encrypted through `EncryptionHelper``.
5. The encrypted value is converted to a URL-safe format.

6. Placeholders in the email body are updated with runtime values.
7. `EmailSender.SendEmailAsync(...)` submits the email through Mailjet.
8. If the send succeeds, the derived sender can mark the email as sent through `SetEmailAsSentAsync(...)`.

8. Design Notes

Strengths

- clear separation between transport logic and context-specific workflow;
- reusable base class for multiple email scenarios;
- integration with centralized logging;
- configuration-driven sender identity, subject, and body template;
- asynchronous API surface for email delivery operations.

Current limitations

- `BaseContextSender.RunAsync(...)` and `SetEmailAsSentAsync(...)` are placeholders and must be implemented by derived classes;
- template file loading uses `File.ReadAllText(...)` directly and assumes the configured path is valid;
- `EmailSender` logs unsuccessful Mailjet responses but does not set the result to failed in that branch;
- naming conventions are not fully aligned with standard C# style in some methods.

9. Typical Usage Scenario

A consuming application can derive a specialized sender from `BaseContextSender` to:

- query pending email notifications;
- call the shared send procedure for each pending item;
- update the application state after successful delivery.

This pattern allows business-specific email workflows to remain outside the reusable infrastructure layer.

10. Summary

`INAF.Libraries.Net.CommonEmailSenders` is a focused reusable library that provides a common foundation for sending templated emails in .NET applications. Its structure is intentionally compact, with one transport class, one execution contract, and one extensible base context class. The current implementation is suitable as a shared infrastructure component for application login validation or similar notification workflows.