## **INTRODUCTION**

This demo application was inspired by a 2016 job interview question:

|  |
| --- |
| Write a simple Email app that uses Dependency Injection.  Use the following struct and interfaces in your solution:  public struct ParsedName  {  public string First { get; set; }  public string Last { get; set; }  }  public interface INameParser  {  ParsedName ParseName(string input);  }  public interface IEmailSender  {  void SendEmail(string to, string from, string subject, string body);  } |

I provided a simplified answer and moved on. After the interview, I returned to the question and created a demo application that was more thorough.

I revisited the demo application in 2022. New versions of .Net 6 (Core) and Visual Studio 2022 justified refreshing parts of the original demo application.

## **SUMMARY**

This demo application is a simple console app based on the Visual Studio 2022 console app template.

The demo application creates a custom message body based on a collection of name strings. The custom message body contains the names of the classes used to generate the message. The class names in the message body demonstrates .Net reflection and confirms that the correct classes were injected into the application interfaces.

The demo application passes the custom message body to an appropriate email sender. The email sender then send’s the message.

An email “message” is displayed in a console window. This is an example of one “message”.

|  |
| --- |
| eMail Sent  ----------  To: joe@gmail.com  From: me@hotmail.com  Subject: Let’s play some music  Let’s get together, play some music and have some fun.  ------------  Original Input Name: **Gaga, Lady**  **First: Lady**  **Last: Gaga**  Name Parsed With: Dependency\_Injection\_SMTP\_Demo\_master\_VS2022.**LastFirstParser**  Message Sent With: Dependency\_Injection\_SMTP\_Demo\_master\_VS2022.**MyEmailSender** |

## **CONDITIONS**

1. The structure of this demo application is intentionally simple in order to not obfuscate the DI example.
2. There is no exception handling in this demo application, such as TRY/CATCH/FINALLY or USE(…){…}. This is intentional because it would obfuscate the DI example.
3. This demo application does not send a real Email message. The message is ‘sent’ to the Console. It is reasonably simple to upgrade this app to send a real message. At least two email services would be required, such as Gmail and Hotmail.
4. All methods are public to simplify experimentation.
5. The ‘To’, ‘From’, and ‘Subject’ text is static. DI could easily be used to automate these values.
6. The EmailSender is selected based on the length of the last name. DI could easily be used to control the EmailSender. (See ‘Final Thoughts’ for enhancements below).

## **BUILD**

The GitHub repo, code/solution, is NOT compatible with console applications generated with Visual Studio 2019 console application templates. See Final Thoughts for a compatible solution.

This is a simple C# console application generated with Visual Studio 2022. The GitHub repo contains the complete solution. To run the demo application, start by downloading the GitHub repo (all files).

The developer has TWO options:

1. Open the downloaded solution with Visual Studio 2022.

**\_OR\_**

1. Create a new Visual Studio 2022 console app and replace the template generated Program.cs file with the same file name down loaded from GitHub. Also add the Contoller.cs file to the console app solution.

At this point, the application can be Built and Tested completely inside Visual Studio.

## **APPLICATION FLOW**

**Code Block One:** Builds a collection of name strings that will be parsed.

**Code Block Two: Dependency Injection** occurs. At the top of the block, GetNameForm(…) decides which parser class should be used and ‘injects’ the proper type into the INameParsr interface.

At the bottom of the block, the correct version of ParseName( ) is called. The results are returned to a ParseName struct.

**Code Block Three:** Decides which mail sender class will be used and initializes the IEmailSender interface. The choice is based on the length of the last name. **Dependency Injection** would be appropriate for this block. See ‘Final Thoughts’ for enhancements to **Selecting Email Sender**.

**Code Block Four:** Builds the message and sends it. At the top of the block, the BuildMessage( ) method creates the body of the message.

At the bottom of the block, the SendMail() method ‘sends’ the Email. In this case the Email is sent to the console window.

## **INTERFACES & CLASSES: [Controller.cs]**

**FirstLastParser : INameParser**

* The implementation of ParseName( ), in the FirstLastParser class, parses a name string with the following pattern: “Lady Gaga”.
* Note the [space] token/delimiter in the name.

**LastFirstParser : INameParser**

* The implementation of ParseName( ), in the LastFirstParser class, parses a name string with the following pattern: “Gaga, Lady”.
* Note the [comma] token/delimiter in the name.

**OneNameParser : INameParser**

* The implementation of ParseName( ), in the OneNameParser class, parses a name string with the following pattern: “Bono”.
* Note the absence of any token/delimiter in the name.

**MyEmailSender : IEmailSender**

**YourEmailSender : IEmailSender**

The implementation of SendEmail( ) is *identical* for both classes. *This is intentional to simplify the example.* The method constructs the message to be sent. The message contains:

* + To
  + From
  + Subject
  + Body. The body of the message is created by the BuildMessage( ) helper method.

## **HELPER METHODS**

**GetNameForm(…)**

* The sole purpose of this method is to help choose the correct name parser class to instantiate.
* This method inspects each input name and returns the ‘form’ of the name as a string:

“LastFirst”, “FirstLast”, “OneName”.

* If “noValidToken(…)” is true, “OneName” form is returned.

It supports names like “Bono”

* If “IndexOf(',')” is true, “LastFirst” form is returned.

It supports names like “Gaga, Lady”.

* If “IndexOf('[space]')” is false, “FirstLast” form is returned.

It supports names like “Lady Gaga”.

**noValidToken(…)**

This searches for a token/delimiter like [space] or [comma]. If no token is found, true is returned.

This is the implementation.

Func<string, bool> noValidToken = (s) => s.IndexOfAny(new char[] { ' ', ',' }) < 0;

**BuildMessage(…)**

* This method creates the message body with this data:
  + [name] // Original input name
  + [parsedName.First] // Parsed first name
  + [parsedName.Last] // Parsed last name
  + [parserName] // Name of the class that was used to parse the name
  + [emailSenderName] // Name of the Email Sender that will be used

## **FINAL THOUGHTS**

**Selecting Email Sender**

This demo application selects the EmailSender based on the length of the last name.

An enhancement could select the EmailSender based on geographic or demographic properties. Another option might be to use the type of customer: Guest, Registered, or Professional in an Ecommerce application.

**Message Delivery: Parallel / Asynchronous**

The messages are currently ‘sent’ serially. The messages could be sent parallel/async. A task would be started for each EmailSender instance. Each name in the name list would be passed to an available EmailSender task. When the task completes, another name in the name list would be assign to the task. This would continue until all of the names in the list had been sent.

This feature was not implemented in this example because it would obfuscate the DI example and would make code-walks/debugging more complicated. In a production environment parallel/async would likely be appropriate.

**Visual Studio 2019 & Earlier Versions**

There is a GitHub repo that contains the ‘original’ version of the demo app that was created in 2016. This repo only contains this source code file: **Program\_DI\_Email\_Example.cs**

After downloading the repo, the **Program\_DI\_Email\_Example.cs** should be renamed to **Program.cs**. Then generate a console app using the Visual Studio 2019 console app template. Replace the template version of **Program.cs** with the renamed **Program.cs** (**Program\_DI\_Email\_Example.cs**) version. Build and test.

Here is a Url to the GitHub repo.

***https://github.com/kevinbowe/Dependency-Injection\_SMTP-Demo***