

## PROJECT: FINCH CONTROL S5 (PERSISTENCE)

## OVERVIEW

Students will implement the use of a data file to store information for the Finch Control application. Possible applications of a data file are indicated below.

- Store and retrieve a console theme, background and foreground colors.
- Store and retrieve login information including a user name and password.
- Store and retrieve sensor data in an array in the Data Recorder module.
- Store and retrieve Finch robot commands in a list in the User Programming module.

## INSTRUCTIONS

1. Extend the application framework with persistence using only **ONE** of the following three methods.  
**Note:** The methods described in the instructions are simplified examples. Students are free to implement these features using any set of methods of their own design.

- a. **Implement User Theme**

- i. **Challenge Levels**

1. Read and set the theme from a data file.
2. Read the current theme from and write a new theme to a data file.
3. Implement the a Try/Catch block for all of the file I/O operations and generate error messages for the user.

- ii. **File:** *Theme.txt*

1. Create a **Data** folder to hold the text file.
  - a. Store background and foreground colors on two lines.

Red

White

iii. **Method:** *static (ConsoleColor foregroundColor, ConsoleColor backgroundColor) ReadThemeData()*

1. Note that the method returns a tuple with two **ConsoleColor** items. The method can also just return the array from the **File.ReadAllLines** method.
2. Declare a variable **dataPath** and store the path to the **theme.txt** file.
3. Declare an array of string to hold the foreground and background colors read from the data file.
4. Read the data file using the **File.ReadAllLines** method and store them in the array.
5. Parse the two values in the array into the **foregroundColor** and **backgroundColor** elements in the tuple.
6. Return the tuple.

```
static (ConsoleColor foregroundColor, ConsoleColor backgroundColor) ReadThemeData()
{
    string dataPath = @"Data/Theme.txt";
    string[] themeColors;

    ConsoleColor foregroundColor;
    ConsoleColor backgroundColor;

    themeColors = File.ReadAllLines(dataPath);

    Enum.TryParse(themeColors[0], true, out foregroundColor);
    Enum.TryParse(themeColors[1], true, out backgroundColor);

    return (foregroundColor, backgroundColor);
}
```

iv. **Method:** *void WriteThemeData(ConsoleColor background, ConsoleColor foreground)*

1. Declare a variable **dataPath** and store the path to the **theme.txt** file.
2. Write the two console colors to the data file as strings using the **WriteAllText** and **AppendAllText** methods.

```
static void WriteThemeData(ConsoleColor foreground, ConsoleColor background)
{
    string dataPath = @"Data/Theme.txt";

    File.WriteAllText(dataPath, foreground.ToString() + "\n");
    File.AppendAllText(dataPath, background.ToString());
}
```

v. **Method:** *static ConsoleColor GetConsoleColorFromUser(string property)*

1. Declare variables.
2. Create a **do/while** loop to validate the user's input as a ConsoleColor enum.
3. Return the value.

```
static ConsoleColor GetConsoleColorFromUser(string property)
{
    ConsoleColor consoleColor;
    bool validConsoleColor;

    do
    {
        Console.Write($"Enter a value for the {property}:");
        validConsoleColor = Enum.TryParse<ConsoleColor>(Console.ReadLine(), true, out consoleColor);

        if (!validConsoleColor)
        {
            Console.WriteLine("\n\t***** It appears you did not provide a valid console color. Please
            try again. *****\n");
        }
        else
        {
            validConsoleColor = true;
        }
    } while (!validConsoleColor);

    return consoleColor;
}
```

vi. **Method:** *void DisplaySetTheme()*

1. Read, set, and display the current theme colors.

```
static void DataDisplaySetTheme()
{
    (ConsoleColor foregroundColor, ConsoleColor backgroundColor) themeColors;
    bool themeChosen = false;

    //
    // set current theme from data
    //
    themeColors = DataReadThemeData();
    Console.ForegroundColor = themeColors.foregroundColor;
    Console.BackgroundColor = themeColors.backgroundColor;
    Console.Clear();
    DisplayScreenHeader("Set Application Theme");

    Console.WriteLine($"Current foreground color: {Console.ForegroundColor}");
    Console.WriteLine($"Current background color: {Console.BackgroundColor}");
    Console.WriteLine();
}
```

2. Query the user to change the current theme.
3. Create a **do/while** loop.
  - a. Call the **GetConsoleColorFromUser** method for both the foreground and background colors.
  - b. Set the new theme, display the colors, and prompt the user to either keep the theme or enter a new one.

```

Console.Write("\tWould you like to change the current theme [ yes | no ]?");
if (Console.ReadLine().ToLower() == "yes")
{
    do
    {
        themeColors.foregroundColor = GetConsoleColorFromUser("foreground");
        themeColors.backgroundColor = GetConsoleColorFromUser("background");

        //
        // set new theme
        //
        Console.ForegroundColor = themeColors.foregroundColor;
        Console.BackgroundColor = themeColors.backgroundColor;
        Console.Clear();
        DisplayScreenHeader("Set Application Theme");
        Console.WriteLine($" \tNew foreground color: {Console.ForegroundColor}");
        Console.WriteLine($" \tNew background color: {Console.BackgroundColor}");

        Console.WriteLine();
        Console.Write("\tIs this the theme you would like?");
        if (Console.ReadLine().ToLower() == "yes")
        {
            themeChosen = true;
            WriteThemeData(themeColors.foregroundColor, themeColors.backgroundColor);
        }

    } while (!themeChosen);
}
DisplayContinuePrompt();
}

```

vii. **Method:** Main

1. Call `DataDisplaySetTheme`.

viii. **Method:** `void DisplayMainMenu()`

1. Add a **Change Theme** option to the menu can call `DisplayMainMenu`.

**b. Implement a login and registration functionality.**

**i. Challenge Levels**

1. Level 1: Store one username  
A single user name is stored, retrieved and authenticated with the user input.
2. Level 2: Store one username and password  
A single user name and password is stored, retrieved and authenticated with the user input.
3. Level 3: Store multiple usernames  
Multiple user names are stored, retrieved and used to authenticate the user input.
4. Level 4: Store multiple usernames and passwords  
Multiple user names and passwords are stored, retrieved and used to authenticate the user input.

**ii. Developer considerations and extensions:**

1. Level 2 and 4: Username and Password – use a tuple to return both values
2. Login – handling incorrect usernames and/or passwords
3. Register – validating username and/or passwords
4. Register – handling usernames currently in the data set

**c. Implement data saving and recovery in the Data Recorder module.**

**i. Challenge Levels**

Add “**Read from Data File**” and “**Write to Data File**” to the **Data Recorder Menu** and create the appropriate methods.

1. Level 1: Store and retrieve temperature **or** light values.
2. Level 2: Store and retrieve temperature **and** light values.
3. Level 3: Store and retrieve temperature **or** light values with time stamp.
4. Level 4: Store and retrieve temperature **and** light values with time stamp.

**ii. Developer considerations and extensions:**

1. Use separate data files for temperature and light values.
2. Use tuples and lists and arrays of tuples to hold sensor value and time stamp.

**d. Implement data saving and recovery in the User Programming module.**

**i. Challenge Levels**

Add “**Load User Program**” and “**Write User Program**” to the **User Programming Menu** and create the appropriate methods

1. Level 1: Store and retrieve one user program.
2. Level 2: Store and retrieve multiple user programs.
3. Level 3: Store and retrieve one user program with extended commands, command and execution time for that command.
4. Level 4: Store and retrieve multiple user programs with extended commands.

2. Test the application thoroughly.

---

SUBMIT THE ASSIGNMENT

1. Complete the **Skills Checklist**.
  - a. [Face-Face only] Demonstrate the application to the instructor.
  - b. [Online only] Upload the checklist in Moodle.
2. Push the VS solution to GitHub.
3. Submit to Moodle.
  - a. Click the **Project: Finch Control S5 (Persistence)** assignment link.
  - b. [Online only] Submit the completed **Skills Checklist**.
  - c. [Online only] Submit a link to the streaming video walk-through.
  - d. Submit the link to the GitHub repository with the solution.
  - e. Click **Save Changes**.
5. After receiving a grade, refer to Moodle to review the graded rubric and additional comments.

## PROJECT: FINCH CONTROL S5 (PERSISTENCE) - SKILLS CHECKLIST

**Author** \_\_\_\_\_ **Reviewer(s)** \_\_\_\_\_

**[In-class Students Only]**

**Code Share** – Discuss the following during the Peer Review.

- Describe the flow of the application, walking through the application's major components.
- State one coding issue you encountered and how you resolved it.
- Highlight one unique block of code (method or function) that you developed and are particularly proud of. Share how the code block functions.
- State something that you learned during the development of this application that will be useful as you develop future applications.

**[All Students]**

**Check all demonstrated skills and submit.**

<b>Skills</b>	
Read a single line of text from a text file.	V
Write a single line of text to a text file.	V
Read more than one line of text from a text file.	V
Write more than one line of text to a text file.	V
Read into an array or list from a text file.	V
Write an array or list to a text file.	V
Read a line of text with multiple values separated by a comma from a text file.	V
Write a line of text with multiple values separated by a comma to a text file.	V
Use multiple data files in an application.	V