Lab 00 - Application Overview

Home Automation

The same application will be used for all of the labs. Please feel free to refer to this overview if need a reminder on the project structure or how to run the application.

This project is home automation software. It is based on a personal project that I've been working with for a number of years. The initial project used X10 hardware to control lighting and air conditioner in my apartment. The software communicates with the hardware through a dongle connected to a serial port (and yes, there are adapters that let you connect a 9 pin serial dongle to a USB C port). The software allows scheduling of events (on/off), supports weekend/weekday schedules, and also supports scheduling relative to sunrise or sunset (such as turning on lights 30 minutes before sunset). The schedule data is stored in a JSON file.

In these labs, you will make this software more flexible and easier to test. This includes allowing the software to work with Tradfri hardware (the IKEA home automation system) -- and it can be extended to work with Zigbee devices. You will also write tests to make sure that the logic works (and keeps working through changes) and remove reliance on a third-party service that provides sunrise and sunset information.

Solution Overview

> Both .NET 8 and .NET 9 versions of the code are provided. The instructions and code are the same. The only difference in the instructions is that if you come across "net80" or "net90", use the value appropriate for your environment.

The "Starter" folder contains the code files for each Lab.

Note: Each Lab also contains a "Completed" folder with the finished solution. If you get stuck along the way or have issues with debugging, take a look at the code in the "Completed" folder for guidance.

> The code has been tested in Visual Studio 2022 (Community edition) and Visual Studio Code. (It should also work with JetBrains Rider, however it has not been tested.) This lab assumes that you know how to open projects in your IDE of choice, start projects in the debugger, add NuGet packages, and various other development tasks. As such, this lab provides general instructions for Visual Studio 2022 and Visual Studio Code rather than step-by-step walkthroughs of what to click on in the IDE.

We'll start by opening the solution.

Visual Studio 2022: Open the `HouseControl.sln` solution.
Visual Studio Code: Open the `Starter` folder in VS Code.

Folders / Projects

- *DataFiles*
  This contains the JSON file with the schedule data. This file is automatically copied to the output folder so

that the application has easy access to it.

- *HouseControl.Library*
  The main application logic: this contains the "HouseController" that coordinates the various pieces, the "Commander" that communicates with hardware, the "Schedule", and various bits of logic used for the schedules.

- *HouseControl.Library.Tests*
  A test project for the HouseControl.Library project.

- *HouseControl.Sunset*
  A library that returns sunrise and sunset times based on a date and location. The data is pulled from an external service (and later, we will calculate it locally).

- *HouseControlAgent*
  A console application that loads a schedule file from the system and starts the "Commander".

- *SolarCalculator.Service*
  An "external" service that provides sunrise and sunset times.

Running the Application

Before running the application, build all of the projects in the solution. One way of doing this is running `dotnet build` from the root solution folder.

```
PS C:\...\Starter> dotnet build
  ... lots of text here

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:00.93
```

Next is to set the "HouseControlAgent" as the startup project and run it in the debugger. (Alternately, you can also use `dotnet run` from the "Starter/HouseControlAgent" folder on the command line).

You should get output similar to the following:

```
Initializing Controller
Initialization Complete
```

If you type "s" and press "Enter" in the console, then the active schedule items are listed:

```
Initializing Controller
Initialization Complete
s
11/2/2025 10:25:12 AM - Standard (00:00:00), Device: 3, Command: On
11/2/2025 10:26:12 AM - Standard (00:00:00), Device: 5, Command: On
11/2/2025 10:27:12 AM - Standard (00:00:00), Device: 3, Command: Off
11/2/2025 10:28:12 AM - Standard (00:00:00), Device: 5, Command: Off
11/2/2025 1:00:00 PM - Standard (00:00:00), Device: 3, Command: Off
11/2/2025 1:00:00 PM - Standard (06:00:00), Device: 3, Command: On
```

In your output, you will see that a number of items are listed as 1 to 4 minutes in the future (based on the current time).

If you wait a minute or two, the controller will try to run the first scheduled item. This will throw an exception unless you have something connected to serial port COM5 (which you probably don't).

Interestingly enough, the exception is a file not found exception:

```
System.IO.FileNotFoundException: 'Could not find file 'COM5'.'
```

This will be one of the first things you address in the first lab.

Stop debugging, and take a look at one more thing.

Using the Sunset Service

To see the sunset service in action, make a small change to the application. Open the `Program.cs` file in the `HouseControlAgent` project and locate the `InitializeHouseController` method:

```
private static HouseController InitializeHouseController()
{
    //45.6382,-122.7013 = Vancouver, WA, USA
    //41.0990,-75.3936 Kalahari Resort, PA

    var fileName = AppDomain.CurrentDomain.BaseDirectory + "ScheduleData";
    var sunsetProvider = new SolarServiceSunsetProvider(41.0990, -75.3936);
    var schedule = new Schedule(fileName, sunsetProvider);
    var controller = new HouseController(schedule);

    //var sunset = sunsetProvider.GetSunset(DateTime.Today.AddDays(1));
    //Console.WriteLine($"Sunset Tomorrow: {sunset:G}");

    return controller;
}
```

At the top of this method are several latitude/longitude locations. These are used to calculate sunrise and sunset times. The locations include "Vancouver, WA, USA" and the workshop location. The location value is passed to the `SolarServiceSunsetProvider`. Make sure that the location is close to where your computer thinks you are, otherwise, you may get unexpected values for sunrise and sunset times.

Uncomment the 2 lines of code near the bottom:

```
    var sunset = sunsetProvider.GetSunset(DateTime.Today.AddDays(1));
    Console.WriteLine($"Sunset Tomorrow: {sunset:G}");
```

This will attempt to call the service that provides sunrise and sunset times. If you build and run the application now, you will get an unhandled exception:

```
Initializing Controller
Unhandled exception. System.AggregateException: One or more errors occurred. (No
connection could be made because the target machine actively refused it.
(localhost:8973))
```

This is because the service is not running.

Stop the debugger. Then you will start the service and try again.

Since you will not edit the service, you may find it easiest to start the service outside of the IDE. Then you can just leave it running while working with the rest of the application. (As an alternative, you can configure your debugger to start both `SolarCalculator.Service` and `HousecontrolAgent` projects.)

From the command line, navigate to the `SolarCalculator.Service` project and type `dotnet run`.

```
PS C:\...\Starter\SolarCalculator.Service> dotnet run
Building...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:8973
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path:
C:\Development\Events\...\Labs\Lab01\dotnet8\Starter\SolarCalculator.Service
```

With the service running, go back an re-run the `HouseControlAgent` application.

```
Initializing Controller
Sunset Tomorrow: 11/1/2025 6:14:04 PM
Initialization Complete
```

When I initially added the sunrise and sunset functionality, I connected with a third-party service (that I have simulated here). One day, that service was throwing 500 errors, and it kept me from being able to run the application.

This is another of the issues that you address in the first lab.

Causing a More Immediate Failure

Let's add some code so that the serial port issue causes failure right at startup (so you do not need to wait 2 minutes to see if it is working or not).

Stop debugging, and open the `Program.cs` file in the `HouseControlAgent` project. Then look at the commented section in the `Main` method.

```
    // For hardware/scheduling testing purposes
    // Uncomment this section to ensure that the hardware
    // and scheduling is working as expected.
    //await controller.SendCommand(5, DeviceCommand.On);
    //await controller.SendCommand(5, DeviceCommand.Off);
```

Uncomment the 2 `SendCommand` method calls:

```
    // For hardware/scheduling testing purposes
    // Uncomment this section to ensure that the hardware
    // and scheduling is working as expected.
    await controller.SendCommand(5, DeviceCommand.On);
    await controller.SendCommand(5, DeviceCommand.Off);
```

Now, rerun the application, and the exception will happen during the initialization process.

```
Initializing Controller
Sunset Tomorrow: 11/1/2025 6:14:04 PM
Unhandled exception. System.IO.FileNotFoundException: Could not find file 'COM5'.
```

Wrap Up

Feel free to refer to this file while working on the labs. A copy of this file appears in the same folder as the instructions for each lab.

END - Lab 00 - Application Overview
