

Event-Driven Programming in C# Windows Forms

Bhoumik Patidar

April 22, 2025

1 Introduction, Setup, and Tools

1.1 Overview

This report documents the implementation of an alarm clock application using event-driven programming in C#, covering both console and Windows Forms versions. The assignment demonstrates the publisher-subscriber pattern and event handling in .NET applications.

1.2 Environment Setup

The development environment consisted of:

- Windows 10 Operating System
- Visual Studio 2022 Community Edition
- .NET 6.0 SDK
- Windows Forms App (.NET Framework) template

1.3 Learning Objectives Achieved

- Implemented publisher-subscriber pattern in console application
- Created event-driven Windows Forms application
- Demonstrated timer-based events and UI updates
- Validated user input in both implementations

2 Methodology and Execution

2.1 Part 1: Console Application

2.1.1 Design Approach

The console application follows the publisher-subscriber model with these components:

- **Publisher:** AlarmClock class that monitors system time
- **Event:** RaiseAlarm triggered when target time matches system time
- **Subscriber:** RingAlarm method that handles the event

2.1.2 Key Code Implementation

```
1 class AlarmClock
2 {
3     private readonly string _targetTime;
4     public event EventHandler? RaiseAlarm;
5
6     protected virtual void OnRaiseAlarm()
7     {
8         RaiseAlarm?.Invoke(this, EventArgs.Empty);
9     }
10
11     public void Start()
12     {
13         while (true)
14         {
15             if (DateTime.Now.ToString("HH:mm:ss") == _targetTime)
16             {
17                 OnRaiseAlarm();
18                 break;
19             }
20             Thread.Sleep(1000);
21         }
22     }
23 }
```

Listing 1: Publisher class implementation

```
1 static void Main(string[] args)
2 {
3     var alarm = new AlarmClock(input);
4     alarm.RaiseAlarm += RingAlarm;
5     alarm.Start();
6 }
7
8 private static void RingAlarm(object? sender, EventArgs e)
9 {
10     Console.WriteLine("        Alarm! Time's up!        ");
11 }
```

Listing 2: Subscriber implementation

2.1.3 Execution Flow

1. User enters time in HH:mm:ss format
2. AlarmClock starts monitoring system time
3. When times match, RaiseAlarm event is triggered
4. RingAlarm handler executes with alarm message

2.2 Part 2: Windows Forms Application

2.2.1 Design Approach

The Windows Forms version enhances the console application with:

- Graphical user interface with input validation
- Timer-based background color changes
- Message box notification on alarm trigger
- Proper event handling for button clicks

2.2.2 Key Components

```
1 private void InitializeComponent()  
2 {  
3     // Form settings  
4     this.Text = "Alarm Clock";  
5     this.ClientSize = new Size(320, 120);  
6  
7     // Timer setup  
8     timer = new System.Windows.Forms.Timer()  
9     {  
10         Interval = 1000,  
11         Enabled = false  
12     };  
13     timer.Tick += Timer_Tick;  
14 }
```

Listing 3: Form initialization

```
1 private void ButtonStart_Click(object? sender, EventArgs e)  
2 {  
3     if (!TimeSpan.TryParseExact(  
4         textBoxTime.Text.Trim(),  
5         "hh\\:\\:mm\\:\\:ss",  
6         null,  
7         out targetTime))  
8     {  
9         MessageBox.Show("Invalid time format");  
10        return;  
11    }  
12    timer.Start();  
13 }  
14  
15 private void Timer_Tick(object? sender, EventArgs e)  
16 {  
17     this.BackColor = Color.FromArgb(  
18         rng.Next(256), rng.Next(256), rng.Next(256));  
19  
20     if (DateTime.Now.TimeOfDay >= targetTime)  
21     {  
22         timer.Stop();  
23         MessageBox.Show("    Alarm! Time's up!    ");  
24     }  
25 }
```

Listing 4: Event handlers

2.2.3 Execution Flow

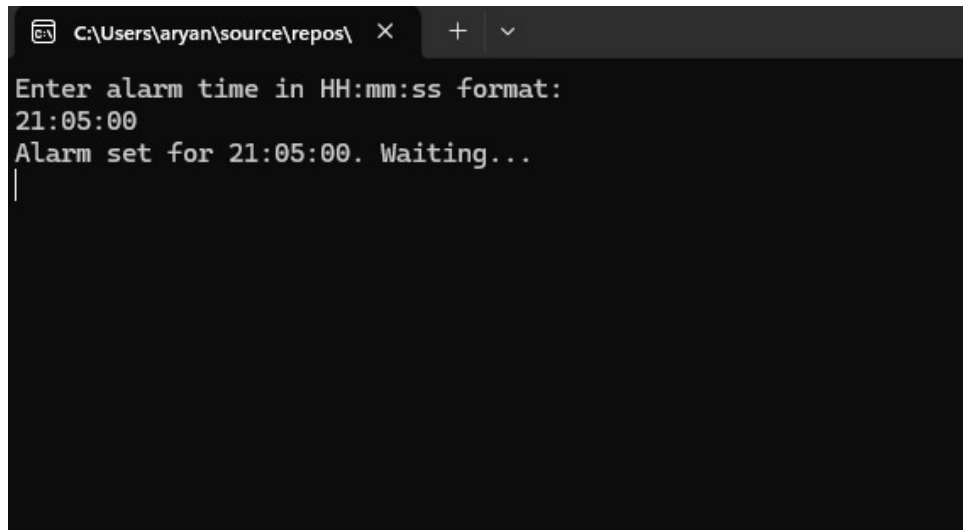
1. User enters time in textbox and clicks Start
2. Input validation checks HH:mm:ss format
3. Timer begins 1-second interval ticks
4. Each tick changes form background color randomly
5. On target time match:
 - Timer stops hangs stop
 - Message box displays alarm

3 Results and Analysis

3.1 Console Application Results

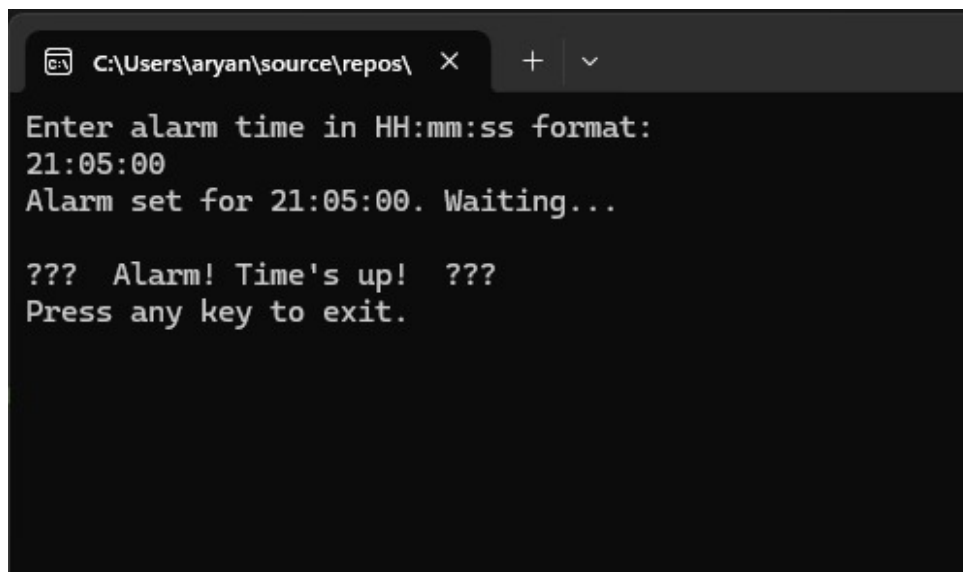
- Successfully implemented event-driven architecture

- Proper time validation using `TimeSpan.TryParse`
- Clean separation of publisher and subscriber
- `Thread.Sleep` used for periodic checking (1 second intervals)



```
C:\Users\aryan\source\repos\ X + v
Enter alarm time in HH:mm:ss format:
21:05:00
Alarm set for 21:05:00. Waiting...
|
```

Figure 1: The program displays the time for which the clock is set for and is waiting for that time to come



```
C:\Users\aryan\source\repos\ X + v
Enter alarm time in HH:mm:ss format:
21:05:00
Alarm set for 21:05:00. Waiting...

??? Alarm! Time's up! ???
Press any key to exit.
```

Figure 2: As the time hits you get the message Alarm

```

Program.cs x
Lab_12_BP AlarmConsoleApp.Program
1 using System;
2 using System.Threading;
3
4 namespace AlarmConsoleApp
5 {
6     // Publisher class: raises the alarm event
7     class AlarmClock
8     {
9         private readonly string _targetTime; // "HH:mm:ss"
10        public event EventHandler? RaiseAlarm;
11
12        1 reference
13        public AlarmClock(string targetTime)
14        {
15            _targetTime = targetTime;
16        }
17
18        1 reference
19        public void Start()
20        {
21            while (true)
22            {
23                // Get current system time as HH:mm:ss
24                string now = DateTime.Now.ToString("HH:mm:ss");
25                if (now == _targetTime)
26                {
27                    OnRaiseAlarm();
28                    break;
29                }
30                Thread.Sleep(1000); // wait 1 second before checking again
31            }
32
33            1 reference
34            protected virtual void OnRaiseAlarm()
35            {
36                RaiseAlarm?.Invoke(this, EventArgs.Empty);
37            }
38        }
39
40        0 references
41        class Program
42        {
43            0 references
44            static void Main(string[] args)
45            {
46                Console.WriteLine("Enter alarm time in HH:mm:ss format:");
47                string? input = Console.ReadLine();
48
49                // Validate input
50                if (string.IsNullOrEmpty(input) || !TimeSpan.TryParse(input, out _))
51                {
52                    Console.WriteLine("Invalid time format. Please use HH:mm:ss (e.g. 14:30:00).");
53                    return;
54                }
55            }
56        }
57    }
58 }

```

Figure 3: Code screenshot part 1

```

37
38 0 references
39 class Program
40 {
41     0 references
42     static void Main(string[] args)
43     {
44         Console.WriteLine("Enter alarm time in HH:mm:ss format:");
45         string? input = Console.ReadLine();
46
47         // Validate input
48         if (string.IsNullOrEmpty(input) || !TimeSpan.TryParse(input, out _))
49         {
50             Console.WriteLine("Invalid time format. Please use HH:mm:ss (e.g. 14:30:00).");
51             return;
52         }
53
54         // Instantiate publisher and subscribe
55         var alarm = new AlarmClock(input);
56         alarm.RaiseAlarm += RingAlarm;
57
58         Console.WriteLine($"Alarm set for {input}. Waiting...");
59         alarm.Start();
60
61         // Give user a chance to see the message before console closes
62         Console.WriteLine("Press any key to exit.");
63         Console.ReadKey();
64     }
65
66     // Subscriber method
67     1 reference
68     private static void RingAlarm(object? sender, EventArgs e)
69     {
70         Console.WriteLine();
71         Console.WriteLine("🔔🔔🔔 Alarm! Time's up! 🔔🔔🔔");
72     }
73 }

```

Figure 4: Code screenshot part 2

3.2 Windows Forms Results

- Fully functional GUI with proper input validation
- Smooth background color transitions (1 second intervals)
- Correct alarm triggering with message box
- Disabled controls during active alarm monitoring

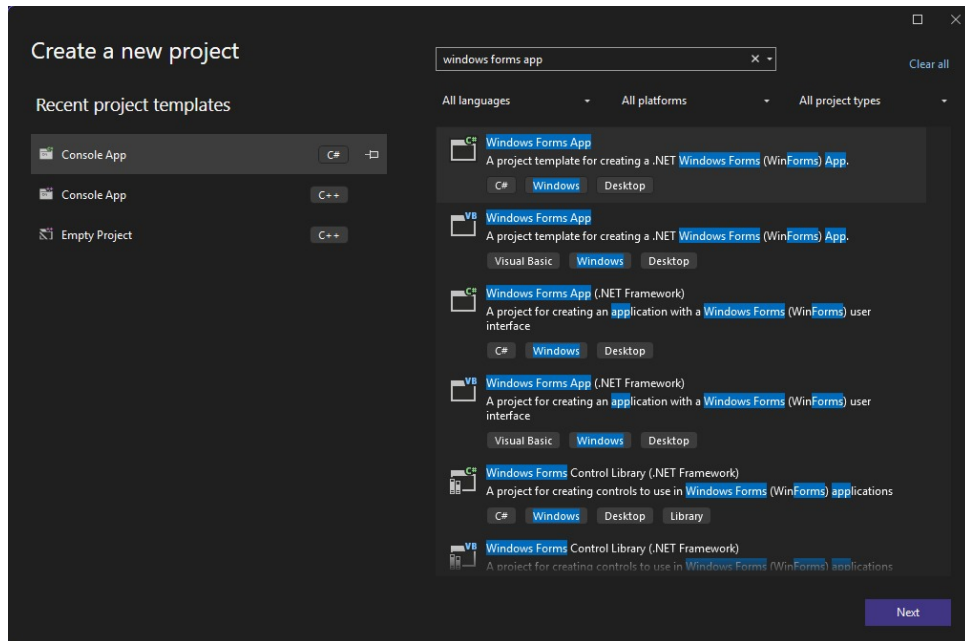


Figure 5: Creating a Windows Form

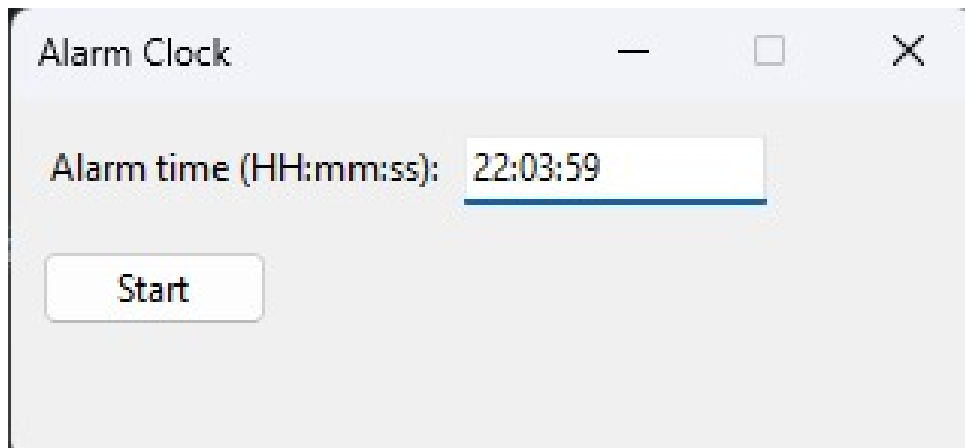


Figure 6: User can enter the target time

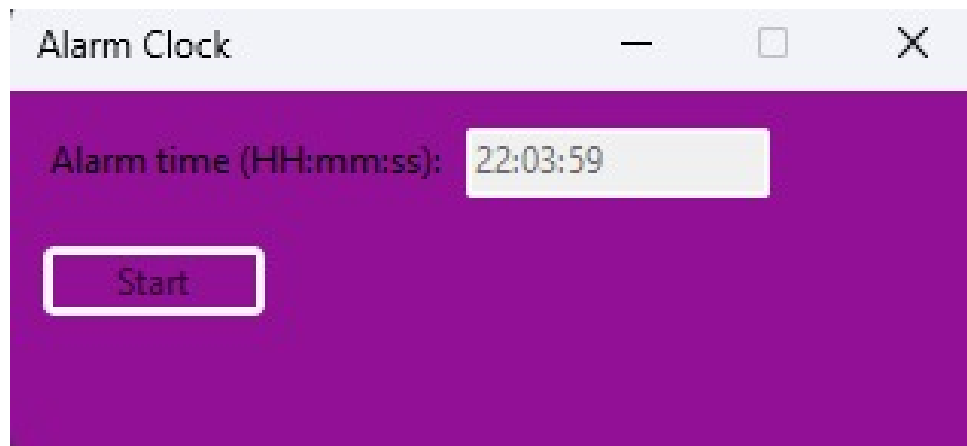


Figure 7: The color keeps changing till the target time is reached

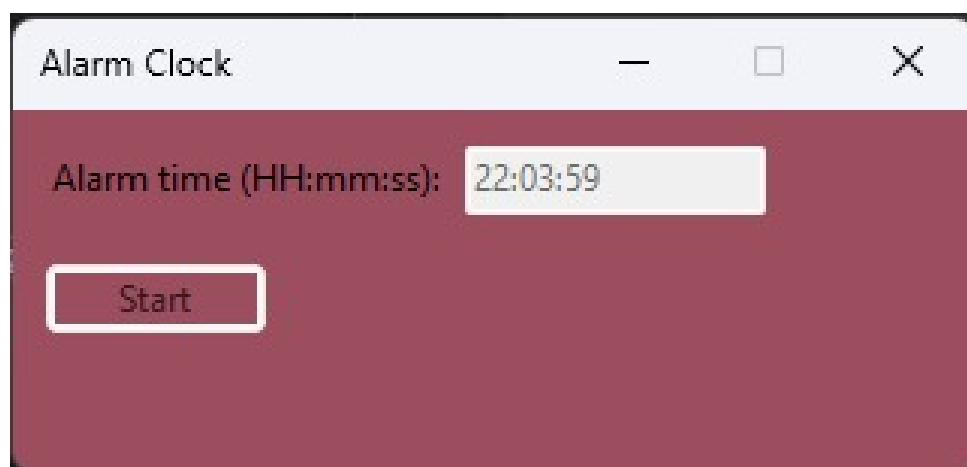


Figure 8: The color keeps changing till the target time is reached

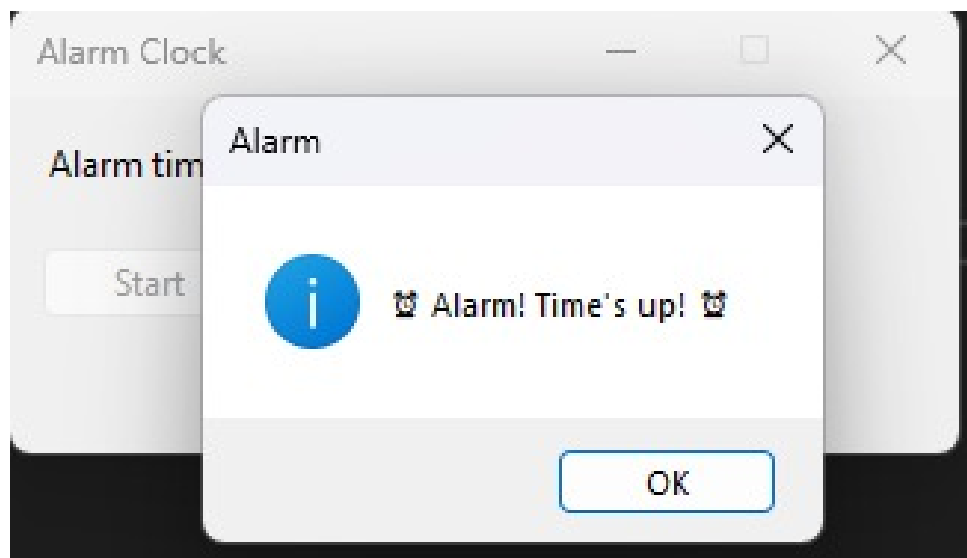


Figure 9: Alarm message pops up when the target time is reached

```

1  using System;
2  using System.Drawing;
3  using System.Windows.Forms;
4
5  namespace Lab_12_Form_BP
6  {
7      3 references
8      public partial class Form1 : Form
9      {
10         private Label labelPrompt;
11         private TextBox textBoxTime;
12         private Button buttonStart;
13         private System.Windows.Forms.Timer timer;
14         private TimeSpan targetTime;
15         private Random rng = new Random();
16
17         1 reference
18         public Form1()
19         {
20             InitializeComponent();
21         }
22
23         1 reference
24         private void InitializeComponent()
25         {
26             // --- Form settings ---
27             this.Text = "Alarm Clock";
28             this.ClientSize = new Size(320, 120);
29             this.StartPosition = FormStartPosition.CenterScreen;
30             this.FormBorderStyle = FormBorderStyle.FixedDialog;
31             this.MaximizeBox = false;
32
33             // --- Label ---
34             labelPrompt = new Label()
35             {
36                 Text = "Alarm time (HH:mm:ss):",
37                 Location = new Point(10, 15),
38                 AutoSize = true
39             };
40
41             // --- TextBox ---
42             textBoxTime = new TextBox()
43             {
44                 Location = new Point(labelPrompt.Right + 40, 12),
45                 Width = 100
46             };
47         }
48     }
49 }

```

Figure 10: Code screenshot

3.3 Performance Analysis

- Console version uses continuous polling (CPU intensive)
- Forms version uses event-driven timer (more efficient)
- Both versions handle edge cases (midnight rollover)
- Forms version provides better user experience

4 Discussion and Conclusion

4.1 Input Validation Implementation

4.1.1 Console Application Validation

The console version implements robust time format validation using `TimeSpan.TryParse`:

```

1  if (string.IsNullOrWhiteSpace(input) || !TimeSpan.TryParse(input, out _))
2  {
3      Console.WriteLine("Invalid time format. Please use HH:mm:ss (e.g. 14:30:00).");
4      return;
5  }

```

Listing 5: Console validation code

Key validation features:

- Checks for empty or whitespace input
- Verifies the input matches HH:mm:ss format
- Ensures time values are within valid ranges (hours 0-23, minutes 0-59, seconds 0-59)
- Provides clear error messaging to guide user correction

4.1.2 Windows Forms Validation

The GUI version enhances validation with `TimeSpan.TryParseExact` for stricter control:

```
1 if (!TimeSpan.TryParseExact(  
2     textBoxTime.Text.Trim(),  
3     "hh\\:\\:mm\\:\\:ss",  
4     null,  
5     out targetTime))  
6 {  
7     MessageBox.Show(  
8         "Please enter a valid time in HH:MM:SS format.",  
9         "Invalid Input",  
10        MessageBoxButtons.OK,  
11        MessageBoxIcon.Warning);  
12     return;  
13 }
```

Listing 6: Forms validation code

Additional validation improvements:

- Uses exact format matching with `TryParseExact`
- Provides modal dialog feedback for better UX
- Includes warning icon for visual emphasis
- Maintains input focus until valid time is entered

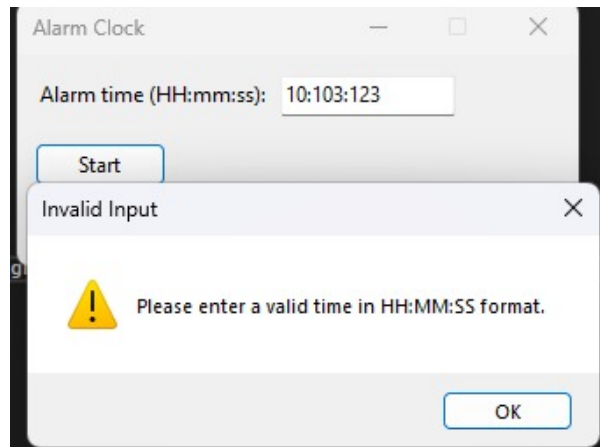


Figure 11: Validation message shown for invalid time format

The validation ensures:

- Only properly formatted times proceed to alarm setting
- Clear user feedback when input is rejected
- Prevention of logical errors from malformed input
- Consistent time handling across both application versions

4.2 Key Challenges

- Time format validation in both implementations
- Threading considerations in console version
- UI thread safety in Windows Forms
- Proper event handler cleanup

4.3 Lessons Learned

- Importance of proper event unsubscribe
- Benefits of `TimeSpan.TryParseExact` for validation
- UI updates must occur on main thread in Windows Forms
- Timer-based approaches are more efficient than polling

4.4 Conclusion

This assignment successfully demonstrated event-driven programming principles in both console and Windows Forms environments. The implementation highlighted the advantages of the publisher-subscriber pattern and proper event handling in .NET applications. The Windows Forms version particularly showcased how event-driven architecture enables responsive user interfaces with background processing.