Mute Scheduler Application - Beginner's Guide

Author: TCHEUFFA DARREN
Date: April 2025

This document explains how the Mute Scheduler application works in simple terms, focusing on the main components and how they interact.

Overview

The Mute Scheduler is a Windows Forms application that lets you automatically mute your computer at specific times. It's built using:

- **C#** - The programming language
- **Windows Forms** - For creating the user interface
- **NAudio** - A library that controls system audio

Project Structure

The application consists of these main files:

1. **Program.cs** - The entry point of the application
2. **MainForm.cs** - Contains all the UI and functionality
3. **MuteScheduler.csproj** - Project configuration file
4. **packages.config** - Lists the external libraries used

How the Code Works

1. Program.cs

This is a simple file that starts the application:

```
using System;
using System.Windows.Forms;

namespace MuteScheduler
{
    static class Program
    {
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new MainForm());
        }
    }
}
```

It does three things: - Enables visual styles for a modern look - Sets rendering defaults - Creates and runs the main form

2. MainForm.cs

This file contains all the application logic and is divided into several parts:

A. Class Variables

```
private List<ScheduleItem> scheduleItems = new List<ScheduleItem>();
private System.Timers.Timer scheduleTimer;
private NotifyIcon trayIcon;
private ContextMenuStrip trayMenu;
```

These store: - The list of mute schedules - A timer that checks if it's time to mute - The system tray icon - The right-click menu for the tray icon

B. User Interface Setup The application has three tabs: 1. **Day Specific** - For selecting individual days 2. **Weekdays** - For scheduling Monday-Friday at once 3. **Schedules** - For viewing and managing all schedules

Each tab contains controls like: - Checkboxes for selecting days - Time pickers for setting mute times - Numeric inputs for duration - Dropdown menus for frequency (one-time or recurring)

C. Schedule Management When you add a schedule, the code: 1. Collects the selected days, time, and options 2. Creates a `ScheduleItem` object to store this information 3. Adds it to the list of schedules 4. Updates the schedule list view

```
private void AddDaySpecificSchedule(FlowLayoutPanel daysPanel, DateTimePicker timePicker,
                                    NumericUpDown durationPicker, ComboBox frequencyComboBox)
{
    List<DayOfWeek> selectedDays = new List<DayOfWeek>();

    // Get selected days from checkboxes
    foreach (Control control in daysPanel.Controls)
    {
        if (control is CheckBox checkBox && checkBox.Checked)
        {
            // Add the day to the list
        }
    }

    // Create and add the schedule
    ScheduleItem item = new ScheduleItem
    {
```

```
        Days = selectedDays,
        Time = timePicker.Value,
        Duration = (double)durationPicker.Value,
        IsRecurring = frequencyComboBox.SelectedIndex == 1
    };

    scheduleItems.Add(item);
    UpdateScheduleListView();
}
```

D. Timer and Mute Functionality The application uses a timer that checks every minute if it's time to mute:

```
private void InitializeScheduleTimer()
{
    scheduleTimer = new System.Timers.Timer(60000); // Check every minute
    scheduleTimer.Elapsed += ScheduleTimer_Elapsed;
    scheduleTimer.AutoReset = true;
    scheduleTimer.Start();
}
```

When the timer triggers, it checks all schedules:

```
private void CheckSchedules()
{
    DateTime now = DateTime.Now;

    foreach (ScheduleItem item in scheduleItems.ToList())
    {
        // If today is in the scheduled days and it's the right time
        if (item.Days.Contains(now.DayOfWeek) &&
            now.Hour == item.Time.Hour && now.Minute == item.Time.Minute)
        {
            MuteSystem();

            // Schedule unmute if duration is set
            if (item.Duration > 0)
            {
                // Unmute after the specified duration
            }

            // Remove if one-time schedule
            if (!item.IsRecurring)
            {
                scheduleItems.Remove(item);
                UpdateScheduleListView();
            }
```

```
        }
    }
}
```

E. NAudio Integration The application uses NAudio to control the system volume:

```
private void MuteSystem()
{
    try
    {
        MMDeviceEnumerator deviceEnumerator = new MMDeviceEnumerator();
        MMDevice device = deviceEnumerator.GetDefaultAudioEndpoint(DataFlow.Render, Role.Mul
        device.AudioEndpointVolume.Mute = true;

        // Show notification
        if (trayIcon != null)
        {
            trayIcon.ShowBalloonTip(3000, "Mute Scheduler",
                "System has been muted according to schedule.", ToolTipIcon.Info);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Error muting system: {ex.Message}", "Error",
                        MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

This code: 1. Creates a device enumerator to find audio devices 2. Gets the default audio output device 3. Sets its mute property to true 4. Shows a notification

F. System Tray Integration The application minimizes to the system tray:

```
private void InitializeTrayIcon()
{
    trayMenu = new ContextMenuStrip();

    // Add menu items (Open, Exit)

    trayIcon = new NotifyIcon
    {
        Text = "Mute Scheduler",
        Icon = SystemIcons.Application,
        ContextMenuStrip = trayMenu,
        Visible = true
```

```
    };

    // Double-click to open the app
    trayIcon.DoubleClick += (sender, e) =>
    {
        this.Show();
        this.WindowState = FormWindowState.Normal;
    };
}
```

When you close the window, it minimizes instead of exiting:

```
private void MainForm_FormClosing(object sender, FormClosingEventArgs e)
{
    if (e.CloseReason == CloseReason.UserClosing)
    {
        e.Cancel = true;
        this.Hide();
        trayIcon.ShowBalloonTip(3000, "Mute Scheduler",
            "Application is still running in the system tray.", ToolTipIcon.Info);
    }
    else
    {
        trayIcon.Dispose();
    }
}
```

3. ScheduleItem Class

This simple class stores information about each mute schedule:

```
public class ScheduleItem
{
    public List<DayOfWeek> Days { get; set; }
    public DateTime Time { get; set; }
    public double Duration { get; set; }
    public bool IsRecurring { get; set; }

    public string GetDaysString()
    {
        // Convert the list of days to a readable string
    }
}
```

External Libraries Used

NAudio (2.2.1)

NAudio is the main external library used in this application. It provides access to Windows audio functionality.

Key components used: - **MMDeviceEnumerator** - Finds audio devices - **MMDevice** - Represents an audio device - **AudioEndpointVolume** - Controls volume and mute state

.NET Framework (4.7.2)

The application uses standard .NET Framework libraries: - **System.Windows.Forms** - For the user interface - **System.Timers** - For scheduling checks - **System.Threading.Tasks** - For delayed unmute operations

How to Modify the Code

Adding a New Feature

To add a new feature (for example, volume control instead of just mute):

1. Find the appropriate section in MainForm.cs
2. Add new UI controls if needed
3. Create methods to handle the new functionality
4. Connect the UI to your new methods

Changing the UI

To modify the user interface:

1. Look for the `InitializeComponent()` method and the related setup methods
2. Modify the controls, their properties, or layout
3. Update any event handlers if needed

Common Questions

How does the application know when to mute?

The application uses a timer that checks every minute if any schedule matches the current time and day. If there's a match, it calls the `MuteSystem()` method.

How does the mute function work?

The application uses NAudio to access the Windows Core Audio API. It gets the default audio device and sets its mute property to true or false.

Can I run this application on startup?

Yes, you can add a shortcut to the application in the Windows Startup folder: 1. Press Win+R 2. Type `shell:startup` 3. Create a shortcut to the application in this folder

Conclusion

The Mute Scheduler application demonstrates how to: - Create a Windows Forms user interface - Work with timers for scheduling - Control system audio using NAudio - Implement system tray functionality

The code is organized in a way that separates different concerns, making it easier to understand and modify.