**Итоговая работа**

**МДК.02.02 Инструментальные средства разработки программного обеспечения**

**Цель**: инспекция кода модулей проекта, документирование результатов тестирования.

**Задачи**:

1. Составить техническое задание (шаблон прилагается) для программного проекта **Будильник**
2. Реализовать программный проект **Будильник** поэтапно (исходя из теоретических сведений, его необходимо доработать)
3. Составить репортинг Ручного тестирования в виде таблицы.

**Ход работы:**

*Задача 1.*

*Программная документация, включает:*

1. техническое задание (назначение, область применения программы, требования, предъявляемые к программе);

2. текст программы (запись программы с необходимыми комментариями);

3. описание программы (сведения о логической структуре и функционировании программы);

4. пояснительная записка (схема алгоритма, общее описание алгоритма и/или функционирования программы, обоснование принятых решений);

5. эксплуатационные документы.

*К эксплуатационным документам относят:*

* описание применения (сведения о назначении программы, области применения, применяемых методах, классе решаемых задач, ограничениях для применения, минимальной конфигурации технических средств);
* руководство системного программиста (сведения для проверки, обеспечения функционирования и настройки программы на условия конкретного применения);
* руководство программиста (сведения для эксплуатации программы);
* руководство оператора (сведения для обеспечения общения оператора с вычислительной системой в процессе выполнения программы);
* описание языка (описание синтаксиса и семантики языка);
* руководство по техническому обслуживанию (сведения для применения тестовых и диагностических программ при обслуживании технических средств)

Основная часть программной документации составляется на стадии рабочего проекта. Необходимость того или иного документа определяется на этапе составления технического задания. Допускается объединять отдельные виды документов.

Эксплуатационный документ "Описание языка" включается в программную документацию, если разработанный программный продукт реализует некий язык программирования, управления заданиями, организации вычислительного процесса и т. п.

Эксплуатационный документ "Руководство по техническому обслуживанию" включается в программную документацию, если разработанный программный продукт требует использования тестовых или диагностических программ.

*Описание применения*

Документ "Описание применения" относится к эксплуатационным документам и состоит из следующих разделов:

* назначение программы (возможности, основные характеристики, ограничения области применения);
* условия применения (требования к техническим и программным средствам, общие характеристики входной и выходной информации, а также требования и условия организационного, технического и технологического характера);
* описание задачи (указываются определения задачи и методы её решения);
* входные и выходные данные.

*Руководство программиста*

Документ "Руководство программиста" относится к эксплуатационным

документам и включается в программную документацию, если разработанный программный продукт требует обслуживания программистом. Документ состоит из следующих разделов:

* назначение и условия применения программы (назначение и функции программы, сведения о технических и программных средствах, обеспечивающих выполнение данной программы);
* характеристики программы (временные характеристики, режимы работы, средства контроля правильности выполнения и т. п.);
* обращение к программе (способы передачи управления и параметров
* данных);
* входные и выходные данные (формат и кодирование);
* сообщения (тексты сообщений, выдаваемых программисту или оператору в ходе выполнения программы и описание действий, которые необходимо предпринять по этим сообщениям).

*Руководство оператора*

Документ "Руководство оператора" относится к эксплуатационным документам и состоит из следующих разделов:

* назначение программы (информация, достаточная для понимания функций программы и её эксплуатации);
* условия выполнения программы (минимальный и/или максимальный набор технических и программных средств и т. п.);
* выполнение программы (последовательность действий оператора, обеспечивающих загрузку, запуск, выполнение и завершение программы;
* описываются функции, форматы и возможные варианты команд, с помощью которых оператор осуществляет загрузку и управляет выполнением программы, а также ответы программы на эти команды);
* сообщения оператору (тексты сообщений, выдаваемых оператору в ходе выполнения программы и описание действий, которые необходимо предпринять по этим сообщениям).

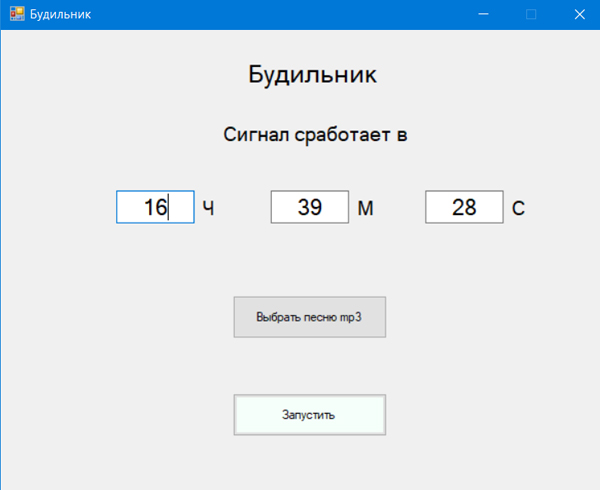
*Задача 2.*

***Разработка программы «Будильник»***

Что она будет уметь делать? Будет три текстовых поля. Там пользователь должен написать, во сколько наша программа должна подать сигнал (часы, минуты, секунды).

Пользователю нужно указать файл **mp3**, который должен зазвучать в час X.

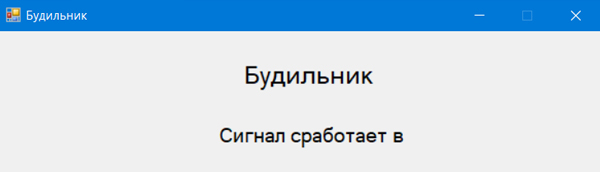
После этого нужно нажать на кнопку "Запустить", чтобы активировать будильник. Вот что мы должны получить в итоге:



Открываем **Visual Studio**. Создаём новый проект "Приложение Windows Forms (.NET Framework)". Зададим свойства для самой формы. Свойство **Text**: "Будильник", это название самой формы, которое мы видим в верху. Свойство **Size**: 800, 600. Это размеры нашего окна. Свойство **StartPosition**: CenterScreen, чтобы программа запускалась по центру экрана. Свойство **MaximizeBox**: Fasle (запретим разворачивать окно на весь экран). Свойство **FormBorderStyle**: FixedSingle (запрещаем менять размеры формы).

Нажмём на панель инструментов. Выберем компонент **Label** и перенесём на форму. Изменим свойства - **Text**: "Будильник", **Font**: размер на 18pt. После этого можно узнать размер этого компонента lable. Посмотрим свойство Size. Ширина получилась равна 175. Рассчитаем, чтобы вывеска программы расположилась посередине: (800 - 175) / 2 = 313. Т.е. с левого края отложим 313 пикселей. Установим свойство **Location**: 313, 35.

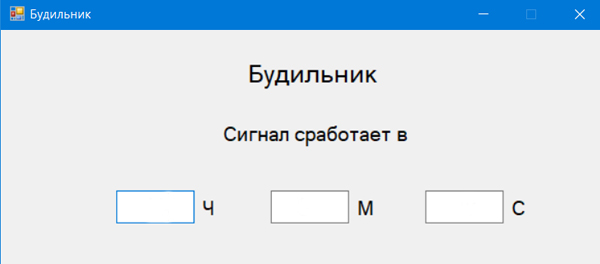
Аналогично перенесём ещё один компонент **Label**. И установим свойства: **Text**: "Сигнал сработает в", **Font**: размер на 14 pt, **Location:** 282, 101.



Перенеcём три элемента **TextBox**. Все элементы будут иметь одинаковые свойства, кроме **Location**. **Font:** шрифт в 16 pt, свойство - **Size**: 100, 38. **Location**: для textBox1 150; 192, для textBox2 350; 192, для textBox3 550; 192. Таким образом, элементы равномерно разместились в нашем окне. Данные для локации всех элементов легко вычисляются, т.к. мы знаем их размеры и знаем размер окна.

Так же к каждому TextBox справа прикрепляем элементы Label. Пишем в каждом Label'e **Ч, М, С** соответственно.

Свойство **Font** у Label'ов: размер устанавливаем в 14pt. Совойство **Text** устанавливаем соответственно в Ч, М, С. Расположение можно установить вручную на одинаковой высоте и на одинковом расстоянии от TextBox'ов.



Добавим на форму две кнопки (два элемента **Button**).

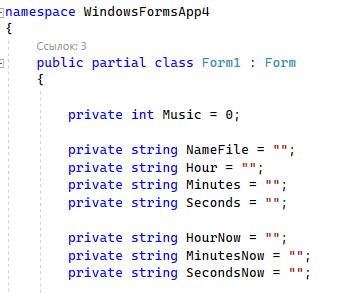
Для первой кнопки название (свойство **Text**): "Выбрать песню mp3". Размеры (**Size**): 200, 52. Расположение (**Location**): 300, 317.

Для второй кнопки. Название (свойство **Text**): "Запустить". Размеры (**Size**): 200, 52. Расположение (**Location**): 300, 434. Так же устанавливаем свойство **BackColor**: MintCream.

Приступаем к программированию. Нажимаем **F7** и переходим к коду.

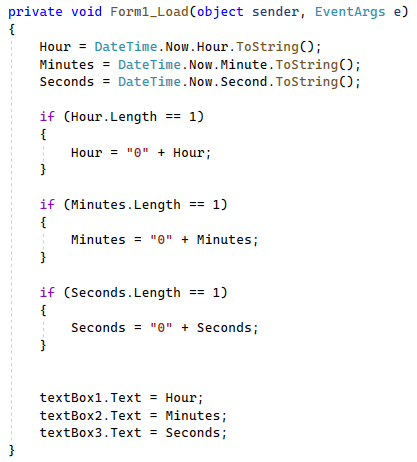
В самом начале класса Form заводим следующие переменные.

|  |  |  |  |
| --- | --- | --- | --- |
| Тип данных | Навзание | Начальное значение | Описание |
| int | Music | 0 | Если звучит музыка будильника в данный момент времени, то переменная Music должна равняться 1, иначе 0. Это переменная флаг. |
| string | NameFile | "" | Здесь будет храниться путь до файла mp3. |
| string | Hour | "" | В эту переменную будем класть значение из textBox1, чтобы узнать часы, которые установил пользователь. |
| string | Minutes | "" | В эту переменную будем класть значение из textBox2, чтобы узнать минуты, которые установил пользователь. |
| string | Seconds | "" | В эту переменную будем класть значение из textBox3, чтобы узнать секунды, которые установил пользователь. |
| string | HourNow | "" | В эту переменную будем класть значение часов из системы Windows, чтобы сравнить с тем значением, которое ввёл пользователь. Если часы, минуты, секунды совпадут, начнёт воспроизводится сигнал. |
| string | MinutesNow | "" | В эту переменную будем класть значение минут из системы Windows, чтобы сравнить с тем значением, которое ввёл пользователь. Если часы, минуты, секунды совпадут, начнёт воспроизводится сигнал. |
| string | SecondsNow | "" | В эту переменную будем класть значение секунд из системы Windows, чтобы сравнить с тем значением, которое ввёл пользователь. Если часы, минуты, секунды совпадут, начнёт воспроизводится сигнал. |



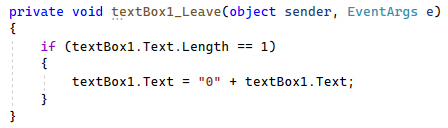
Все переменные делаем закрытыми **private**.

Нужно, чтобы при запуске в TextBox'ах появлялось текущее время. Выделяем форму. В окне свойств переходим на значок событий (иконка в виде молнии). Ищем событие **Load** (загрузка формы). Кликаем два раза на ячейку справа от этого события. Visual Studio автоматически создаст функцию обработчика события загрузки формы в коде. Там пропишем код:



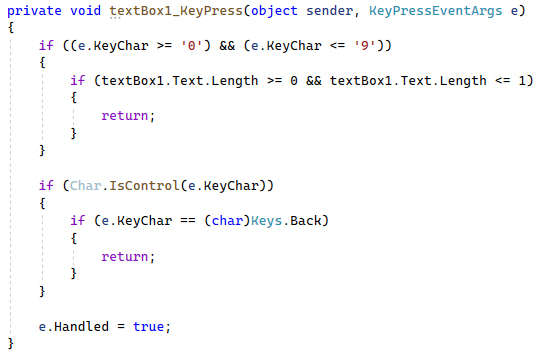
В первых трёх строчках получаем системное время. Следующие три условия приписывают ноль, если число состоит из одной цифры. Вместо, к примеру, числа 3, нам хотелось бы видеть 03 в этих полях. Последние три строчки уже записывают результат в textBox'ы. Можете запустить и проверить, как всё получилось.

После редактирования пользователем полей, тоже нужно подписывать нули к одиночным цифрам. Для каждого текстового поля пропишем событие потери фокуса (**Leave**).



Так делаем для всех полей.

Так же пропишем для элементов **TextBox** событие **KeyPress** (нажатие клавиши т.е. изменение значения Textbox):



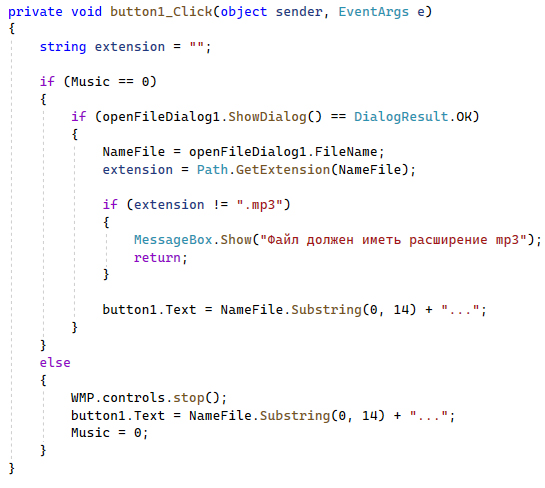
В конце функции прописано **e.Handled = true**. Если мы дойдём до этой строчки, то нажатие клавиши заблокируется, т.е. при нажатии клавиши в этом поле ничего не произойдёт. Если пользователь ввёл цифры и при этом количество этих цифр в TextBox'е равно 0 или 1, то мы прописываем **return** (выход из функции обработчика события), тем самым, избежим блокировки нажатия. Другими словами, мы ждём от пользователя только цифры, и количество этих цифр не может превышать двух в любом из TextBox'ов.

Так же мы разрешаем нажимать клавишу Backspace, за это отвечает второе условие. Если клавиши "легальны", то выходим из функции и блокировка события не происходит, в остальных случаях нажатие клавиши будет проигнорировано.

Тоже самое сделаем и для остальных TextBox'ов. Только заменим в пободных функциях имена на textBox2 и textBox3 соответственно.

Напишем обраточик события нажатия кнопки (**Click**) для button1, т.е. той кнопки, которая отвечает за выбор песни. При нажатии на неё должен появится файловый диалог. Поэтому добавим на форму ещё один элемент **OpenFileDialog**.

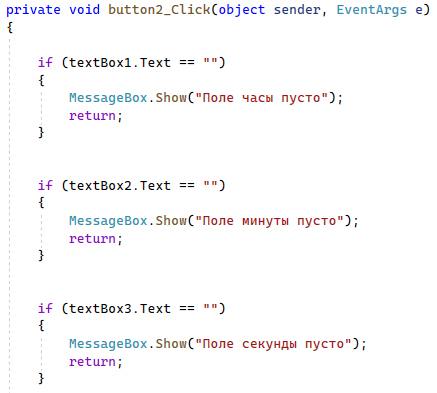
Переходим в обработчик события **Click** и пропишем:



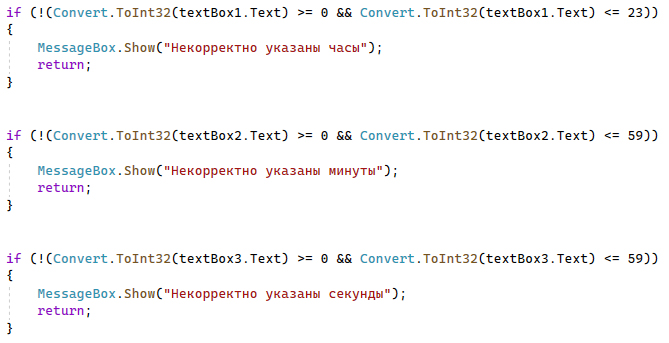
Переменная **extension** будет хранить в себе расширение выбранного файла. Если сейчас не играет музыка, тогда открываем файловый диалог, и после того, как файл выбран, помещаем путь к файлу в переменную **NameFile**. Узнаём расширение выбранного файла, если он не mp3, выводим сообщение об ошибке и выходим из функции. Если с расширением всё нормально, меняем надпись на кнопке на путь к песне, укороченного до 14 символов. В конце к названию кнопки добавляется троеточие.

Если же песня играет, то кнопка **button1**, превращается в кнопку **Выключить музыку**. Мы об этой ветке **else** поговорим чуть попозже. Пока эту ветку не пишите.

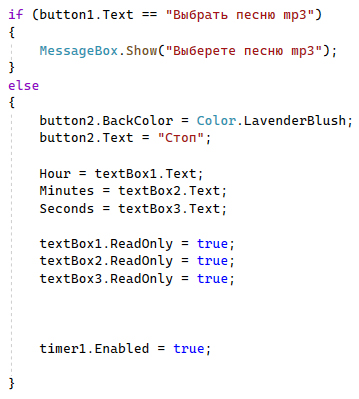
Сделаем обработчик события нажатия для второй кнопки **button2** (Запустить).



Здесь всё понятно, если какое-то поле осталось пустым, сообщить пользователю и выйти из функции. Добавим ещё в **эту же функцию**.

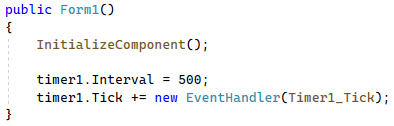


Часы не должны превышать значение 23. Минуты и секунды не должны превышать значение 59. Добавим самое главное условие.

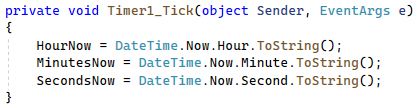


Если файла mp3 нет, то и кнопка "Запустить" не будет работать. Иначе (в названии button1 находится путь до файла), у первой кнопки меняем название на "Стоп". Помещаем в переменные **Hours**, **Minutes**, **Seconds** показание часов, которое ввел пользователь. Свойство **ReadOnly** блокирует дальнейшее редактирование полей.

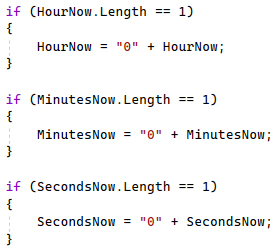
Запускается таймер **timer1**. Его нужно с панели элементов так же добавить на форму. Таймер должен запускать определённую функцию каждые пол секунды, где и будет сравниваться время введённое пользователем и системное время. Т.к. запускать сравнение будем достаточно часто, мы точно не "прозеваем" время совпадения. Пропишем настройки для таймера в функции **Form1**.



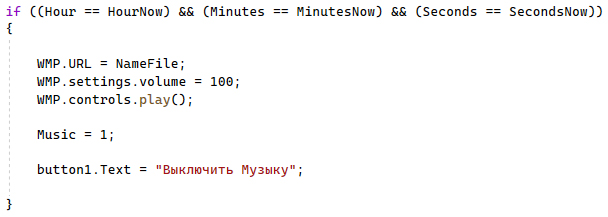
Функция **Timer1\_Tick** должна запускаться каждые 500 милисекунд. Эту функцию тоже пропишем внутри **класса Form**.



Здесь уже снимаем текущее системное время. Время, в которое будильник должен зазвонить находится в переменных Hours, Minutes, Seconds. Мы здесь к переменным HoursNow, MinutesNow, SecondsNow тоже должны приписвать ноль к одиночным цифрам, как это делали в функции **Form1\_Load**. Продолжаем прописывать в функции **Timer1\_Tick**:



И в эту же функцию пропишем самое главное условие, ради чего всё и делалось. Сравнение значений системного времени и значений от пользователя.

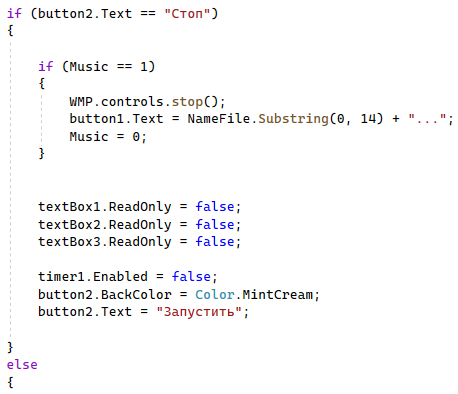


Если значения одинаковые, то воспроизводим файл mp3. Воспроизведения файла mp3 осуществляется с помощью библиотеки **Windows Media Player (WMP)**. Её нужно подключить к проекту. Нажимаем **Проект -> Добавить ссылку -> COM**. В самом низу ищем Windows Media Player, при наведении курсора должна всплыть подсказка, что это файл **wmp.dll**. Добавляем данную библиотеку в наш проект. Так же в том месте, где создавали все переменные, создадим объект WMP, который мы использовали ранее.

Программа будильник на C# - создаём объект WMP

Изменяем название для первой кнопки, где раньше был путь к файлу. Теперь с помощью неё можно отключить музыку. Вспоминаем про ветку else в функции button1\_Click и пропишем её. Теперь-то переменная **Music** установлена в 1, если звучит песня, и мы долны остановить воспроизведение по нажатию **button1**.

После нажатия по второй кнопке, когда мы активировали наш будильник, её название заменяется на "Стоп". Пользователь имет право в любой момент деактивировать будильник. Тогда вопроизведение не произойдёт. Напишем код в функции **button2\_Click**, для того случая, когда вторая кнопка превратилась в кнопку "Стоп".



А вот что мы до этого написали в этой функции, перемещаем в ветку else. Здесь отключаем таймер, даём возможность снова редактировать поля. Так же эта кнопка останавливает воспроизведение музыки, если она была нажата в момент, когда звучала музыка.

*Задача 3*

Провести ручное тестирование и зафиксировать результаты

|  |  |  |  |
| --- | --- | --- | --- |
| **Действие** | **Входные данные** | **Ожидаемый результат** | **Фактический результат** |
| *пр. ввод числа в поля Ч М С* | *11 20 00* | *Будильник воспроизведется в 11 20 00* | *Будильник воспроизвелся в 11 20 00* |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |