Оглавление

[Введение 2](#_Toc405273183)

[Описание предметной области 2](#_Toc405273184)

[Неформальная постановка задачи 3](#_Toc405273185)

[Обзор существующих методов решения 5](#_Toc405273186)

[План робот 7](#_Toc405273187)

[Требования к окружению 7](#_Toc405273188)

[Требования к аппаратному обеспечению 7](#_Toc405273189)

[Требования к програмному обеспечению 8](#_Toc405273190)

[Требования к пользователю 8](#_Toc405273191)

[Организационные требования 8](#_Toc405273192)

[Архитектура системы 8](#_Toc405273193)

[Варианты использования 10](#_Toc405273194)

[Проект программной системы 11](#_Toc405273195)

[Средства реализации 11](#_Toc405273196)

[Проект Интерфейса 11](#_Toc405273197)

[Вывод 15](#_Toc405273198)

## Введение

В нынешний век высоких технологий, наука и техника проникли практически во все аспекты современной жизни. В связи с быстрым темпом жизни в современном обществе стоит проблема распределением человеком времени для повышения своей результативности. Чтобы решить эту проблему нужно грамотно распределять свое время.

Для контроля и грамотного распределения времени лучше всего помогают часы-будильник, так как проблема нехватки времени касается всех без исключения начиная от студентов заканчивая всевозможными директорами банков и других крупных предприятий, актуальность разработки программного обеспечения будильника стоит остро и не вызывает сомнений.

## Описание предметной области

Практическая ценность. Создание программы часы-будильник решает следующие задачи:

* Помогает встать по утрам не пропустив работу/учебу
* Напоминает о важных, запланированных на сегодня, мероприятиях
* Составляет четкий распорядок дня

В данной курсовой работе представлен пример объектно-ориентированной разработки программного обеспечения на основе проектирования в case-средстве Rational Rose, используя ее возможности для построения UML-диаграмм, анализа, моделирования и преобразования моделей и генерации кода.

Исходными данными для выполнения курсовой работы является постановка задачи, приведенная в пункте 2, результатом являются диаграммы вариантов использования, классов, состояний, деятельности, последовательностей, коопераций, компонентов и размещения, а так же сгенерированный код.

Функциональность объекта изначально ограничена, но полученные материалы могут быть использованы для программирования реально существующего объекта, то есть, данный способ разработки имеет практическую ценность.

Для достижения поставленной задачи требуется решить ряд конкретизирующих её задач:

* Выявить основные тенденции развития и особенности объектно-ориентированного программирования и проектирования;
* Выделить основные функции системы, объекты, взаимодействующие с ними, и реакции системы на действия этих объектов;
* Проанализировать основные компоненты системы и составить их подробное описание
* На основе полученных данных осуществить проектирование системы при помощи диаграмм на языке UML.
* Составить инструкцию пользователя для того, чтобы обеспечить легкость внедрения её в структуру предприятия и упростить обучение сотрудников работе в данной программе.

В качестве объекта исследования выбраны часы-будильник – в данной работе – это готовая автоматизированная информационная система предназначенная для помощи распределения времени.

Неформальная постановка задачи

Требуется разработать модель программного обеспечения встроенного микропроцессора для будильника.

Будильник постоянно отображает текущее время (часы, минуты). Управление будильником осуществляется следующими кнопками:

1) кнопкой режима установки времени, 2) кнопкой режима установки времени срабатывания, 3) двумя отдельными кнопками для установки часов и минут, 4) кнопкой сброса сигнала «СБРОС».

На будильнике имеется переключатель режима работы со следующими положениями: «ВЫКЛ», «ВКЛ», «РАДИО» и «ТАЙМЕР». Для установки текущего времени нужно нажать на кнопку режима установки и, при нажатой кнопке, нажимать на кнопки установки часов и минут.

При каждом нажатии на кнопки, устанавливаемое значение увеличивается на одну единицу (один час или одну минуту соответственно). При достижении максимального значения производится сброс.

Для установки времени срабатывания будильника нужно нажать на кнопку режима установки времени срабатывания и, держа кнопку нажатой, нажимать на кнопки установки часов и минут.

Когда переключатель режима работы находится в положении «ВКЛ», при достижении времени срабатывания происходит подача звукового сигнала в течение одной минуты.

Сигнал можно прервать, нажав на кнопку «СБРОС». При этом сигнал должен быть возобновлен через пять минут. При установке переключателя в положение «ВЫКЛ» звуковой сигнал не подается.

Когда переключатель находится в положении «РАДИО» работает радиоприемник.

При переводе переключателя в положение «ТАЙМЕР» включается радиоприемник на тридцать минут, а затем часы переходят в состояние будильника (аналогично положению «ВКЛ»).

При нажатии на кнопку режима установки времени, будильник должен отображать время срабатывания.

Будильник – это программа позволяющая создать задачи и вызов звукового сигнала, если прошло определенное время.

В проектируемой системе, также предусмотрено использование Радиоприемника, для прослушивания радиоволн, в режиме ожидания, или во время отсчета таймера.

Иногда будильник сможет заменять Таймер, который устанавливается на 30 минут, и на протяжении этого времени играет музыка с Радиоприемника

## Обзор существующих методов решения

На сегодняшний день существует множество всевозможных будильников.

Вот некоторые из них :



**Рисунок 1 - ASSISTANT AH-1060 RED**



**Рисунок 2 - ASSISTANT AH-1063 GREEN**



**Рисунок 3- LAMARK LK-2101 BK**



**Рисунок 4 - LAMARK LK-2101**

## 

## План робот

## Требования к окружению

Требования к аппаратному обеспечению

Минимальные требования к аппаратному обеспечению:

|  |
| --- |
| 1.Процессор з тактовой частотой не меньше 1 ГГц. |
| 2.Оперативна память объемом 128 Мб. |
| 3.Жесткий диск с доступными 200 Мб памяти |
| 4.Монитор с разрешением 1024х768. |

## Требования к програмному обеспечению

Windows система, начиная с Windows XP до Windows 8.

Платформа .NET Framework 4.0

## Требования к пользователю

Пользователь должен пройти базовый курс использования системы Windows, в другом случае, программа не требует особенных навыков работы с Windows.

## Организационные требования

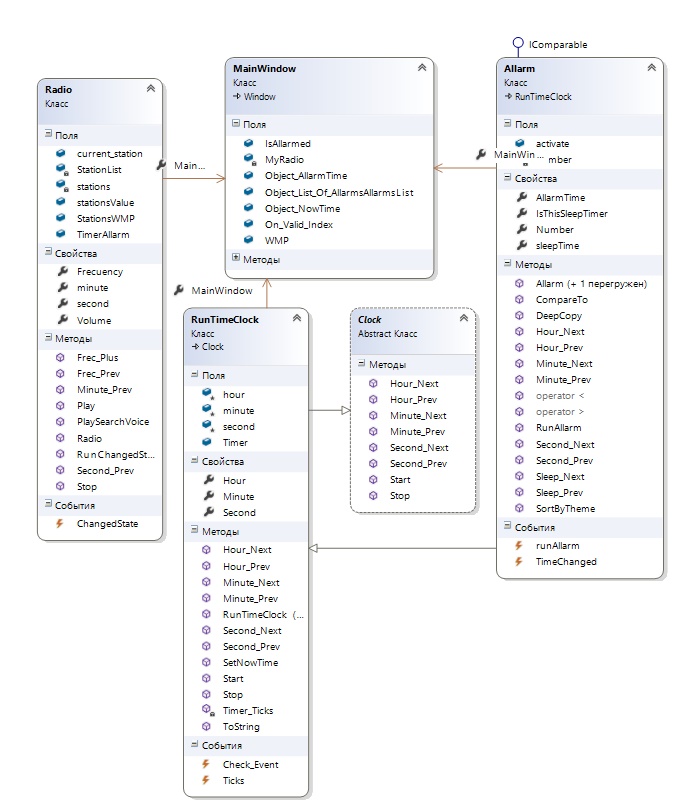
Для использования программы, необходим человек, с базовыми познаниями Windows, и опытом использования ПК.

## **Архитектура системы**

Анализируя поставленную задачу было решено определить такие классы:

* Clock; - Базовый, абстрактный класс для времени
* MainWindow; - Класс отображает окна программы
* RunTimeClock; - Класс работает с текущим временем
* Alllarm; - Класс работает с временем будильника
* Radio; - Класс отвечающий за работу радио

Отримуємо таку діаграму класів, де всі класи пов’язані між собою відношенням асоціаціі:



**Рисунок 5 – Диаграмма классов**

## Варианты использования

Модель вариантов использования представляет собой модель взаимодействия пользователей для решения проблем и задач.

Модель вариантов использования описывает цели пользователей, взаимодействие между пользователями и системой и требуемое поведение системы для удовлетворения этих целей.

Модель вариантов использования состоит из ряда модельных элементов (или графических примитивов). К наиболее важным из них относятся варианты использования [use cases], действующие лица или акторы [actors] и отношения (связи) между ними [relationships].

Для того, чтобы описать то, как может использоваться данная проектируемая система, необходимо выявить действующие лица – то есть, все объекты, которые будут взаимодействовать с системой. В данном случае, выделяется один объект и рассматривается его поведение в системе.

Действующее лицо:

Пользователь – Любой человек, пользующееся программным обеспечением будильника.

Варианты использования:

Исходя из потребностей пользователя, выделяются следующие варианты использования:

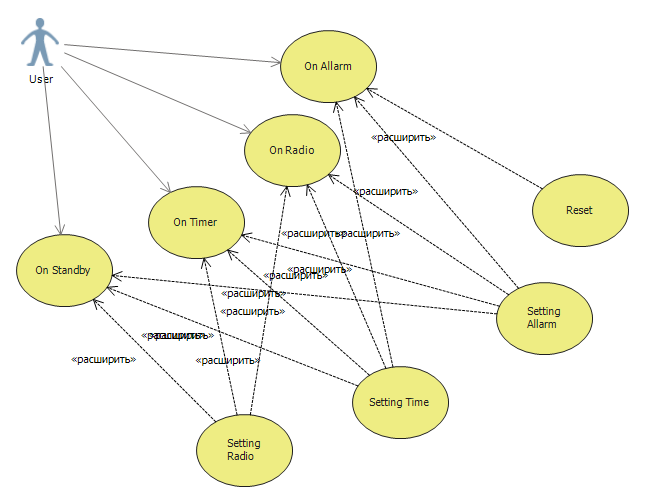


Рисунок 6 - Диаграмма вариантов использования

## Проект программной системы

## Средства реализации

Модель программного обеспечения была реализована посредством языка С#

Среда разработки: Microsoft Visual Studio 2013.

## Проект Интерфейса

Интерфейс программы был выполнен в черно-зеленой цветовой схеме- аналог монохромного дисплея в настольных часах (електрических).

***\Главное окно:***



***Установка текущего времени:***



***Установка времени будильника:***



***Установка Радио:***



***Состояние Будильника:***



***Состояние Радио:***



***Состояние Таймера:***



## Вывод

В ходе выполнения курсовой работы , я достиг некоторых результатов , а именно:

* Усовершенствовал свои навыки работы с WPF
* Применил свои знания из области ООП
* Научился строить диаграммы классов
* Научился строит диаграммы вариантов использования программы Use Case
* Усвоил новые методы проектирования и разработки программ, по принципу ООП.

**Код Программы:**

**Абстрактный класс «Часы»**

public abstract class Clock

{

public virtual void Hour\_Next() { }

public virtual void Minute\_Next() { }

public virtual void Second\_Next() { }

public virtual void Start() { }

public virtual void Stop() { }

public virtual void Hour\_Prev() { }

public virtual void Minute\_Prev() { }

public virtual void Second\_Prev() { }

}

**Класс «Текущее время»**

public class RunTimeClock : Clock

{

protected int hour;

protected int minute;

protected int second;

public System.Windows.Threading.DispatcherTimer Timer;

public int Hour

{

get

{

return hour;

}

}

public int Minute

{

get

{

return minute;

}

}

public int Second

{

get

{

return second;

}

}

public override void Hour\_Next()

{

if (hour == 23)

hour = 0;

else hour++;

}

public override void Minute\_Next()

{

if (minute == 59)

{

minute = 0;

Hour\_Next();

}

else

minute++;

if (Check\_Event != null) // Вызываем событие

Check\_Event(this, new EventArgs());

}

public override void Second\_Next()

{

if (second == 59)

{

second = 0;

Minute\_Next();

}

else

second++;

}

public override void Hour\_Prev()

{

if (hour == 0)

hour = 23;

else hour--;

}

public override void Minute\_Prev()

{

if (minute == 0)

{

minute = 59;

Hour\_Prev();

}

else

minute--;

}

public override void Second\_Prev()

{

if (second == 0)

{

second = 59;

Minute\_Prev();

}

else

second--;

}

public RunTimeClock()

{

Timer = new System.Windows.Threading.DispatcherTimer();

SetNowTime();

}

public RunTimeClock(int Hours, int Minutes, int Seconds)

{

hour = Hours;

minute = Minutes;

second = Seconds;

}

public void SetNowTime()

{

DateTime Now = DateTime.Now;

hour = Now.Hour;

minute = Now.Minute;

second = Now.Second;

}

public override void Start()

{

Timer.Tick+=new EventHandler(Timer\_Ticks);

Timer.Interval = new TimeSpan(0, 0, 1);

Timer.Start();

}

public override void Stop()

{

Timer.Stop();

}

public override string ToString()

{

return new TimeSpan(hour, minute, second).ToString();

}

public event EventHandler Ticks;

public event EventHandler Check\_Event;

private void Timer\_Ticks(object sender, EventArgs e)

{

Second\_Next();

if (Ticks != null) // Вызываем событие

Ticks(this, new EventArgs());

}

public MainWindow MainWindow

{

get

{

throw new System.NotImplementedException();

}

set

{

}

}

}

**Класс «Будильник»**

public class Allarm : RunTimeClock, IComparable

{

public Allarm() : base()

{

second = 0; sleepTime = 5; activate = true; IsThisSleepTimer = false; //1 минута

}

public bool IsThisSleepTimer { get; set; }

public Allarm(int Hours, int Minutes, int Seconds, int \_number, int \_allarmTime) : base(Hours, Minutes, Seconds)

{

activate = true;

number = \_number;

sleepTime = \_allarmTime;

}

public static IComparer<Allarm> SortByTheme()

{

return (IComparer<Allarm>)new Allarm();

}

public bool activate;

private int number;

public int sleepTime { get; set; }

public Allarm DeepCopy()

{

return new Allarm(Hour, Minute, Second, number,sleepTime);

}

public int AllarmTime { get; set; }

public int Number { get; set; }

public event EventHandler TimeChanged;

public event EventHandler runAllarm;

public void Sleep\_Next()

{

sleepTime += 1;

}

public void Sleep\_Prev()

{

sleepTime -= 1;

}

public override void Hour\_Next()

{

if (this.hour == 23)

hour = 0;

else hour++;

if (TimeChanged != null) // Вызываем событие

TimeChanged(this, new EventArgs());

}

public static bool operator >(Allarm a, Allarm b)

{

if ((a.hour \* 100) + a.minute > (b.hour + 100) + b.minute)

return true;

else return false;

}

public static bool operator <(Allarm a, Allarm b)

{

if (a > b)

return false;

else return true;

}

public override void Minute\_Next()

{

if (minute == 59)

{

minute = 0;

Hour\_Next();

}

else

minute++;

if (TimeChanged != null) // Вызываем событие

TimeChanged(this, new EventArgs());

}

public override void Second\_Next()

{

if (second == 59)

{

second = 0;

Minute\_Next();

}

else

second++;

if (TimeChanged != null) // Вызываем событие

TimeChanged(this, new EventArgs());

}

public void RunAllarm()

{

if (runAllarm != null) // Вызываем событие

runAllarm(this, new EventArgs());

}

public override void Hour\_Prev()

{

if (hour == 0)

hour = 23;

else hour--;

if (TimeChanged != null) // Вызываем событие

TimeChanged(this, new EventArgs());

}

public override void Minute\_Prev()

{

if (minute == 0)

{

minute = 59;

Hour\_Prev();

}

else

minute--;

if (TimeChanged != null) // Вызываем событие

TimeChanged(this, new EventArgs());

}

public override void Second\_Prev()

{

if (second == 0)

{

second = 59;

Minute\_Prev();

}

else

second--;

if (TimeChanged != null) // Вызываем событие

TimeChanged(this, new EventArgs());

}

public int CompareTo(object obj)

{

Allarm Obg = obj as Allarm;

if (obj == null) return 1;

if (obj is Allarm)

{

return (hour \* 100 + minute).CompareTo((Obg.hour \* 100 + Obg.minute));

}

else

throw new ArgumentException("Object is not a Temperature");

}

public MainWindow MainWindow

{

get

{

throw new System.NotImplementedException();

}

set

{

}

}

}

**Класс «Главное Окно»**

public partial class MainWindow : Window

{

public List<Allarm> Object\_List\_Of\_AllarmsAllarmsList = new List<Allarm>();

public int On\_Valid\_Index=0;

public Allarm Object\_AllarmTime = new Allarm();

Radio MyRadio = new Radio();

public RunTimeClock Object\_NowTime = new RunTimeClock();

public WMPLib.WindowsMediaPlayer WMP = new WMPLib.WindowsMediaPlayer();

public bool IsAllarmed = true;

public MainWindow()

{

InitializeComponent();

String path = System.IO.Path.Combine(System.IO.Path.GetDirectoryName(Process.GetCurrentProcess().MainModule.FileName), "clock.jpg");

Wrapper.Background = new ImageBrush(new BitmapImage(new Uri(path)));

Object\_NowTime.Ticks+=new EventHandler(Timer\_Ticks);

Object\_NowTime.Start();

Object\_NowTime.Check\_Event += IsAllarmed\_Change;

off.IsChecked = true;

Main\_ListBox\_Of\_AllarmTime.Items.Add("<-- No Allarms Set -->");

}

#region main Grid

private void Timer\_Ticks(object sender, EventArgs e)

{

clock.Text = Object\_NowTime.ToString();

date.Text = DateTime.Now.ToShortDateString();

}

private void setTime\_Click(object sender, RoutedEventArgs e)

{

Grid\_Timer\_Allarm.Visibility = Visibility.Hidden;

MainClock.Visibility = Visibility.Visible;

setAllarmGrid.Visibility = Visibility.Hidden;

setTimeGrid.Visibility = Visibility.Visible;

}

private void setAllarm\_Click(object sender, RoutedEventArgs e)

{

Grid\_Timer\_Allarm.Visibility = Visibility.Hidden;

MainClock.Visibility = Visibility.Hidden;

setTimeGrid.Visibility = Visibility.Hidden;

setAllarmGrid.Visibility = Visibility.Visible;

Clock\_On\_The\_Grid\_Allarm.Text = Object\_AllarmTime.ToString();

Object\_AllarmTime.TimeChanged += new EventHandler(Allarm\_Set\_Clock\_Changed);

}

private void Allarm\_Set\_Clock\_Changed(object sender, EventArgs e)

{

Clock\_On\_The\_Grid\_Allarm.Text = Object\_AllarmTime.ToString();

}

private void mainClock\_Click(object sender, RoutedEventArgs e)

{

MainClock.Visibility = Visibility.Visible;

setTimeGrid.Visibility = Visibility.Hidden;

setAllarmGrid.Visibility = Visibility.Hidden;

}

private void reSet\_Click(object sender, RoutedEventArgs e)

{

Main\_ListBox\_Of\_AllarmTime.Items.Clear();

SetAllarm\_ListBox\_AlarmList.Items.Clear();

Object\_List\_Of\_AllarmsAllarmsList.Clear();

Main\_ListBox\_Of\_AllarmTime.Items.Add("<-- No Allarms Set -->");

}

#endregion

#region SetTime\_Grid

private void hour\_plus\_Click(object sender, RoutedEventArgs e)

{

Object\_NowTime.Hour\_Next();

}

private void hour\_minus\_Click(object sender, RoutedEventArgs e)

{

Object\_NowTime.Hour\_Prev();

}

private void minut\_plus\_Click(object sender, RoutedEventArgs e)

{

Object\_NowTime.Minute\_Next();

}

private void minute\_minus\_Click(object sender, RoutedEventArgs e)

{

Object\_NowTime.Minute\_Prev();

}

#endregion

#region Set\_Allarm\_Grid

private void Button\_Click(object sender, RoutedEventArgs e)

{

MainClock.Visibility = Visibility.Visible;

setTimeGrid.Visibility = Visibility.Hidden;

setAllarmGrid.Visibility = Visibility.Hidden;

}

private void hour\_plus\_a\_Click(object sender, RoutedEventArgs e)

{

Object\_AllarmTime.Hour\_Next();

}

private void hour\_minus\_a\_Click(object sender, RoutedEventArgs e)

{

Object\_AllarmTime.Hour\_Prev();

}

private void minut\_plus\_a\_Click(object sender, RoutedEventArgs e)

{

Object\_AllarmTime.Minute\_Next();

}

private void minute\_minus\_a\_Click(object sender, RoutedEventArgs e)

{

Object\_AllarmTime.Minute\_Prev();

}

private void add\_Click(object sender, RoutedEventArgs e)

{

IsAllarmed = false;

SetAllarm\_ListBox\_AlarmList.Items.Clear();

Main\_ListBox\_Of\_AllarmTime.Items.Clear();

Object\_List\_Of\_AllarmsAllarmsList.Add(Object\_AllarmTime.DeepCopy());

Object\_List\_Of\_AllarmsAllarmsList.Sort();

for (int i = 0; i < Object\_List\_Of\_AllarmsAllarmsList.Count;i++)

SetAllarm\_ListBox\_AlarmList.Items.Add((i + 1).ToString() + ". " + Object\_List\_Of\_AllarmsAllarmsList[i]);

for (int i = 0; i < Object\_List\_Of\_AllarmsAllarmsList.Count; i++)

Main\_ListBox\_Of\_AllarmTime.Items.Add((i + 1).ToString() + ". " + Object\_List\_Of\_AllarmsAllarmsList[i]);

}

private void plus\_sleep\_time\_Click(object sender, RoutedEventArgs e)

{

Object\_AllarmTime.Sleep\_Next();

TextBox\_Sleep\_Time.Text = Object\_AllarmTime.sleepTime.ToString();

}

private void minus\_sleep\_time\_Click(object sender, RoutedEventArgs e)

{

Object\_AllarmTime.Sleep\_Prev();

TextBox\_Sleep\_Time.Text = Object\_AllarmTime.sleepTime.ToString();

}

#endregion

#region Allarm\_View\_Grid

private void on\_Checked(object sender, RoutedEventArgs e)

{

MyRadio.Stop();

Main\_ListBox\_Of\_AllarmTime.Visibility = Visibility.Visible;

TextBox\_RadioIndicator.Visibility = Visibility.Hidden;

TextBox\_FM\_Indicator.Visibility = Visibility.Hidden;

Grid\_Timer\_Allarm.Visibility = Visibility.Hidden;

Object\_NowTime.Ticks += Allarm\_Enable;

Object\_NowTime.Check\_Event += Check\_Event\_Event;

Object\_NowTime.Ticks -= Allaert\_Indicator;

}

private void IsAllarmed\_Change(object sender, EventArgs e)

{

IsAllarmed = false;

}

private void Check\_Event\_Event(object sender, EventArgs e)

{

for (int i = 0; i < Object\_List\_Of\_AllarmsAllarmsList.Count; i++)

{

if (Object\_List\_Of\_AllarmsAllarmsList[i].Hour == Object\_NowTime.Hour &&

Object\_List\_Of\_AllarmsAllarmsList[i].Minute < Object\_NowTime.Minute)

Object\_List\_Of\_AllarmsAllarmsList[i].activate = false;

if(Object\_List\_Of\_AllarmsAllarmsList[i].Hour<Object\_NowTime.Hour)

Object\_List\_Of\_AllarmsAllarmsList[i].activate = false;

}

}

private int Get\_Valid\_Allarm\_Index()

{

for (int i = 0; i < Object\_List\_Of\_AllarmsAllarmsList.Count; i++)

if (Object\_List\_Of\_AllarmsAllarmsList[i].activate)

return i;

return 0;

}

private void On\_Signal(bool button)

{

if(button)

{

WMP.URL = @"back.mp3"; // файл музыкальный

WMP.settings.volume = 100; // меняя значение можно регулировать громкость

WMP.controls.play(); // Старт

}

else

{

WMP.controls.stop();

}

}

private void Allarm\_Enable(object sender, EventArgs e)

{

if (Object\_List\_Of\_AllarmsAllarmsList.Count!=0)

if (!IsAllarmed)

{

On\_Valid\_Index = Get\_Valid\_Allarm\_Index();

if (Object\_NowTime.Hour == Object\_List\_Of\_AllarmsAllarmsList[On\_Valid\_Index].Hour)

if (Object\_NowTime.Minute == Object\_List\_Of\_AllarmsAllarmsList[On\_Valid\_Index].Minute)

if (Object\_NowTime.Second == Object\_List\_Of\_AllarmsAllarmsList[On\_Valid\_Index].Second)

{

Object\_NowTime.Ticks += Run\_Allarm\_Clock;

On\_Signal(true);

Grid\_Allarm\_View.Visibility = Visibility.Visible;

}

}

}

private void off\_Checked(object sender, RoutedEventArgs e)

{

MyRadio.Stop();

Main\_ListBox\_Of\_AllarmTime.Visibility = Visibility.Hidden;

TextBox\_RadioIndicator.Visibility = Visibility.Hidden;

TextBox\_FM\_Indicator.Visibility = Visibility.Hidden;

Grid\_Timer\_Allarm.Visibility = Visibility.Hidden;

Object\_NowTime.Ticks -= Allarm\_Enable;

Object\_NowTime.Ticks -= Run\_Allarm\_Clock;

IsAllarmed = true;

Object\_NowTime.Ticks -= Allaert\_Indicator;

}

private void Run\_Allarm\_Clock(object sender, EventArgs e)

{

Allarm\_time.Text = Object\_NowTime.ToString();

}

private void off\_allarm\_Click(object sender, RoutedEventArgs e)

{

On\_Signal(false);

Grid\_Allarm\_View.Visibility = Visibility.Hidden;

MainClock.Visibility = Visibility.Visible;

}

#endregion

#region Set\_Radio\_Grid

//Set\_Radio\_Grid

private void set\_Radio\_Click(object sender, RoutedEventArgs e)

{

MainClock.Visibility = Visibility.Hidden;

Set\_Radio.Visibility = Visibility.Visible;

}

private void back\_to\_main\_Click(object sender, RoutedEventArgs e)

{

MainClock.Visibility = Visibility.Visible;

Set\_Radio.Visibility = Visibility.Hidden;

}

private void fm\_plus\_Click(object sender, RoutedEventArgs e)

{

MyRadio.RunChangedState();

MyRadio.Frec\_Plus();

Slider\_State\_Control.Value += 0.1;

radio\_frequency.Text = "" + String.Format("{0:F1}", MyRadio.Frecuency) + " FM";

}

private void fm\_minus\_Click(object sender, RoutedEventArgs e)

{

MyRadio.RunChangedState();

MyRadio.Frec\_Prev();

Slider\_State\_Control.Value -= 0.1;

radio\_frequency.Text = "" + String.Format("{0:F1}", MyRadio.Frecuency) + " FM"; ;

TextBox\_FM\_Indicator.Text = "" + String.Format("{0:F1}", MyRadio.Frecuency) + " FM";

}

private void Slider\_State\_Control\_ValueChenged(object sender, RoutedEventArgs e)

{

MyRadio.RunChangedState();

MyRadio.Frecuency = Slider\_State\_Control.Value;

radio\_frequency.Text = "" + String.Format("{0:F1}", MyRadio.Frecuency) + " FM";

TextBox\_FM\_Indicator.Text = "" + String.Format("{0:F1}", MyRadio.Frecuency) + " FM";

}

private void State\_Control\_Voice(object sender, EventArgs e)

{

if (Convert.ToDouble(String.Format("{0:F1}", MyRadio.Frecuency)) == 106.6)

{

MyRadio.PlaySearchVoice(false);

MyRadio.Play(1);

}

else if (Convert.ToDouble(String.Format("{0:F1}", MyRadio.Frecuency)) == 103.5)

{

MyRadio.PlaySearchVoice(false);

MyRadio.Play(2);

}

else if (Convert.ToDouble(String.Format("{0:F1}", MyRadio.Frecuency)) == 107.3)

{

MyRadio.PlaySearchVoice(false);

MyRadio.Play(3);

}

else

{

MyRadio.Stop();

MyRadio.PlaySearchVoice(true);

}

}

#endregion

private void sleep\_Click(object sender, RoutedEventArgs e)

{

;

for (int i = 0; i < Object\_List\_Of\_AllarmsAllarmsList[On\_Valid\_Index].sleepTime; i++)

Object\_List\_Of\_AllarmsAllarmsList[On\_Valid\_Index].Minute\_Next();

SetAllarm\_ListBox\_AlarmList.Items.Clear();

Main\_ListBox\_Of\_AllarmTime.Items.Clear();

Object\_List\_Of\_AllarmsAllarmsList.Sort();

for (int i = 0; i < Object\_List\_Of\_AllarmsAllarmsList.Count; i++)

SetAllarm\_ListBox\_AlarmList.Items.Add((i + 1).ToString() + ". " + Object\_List\_Of\_AllarmsAllarmsList[i]);

for (int i = 0; i < Object\_List\_Of\_AllarmsAllarmsList.Count; i++)

Main\_ListBox\_Of\_AllarmTime.Items.Add((i + 1).ToString() + ". " + Object\_List\_Of\_AllarmsAllarmsList[i]);

Grid\_Allarm\_View.Visibility = Visibility.Hidden;

On\_Signal(false);

Object\_NowTime.Ticks -= Run\_Allarm\_Clock;

IsAllarmed = true;

}

private void radio\_Checked(object sender, RoutedEventArgs e)

{

MyRadio.ChangedState -= State\_Control\_Voice;

MyRadio.ChangedState += State\_Control\_Voice;

MyRadio.RunChangedState();

Main\_ListBox\_Of\_AllarmTime.Visibility = Visibility.Hidden;

TextBox\_RadioIndicator.Visibility = Visibility.Visible;

TextBox\_FM\_Indicator.Visibility = Visibility.Visible;

Object\_NowTime.Ticks -= Allaert\_Indicator;

Grid\_Timer\_Allarm.Visibility = Visibility.Hidden;

Object\_NowTime.Ticks -= Allarm\_Enable;

}

private void timer\_Checked(object sender, RoutedEventArgs e)

{

MyRadio.ChangedState -= State\_Control\_Voice;

MyRadio.ChangedState += State\_Control\_Voice;

MyRadio.RunChangedState();

Main\_ListBox\_Of\_AllarmTime.Visibility = Visibility.Hidden;

TextBox\_FM\_Indicator.Visibility = Visibility.Visible;

MyRadio.TimerAllarm=30;

Object\_NowTime.Ticks += Allaert\_Indicator;

MyRadio.minute = 29;

MyRadio.second = 59;

Grid\_Timer\_Allarm.Visibility = Visibility.Visible;

}

private void Allaert\_Indicator(object sender, EventArgs e)

{

TextBox\_Time\_To\_Allert.Text = MyRadio.minute + " min " + MyRadio.second + " sec";

MyRadio.Second\_Prev();

if(MyRadio.minute==0 &&MyRadio.second==0)

{

Grid\_Timer\_ShowAllarmScreen.Visibility = Visibility.Visible;

}

}

private void Slider\_Volume\_Control\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

MyRadio.Volume = (int)Slider\_Volume\_Control.Value;

}

private void off\_allert\_Click(object sender, RoutedEventArgs e)

{

Grid\_Timer\_ShowAllarmScreen.Visibility = Visibility.Hidden;

}

private void Find\_Station\_Click(object sender, RoutedEventArgs e)

{

for (int i = 0; i < MyRadio.stationsValue.Length; i++)

if (MyRadio.Frecuency < MyRadio.stationsValue[i])

{

Slider\_State\_Control.Value = MyRadio.stationsValue[i];

MyRadio.RunChangedState();

break;

}

}

private void On\_Radio\_Click(object sender, RoutedEventArgs e)

{

TextBox\_FM\_Indicator.Visibility = Visibility.Visible;

MyRadio.ChangedState -= State\_Control\_Voice;

MyRadio.ChangedState += State\_Control\_Voice;

MyRadio.RunChangedState();

}

private void Off\_Radio\_Click(object sender, RoutedEventArgs e)

{

MyRadio.Stop();

TextBox\_FM\_Indicator.Visibility = Visibility.Hidden;

}

}