
Урок 2

# Введение в WPF.

# Часть 2

Добавляем таймер, календарь и напоминание в приложение «Рассыльщик». Создаем собственные библиотеки DLL и свои контролы (элементы управления).

# Введение

Сегодня наша цель — создать общий прототип приложения (внешний вид и архитектуру). Поэтому добавим:

- контролы на форму, чтобы было понятно, как будет выглядеть приложение;
- базу данных с одной таблицей;
- основные классы, при помощи которых будут отправляться письма.

# Обзор контролов WPF

В WPF входит ряд контролов, которые применяются в пользовательском интерфейсе. Они похожи на те, которые используются в Windows Forms.

## Menu

На прошлом уроке мы добавили меню на форму, теперь сделаем его пункты.

Есть три способа добавить пункты меню, используя возможности VS и не прибегая к кодированию на C#:

1. При помощи **Properties** в категории **Common**, пункт **Items**.
2. Кликнуть правой кнопкой мыши по меню и выбрать пункт **Add MenuItem**.
3. Прописать в XAML-коде.

Добавим пункт «Файл» любым из способов. Предпочтительнее правая кнопка мыши, но если надо добавить не **MenuItem**, а другой элемент, то лучше использовать **Properties**.

Чтобы добавить к строкам меню раскрывающиеся подпункты, используем те же способы. Еще добавим пункт «Закрыть». Сразу переименуем его: изменим свойство **Name**, назовем **miClose**. Сразу добавим обработчик и пропишем внутри него **this.Close();**.

## TabControl

По умолчанию у контрола уже есть две вкладки, которые называются **TabItem**. Добавим еще две (так же, как пункты для меню) и переименуем:

1. Формирование группы рассылки.
2. Планировщик.
3. Редактор писем.
4. Статистика.

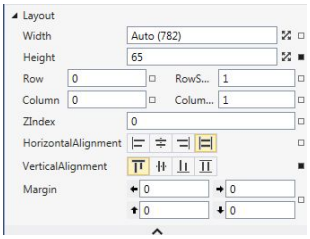
На первой вкладке «Формирование группы рассылки» будет находиться **DataGrid** с адресатами и меню с возможностью редактировать адресатов в DataGrid и выбрать отправителя и smtp-сервер.

## ToolBarTray

Этот элемент управления упрощает размещение панелей инструментов на одной строке. Контрол **ToolBarTray** содержит коллекцию элементов **ToolBar**. Панели инструментов **ToolBar** можно перетаскивать уже в рабочем приложении, если не устанавливать свойство **ToolBarTray.IsLocked =**

**“true”** для конкретной панели.

Разместим **ToolBarTray** во вкладке «Формирование группы рассылки» сверху и сделаем ширину 65. Длину — **AUTO** и растянутой по всей длине. **VerticalAlignment** — **Top**. Для **Margin** выполним **Reset**, чтобы везде были нули. Таким образом элемент управления **ToolBarTray** разместится на самом верху вкладки с фиксированной шириной и длиной по всей длине окна.




# ToolBar

Элемент управления **ToolBar**, как правило, заполняется контролами: кнопками, **ComboBox**, **Checkbox**, **RadioButton** и другими.

Добавим **ToolBar** и снабдим его кнопками и **ComboBox**-ом для выбора почтового ящика отправителя.

Кидаем элемент **ToolBar** на **ToolBarTray**. В свойствах устанавливаем фиксированную длину 500 и ширину 30. Переименовываем его в **tbSender**. Кидаем на **ToolBar** элемент **Label**, переименовываем его в **lSender**. В **Content** прописываем «Выбрать отправителя». Длину и ширину лейбла сделаем фиксированными — 130 и 25 соответственно.

# ComboBox

Добавим на **ToolBar** элемент выбора адреса отправителя. Перетащим мышью данный контрол и разместим его рядом с лейблом **lSender**. Сразу переименуем его в **cbSenderSelect**.

Если вы работали в WinForms, то помните, для чего нужен **ComboBox** — для выбора конкретного элемента из выпадающего списка. В WPF **ComboBox** служит такой же цели. Сделаем фиксированную длину и ширину 270 и 25 и добавим два (или более) элемента **ComboBoxItem** в данный контрол. Для этого есть те же три способа:

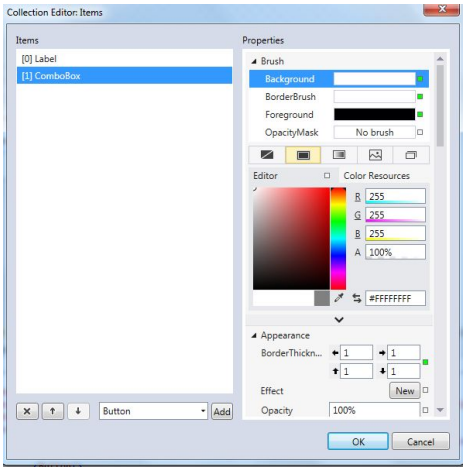
1. При помощи **Properties** в категории **Common**, пункт **Items**.
2. Кликнуть правой кнопкой мыши по меню и выбрать пункт **Add ComboBoxItem**.
3. Прописать в XAML-коде.

В свойство **Content** каждого **Item**-а прописываем email отправителя.

В свойстве **SelectedIndex** элемента **ComboBox** можем задать тот **Item**, который будет выбран по умолчанию. В свойстве **ToolTip** напишем «Выбрать email отправителя» — эта подсказка будет появляться, если навести курсор на **ComboBox**.

# Button

Чтобы добавить кнопку на элемент ToolBar, для разнообразия воспользуемся **Properties**. Выберем категорию **Common** и свойство **Items**. Появится такое окно:



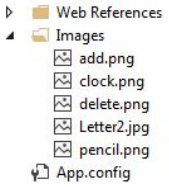

**Label** и **ComboBox** в списке уже присутствуют. Внизу есть кнопка **Add**, рядом — выпадающий список, через который можно выбрать элемент, чтобы добавить на **ToolBar**. В принципе, таким же образом можно добавить и лейбл, и выпадающий список, но их нет в выпадающем меню, а выбирать их через пункт «другие» очень неудобно. Теперь добавим три кнопки и зададим им фиксированную длину и ширину — 25 и 25. Делаем их растянутыми.

Первая кнопка — добавить нового отправителя. Переименовываем ее в **btnAddSender** и в свойство **ToolTip** пишем «Добавить».

Вторая — редактировать нового отправителя. Переименовываем ее в **btnEditSender** и в свойство **ToolTip** пишем «Редактировать».

Третья — удалить нового отправителя. Переименовываем ее в **btnDeleteSender** и в свойство **ToolTip** пишем «Удалить».

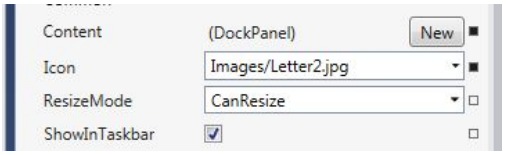
Теперь делаем кнопки красивыми. Добавляем в проект папку **Images** и добавляем в нее картинки, которые нужны для проекта.




Цель — поместить красивую иконку на кнопку. Проще всего сделать это через XAML.

```
<Button x:Name="btnAddSender" VerticalAlignment="Stretch"
HorizontalAlignment="Stretch" Width="25" ToolTip="Добавить">

         <Image Source="Images/add.png" HorizontalAlignment="Right"    />

</Button>

<Button x:Name="btnEditSender" HorizontalAlignment="Center"
VerticalAlignment="Bottom" Width="25" Height="25" ToolTip="Редактировать">

        <Image Source="Images/pencil.png"/>

</Button>

<Button x:Name="btnDeleteSender" HorizontalAlignment="Right"
VerticalAlignment="Bottom" Width="25" Height="25" ToolTip="Удалить">

        <Image Source="Images/delete.png" HorizontalAlignment="Right"/>

</Button>
```

Добавим сразу иконку и на весь проект. Заходим в свойства главного окна и прописываем в свойстве **Icon — Images/Letter2.jpg**.




По аналогии с комбобоксом по выбору отправителя, создаем комбобокс с выбором smtp-сервера. В поле **Content** указываем название «Выбрать smtp-server».

```
<ToolBar x:Name="tbSmtp" HorizontalAlignment="Left" Height="30"
VerticalAlignment="Top" Width="Auto" Band="1" BandIndex="0">
      <Label x:Name="lSmtp" Content="Выбрать smtp-сервер" Height="25"
Width="Auto"/>
      <ComboBox x:Name="cbSmtpSelect" Height="25" Margin="0"
VerticalAlignment="Top" Width="270" SelectedIndex="0" ToolTip="Выбрать
smtp-сервер"/>
      <Button x:Name="btnAddSmtp" Width="25" Height="25" ToolTip="Добавить">
           <Image Source="Images\add.png"/>
      </Button>
      <Button x:Name="btnEditSmtp" Width="25" Height="25"
ToolTip="Редактировать">
          <Image Source="Images\ pencil.png"/>
      </Button>
      <Button x:Name="btnDeleteSmtp" Width="25" Height="25" ToolTip="Удалить">
          <Image Source="Images\delete.png"/>
   </Button>
</ToolBar>
```

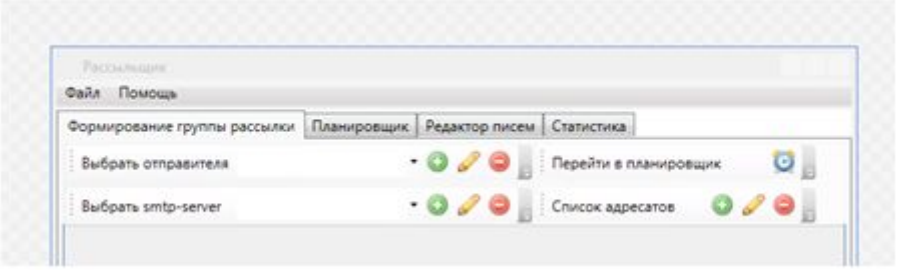
Затем добавляем **ToolBar** «Перейти в планировщик» без комбобокса с одной кнопкой **clock.png**:

```
<ToolBar x:Name="tbPlanner" HorizontalAlignment="Left" Height="30"
VerticalAlignment="Top" Width="230" Band="0" BandIndex="1" >
  <Label x:Name="lPlanner" Content="Перейти в планировщик" Height="25"
Width="Auto"/>
  <Button x:Name="btnClock" Width="25" Height="25" ToolTip="Перейти в
планировщик"
    <Image Source="Images\clock.png"/>
  </Button>
</ToolBar>
```

После этого добавляем последний **ToolBar** «Список адресатов» без комбобокса с тремя кнопками.

```
<ToolBar x:Name="tbAddressee" HorizontalAlignment="Left" Height="30"
VerticalAlignment="Top" Width="230" Band="1" BandIndex="1" >
    <Label x:Name="lAddressee" Content="Список адресатов" Height="25"
Width="Auto"/>
    <Button x:Name="btnAddAddressee" Width="25" Height="25" ToolTip="Добавить">
       <Image Source="Images\add.png"/>
    </Button>
    <Button x:Name="btnEditAddressee" Width="25" Height="25"
ToolTip="Редактировать">
       <Image Source="Images\pencil.png"/>
    </Button>
    <Button x:Name="btnDeleteAddressee" Width="25" Height="25"
ToolTip="Удалить">
       <Image Source="Images\delete.png"/>
    </Button>
</ToolBar>
```

Результат должен выглядеть так:




Сделаем так, чтобы по нажатию кнопки «Перейти в планировщик» пользователь переходил на вкладку «Планировщик». Добавим для этой кнопки обработчик **Click="btnClock_Click">**:

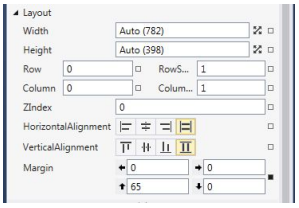
```
private void btnClock_Click(object sender, RoutedEventArgs e)
{
     tabControl.SelectedItem = tabPlanner;
}
```

## DataGrid

Этот элемент необходим для управления визуализацией данных, которые поступают из коллекции

объектов (как правило, таблицы базы данных). Выглядит **DataGrid** как сетка со строками и ячейками. И он совсем не похож на компоновочный элемент управления Grid.

Кинем его на нашу вкладку и выставим свойства **Layout**, как показано на рисунке:



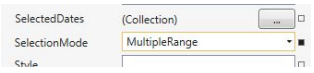

Остальные **ToolBar**-ы, выбор smtp-сервера, переход на вкладку «Планировщик» и редактирование **DataGrid**, нужно будет сделать по аналогии в домашнем задании.

## Calendar

Переходим на вкладку «Планировщик» и добавляем на нее календарь. Он нужен, чтобы выбрать даты и запланировать на них рассылку писем.

Поместим календарь на вкладке слева.

Чтобы появилась возможность выбирать сразу несколько дат в свойствах в категории **Miscellaneous**, выберем свойство **SelectionMode** и установим его в **MultipleRange**.




По умолчанию календарь выбирает сегодняшнюю дату. Пока не будем организовывать сложную рассылку — ограничимся одним днем. Значение свойства **Name** для **Calendar** — **cldSchedulDateTimes**.

## TextBox

Чтобы задать время рассылки, лучше всего подошел бы **TimePicker**, но его нет в стандартной библиотеке контролов. Чтобы его получить, надо скачать и установить **WPF Toolkit** — сделайте это в домашнем задании. Сейчас разместим **TextBox** прямо под календарем и установим им равную длину. В свойстве **Name** укажем **tbTimePicker**.

## Опять Button

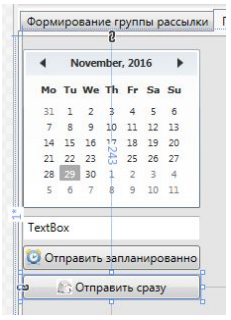
Разместим две кнопки под **TextBox**-ом. По длине сделаем их равными календарю и **TextBox**-у. Назовем их «Отправить запланировано» и «Отправить сразу», а в свойствах **Name** зададим **btnSend** и **btnSendAtOnce** соответственно.

Добавим маленькую иконку рядом с текстом картинки. Свойство **Content** позволяет присвоить ему элемент **StackPanel**, который будет содержать **Image** и **TextBlock**. Добавим в XAML, соответствующий кнопке «Отправить запланировано», следующий код:

```
<Button    x:Name="btnSend"       HorizontalAlignment="Left"    Margin="0,213,0,0"
VerticalAlignment="Top"     Width="179"     RenderTransformOrigin="-1.12,-0.727"
Click="btnSend_Click" Height="25">
```

```
    <Button.Content>
          <StackPanel Orientation="Horizontal">
                <Image Source="Images/clock.png" Width="16" Height="16" />
                    <TextBlock Margin="3,0,0,0" Text="Отправить запланированно"
VerticalAlignment="Center" />
          </StackPanel>
     </Button.Content>
</Button>
```

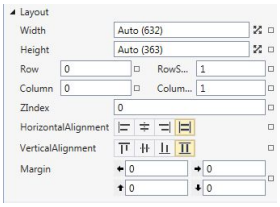
Должно получиться:




Добавить картинку ко второй кнопке будет домашним заданием.

## RichTextBox

Открываем вкладку «Редактор текста» и просто кидаем на нее элемент **RichTextBox**. В свойствах **Layout** делаем так:




На этом визуальная часть прототипа закончена. Можно показывать потенциальному заказчику или начальнику.

# Привязка к данным

Мы придерживаемся мнения, что прототип должен уметь что-то делать. Тем более тестовое приложение, которое отправляет письма, у нас уже есть. Осталось только код по отправке писем перенести из тестового проекта в основной и адаптировать его. Еще надо подключить базу данных с одной таблицей и загрузить email-ы адресатов в проект.

# Привязка к данным ComboBox

Сначала привяжем к **ComboBox cbSenderSelect** данные по адресатам.

Добавим статический класс **VariablesClass**:

```
public static class VariablesClass
    {
        public static Dictionary<string, string> Senders
        {
            get { return dicSenders; }
        }
                private static Dictionary<string, string> dicSenders = new
Dictionary<string, string>()
        {
            { "79257443993@yandex.ru",PasswordClass.getPassword("1234l;i") },
            { "sok74@yandex.ru",PasswordClass.getPassword(";liq34tjk") }
        };
    }
```

Здесь будем хранить возможных отправителей электронных писем.

Хранить в одном файле логин ящика и его пароль — плохая затея даже в учебных и тестовых проектах. Поэтому добавим еще один статический класс с двумя статическими методами.

```
/// <summary>
/// На вход подаем зашифрованный пароль, на выходе получаем пароль для email
/// </summary>
public static class PasswordClass
{
    public static string getPassword(string p_sPassw)
    {
        string password = "";
        foreach (char a in p_sPassw)
        {
            char ch = a;
            ch--;
            password += ch;
        }

        return password;
    }
    /// <summary>
    /// На вход подаем пароль, на выходе получаем зашифрованный пароль
    /// </summary>
    /// <param name="p_sPassword"></param>
    /// <returns></returns>
    public static string getCodPassword(string p_sPassword)
    {
        string sCode = "";
        foreach (char a in p_sPassword)
        {
            char ch = a;
            ch++;
            sCode += ch;
        }
        return sCode;
    }
}
```

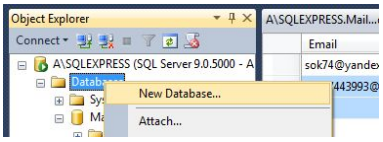
Подобный метод шифрования пароля — скорее от честных людей, но как защита от того, что пароль случайного попадет в руки недоброжелателя, вполне подходит. Осталось привязать коллекцию **Senders** к **ComboBox**.

Добавим код, приведенный ниже, в конструктор основного класса проекта **WPFMailSender**. В файле **WPFMailSender.xaml** у элемента **cbSenderSelect** необходимо удалить **ComboBoxItem**, поскольку указание **ItemsSource** возможно только без него:

```
public WPFMailSender()
{
    InitializeComponent();
    cbSenderSelect.ItemsSource = VariablesClass.Senders;
    cbSenderSelect.DisplayMemberPath = "Key";
    cbSenderSelect.SelectedValuePath = "Value";
}
```

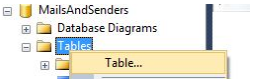
## Привязываем к базе данных DataGrid

Сначала надо создать базу данных в **MS SQL Server**. Открываем **MS SQL Server Management Studio**:




Назовем БД **MailsAndSenders**.

Создаем таблицу:

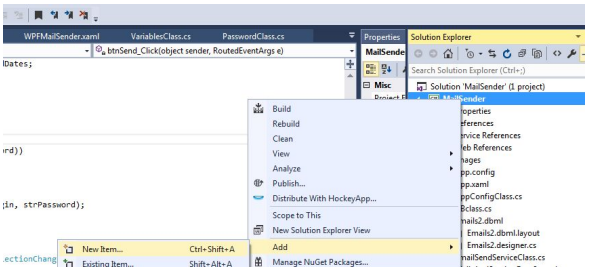



Назовем таблицу **Email**.

```
CREATE TABLE [dbo].[Email] (

    [Id] int NOT NULL,

    [Value]  NVARCHAR (MAX) NOT NULL,

    [Name]   NVARCHAR (MAX) NOT NULL

);

ALTER TABLE [dbo].[Email]

ADD CONSTRAINT PK_Email_Id PRIMARY KEY CLUSTERED (Id);
```

Заполним таблицу тестовыми email-ами и именами.

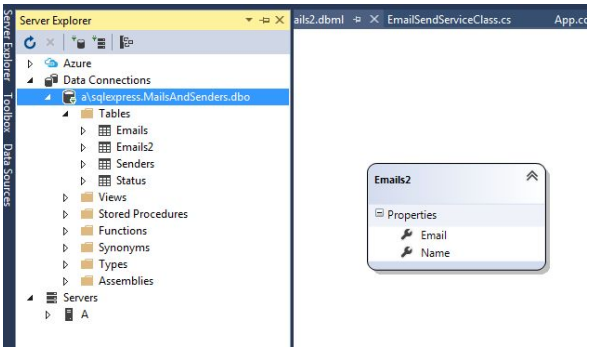
Теперь надо привязать таблицу базы к проекту. Заходим в проект, кликаем правой кнопкой мыши по нему и выбираем **Add — New Item**.



Выбираем **Ling to SQL Classes**:


Назовем новый класс **Emails**

Открываем полученный dbml-файл и просто перетаскиваем на экран мышью таблицу из **Server Explorer**:




В класс, который отвечает за базу данных, добавляем код:

```
/// <summary>
/// Класс, который отвечает за работу с базой данных
/// </summary>
public class DBclass
{
    private EmailsDataContext emails = new EmailsDataContext();
    public IQueryable<Email> Emails
    {
        get
        {
            return from c in emails.Emails select c;
        }
    }

}
```

В конструкторе основного класса добавляем код по привязке данных из базы данных к элементу управления **DataGrid**, который отвечает за отображение адресатов.

```
DBclass db = new DBclass();
dgEmails.ItemsSource = db.Emails;
```

# Отправление писем

Теперь нужно заполнить кодом класс **EmailSendServiceClass**. Почти полностью переносим класс из тестового проекта по отправке писем из прошлого урока — за исключением ряда изменений.

```
class EmailSendServiceClass
{
    #region vars
    private string strLogin;         // email, с которого будет рассылаться почта
     private string strPassword;  // пароль к email, с которого будет рассылаться
почта
    private string strSmtp = "smtp.yandex.ru"; // smtp-server
    private int iSmtpPort = 25;                // порт для smtp-server
    private string strBody;                    // текст письма для отправки
    private string strSubject;                 // тема письма для отправки
    #endregion
    public EmailSendServiceClass(string sLogin, string sPassword)
    {
        strLogin = sLogin;
        strPassword = sPassword;
    }
     private void SendMail(string mail, string name) // Отправка email конкретному
адресату
    {
        using (MailMessage mm = new MailMessage(strLogin, mail))
        {
            mm.Subject = strSubject;
            mm.Body = "Hello world!";
            mm.IsBodyHtml = false;
            SmtpClient sc = new SmtpClient(strSmtp, iSmtpPort);
            sc.EnableSsl = true;
            sc.DeliveryMethod = SmtpDeliveryMethod.Network;
            sc.UseDefaultCredentials = false;
            sc.Credentials = new NetworkCredential(strLogin, strPassword);
            try
            {
                sc.Send(mm);
            }
            catch (Exception ex)
            {
                MessageBox.Show("Невозможно отправить письмо " + ex.ToString());
            }
        }
    }//private void SendMail(string mail, string name)
    public void SendMails(IQueryable<Email> emails)
    {
        foreach (Email email in emails)
        {
            SendMail(email.Email, email.Name);
        }
    }
}  //private void SendMail(string mail, string name)
```

Появился конструктор, в котором задаем логин и пароль отправителя, а также метод **SendMails**, в который передаем коллекцию объектов класса, связанного с базой данных.

## Отправить сразу

Осталось прописать отправление писем. Письма будем отправлять по нажатию на кнопку «Отправить сразу» на вкладке «Планировщик».

Создаем обработчик и вставляем код.

```
private void btnSendAtOnce_Click (object sender, RoutedEventArgs e)
{
    string strLogin = cbSenderSelect.Text;
    string strPassword = cbSenderSelect.SelectedValue.ToString();
    if (string.IsNullOrEmpty(strLogin))
     {
        MessageBox.Show("Выберите отправителя");
        return;
     }
    if (string.IsNullOrEmpty(strPassword))
     {
        MessageBox.Show("Укажите пароль отправителя");
        return;
     }

    EmailSendServiceClass emailSender = new EmailSendServiceClass(strLogin,
strPassword);
    emailSender.SendMails((IQueryable<Email>)dgEmails.ItemsSource);
}
```

Теперь если нажать на кнопку «Отправить сразу», письма отправятся по указанным адресам.

## Отправить запланировано

Поработаем над классом **SchedulerClass**, который отвечает за планировщик. Добавим две библиотеки сразу: одну для таймера, вторую для **MessageBox**.

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

Вот код, который надо добавить в класс **SchedulerClass**:

```
/// <summary>
/// Класс-планировщик, который создает расписание, следит за его выполнением и напоминает о событиях
/// Также помогает автоматизировать рассылку писем в соответствии с расписанием
/// </summary>
class SchedulerClass
{
    DispatcherTimer timer = new DispatcherTimer(); // таймер
     EmailSendServiceClass emailSender;           // экземпляр класса, отвечающего за отправку писем
    DateTime dtSend;                              // дата и время отправки
    IQueryable<Email> emails;                     // коллекция email-ов адресатов
    /// <summary>
    /// Метод, который превращает строку из текстбокса tbTimePicker в TimeSpan
    /// </summary>
    /// <param name="strSendTime"></param>
    /// <returns></returns>
    public TimeSpan GetSendTime(string strSendTime)
    {
        TimeSpan tsSendTime = new TimeSpan();
        try
        {
            tsSendTime = TimeSpan.Parse(strSendTime);
        }
        catch{}
        return tsSendTime;
    }
    /// <summary>
    //// Непосредственно отправка писем
    /// </summary>
    /// <param name="dtSend"></param>
    /// <param name="emailSender"></param>
    /// <param name="emails"></param>
     public void SendEmails(DateTime dtSend, EmailSendServiceClass emailSender, IQueryable<Email> emails)
    {
           this.emailSender = emailSender; // Экземпляр класса, отвечающего за отправку писем, присваиваем
        this.dtSend = dtSend;
        this.emails = emails;
        timer.Tick += Timer_Tick;
        timer.Interval = new TimeSpan(0, 0, 1);
        timer.Start();
    }
    private void Timer_Tick(object sender, EventArgs e)
    {
        if (dtSend.ToShortTimeString() == DateTime.Now.ToShortTimeString())
        {
            emailSender.SendMails(emails);
            timer.Stop();
            MessageBox.Show("Письма отправлены.");
        }
    }
}
```

Добавляем код в обработчик кнопки «Отправить запланировано»:

```
private void btnSend_Click(object sender, RoutedEventArgs e)
{
    SchedulerClass sc = new SchedulerClass();
    TimeSpan tsSendTime = sc.GetSendTime(tbTimePicker.Text);
    if (tsSendTime == new TimeSpan())
    {
        MessageBox.Show("Некорректный формат даты");
        return;
    }
    DateTime dtSendDateTime = (cldSchedulDateTimes.SelectedDate ??
DateTime.Today).Add(tsSendTime);
    if (dtSendDateTime < DateTime.Now)
    {
        MessageBox.Show("Дата и время отправки писем не могут быть раньше, чем
настоящее время");
        return;
    }
    EmailSendServiceClass emailSender = new
EmailSendServiceClass(cbSenderSelect.Text,
cbSenderSelect.SelectedValue.ToString());
    sc.SendEmails(dtSendDateTime, emailSender,
(IQueryable<Email>)dgEmails.ItemsSource);
}
```

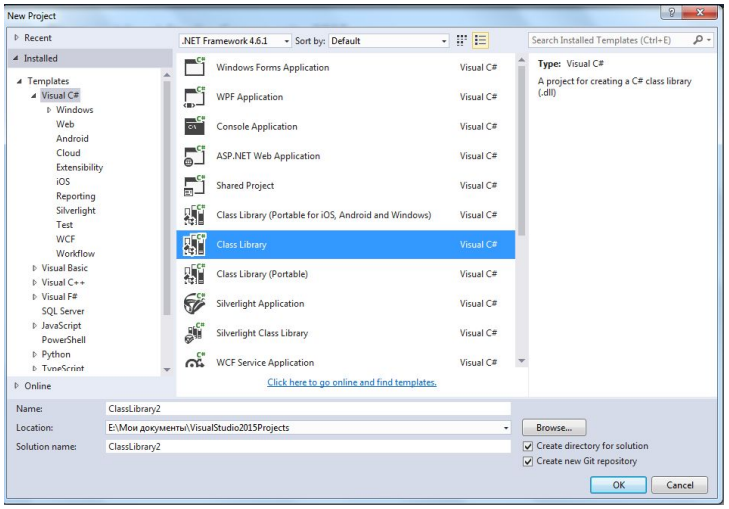
# DLL

## Что такое DLL

DLL — это библиотека динамической компоновки, или динамически подключаемая библиотека (**dynamic link library**). Служит для многократного использования различными программными приложениями.

## Создание файла DLL

Создадим новый проект, выберем тип **Class Library**.

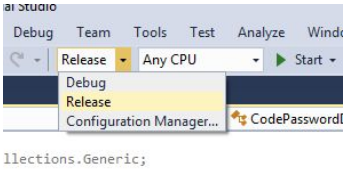



Назовем новый проект **CodePasswordDLL**.

Классу дадим имя **CodePassword**. Наименования пространства имен и класса лучше делать разные, иначе потом будут сложности с использованием этой библиотеки.

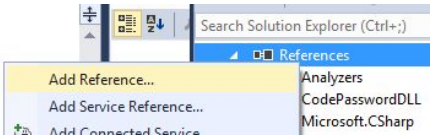
Теперь перенесем код из класса **PasswordClass** нашего проекта **MailSender** в новый проект по созданию **DLL**.

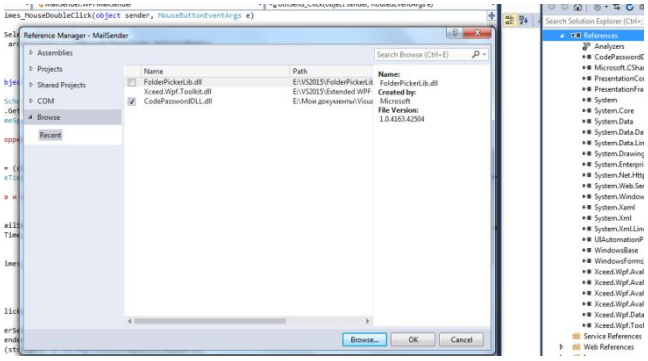
Переключим проект из **Debug** в **Release**:




Соберем проект. В папке **Bin — Release** найдем файл **CodePasswordDLL.dll**.

Вернемся в проект **MailSender** и подключим новую библиотеку к ссылкам проекта:




Закодированный пароль мы использовали в файле **VariableClass.cs**, значит там нужно добавить **using CodePasswordDLL** и немного подправить код.

```
using System.Collections.Generic;
using CodePasswordDLL;
namespace MailSender
{
    public static class VariablesClass
    {
        public static Dictionary<string, string> Senders
        {
            get { return dicSenders; }
        }
                private static Dictionary<string, string> dicSenders = new
Dictionary<string, string>()
        {
            { "79257443993@yandex.ru", CodePassword.getPassword("{3t1l2m6") },
            { "sok74@yandex.ru", CodePassword.getPassword("{3t1l2m6") }
        };
    }
}
```

Удаляем файл **PasswordClass.cs** из проекта. Запускаем — все работает. Можно не создавать отдельное решение, а в том же добавить еще один проект.

## Помещаем DLL в GAC

При создании **DLL** мы сделали закрытую сборку. Использовать ее можно, просто подключив к любому проекту и положив рядом с файлом **.exe**.

Если хотим использовать сборку не с одним проектом, а с несколькими, то и положить ее надо во все.

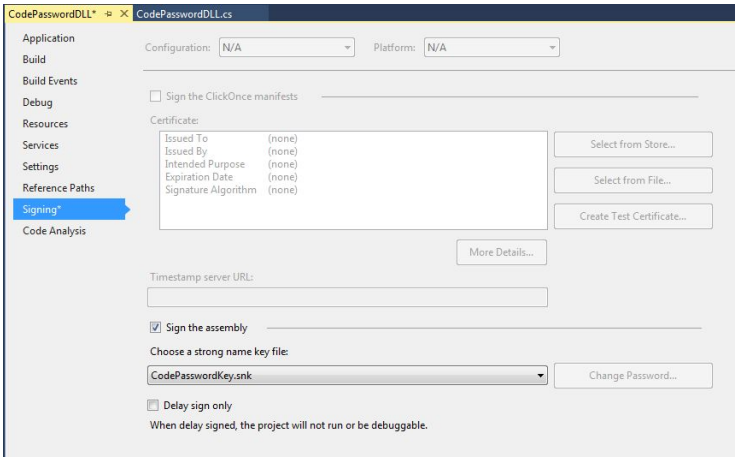
Вариант — положить DLL-ку в **GAC** — Global Assembly Cache — глобальный кэш сборок.

Для этого **DLL** нужно задать строгое имя, в которое входит:

1. Наименование сборки без расширения.
2. Номер версии — при его помощи сможем воспользоваться разными версиями одной сборки.
3. Открытый ключ.
4. Цифровая подпись.

Кликаем правой кнопкой мыши по проекту, который создает **DLL**.

Нажимаем **Properties** и идем в **Signing**.



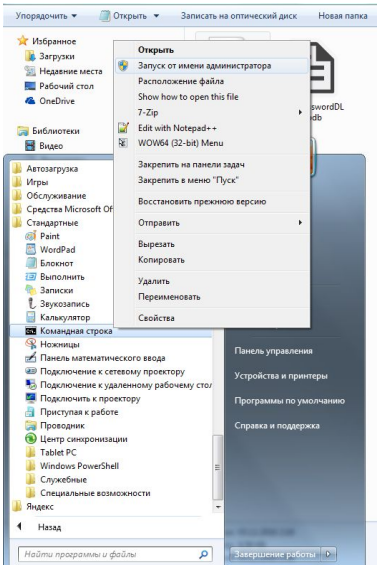

Выбираем галочку **Sing the assembly**, далее в списке **Choose a strong name key file** выбираем **New** и придумываем имя ключа — например, **CodePasswordKey.snk**.

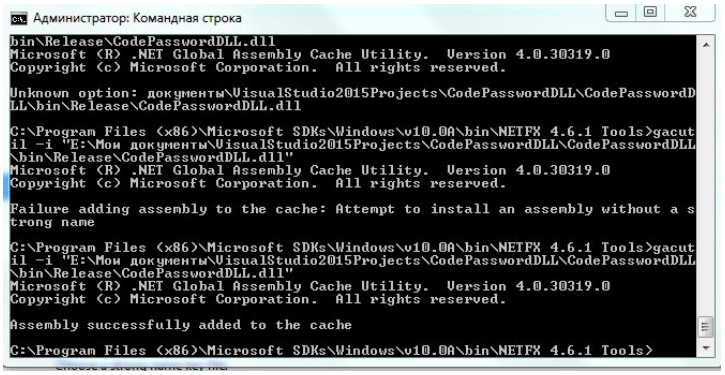
Поместим **DLL** в **GAC**. Для этого понадобится утилита **gacutil.exe**, на компьютере она может располагаться здесь: **C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.6.1 Tools**.

У этой утилиты есть три команды:

1. –i — поместить **DLL** в **GAC**;
2. –l — просмотреть весь список сборок;
3. –u — удалить из **GAC**.

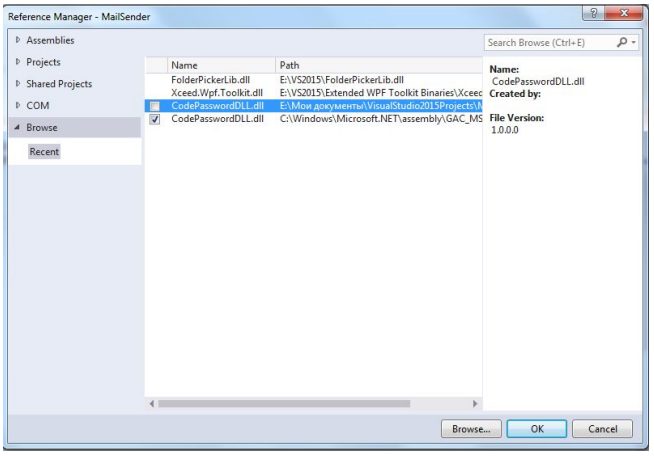
Заходим в консоль как администратор:






При успешном добавлении **DLL** в **GAC** появится надпись в консоли: ***Assembly successfully added to the cache***.

Чтобы подключиться к библиотеке из **GAC**, удаляем сборку **CodePasswordDLL.dll** из **References** и добавляем снова, но уже из **GAC**.



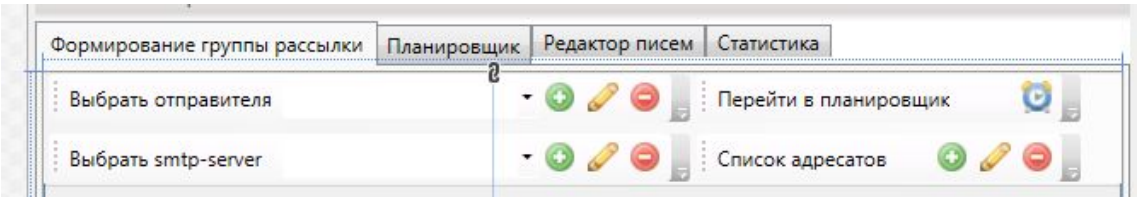

# Создание собственного элемента управления (контрола)

Есть два способа создать свой контрол:

1. Композиция (составной элемент управления) — из существующих контролов создать новый. Наследуется от **UserControl**.
2. Расширить функционал другого контрола — **CustomControl**.

## Создаем составной элемент управления

В «Рассыльщике» на первой вкладке видим **ToolBarTray** с панелями.

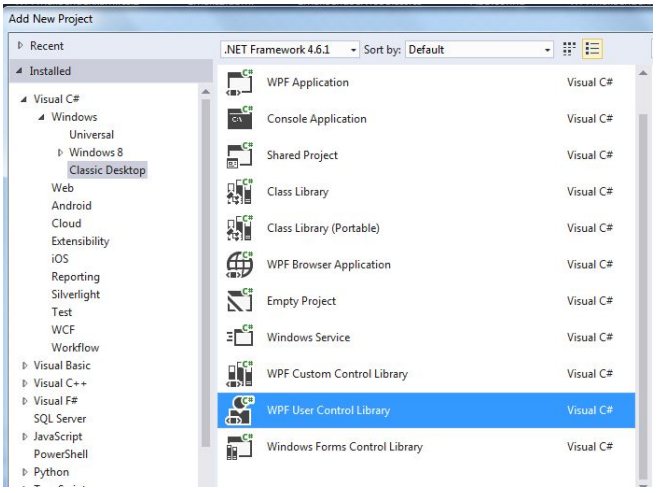


По клику на панель «Перейти в планировщик» переключаемся на следующую вкладку — «Планировщик».

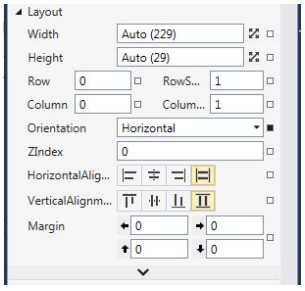
Поменяем эту панель и поместим на ее место кнопки, позволяющие переключаться между вкладками **TabControl**-а.

Создадим свой контрол с кнопками «Следующий» и «Предыдущий», дополним надписи на кнопках иконками со стрелками и сделаем так, чтобы упомянутые кнопки можно было скрыть при необходимости. По размеру новый контрол будет как панель «Перейти в планировщик». Поместить его сможем на все вкладки TabControl-а.

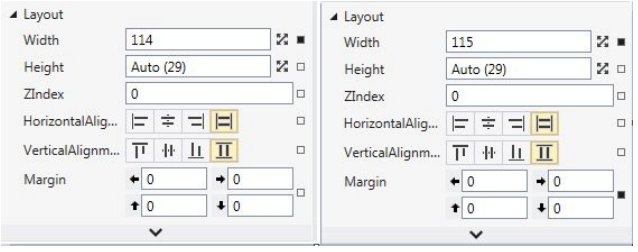
Для этой цели выберем проект **WPF User Control Library**:




Иногда не сразу находится этот проект, он может спрятаться в **Visual C# — Windows — Classic Desktop**. Назовем новый проект **TabSwitcher**. Класс, который создается автоматически, назовем **TabSwitcherControl**. В дизайнере увидим квадрат, куда сможем добавлять элементы управления, — зададим ему размер 229 на 30 и кинем на форму **StackPanel**, а настройки сделаем такие:

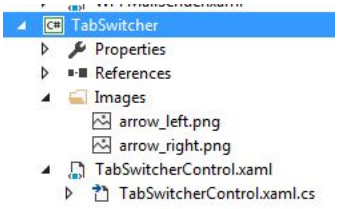



Уже сверху на **StackPanel** расположим две кнопки:




Переименуем их в **btnNext** и **btnPrevious**.

Добавим в новый проект папку **Images** и добавим в нее два файлика с иконками со стрелками.




Поместим на кнопки надпись «Предыдущий» и «Следующий» и иконки со стрелками рядом.

Мы уже добавляли на кнопки **StackPanel**, чтобы поместить на нем и надпись, и картинку.

XAML-код контрола:

```
<UserControl x:Class="TabSwitcher.TabSwitcherControl"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:TabSwitcher"
             mc:Ignorable="d"
             d:DesignHeight="28.895" d:DesignWidth="300" Height="30" Width="229">
    <Grid Height="29" VerticalAlignment="Top">
        <StackPanel Orientation="Horizontal">
            <Button x:Name="btnPrevious"  Width="114" >
                <Button.Content>
                    <StackPanel Orientation="Horizontal">
                                <Image Source="Images/arrow_left.png" Width="30"
Height="16" />
                                   <TextBlock Margin="3,0,0,0" Text="Предыдущий"
VerticalAlignment="Center" />
                    </StackPanel>
                </Button.Content>
                </Button>
                <Button x:Name="btnNext" Width="115 Margin="0">
            <Button.Content>
                <StackPanel Orientation="Horizontal">
                    <TextBlock Margin="3,0,0,0" Text="Следующий" />
                               <Image Source="Images/arrow_right.png" Width="30"
Height="16" Margin="5,0,0,0" />
                    </StackPanel>
            </Button.Content>
                    </Button>
        </StackPanel>
    </Grid>
</UserControl>
```

Добавим возможность скрывать кнопки по желанию пользователя. Если скрываем одну кнопку, то вторая заполняет весь контрол. Добавляем этот код в класс **TabSwitcherControl.xaml.cs**:

```
public partial class TabSwitcherControl : UserControl
{
    #region properties
     private bool bHidebtnPrevious = false; // поле, соответствующее тому, будет ли скрыта кнопка «Предыдущий»
     private bool bHideBtnNext = false; // поле, соответствующее тому, будет ли скрыта кнопка «Следующий»
    /// <summary>
    /// Свойство, соответствующее тому, будет ли скрыта кнопка «Предыдущий».
     /// Чтобы Свойство отразилось на PropertiesGrid у нашего контрола, его нужно представить именно свойством, а не полем
    /// </summary>
    public bool IsHidebtnPrevious
    {
        get { return bHidebtnPrevious; }
        set
        {
            bHidebtnPrevious = value;
            SetButtons(); // метод, который отвечает на отрисовку кнопок
        }
    }// IsHidebtnPrevious
    public bool IsHideBtnNext
    {
        get { return bHideBtnNext; }
        set
        {
            bHideBtnNext = value;
            SetButtons(); // метод, который отвечает за отрисовку кнопок
        }
    }// IsHidebtnPrevious
    private void BtnNextTruebtnPreviousFalse()
    {
        btnNext.Visibility = Visibility.Hidden;
        btnPrevious.Visibility = Visibility.Visible;
        btnPrevious.Width = 229;
        btnNext.Width = 0;
        btnPrevious.HorizontalAlignment = HorizontalAlignment.Stretch;
    }
    private void btnPreviousTrueBtnNextFalse()
    {
        btnPrevious.Visibility = Visibility.Hidden;
        btnNext.Visibility = Visibility.Visible;
        btnNext.Width = 229;
        btnPrevious.Width = 0;
        btnNext.HorizontalAlignment = HorizontalAlignment.Stretch;
    }
    private void btnPreviousFalseBtnNextFalse()
    {
        btnNext.Visibility = Visibility.Visible;
        btnPrevious.Visibility = Visibility.Visible;
        btnNext.Width = 115;
        btnPrevious.Width = 114;
    }
    private void btnPreviousTrueBtnNextTrue()
    {
        btnPrevious.Visibility = Visibility.Hidden;
        btnNext.Visibility = Visibility.Hidden;
    }
    /// <summary>
```

```
   /// Метод, который отвечает за отрисовку кнопок.
    /// Есть три варианта: когда обе кнопки доступны; доступна одна и недоступна
вторая; обе кнопки недоступны
   /// </summary>
    private void SetButtons()
    {
        if (bHidebtnPrevious && bHideBtnNext) btnPreviousTrueBtnNextTrue();
                           else    if    (!bHideBtnNext    &&    !bHidebtnPrevious)
btnPreviousFalseBtnNextFalse();
                            else    if    (bHideBtnNext    &&    !bHidebtnPrevious)
BtnNextTruebtnPreviousFalse();
                            else    if    (!bHideBtnNext    &&    bHidebtnPrevious)
btnPreviousTrueBtnNextFalse();
    }
    #endregion
```

Если мы пересоберем все решение и добавим новый контрол на панель, то увидим в **Properties** нового контрола свойства **IsHideBtnNext** и **IsHidebtnPrevious**, при помощи которых можем скрыть одну из кнопок или обе.

Теперь нужно добавить возможность добавлять обработчик по клику на эти кнопки на контроле **TabSwitcherControl**.

Сначала добавим обработчики на каждую из кнопок в проекте **TabSwitcher**. Затем — два события (event) для каждой из кнопок.

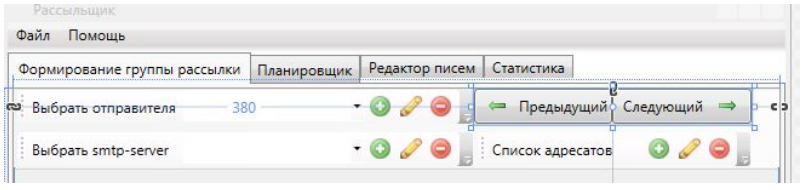
```
public event RoutedEventHandler btnNextClick;
public event RoutedEventHandler btnPreviosClick;
```

А в обработчиках будем эти события вызывать:

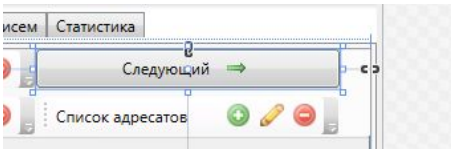
```
public event RoutedEventHandler btnNextClick;
public event RoutedEventHandler btnPreviousClick;
private void btnNext_Click(object sender, RoutedEventArgs e)
{
    btnNextClick?.Invoke(sender, e);
}
private void btnPrevious_Click(object sender, RoutedEventArgs e)
{
    btnPreviousClick?.Invoke(sender, e);
}
```

Пересоберем решение: элемент управления должен появиться в **ToolBox**-е.

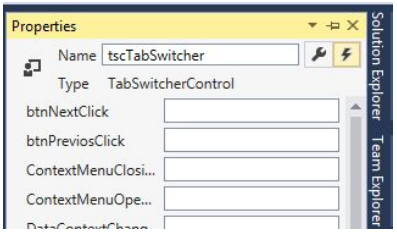
Поместим контрол на нашей форме и сразу переименуем его в **tscTabSwitcher**.




Заходим в свойствах контрола в категорию **Miscellaneous**, находим там свойство **IsHidebtnPrevious** и ставим для него галочку.




Добавим обработчик по нажатию на кнопку «Следующий». Если зайдем в обработчики событий **Properties** контрола, должны увидеть два обработчика.

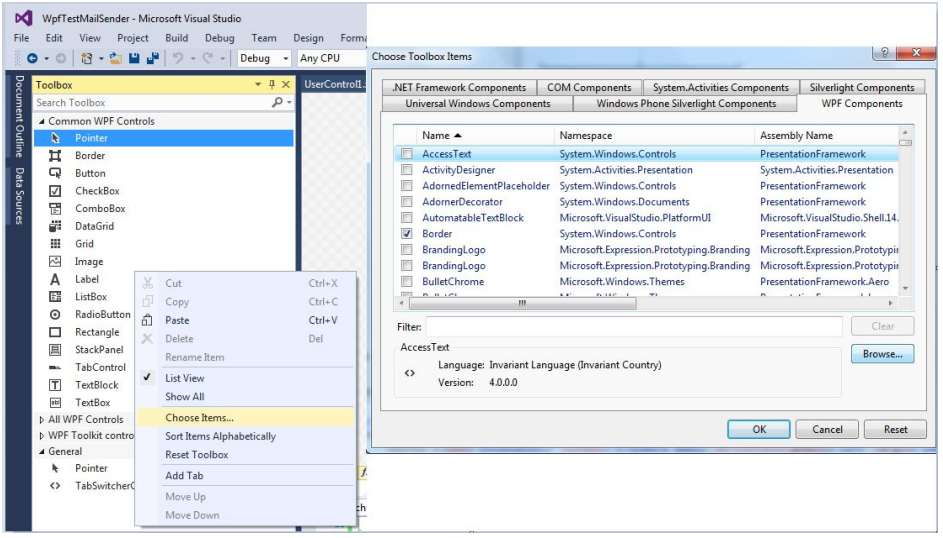



Но они могут не проявиться там до тех пор, пока вы не закроете и вновь не откроете решение. Даже полный **Rebuild** может не помочь — в этом случае можно добавить обработчик события вручную в коде:

```
          tscTabSwitcher.btnNextClick += TscTabSwitcher_btnNextClick;
        }
         private void TscTabSwitcher_btnNextClick(object sender, RoutedEventArgs
e)
        {
            tabControl.SelectedIndex = 1;
        }
```

В домашнем задании добавьте этот контрол на все остальные вкладки **TabControl**. Выберите нужные кнопки и добавьте обработчики по переключению вкладок.

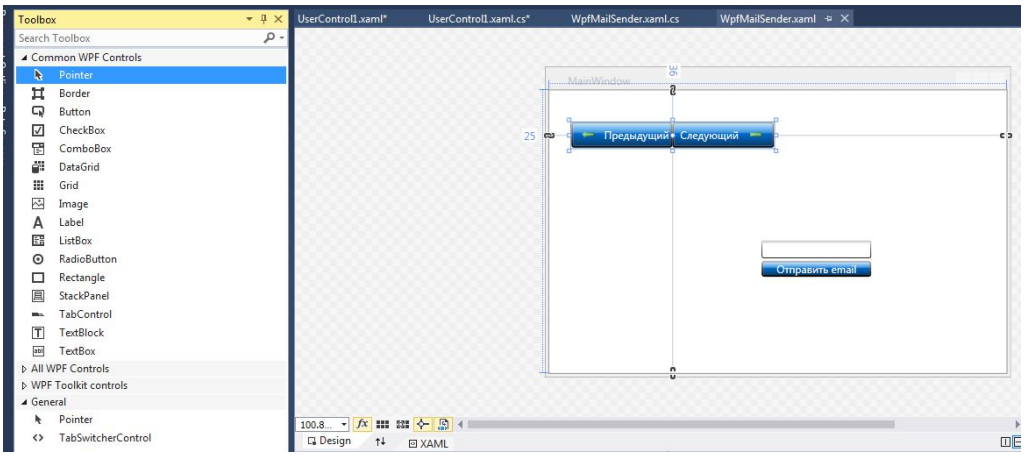
Проверим, что будет, если добавить контрол в сторонний проект. Для этого откроем тестовый проект с первого занятия — **WpfTestMailSender**. Откроем дизайнер основной формы и кликнем правой кнопкой мыши по **ToolBox**.




Нажмем **Browse** и найдем проект, соответствующий нашему контролу, и выберем **DLL**.

В **ToolBox** появится новый контрол **TabSwitchControl**, который мы можем спокойно кинуть на форму и

использовать.




# Домашнее задание

1. К комбобоксу «Выбрать smtp-server» привязать **Dictionary** по аналогии с комбобоксом «Выбрать отправителя». Создать **Dictionary** в том же классе, где ключом будет smtp-сервер, а значением (с типом **int**) — порт smtp-сервера. Сделать так, чтобы значения сервера и порта передавались в экземпляр класса, который отвечает за отправку почты.

2. При отправке писем сделать проверку, есть ли текст в элементе **RichTextBox** во вкладке «Редактор писем». Если он пуст, то на экран выдавать окно «Письмо не заполнено» и открывать вкладку «Редактор писем».

3. Скачать и установить WPF Toolkit с сайта http://wpftoolkit.codeplex.com/:

    a. Добавить на **ToolBox** вкладку **Wpf Toolkit controls**;

    b. Затем кликнуть правой кнопкой мыши по **Choose item**;

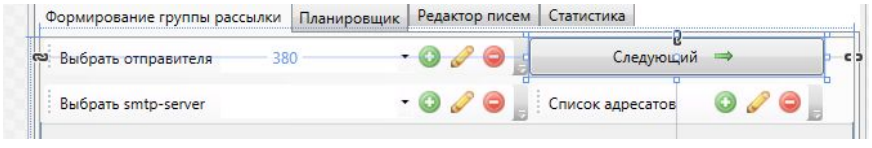
    c. В диалоге, который появился, выбрать кнопку **Browse** и выбрать **DLL** с **Toolkit**.

4. Во вкладке «Планировщик» заменить **TextBox** для ввода времени на элемент **TimePicker**.

5. Добавить картинку **Letter2.jpg** к кнопке «Отправить сразу» во вкладке «Планировщик».

6. По аналогии с тем, как создавали **DLL** из класса, который шифрует пароли, создать **DLL** из класса **EmailSendServiceClass**, который занимается рассылкой писем.

7. Посмотреть на **ToolBarTray**:




Некоторые панели **ToolBar** похожи, и из них можно сделать контрол и добавлять на **ToolBar**. Задание: сделать контрол из панели «Выбрать отправителя» и добавить его в качестве контрола «Выбрать smtp-server». У этого контрола должна быть возможность заменить текст у лейбла, должен функционировать комбобокс и все три кнопки.

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:


1. Ник Рандольф, Дэвид Гарднер, Майкл Минутилло, Крис Андерсон. Visual Studio 2010 для профессионалов.
2. MSDN.