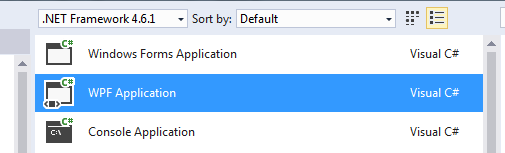
C# Уровень 3.

# Продвинутый курс с WPF.

# Урок 1

# Введение в WPF. Архитектура создания приложения на C#.



Создание приложений на C#, общие рекомендации. Схема приложения. Введение в WPF.

# 

# 

[Продвинутый курс с WPF. 0](#_Toc467090852)

[Урок 1 1](#_Toc467090853)

[Введение в WPF. Архитектура создания приложения на C#. 1](#_Toc467090854)

[Введение 2](#_Toc467090855)

[Введение в WPF 4](#_Toc467090856)

[Знакомсво с XAML 6](#_Toc467090857)

[Знакомство с контролами WPF. 7](#_Toc467090858)

[Изменение стиля приложения WPF. 10](#_Toc467090859)

[Итог 15](#_Toc467090860)

[Рекомендации по созданию приложений. 15](#_Toc467090861)

[Постановка цели. 16](#_Toc467090862)

[С чего начать. План. 16](#_Toc467090863)

[Первый пункт плана. 16](#_Toc467090864)

[Второй пункт плана. 17](#_Toc467090865)

[Третий пункт плана. 18](#_Toc467090866)

[Итог. 19](#_Toc467090867)

[Приложение «Рассыльщик». Интерфейс приложения и WPF. 20](#_Toc467090868)

[Прототип приложения «Рассыльщик». 20](#_Toc467090869)

[Компоновочные элементы управления в системе WPF (панели). 21](#_Toc467090870)

[Итог. 25](#_Toc467090871)

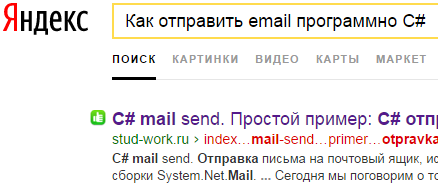
[Домашнее задание. 25](#_Toc467090872)

[Список использованной литературы 26](#_Toc467090873)

# Введение

Предположим, вам пришла в голову замечательная идея, написать программу, которая рассылает электронную почту автоматически всем вашим друзьям, коллегам или какой-либо другой группе лиц. На первом этапе вы ещё четко не представляете интерфейс этой программы. Вы не представляете, какие конкретно задачи она будет выполнять. Нужно проверить, возможно, ли в принципе отправить электронное письмо средствами C#.

На помощь нам придет yandex.ru или google.com. В строке поиска просто набираем "отправка email c#" или "рассылка писем программно C#" и др.



Зачастую, работа программиста заключается в том, что вы ищете в интернете куски кода, подходящего для той задачи, которую вы решаете. В этом нет ничего зазорного или неправильного. Главное понимать и уметь адаптировать тот код, который вы нашли под ваш проект и под ваши задачи.

Вернемся к нашей задаче. В интернете вы можете найти множество самых различных вариантов отправки электроного письма программным способом. Давайте теперь и я предложу один из вариантов.

Для нашей задачи нам нужно использовать протокол SMTP (Просто протокол передачи почты – simple mail transfer protocol).

В .NET Framework, существует специальная библиотека, при помощи которой можно отправить электронное письмо, используя протокол smtp.

Подключим два пространства имен.

|  |
| --- |
| using System.Net;  using System.Net.Mail; |

Для отправки email в пространстве имен System.Net.Mail реализованно два класса SmtpClient и MailMessage.

В коде это может быть реализованно вот так.

|  |
| --- |
| // Формирование письма  MailMessage mm = new MailMessage("оправитель@yandex.ru", [получатель@yandex.ru](mailto:получатель@yandex.ru))  mm.Subject = "Заголовок письма";  mm.Body = "Содержимое письма";  mm.IsBodyHtml = false; // Не используем html в теле письма |

Далее нам нужно организовать авторизацию на smtp сервере.

Здесь нам пригодится класс NetworkCredential из System.Net

|  |
| --- |
| // Авторизуемся на smtp сервере и отправляем письмо  SmtpClient sc = new SmtpClient("smtp.yandex.ru", 25);  sc.EnableSsl = true;  sc.DeliveryMethod = SmtpDeliveryMethod.Network;  sc.UseDefaultCredentials = false;  sc.Credentials = new NetworkCredential("отправитель@yandex.ru", "password");  sc.Send(mm); |

Мы использовали smtp.yandex.ru в качестве службы отправки почты, где 25 это порт. Мы выбрали яндекс, потому что email отправителя находится на yandex.ru.

Если почта отправителя находится на gmail.com, то строка получения экземпляра класса SmtpClient будет выглядеть вот так SmtpClient sc = new SmtpClient("smtp.gmail.com", 587) и необходимо разрешить доступ ненадежным приложениям к аккаунту.

Если на mail.ru, то вот так SmtpClient sc = new SmtpClient("smtp.mail.ru", 25);

Если ящик отправителя находится где-то ещё, то вам придется воспользоваться интернетом, что бы определить какой smtp – сервер и порт нужно использовать.

Немного усовершенствуем наш код. Так как при отправке письма может всякое произойти. То добавим блок try…catch к строке sc.Send(mm).

|  |
| --- |
| try  {  sc.Send(mm);  }  catch (Exception ex)  {  MessageBox.Show("Невозможно отправить письмо " + ex.ToString());  } |

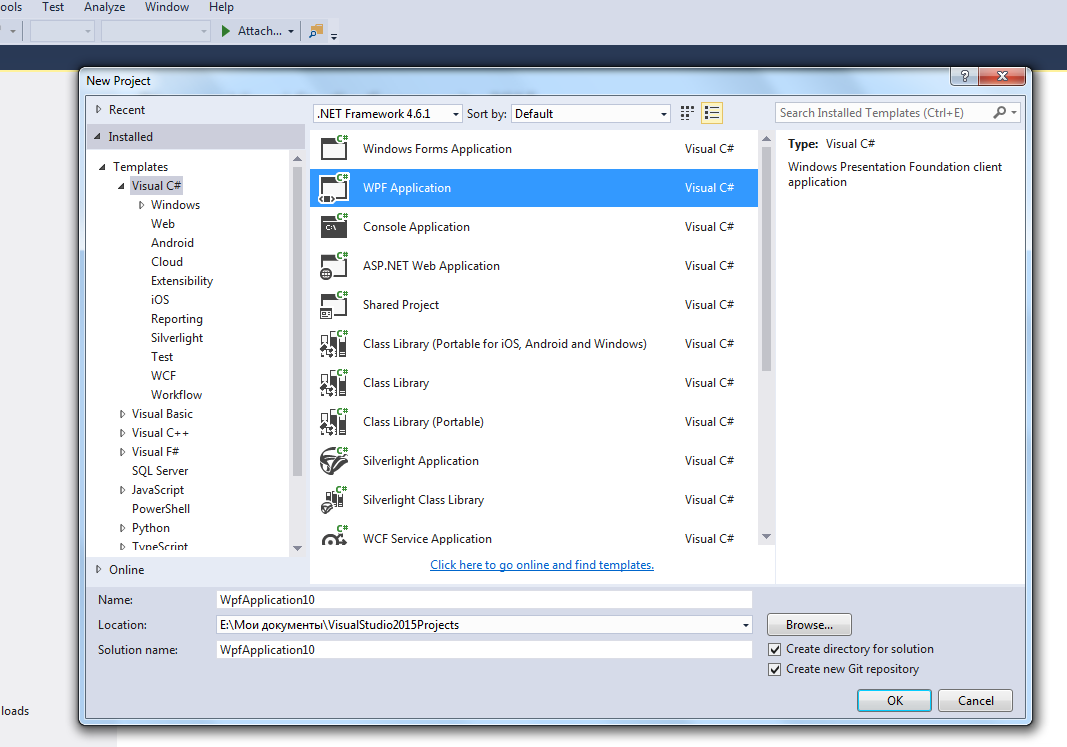
# Введение в WPF

Пришло время проверить тот код, который мы написали выше и посмотреть, отправиться ли письмо из нашей программы.

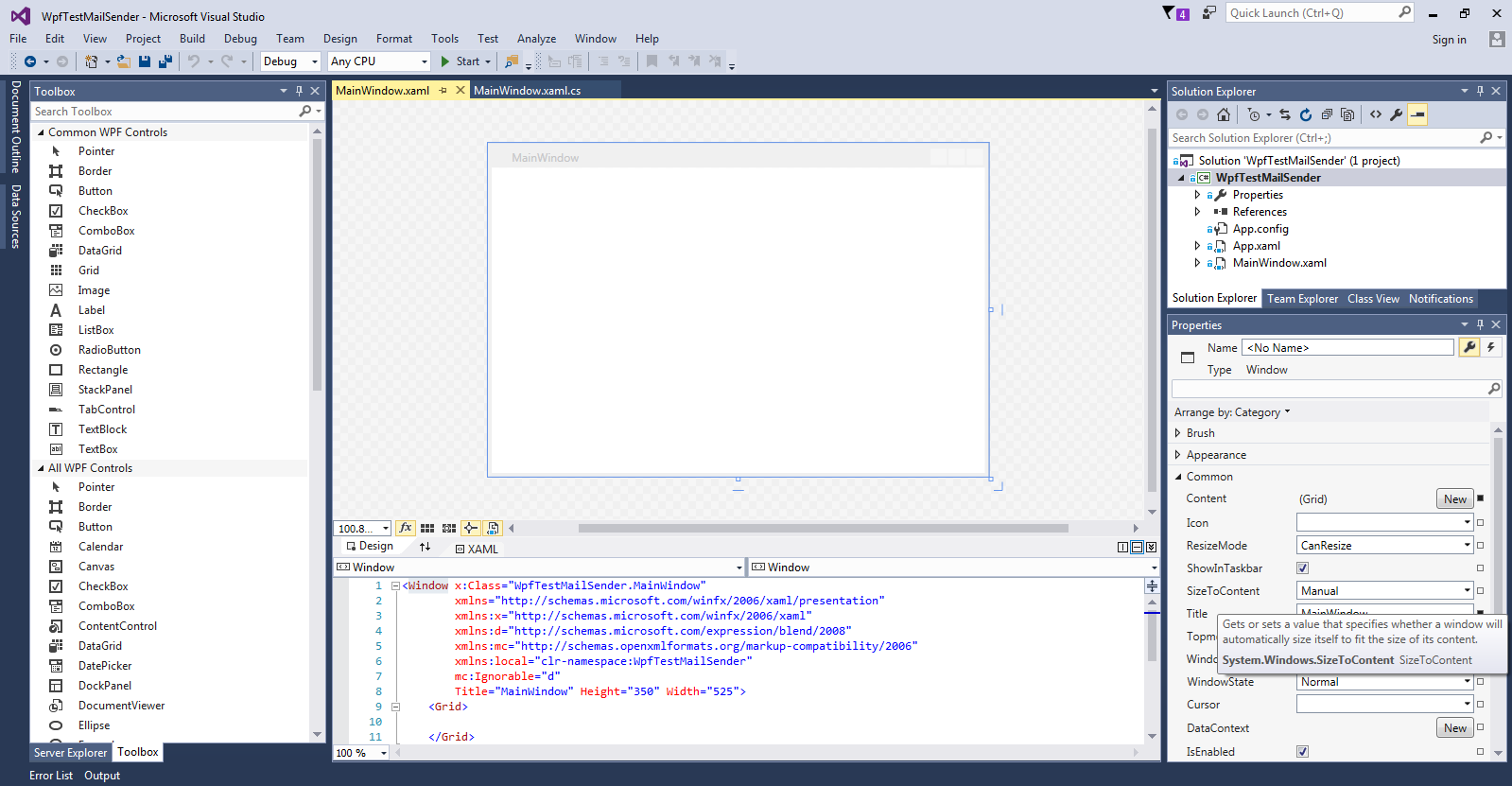
Не будем создавать консольное приложение, а перейдем сразу к WPF.

Если мы разрабатываем клиентское приложение под Windows, то на сегодняшний момент у нас на выбор есть две технологии Windows Forms и WPF (Windows Presentation Foundation). WPF – это новый интерфейс прикладного программирования, для управления слоем презентации пользовательсокого приложения.

Приступим. Отрываем Visual Studio, кликаем New Project (новый проект) и выбираем WPF Application

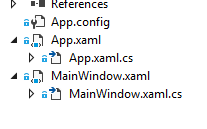


Не ленимся и переименуем проект. Например, назовем его WpfTestMailSender. Нажимаем ОК и видим структуру проекта

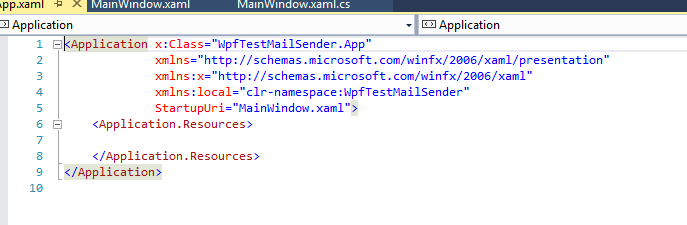


Сгенерированная структура состоит из файлов MainWindow.xaml и App.xaml.

Если развернуть узлы проекта соответствующие этим файлам, то мы увидим исходные файлы .cs



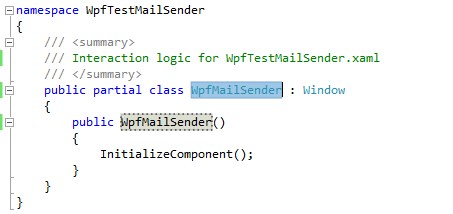
Файл App.xaml определяет первоначально загружаемый файл. Если мы откроем файл App.xaml то увидим элемент Application XAML, в котором в атрибуте StartupUri, по умолчанию прописан MainWindow.xaml, то есть первоначально загружаемый XAML файл.



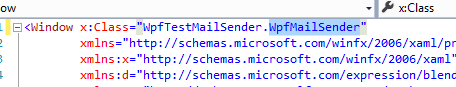
Начнем с того, что переименуем файл MainWindow во что-то другое, например на WpfMailSender.

Переименуем файл MainWindow.xaml на WpfMailSender.xaml. Исходный файл .cs переименуется автоматически. Делайте так, что бы название файла не совпадало с названием проекта и соответственно с названием основного Namespace. Самые любопытные могут проверить, какие ошибки при этом появятся.

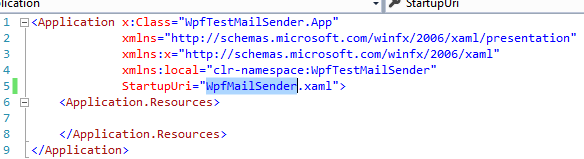
Переименуем название класса в файле .cs



Переименуем в файле .xaml значение атрибута x:Class элемента Window



Переименуем в файле App.xaml атрибут StartupUri



Если мы посмотрим на окно визуального дизайнера основного файла проекта WpfMailSender.xaml. То увидим, что по сравнению с технологией Windows Forms, почти ничего не поменялось. Мы точно так же можем мышью перетаскивать контролы из Toolbox на наше окно, но появилось одно новшество.

Посмотрите на область предварительного просмотра и редактирования, и увидите область, для редактирования кода XAML.

Не смотря на внешнее сходство, технология WinForms отличается от технологии WPF. На WinForms программист, как правило, с помощью визуального конструктора определяет пользовательский интерфейс, при этом генерируется код в файле с расширением .designer. Таким образом, интерфейс определяется кодом на языке C#. На WPF для этой цели есть язык XAML.

XAML или Extensible Application Markup Language – это язык разметки, для определения пользовательского интерфейса. XAML разрабатывался на основе языка XML и внешне он очень на него похож, и благодаря которому у приложения появилось много новых возможностей, сделать более красивый дизайн, добавить 3D графику, а так же другие возможности. Но код, написанный на C#, практически не отличается, если бы вы делали приложение на WinForms.

## Знакомсво с XAML

Язык XAML покажется вам простым, если вы знакомы с языком XML, так как синтаксис этих языков практически совпадает.

|  |
| --- |
| <Window x:Class="WpfTestMailSender.WpfMailSender"  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  xmlns:local="clr-namespace:WpfTestMailSender"  mc:Ignorable="d"  Title="MainWindow" Height="350" Width="525">  <Grid>    </Grid>  </Window> |

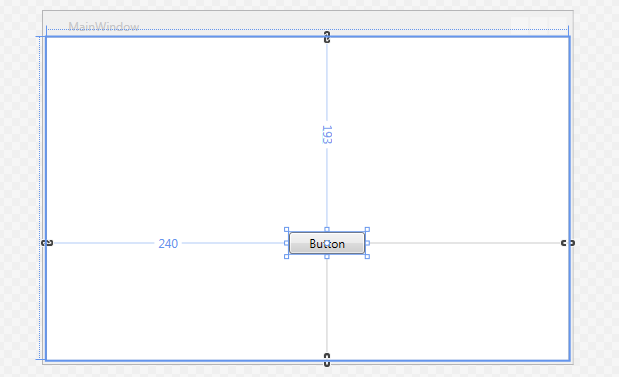
Выше XAML код, который студия создала автоматички при создании проекта. Единственное, что мы сделали, это заменили название класса основного окна MainWindow на WpfMailSender.

XAML может иметь только один корневой узел, в данном случае это Window. Элемент, который в нем содержится это Grid, в принципе в XAML элементы вкладываются в корневой элемент и друг, в друга определяя, как выглядит пользовательский интерфейс и что в нём содержится. Наименования атрибутов и элементов зависимы от регистра. Каждому элементу XAML соответствует класс .NET, а атрибутам события или свойства этого класса. Атрибут x:Class содержит название класса, который соответствует корневому узлу. Корневой узел, содержит различные свойства Height, Width, Title и префиксы пространств имен. Пространства имен объявлены атрибутом xmlns и соответствуют определению пространств имен в .cs файлых при помощи using.

## Знакомство с контролами WPF.

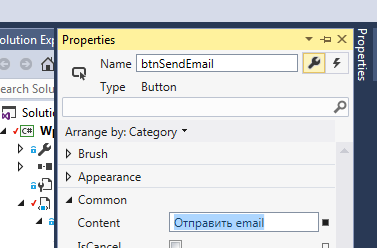
Пришло время поместить кнопку (button) на форму нашего приложения, определить метод по событию нажатия этой кнопки и поместить код, который рассылает письма, в тело этого метода.

Поместим кнопку на наш проект



Заходим в Property и видим, что свойства Text у кнопки нет, вместо него есть свойство Content. Свойство Name осталось и также находится на самом верху. Заменим наименование контрола (свойство Name) с «Button», на «btnSendEmail». Что бы было понятнее разбираться с нашей программой, придумываем наименование контролам таким образом - вначале идет краткое обозначение контрола, для кнопки это btn, затем, что он делает, каждое слово начинается с больших букв, без пробелов и подчеркиваний.

Теперь заменим надпись на кнопке (свойство Content) с «Button» на «Отправить email».



Теперь посмотрите на XAML код, внутри элемента Grid, появился элемент Button.

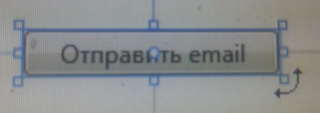
|  |
| --- |
| <Grid>  <Button x:Name="btnSendEmail" Content="Отправить email" HorizontalAlignment="Left" Margin="240,193,0,0" VerticalAlignment="Top" Width="124"/>  </Grid> |

Кстати в XAML коде вы можете заменить название кнопки и другие свойства.

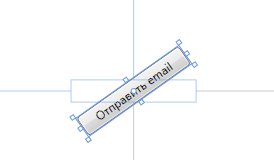
Поиграйтесь с кнопкой, поменяйте ей различные свойства, такие как наименование, длина, ширина, расположение на экране и др. самыми разными способами

* На экране в дизайнере
* В XAML коде
* В свойствах Property

Если вы подведете курсор мыши к любому углу кнопки, то увидите как курсор стал изогнутым, как на рисунке ниже.

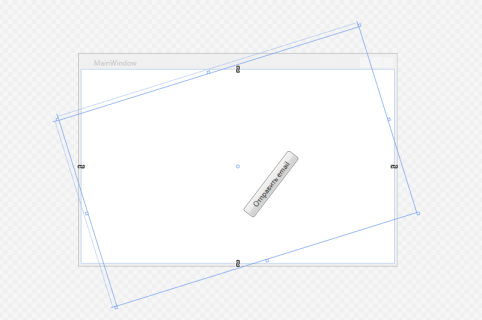


Таким образом, вы можете повернуть кнопку под углом.



Посмотрите, как поменяется при этом XAML код.

Точно также мы можем повернуть и весь Grid



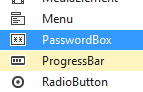
Нам не нужны под углом ни Grid, ни кнопка, давайте вернем их в обычное «ровное» состояние.

Для этой цели просто удалим подузлы Grid.RenderTransform и Button.RenderTransform.

Кстати если хотите работать только в дизайнере, то можете два раза кликнуть по вкладке Design. Если только с XAML, то по вкладке XAML. Что бы вернуть отображение и дизайнера и кода XAML кликните справа по кнопке.

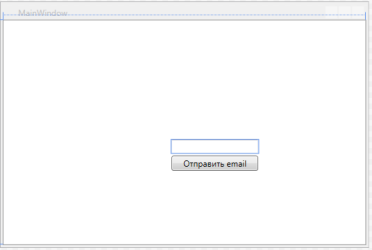


Добавим на форму ещё один контрол – PasswordBox



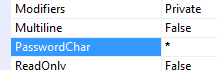
Этот контрол нужен для ввода пароля к нашему ящику, при отправке писем. Не нужно писать свои пароли, да ещё и рядом с логином, даже в таких вот тестовых приложениях, вдруг увидит какой-нибудь недоброжелатель или вирус.

Свойство Name менять этому контролу не будем, пусть так и называется «passwordBox».



Теперь что бы почувствовать разницу, создайте точно такой же проект только на Windows Forms.

Киньте на форму кнопку, переименуйте свойство Name также btnSendEmail и назовите эту кнопку «Отправить email». Контрола PasswordBox в Windows Forms нет, поэтому киньте на форму обычный TextBox, поменяйте ему свойство Name с «textBox1» на «passwordBox», найдите свойство PasswordChar и поставьте там символ \* или любой другой символ, которым вы бы хотели скрыть свой пароль.



Теперь добавляем метод обрабатывающий событие клик кнопки «отправить email» в обоих приложениях. Оба приложения могут добавить обработчик, два раза кликнув, по кнопке в дизайнере, либо добавив событие Click в списке событий в свойствах кнопки.

В приложении WPF появился ещё один способ добавить обработчик события. Просто добавьте в XAML код в свойства кнопки Button вот такой атрибут Click="btnSendEmail\_Click".

|  |
| --- |
| <Grid  <Button x:Name="btnSendEmail" Content="Отправить email" HorizontalAlignment="Left" Margin="240,193,0,0" VerticalAlignment="Top" Width="124" Click="btnSendEmail\_Click" />  <PasswordBox x:Name="passwordBox" HorizontalAlignment="Left" Margin="240,170,0,0" VerticalAlignment="Top" Width="124"/>  </Grid>  </Window> |

В этом случае придется вручную добавить код обработчика в файл .cs.

|  |
| --- |
| private void btnSendEmail\_Click(object sender, RoutedEventArgs e)  {  } |

Теперь в двух проектах нужно добавить две директивы using

using System.Net;

using System.Net.Mail;

А в тело обработчика btnSendEmail\_Click добавить тот код, который мы писали в самом начале нашего урока.

Ниже я приведу его с некоторыми изменениями

|  |
| --- |
| List<string> listStrMails = new List<string> { "testEmail@gmail.com", "email@yandex.ru" }; // Список email'ов //кому мы отправляем письмо  string strPassword = passwordBox.Password; // для WinForms - string strPassword = passwordBox.Text;  foreach (string mail in listStrMails)  {  // Используем using, что бы гарантированно удалить объект MailMessage после использования  using (MailMessage mm = new MailMessage("sender@yandex.ru", mail))  {  // Формируем письмо  mm.Subject = "Привет из C#"; // Заголовок письма  mm.Body = "Hello, world!"; // Тело письма  mm.IsBodyHtml = false; // Не используем html в теле письма  // Авторизуемся на smtp сервере и отправляем письмо  // Оператор using гарантирует вызов метода Dispose, даже если при вызове  // методов в объекте происходит исключение.  using (SmtpClient sc = new SmtpClient("smtp.yandex.ru", 25))  {  sc.EnableSsl = true;  sc.Credentials = new NetworkCredential("sender@yandex.ru", strPassword);  try  {  sc.Send(mm);  }  catch (Exception ex)  {  MessageBox.Show("Невозможно отправить письмо " + ex.ToString());  }  }  }//using (MailMessage mm = new MailMessage("sender@yandex.ru", mail))  }  MessageBox.Show("Работа завершена."); |

Теперь вам остается заменить список email’ов кому вы хотите отправить письмо.

И свой email от кого вы хотите запустить рассылку. Не забудьте выбрать правильный smtp сервер.

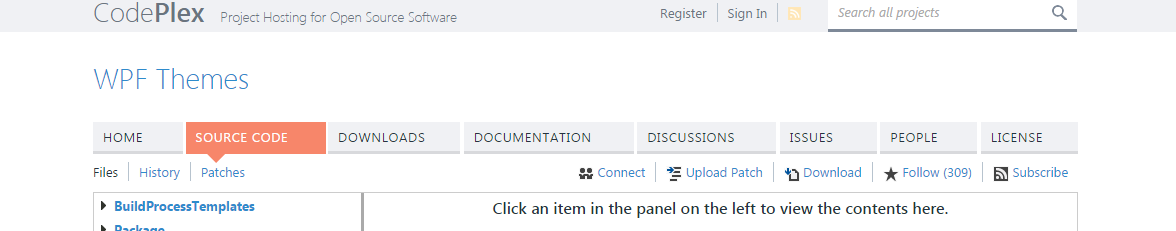
Открываю тестовые email’ы, и я вижу письма с темой «Привет из C#» внутри текст «Hello, world!». Ура всё работает, письма отправляются. Причем отправляются они как из приложения основанного на технологии WPF, так и из приложения основанного на WinForms.

## Изменение стиля приложения WPF.

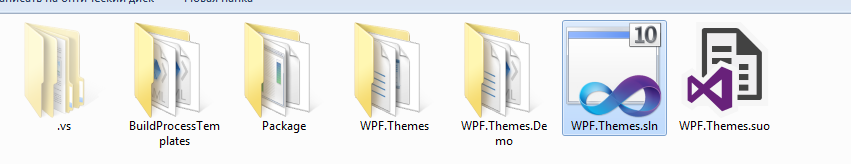
Давайте подчеркнем преимущество технологии WPF и добавим какой-нибудь красивый стиль для нашего приложения.

Мы можем создать стиль в отдельном XAML файле и привязать его к свойству Resources любого элемента или даже всему приложению. Для всего приложения мы добавляем стиль в файле App.xaml в элементе Application в свойстве Application.Resources.

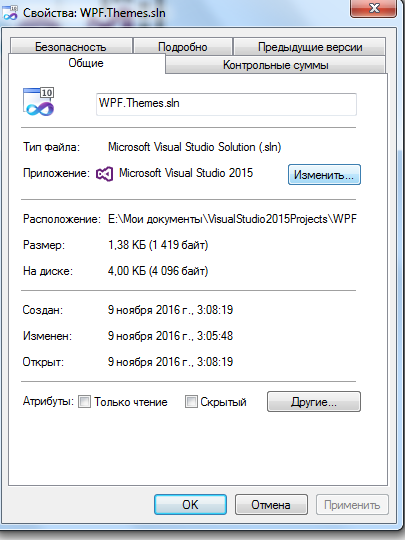
Давайте скачаем целую библиотеку стилей или тем (theme) с сайта <http://wpfthemes.codeplex.com>



Заходим на сайт и переключаемся на вкладку SOURCE CODE, затем кликаем на кнопку Download и загружаем проект с темами. Распаковываем проект.



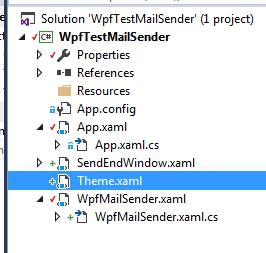
В папке WPF.Themes мы найдем список тем. Если откроем проект WPF.Themes.sln и запустим его, то можем посмотреть, как внешне выглядят разные темы, что бы выбрать то, что нравится. Если вдруг, при клике на WPF.Themes.sln, проект не хочет открываться, тогда нажимаем на него правой кнопкой мыши, выбираем «Свойства» и нажимаем на кнопку «Изменить» и принудительно задаем, чем открывается приложение.



Не буду описывать, как устроено приложение WPF.Themes, если оно у вас откроется в вашей версии Visual Studio и запуститься, то разберетесь сами. Заодно можете оценить код приложения и оформление контролов.

Наша цель, выбрать тему и добавить её на наше приложение.

Отрываем папку «WPF.Themes», находим внутри неё папку «ShinyBlue», и копируем файл Theme.xaml в корневую папку нашего проекта.



И добавляем в файл App.xaml, элемент Application, свойство Application.Resources, вот такой вот код.

|  |
| --- |
| <ResourceDictionary>  <ResourceDictionary.MergedDictionaries>  <ResourceDictionary Source="Theme.xaml"/>  </ResourceDictionary.MergedDictionaries>  </ResourceDictionary> |

Вставьте код, и посмотрите, что получилось. Если запустить приложение, то можете увидеть, что оно стало синеватым.

Теперь уберите новый код из файла App.xaml и добавьте его к Resources кнопки btnSendEmail нашего основного файла WpfMailSender.xaml.

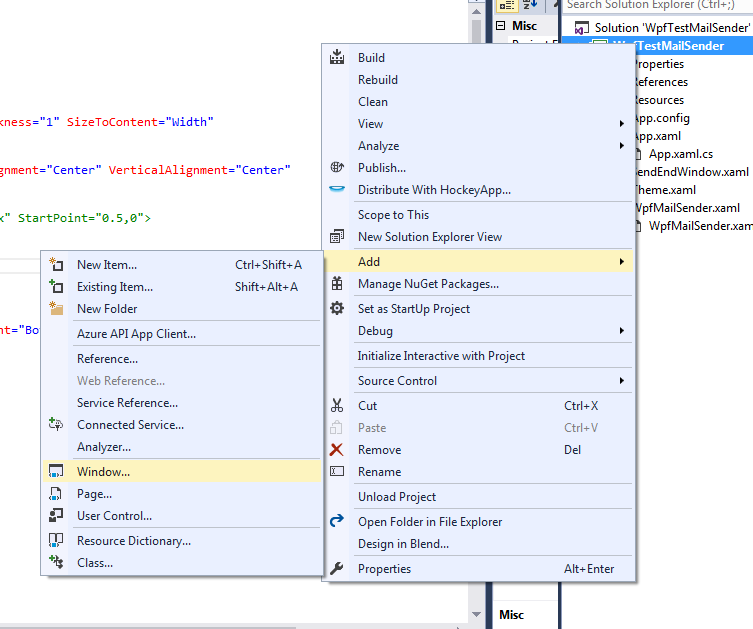
|  |
| --- |
| <Button x:Name="btnSendEmail" Content="Отправить email" HorizontalAlignment="Left" Margin="240,193,0,0" VerticalAlignment="Top" Width="124" RenderTransformOrigin="0.5,0.5" Click="btnSendEmail\_Click" >  <Button.Resources>  <ResourceDictionary>  <ResourceDictionary.MergedDictionaries>  <ResourceDictionary Source="Theme.xaml"/>  </ResourceDictionary.MergedDictionaries>  </ResourceDictionary>  </Button.Resources>  </Button> |

Теперь вы можете увидеть, что синеватой осталась только одна кнопка, поле для ввода пароля, не отдаёт синевой при наведении мыши.

Добавить стиль, можно на все главное окно нашего приложения, добавив тот же самый узел ResourceDictionary в подузел Window.Resources узла Window. В этом случае мы оставляем новый стиль, только на все контролы основного окна нашего приложения. Так как других окон у нашего приложения нет, то давайте создадим окно вместо **MessageBox.Show("Работа завершена.").**

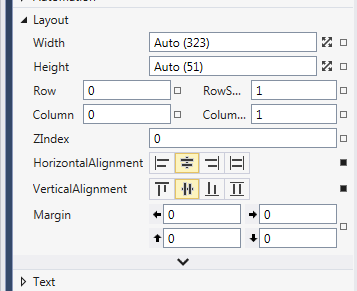
В любом случае MessageBox выглядит не красиво и на него новый стиль не действует и поэтому лучше их заменять своими окнами с сообщениями.

Кликнем правой кнопкой мыши в Solution Explorer по названию проекта и выберем добавить новое окно.



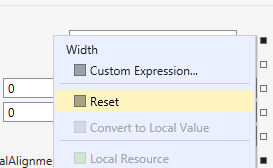
И назовем его SendEndWindow. Кинем на это окно Label.

Теперь идем вправо в Properties. Меняем название контрола, назовем его (поле Name на самом верху) lSendEnd, поле контент (Content) заполним «Работа завершена». В категории Layout сделаем так, что бы лейбл находился в центре окна

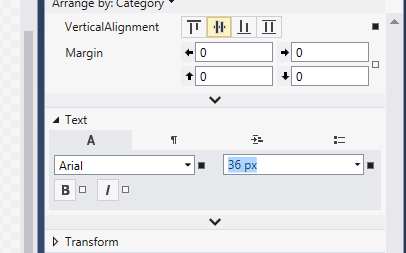


HorizontalAlignment и VerticalAlignment делаем по центру (как на рисунке).

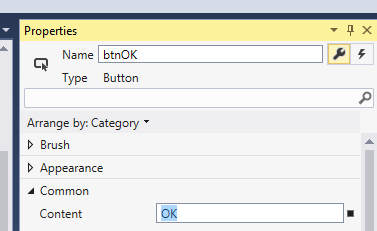
Свойства Width и Height делаем Auto (если они не авто, то нажимаем на маленькие квадратики спарава и в появившемся меню выбираем Reset).



В категории Text увеличим шрифт до 36, что бы надпись была крупнее.



Теперь кидаем кнопку на нашу новую форму. Свойство Name - btnOK, Content – OK.



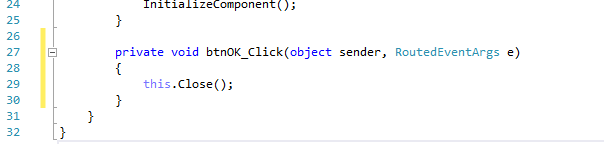
Теперь идем в категорию Layout и устанавливаем Margin (расстояния до границ родительского контейнера в данном случае окна SendEndWindow) 50 до правого края и 20 до нижнего края. Длину и ширину кнопки сделайте по своему усмотрению мышью.

Основное окно, сделаем так, что бы его размеры нельзя было поменять. Мы можем случайно кликнуть мышью не по основному окну, а по элементу Grid, который лежит на нем. Удостоверимся, что бы выбрали основное окно. Кликнем в xaml коде по узлу Window, прямо по слову Window, и увидим, как в дизайнере подсветится само окно, а в свойствах (Properties) появятся свойства основного окна. Если нам нужно выбрать любой другой элемент, то соответственно нужно кликнуть по названию другого элемента (Grid или Button, например).

Заходим в свойства (Properties), основного окна, заходим в категорию Appearance, выбираем WindowsStyle – None. Заходим в категорию Common, выбираем ResizeMode – NoResize, WindowStarupLocation – CenterOwner. Заходим в категорию Layout и делаем Width – 400, Height – 175.

И получаем красивое окошко для надписи «Работа завершена».

Теперь два раза кликаем по кнопке ОК (или в свойствах выбираем событие Click), и в обработчике добавляем строку this.Сlose();



Теперь заходим в файл WpfMailSender.xaml.cs и в самом низу обработчика

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

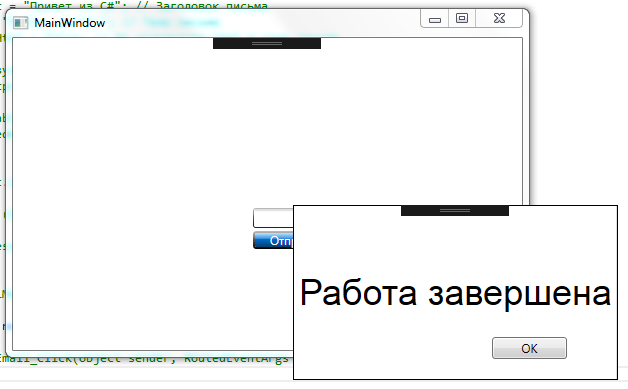
Вместо вызова MessageBox.Show("Работа завершена.");

Добавляем вот такой код.

|  |
| --- |
| SendEndWindow sew = new SendEndWindow();  sew.Show(); |

То есть мы создаем экземпляр нашего нового окна и открываем его.

Кстати пока мы отлаживаем наше новое окошко, можете закоментировать код по отправке писем и оставить только вызов нового окна.



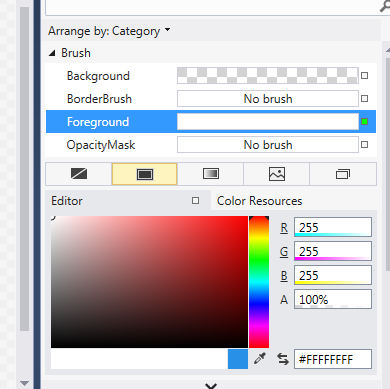
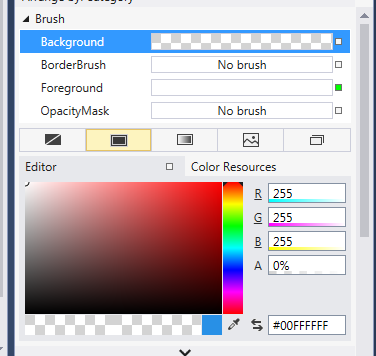
Как видим, дизайн нового окна черный и не красивый. Давайте вернем стиль на все приложение. Вернем в файл App.xaml, элемент Application, свойство Application.Resources наш стиль

|  |
| --- |
| <ResourceDictionary>  <ResourceDictionary.MergedDictionaries>  <ResourceDictionary Source="Theme.xaml"/>  </ResourceDictionary.MergedDictionaries>  </ResourceDictionary> |

И что мы теперь видим. Да всё красиво, за исключением, того, что пропал лейбл с надписью «Работа завершена». Да новое окно появляется, кнопка «ОК» синеватая, а вот надписи на окне никакой нет.

Видимо без танца с бубном, не обойтись.

К сожалению, в работе программистом, встречаются такие вот мелкие недостатки, которые отъедают часть времени. Здесь либо мы неправильно прикрутили новый стиль, либо в самом стиле что-то не так. Открываем свойства (Properties) элемента Lable, категорию Brush, которая как раз и отвечает за отрисовку цвета фона, цвета шрифта, границ и др.

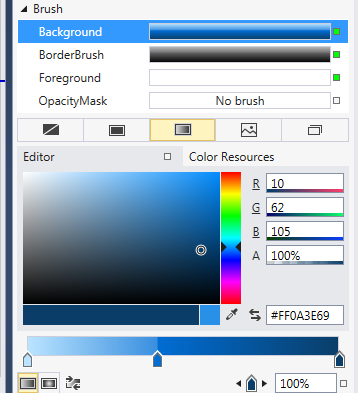


Если кликнуть на Background, то мы увидим прозрачный фон (#00FFFFFF), то есть фон лейбла будет совпадать с фоном основного окна. Если кликнем на Foreground, то увидим белый цвет (#FFFFFFFF), то есть цвет шрифта нашей надписи, будет тоже белый, как и цвет фона.

И для способа решения этой проблемы, надо либо сделать фон каким-нибудь цветом, что бы на нем были видны белые буквы, либо сделать цветными буквы.

Мне больше нравится идея сделать цветными буквы. Если мы посмотрим на фон кнопок этой темы, то увидим синий переливающийся цвет.

Давайте посмотрим, как устроены свойства кноки, зайдем в категорию Brush.



Давайте сделаем таким же цветом буквы лейбла. Можно это сделать прямо в свойствах, но правильнее скопировать xaml код цветов фона кнопки.

Если мы заглянем в файл Theme.xaml и найдем оформление кнопки, то увидим вот такой вот код.

|  |
| --- |
| <LinearGradientBrush x:Key="NormalBrush" EndPoint="0.5,1" StartPoint="0.5,0">  <GradientStop Color="{StaticResource NormalBrushGradient1}" Offset="0" />  <GradientStop Color="{StaticResource NormalBrushGradient2}" Offset="0.41800001263618469" />  <GradientStop Color="{StaticResource NormalBrushGradient3}" Offset="0.418" />  <GradientStop Color="{StaticResource NormalBrushGradient4}" Offset="1" />  </LinearGradientBrush> |

Эти цвета нужно добавить в описание лейбла. Нужно только удалить x:Key=”NormalBrush”, всё остальное должно перекочевать в лейбл.

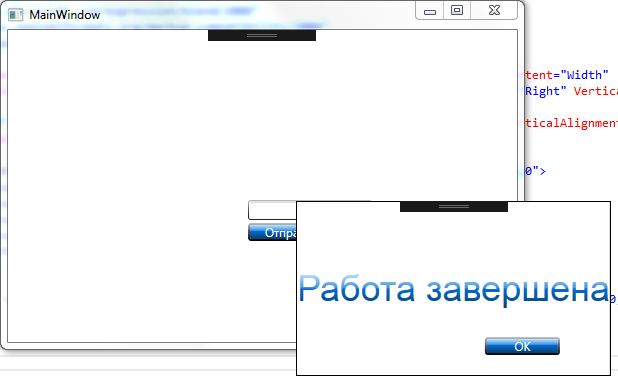
Сейчас описание лейбла выглядит так

|  |
| --- |
| <Label x:Name="lSendEnd" Content="Работа завершена" FontSize="36" HorizontalAlignment="Center" VerticalAlignment="Center" FontFamily="Arial"/> |

Мы сделаем вот так

|  |
| --- |
| <Label x:Name="lSendEnd" Content="Работа завершена" FontSize="36" HorizontalAlignment="Center" VerticalAlignment="Center"  FontFamily="Arial" >  <Label.Foreground>  <LinearGradientBrush EndPoint="0.5,1" MappingMode="RelativeToBoundingBox" StartPoint="0.5,0">  <GradientStop Color="#FFBAE4FF" Offset="0.013"/>  <GradientStop Color="#FF398FDF" Offset="0.41800001263618469"/>  <GradientStop Color="#FF006DD4" Offset="0.418"/>  <GradientStop Color="#FF0A3E69" Offset="1"/>  </LinearGradientBrush>  </Label.Foreground>  </Label> |

Запускаем проект



Всё выглядит красиво и в одном стиле.

# Итог

Подведем итог нашей работы.

Мы проверили код по отправке писем из приложения созданного на языке C#, это вполне возможно.

Мы познакомились с технологией визуального интерфейса WPF. И сравнили с технологией Windows Forms.

Как итог могу сказать, что основным отличием WPF от WinForms – это создание красивых стилей, украшений проекта. Да я уверен, что если как следует порыться в интернете и «станцевать с бубном», то мы можем и приложение, написанное на WinForms сделать более красивым и натянуть на него какой-нибудь свой корпоративный стиль. Точно могу сказать, что для целей красивого приложения WPF подходит больше. Остаётся вопрос, какая технология лучше, и какую и когда лучше применять. Мои рекомендации такие:

* WinForms надо знать и уметь применять, так как, устроившись в какую-нибудь компанию, наверняка там остались старые программы, работающие под WinForms, которые нужно поддерживать. Но не нужно сильно вкладываться в изучение Windows Forms, эту технлогию достаточно знать на поверхностном уровне.
* WPF следует изучать более серьезно, и если создаете новое приложение, то делайте его с использованием именно этой технологии, даже если не нужно создавать красивый дизайн. WPF более современная технология и вы будете более востребованы на рынке труда если хорошо её освоете.

# Рекомендации по созданию приложений.

Мы только, что написали приложение, которое рассылает электронные письма. И у нас есть два пути возможного дальнейшего развития.

1. Мы можем продолжать дальше развивать наше приложение и «по ходу пьесы» дополнять разные фичи и новые возможности в нашу программу, надеясь, что из неё получится что-то хорошее.
2. Сесть, хорошенько подумать, что мы хотим от нашей программы.
   1. Какие цели мы преследуем, что хотим получить
   2. Как должен выглядеть интерфейс приложения, какой дизайн
   3. Продумать структуру программы

То есть написать ТЗ (техническое задание).

Первый путь, к сожалению, приводит к спагетти-коду, то есть когда в вашем приложении «чёрт ногу сломит», ничего не понятно, какой метод, когда вызывается, какая переменная и для чего нужна. И, конечно же, отсутствие чёткой цели, приводит к непонятному результату. Поэтому выбираем второй путь.

## Постановка цели.

То с чего обычно начинается создание нового проекта или даже нового приложения это с постановки цели и составления технического задания. В компаниях обычно для этого собираются митинги, устраиваются мозговые штурмы, заказчикам отправляются брифы, анкеты. Если приложение не очень большое и под силу одному программисту, то достаточно просто посидеть и подумать, каким оно должно быть и как оно должно быть устроено.

Предвижу вопрос, почему урок не начался с пункта постановка цели. Ответ простой. Урок построен так, каким бы он был в реальной жизни. Вначале я бы изучил задачу, действительно ли можно отправить письмо программно. Затем хотя бы поверхностно изучил бы новую технологию. А затем, когда я уверен, что решить вопрос рассылки средствами .NET с использованием технологии WPF возможно, уже тогда приступил бы к продумыванию цели.

### С чего начать. План.

Будем следовать небольшому плану, по созданию программного продукта

1. Первоначально надо ответить на вопросы, для чего нужно наше приложение, кто его будет использовать, что оно будет делать.
2. Определите структуру будущего программного продукта.
3. Продумайте интерфейс или внешний вид приложения и нарисуйте его, например на бумаге.
4. Создание прототипа программы, который содержит весь графический интерфейс (кнопки, диалоговые окна и др.), но не обладает функциональной полнотой.
5. Непосредственно сам кодинг, то есть добавление функциональных блоков, методов и т.п. в программу

### Первый пункт плана.

**Начнем с первого пункта**

Так что откиньтесь на спинку кресла и подумайте, что вы хотите от приложения, что оно должно делать.



Не возбраняется думать с открытыми глазами, или даже расхаживая по комнате взад-вперед, или даже гуляя в парке. А можно просто начать записывать свои мысли.

И вот какое приложение мы будем делать

1. Приложение «Рассыльщик», занимается рассылкой электронных писем по адресатам из базы данных адресов.
2. В базе данных хранятся адреса (email) адресатов, имена и другая информация.
3. В базе данных хранятся адреса (email) от чьего имени делается рассылка, так же smtp серверы и порты.
4. В базе данных хранятся тексты писем для отправки, названия писем и ссылки на файлы, которые нужно к письму прикрепить
5. В базе данных хранится статистика, кому и что мы отправили.
6. В приложении можно выбрать из базы, набор адресатов, кому отправляется письмо. Можно добавить, редактировать и удалить адресата. Можно загрузить из файла несколько адресатов сразу.
7. В приложении можно выбрать адрес от чьего имени отправляется письмо и smtp сервер. Можно добавить, редактировать и удалить отправителя.
8. В приложении можно составить календарный план рассылки, когда и что мы отправляем.
9. В приложении можно отправить письмо сразу по нажатию кнопки.
10. В приложении есть напоминание, о том, что нужно составить и осуществить рассылку к определенной дате и времени.
11. В приложении можно посмотреть статистику, отправленных писем.
12. Использовать приложение вы будете сами, для рассылки писем, а так же для демонстрации потенциальному работодателю на собеседовании.

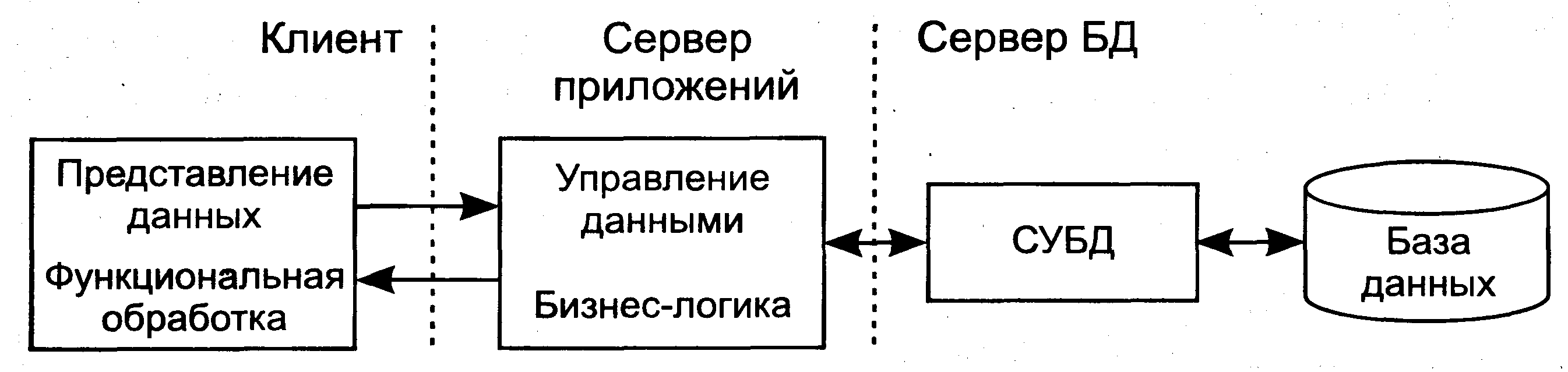
### Второй пункт плана.

**Теперь перейдем ко второму пункту нашего плана**

Нужно продумать внутреннюю структуру приложения.

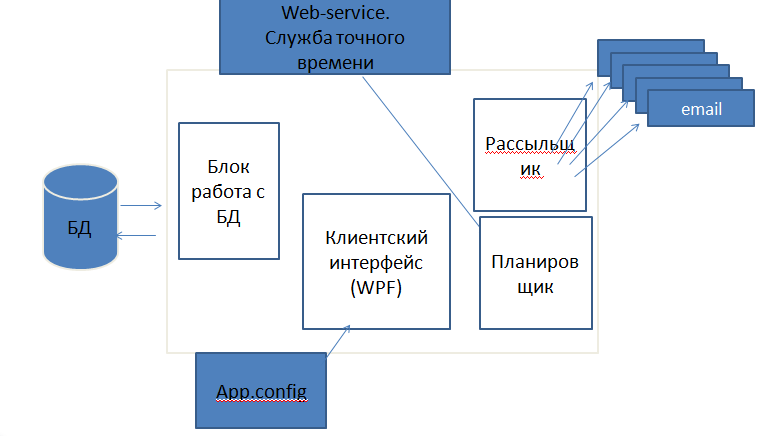
Надо отдавать себе отчет, что в процессе работы над приложением, оно может поменяться, как структура, так и внешний вид, в зависимости от ситуации. Но мы постараемся сделать так, что бы на самом начальном этапе все было сделано качественно.

Если мы заглянем в интернет и начнем изучать архитиктуру различных приложений, то увидим множество рекомендаций по созданию трехзвенной архитектуры.



То есть, существуют клиентское приложение, которое установлено на компьютере клиента, так называемый «тонкий клиент», на котором находится только интерфейс и больше ничего. Он в свою очередь отправляет задание серверу приложения, на котором находится вся бизнес-логика, то есть он и занимается рассылкой писем. И отдельно находится сервер-базы данных, к нему в свою очередь обращается сервер бизнес-логики. Все сервера находятся на разных компьютерах.

Но так выглядит идеальный программный комплекс. Мы же сделаем немного попроще. У нас будет одно приложение, которое находится на одном компьютере, к которому присоединяется база данных.



Выше структура приложения, которое мы будет создавать. Состоит оно из нескольких блоков. По своей сути наше приложение тоже похоже на идеальный программный комплекс созданный по принципу трехзвенной архитектуры. Те блоки, которые находятся на отдельных серверах в идеальном программном комплексе, у нас находятся в одной программе в разных блоках.

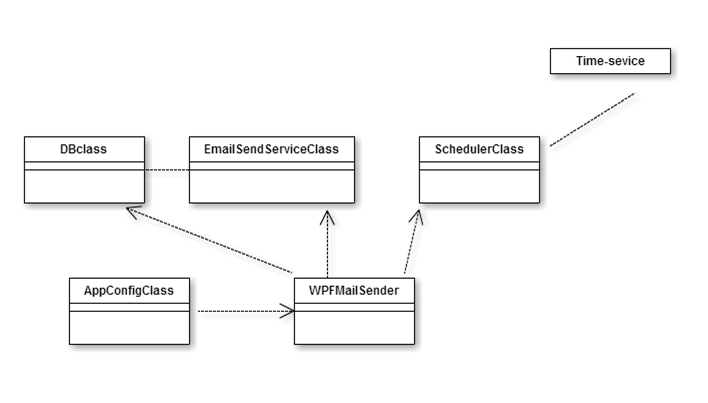
Каждый отдельный блок это отдельный класс или сборка (namespace) в зависимости от ситуации.

Вообще старайтесь делать так, что бы каждая логическая структура программы была отдельным классом. Сейчас если посмотреть на первоначальную схему, мы можем увидеть 5 различных классов и web-service служба точного времени.

* Блок работы с базой данных. То есть класс, который отвечает за подключение к базе данных, за получение данных из неё, за добавление новых данных (адресатов) и другие манипуляции с данными.
* Клиентский интерфейс. Класс главного окна программы. Возможно, появятся другие классы, для вспомогательных форм.
* Рассыльщик. Класс, который отвечает за логику программы. Именно он производит рассылку писем.
* Планировщик. Класс, который планирует рассылку, присылает уведомления.
* App.config – это файл в котором хранятся настройки приложения. И ему будет соответствовать отдельный класс, который будет отвечать на работы с этим файлом.
* Web-service – служба точного времени. Служба, к которой мы подключаемся, нужна в первую очередь, для планировщика. А также для того, что бы напомнить, как подключаются веб-службы.

Для того, что бы лучше понимать, как должно быть устроено наше приложение, рекомендую использовать UML схемы.

UML схемы удобно использовать когда нам нужно понять, как должен быть устроен наш программный продукт. Что он должен делать, и в какой период времени, как взаимодействует с пользователем. Рекомендую потратить немногно времени на изучение UML, таким образом, вы повысити свою стоимость на рынке труда. Тем более язык UML достаточно простой и наглядный и действительно упрощает понимание, что будет делать ваша программа или что делает уже существующее приложение.



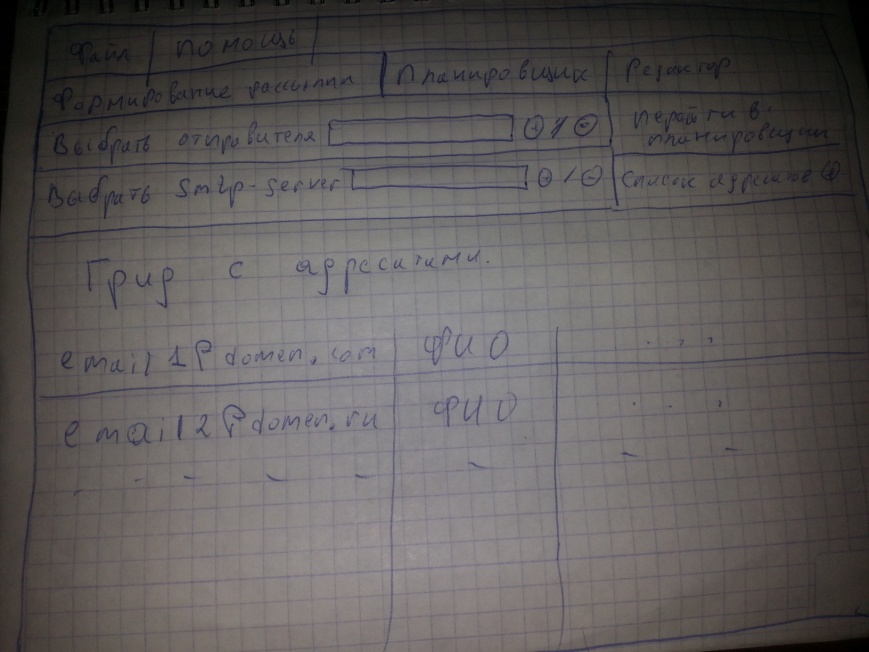
Выше представленна UML диаграмма классов. На ней изображены классы, которые предположительно будут использоваться в нашем приложении. Эта диаграмма похожа на схему структуры приложения, которая была представлена выше. При разработке серьезного корпоративного программного продукта, рисуется множество всевозможных схем, диаграмм, поэтому не жалейте времени на продумывание структуры приложения. Зачастую бывает так, что на продумывание цели программы, написание ТЗ, рисование всевозможных схем и диаграмм, уходит больше сил и времени, чем на сам кодинг, на непосредственно само программирование.

### Третий пункт плана.

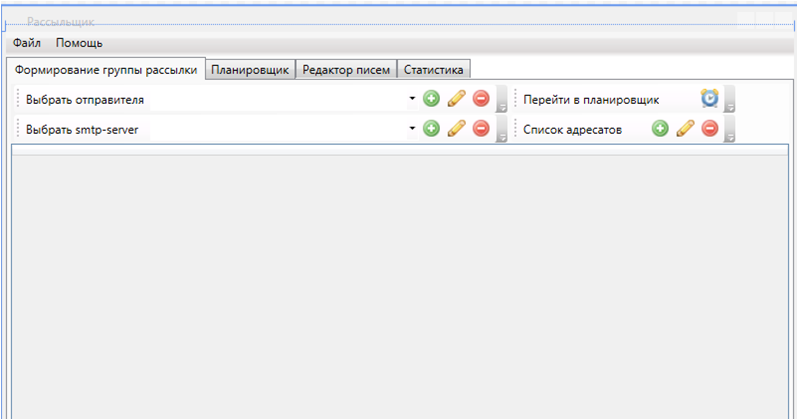
**Теперь третий пункт нашего плана**

Продумаем, как будет выглядеть наше приложение внешне. Как бы вам хотелось, что бы оно выглядело.

Для этой цели я рекомендую использовать бумагу и ручку. Просто взять и нарисовать в тетрадке формы, как они будут выглядеть, какие контролы будут использоваться. В тетради рисовать проще и быстрее, чем каком-нибудь редакторе на компьютере.



На рисунке представлено, как я рисовал на листочке в тетради, внешний вид приложения. К сожалению, сфотографировать эскиз приложения, качественно не получилось. Поэтому ниже показываю скриншот приложения, для лучшего пониманию, что мы будем делать.



То есть у нас будет основное окно, на котором расположен TabControl с вкладками.

На первой вкладке, будет возможность выбрать smtp сервер, и почтовый ящик от чьего имени вы будете отправлять письма и список адресатов.

На второй вкладке, будет планировщик с возможностью запланировать время и дату рассылки, напоминания о рассылках и возможность отправить письма сразу.

На третей вкладке это просто TextBox, где будет сохраняться текст письма.

И на последней вкладке таблица со статистикой, кому, когда и какие письма мы отправляли.

## Итог.

И вот на третьем пункте нашего плана заканчивается подготовительная работа по написанию нашего приложения. Мы сформулировали цель и теперь четко представляем, что нужно делать.

И оставшиеся четвертый и пятый пункт по созданию приложения – это уже непосредственно программирование.

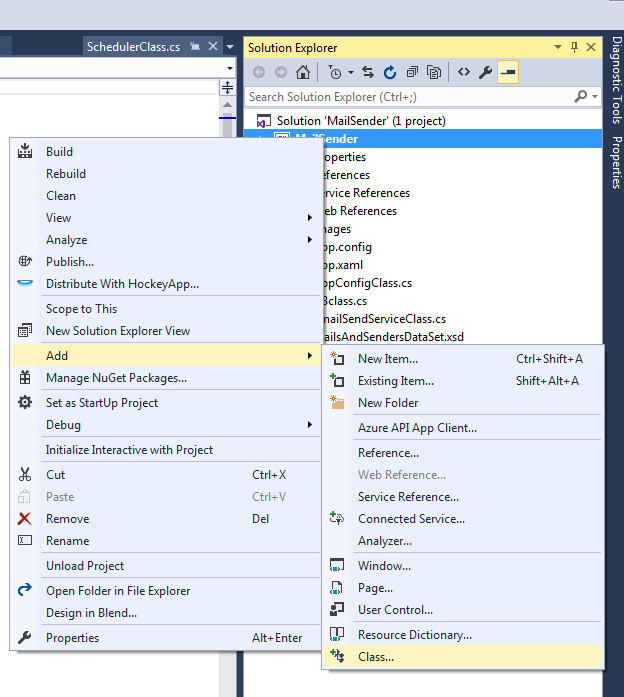
# Приложение «Рассыльщик». Интерфейс приложения и WPF.

Если мы посмотрим на план создания приложения, то у нас осталось два пункта 4й и 5й. Четвертый пункт – нам нужно создать прототип с интерфейсной частью. Пятый пункт – создание самого тела приложения, сам кодинг. Возможно, на этапе пятого пункта, мы будем добавлять что-то в интерфейс или каким-либо образом его изменять, но это не означает, что мы возвращаемся к четвертому пункту. Четвертый пункт – это создание прототипа, не означает, что мы должны с педантичной точностью добавить все контролы, формы и стиль. Нет, это примерный внешний вид и внутренний каркас нашего приложения. А вот заполнение внутреннего каркаса кодом и шлифовка внешного вида, а так же отладка – это уже пятый пункт нашего плана.

## Прототип приложения «Рассыльщик».

Открываем Visual Studio и создаем новый WPF проект. Назовем его MailSender.

В соответствии с нашей внутренней структурой у нас должно быть 5 классов. Сразу же их добавим в проект. Кликаем правой кнопкой мыши по названию проекта, нажимаем добавить, и добавляем класс. И повторяем этот процесс четыре раза (один класс у нас уже создан).



*Небольшое отступление по правилу наименования классов и переменных.*

* Классы начинаем с заглавных букв.
* Переменные с маленьких.
* Без пробелов и подчеркиваний.
* Каждое новое слово с заглавной буквы
* Названия подробные и наглядные.

В соответствии с правилами наименования классов добавим классы

* AppConfigClass – для работы с App.config
* DBClass – для работы с базой данных
* EmailSendServiceClass – класс который непосредственно отвечает за рассылку писем
* SchedulerClass – планировщик
* Класс, который отвечает за интерфейс, добавился автоматически при создании проекта

## Компоновочные элементы управления в системе WPF (панели).

Пришло время вернуться к визуальному интерфейсу.

В технологии WPF, как и в WinForms, есть компоновочные элементы управления или панели, которые нужны для позиционирования на своей поверхности других элементов управления. Панели являются невидимыми и определяют поверхность, на которой работает программист, так как поверхности предназначенной для позиционирования элементов управления в WPF нет. Компоновочные элементы управления в иерархии XAML файла находятся сразу под корневым узлом.

Существует несколько основных компоновочных элементов

1. Grid – по умолчанию он помещается на поверхность окна нашего приложения. Определяет гибкую область сетки, состоящей из строк и столбцов. Дочерние элементы [Grid](https://msdn.microsoft.com/ru-ru/library/system.windows.controls.grid(v=vs.110).aspx) могут быть расположены с помощью свойства Margin.
2. Canvas – использует абсолютную систему координат.
3. DockPanel - Определяет область, внутри которой можно упорядочить дочерние элементы по горизонтали или по вертикали друг относительно друга.
4. StackPanel - Располагает дочерние элементы в одну строку, которую можно ориентировать по горизонтали или по вертикали.
5. WrapPanel - размещает дочерние элементы в последовательности слева направо, перенося содержимое на следующую строку на границе содержащего поля. Дальнейшее упорядочивание происходит последовательно сверху вниз или справа налево в зависимости от значения свойства Orientation.

Разработчик может, по своему усмотрению, заменить компоновочную панель на ту, которая ему больше подходит в настоящий момент. Или разместить одну панель на поверхности другой.

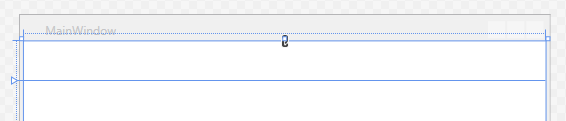
Панель Canvas в WPF используют крайне редко. Поэтому рассмотрим Grid.

Grid позволяет разделить свою поверхность на ячейки, в которых можно разместить другие контролы.

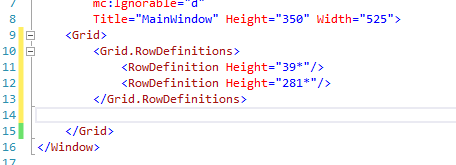
Grid разбивается на ячейки при помощи свойств RowDefinitions и ColumnDefinitions. Каждое из этих свойств это коллекции колонок и рядов, пересечения которых и определяют ячейки.

Рассмотрим на практике

Если мы подведем мышь к левому краю грида, то на самом гриде появится линия, которая определяет в каком месте мы размещаем границу ячейки. Кликните мышью в получившейся ячейке, состоящей из одного ряда, чтобы в ней можно было бы разместить контрол меню.

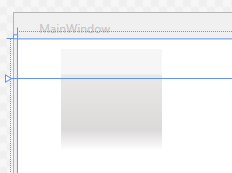


Посмотрите на XAML код, который появился.

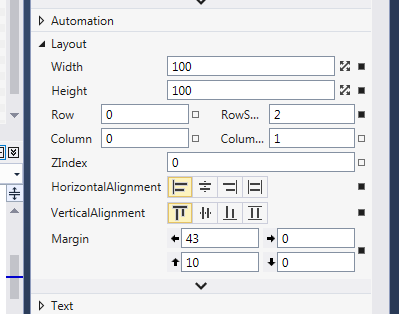


Если мы удалим свойства Height из RowDefinition, то черточка, определяющая ряды переместится в середину окна.

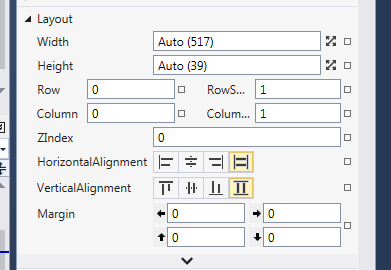
Но мы оставим определения ряда ближе к верху. И поместим на нашу форму контрол «меню». Выберете его из ToolBox и перетащите мышью на экран, что бы он попал на нужный нам ряд.



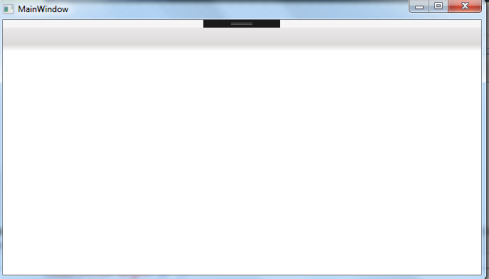
Заходим в свойства (properties) этого контрола и заходим в категорию Layout.



Видим, что под определением длины и ширины (Width и Height) появились определения, в каком ряду и в какой колонке находится наш контрол, в полях Row и Column соответственно слева. Справа находтся поля RowSpan и ColumnSpan, в них содержится определение, на какое количество рядов и колонок растягивается контрол. У нас сейчас на поверхности Grid, находится два ряда и всего одна колонка. Так как пункт меню будет находиться только в верхней ячейке, то мы оставляем значение Row 0 и значение Column тоже 0. RowSpan сейчас равен 2, нам же нужно, что бы меню находилось только в верхем ряду, поэтому занимать оно будет только ячейки верхнего ряда, поэтому устанавливаем свойству RowSpan значение 1. Поля Width и Height устанавливаем в значение Auto, то есть, нажимаем на маленький квадратик справа от поля и выбираем Reset. Теперь посмотрим на свойство Margin, оно определяет на каком расстоянии от границ ячейки будет находится наш контрол, так как мы хотим, что бы он занимал всё пространство ячейки, так же нажимаем на маленький квадратик справа и выбираем Reset. И последнее выбираем свойства HorizontalAlignment и VerticalAlignment выбираем значения крайние справа, то есть растягиваем по всей длине и ширине ячейки. Получаем вот такие свойства.



Теперь запускаем проект, нажимаем кнопку Start и мы видим окно с меню на самом верху.



Если мы поиграем с размерами окна, то увидим, что меню становится шире, если и окно становится шире. Но я нигде не видел, что бы меню меняло свой размер в зависимости от размера окна. Поэтому Grid в качестве основной панели для размещения контролов, для нашего приложения не совсем подходит. Grid очень хорош, если мы хотим сделать так называемый «резиновый» интерфейс. Мы можем разбить его на колонки и ряды и размещать контролы в ячейках, контролы в свою очередь будут менять размер вместе с размерами ячеек. Вообще Grid очень часто используют для WPF приложений, в качестве основной панели, поэтому рекомендую его изучить.

Сейчас давайте рассмотрим оставшиеся панели StackPanel, DockPanel, WrapPanel.

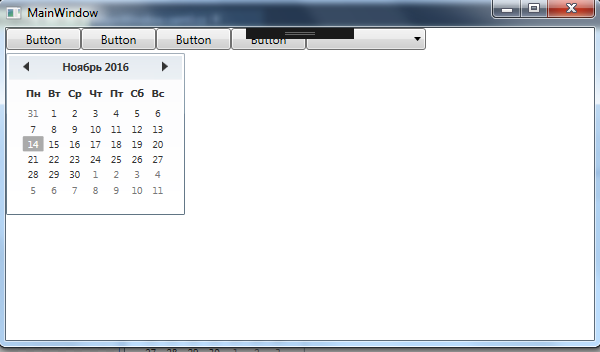
Удалим и Grid и Menu прямо из xaml кода.

Поместим вместо Grid в качестве основной панели WrapPanel. Мы можем просто в xaml коде добавить узел

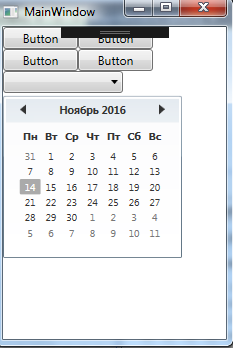
|  |
| --- |
| <Window x:Class="WpfApplication9.MainWindow"  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  xmlns:local="clr-namespace:WpfApplication9"  mc:Ignorable="d"  Title="MainWindow" Height="350" Width="525">  <WrapPanel >    </WrapPanel>  </Window> |

А можем просто кинуть панель из ToolBox на поверхность окна. В этом случае, нужно просто удалить всё лишнее (размеры и другие свойства) соответсвующее WrapPanel из xaml кода.

Теперь давайте кинем на поверхность WrapPanel несколько любых контролов. Посмотрите, как они удобно сами ложатся рядышком друг с другом.



Если мы растягиваем окно делаем его шире, то и контролы растягиваются в одну линию. Если мы сжимаем окно по длине, то и контролы как бы перемещаются влево.

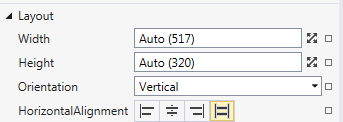


Интересное свойство у этой панели, но не совсем подходит для наших задач.

Рассмотрим StackPanel.

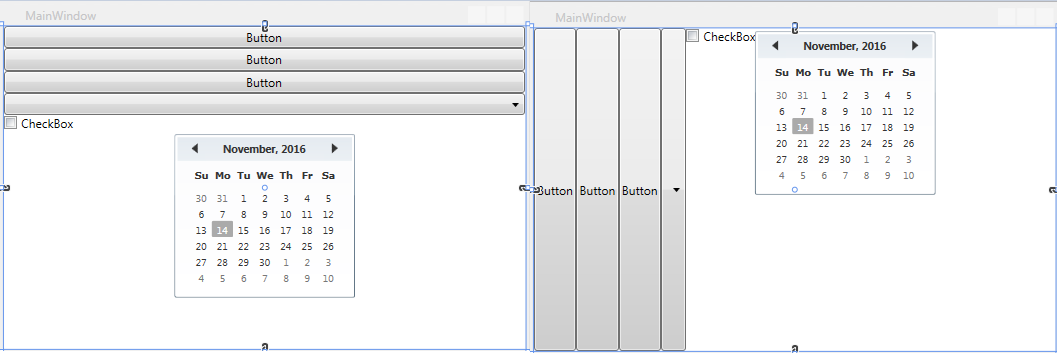
Давайте удалим WrapPanel и все контролы находящиеся на ней. И разместим StackPanel на нашей форме.

Если мы посмотрим на свойства StackPanel и заглянем в категорию Layout, то увидим там свойство Orientation



У меня по умолчанию стоит Vertical. Давайте кинем, так же как и в прошлый раз несколько различных контролов на панель.

Посмотрите, как они аккуратно ложатся сверху вниз. Теперь поменяйте Orientation с Vertical на Horizontal. Теперь контролы аккуратно размещаются слева направо.



Теперь запустите приложение и с вертикальной и с горизонтальной ориентацией, посмотрите что происходит.

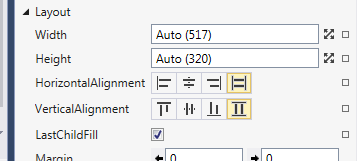
StackPanel удобно использовать, когда нам нужно аккуратно разместить несколько контролов подряд, либо слева направо, либо сверху вниз.

Причем некоторые панели можно использовать не только как основные, но и как вспомогательные. Например, если в качестве основной панели использовать Grid, то одну из панелей (например StackPanel) можно разместить на той части Grid, где мы бы хотели разместить контролы определенным образом.

Рассмотрим DockPanel.

Точно также удаляем StackPanel и все контролы на ней. Размещаем DockPanel в качестве основной панели. DockPanel, определяет область, внутри которой можно упорядочить дочерние элементы по горизонтали или по вертикали, друг относительно друга.

Посмотрите на свойства панели категорию Layout.



Появилось свойство LastChildFill. То есть если мы выберем галочкой это свойство, то последний контрол который мы разместим на панели, заполнит всё оставшееся место.

Давайте поместим меню контрол на нашей форме. Кинем его на DockPanel и увидим, как он заполнил всю поверхность формы. Нам не нужно такоё большое меню. Поэтому идем в свойства меню в категорию Layout.

Свойство Width, если оно у вас не Auto делаем Reset.

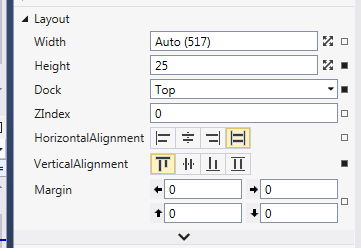
Свойство Height определяет высоту менюшки, так как по высоте я никогда не видел, что бы меню меняло свои размеры, оно не может быть Auto, поэтому устанавливаем его 25.

Появилось свойство Dock, выбираем Top.

HorizontalAlignment, делаем растянутым по всей ширине.

VerticalAlignment, делаем Top.

Margin, если значения отличны от 0, то делаем Reset.



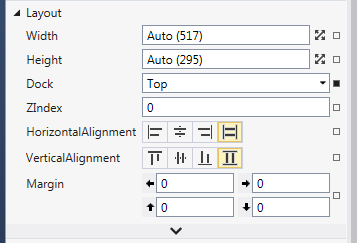
Теперь добавляем на нашу форму TabControl, нам надо, что бы он заполнил всё оставшееся место на нашей форме. Идем в свойства

Свойствам Width и Height делаем Reset, что бы они были Auto.

К сожалению, у свойства Dock, нет значения Fill, как это было в Windows Forms. Вместо Fill у панели DockPanel есть свойство LastChildFill. Так что, у свойства Dock просто выбераем Top.

У свойств HorizontalAlignment и VerticalAlignment выбираем Stretch, то есть крайнее справа свойство, означающее, что контрол будет, растянут по максимальной длине и ширине.

Свойству Margin, если там не все нули, делаем Reset.



Теперь запустите и посмотрите, как выглядит окно нашего приложения. Именно DockPanel мы и будем использовать в качестве основной панели.

# Итог.

Подошел к концу сегодняшний урок. Сегодня мы выполнили много интересных задач.

* Проверили возможность рассылки писем средствами C#.
* Познакомились с технологией WPF,
* сравнили технологию WPF с Windows Forms.
* Ознакомились с порядком действий, при разработке нового программного продукта
* Начали работу над прототипом нашего приложения «Рассыльщик»

# Домашнее задание.

1. Посмотрите внимательно на наше приложение, при помощи которого мы тестировали возможность отправлять электронные письма и начали изучать WPF. Посмотрите внимательно на код. В коде есть несколько моментов, которые простительны для теста, но не простительны для серьёзного приложения.
   1. Первый момент это жестко заданные переменные в коде. Например, new SmtpClient("smtp.yandex.ru", 25), в этой строке две жестко заданные переменные это "smtp.yandex.ru" – smtp сервер и 25 – порт для этого сервера. В коде много и других жестко заданных переменных, адреса почтовых ящиков, тексты писем, тексты ошибок и др. ***Теперь непосредственно задание, добавить в проект WpfTestMailSender возможность чтения парметров из конфигурационного файла.. В коде использовать эти переменные вместо жестко заданных.***
   2. Второй момент, который также простителен для тестовой программки и не желателен для более или менее серьезного приложения. Код, который описывает форму и код который занимается непосредственно рассылкой, содержится в одном классе. ***Теперь непосредственно задание, добавить к проекту WpfTestMailSender public class, назвать его, например EmailSendServiceClass с конструктором. Создать в этом классе методы (или метод), которые будут заниматься непосредственно рассылкой писем. Причём класс надо создать таким образом, что бы его можно было легко перенести в другой проект.***
   3. В коде присутствует MessageBox с выводом ошибки, в случае невозможности отправить письма. В принципе, MessabeBox это не криминал и даже в серьезных проектах они присутствуют, но всё-таки, окно со своим стилем выглядит лучше. ***Теперь непосредственно задание, по аналогии с формой, которая выводит сообщение «Работа завершена» создайте ещё одну для вывода текста ошибки, цвет текста ошибки пусть будет красным. Также добавьте кнопку «ОК» которая будет закрывать форму.***
2. Теперь задание на укрепление знаний технологии WPF.
   1. ***Добавьте на главное окно тестового проекта WpfTestMailSender, в любое место формы два контрола TextBox, одно для названия письма, второе для самого текста письма. И сделайте так, что бы название письма и текс письма брались из этих контролов.***
   2. ***Скачайте библиотеку стилей или тем (theme) с сайта*** [***http://wpfthemes.codeplex***](http://wpfthemes.codeplex)***. Как описано в главе*** Изменение стиля приложения WPF. ***Выберете любой стиль по своему усмотрению и замените стиль, сделайте так же, как было описано на нашем уроке.***
   3. ***Поиграйтесь с контролами тестового приложения WpfTestMailSender, поменяйте им свойства, поразмещайте в разных местах окна. Поменяйте основную панель Grid на другие панели, которые мы рассматривали на нашем уроке.***
3. ***Замените название основного окна и класса приложения MailSender, MainWindow на WpfMailSender, сделайте по аналогии с тем, как мы меняли название главного окна у тестового приложения WpfTestMailSender на нашем уроке.***

# Список использованной литературы

1. Н. Рандольф, Д. Гарднер, М. Минутилло, К. Андерсон. Visual Studio 2010 для профессионалов
2. MSDN
3. Статья «Меню в WPF» <http://www.c-sharpcorner.com/uploadfile/mahesh/menus-in-wpf/>
4. Михаил Смирнов «Стандарты и правила оформления кода C#» <http://michaelsmirnov.blogspot.ru/2011/01/c.html>