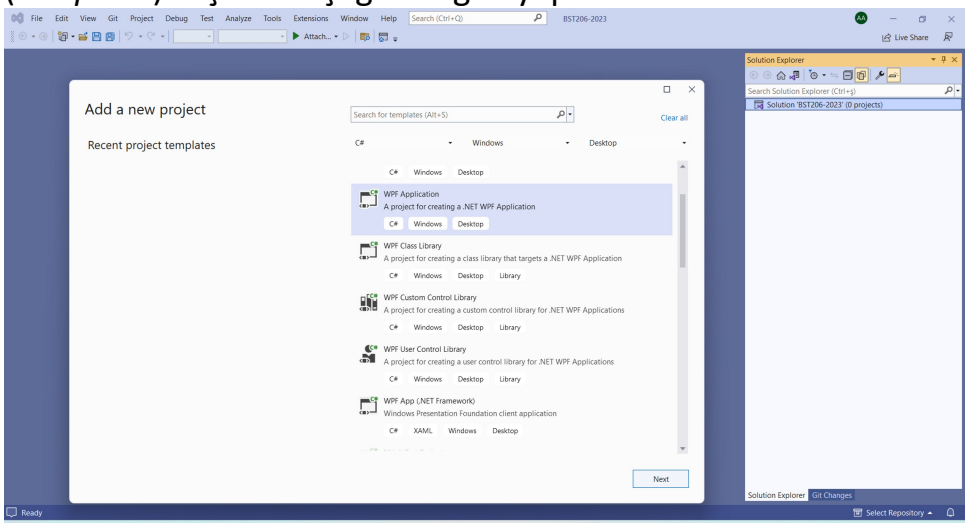
Grid1

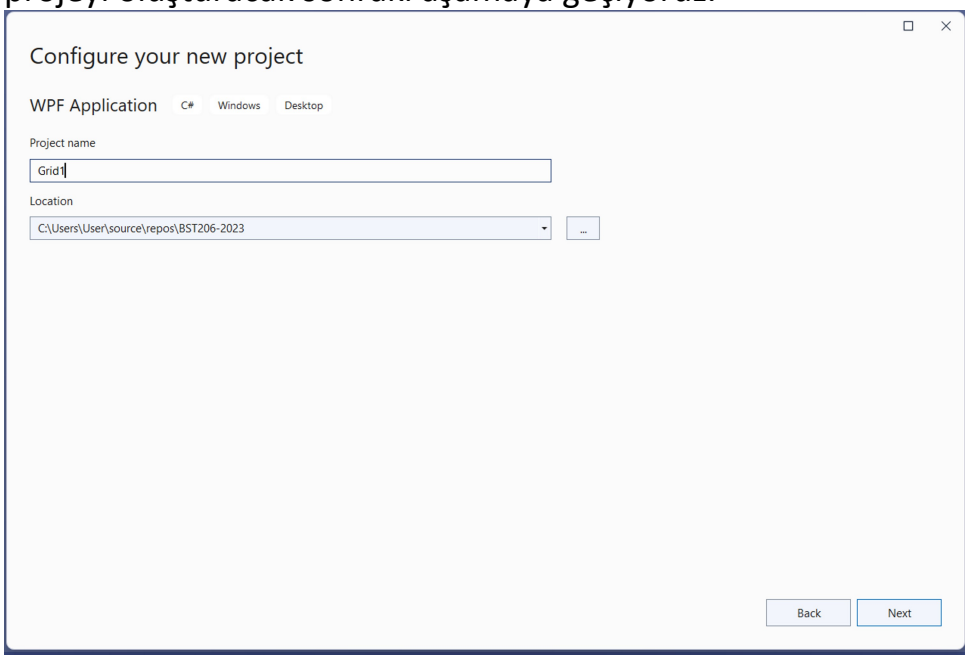
Thursday, March 2, 2023 18:41

Bu başlangıç örneğinde C# programlama diliyle bir WPF masaüstü uygulaması oluşturuyoruz. Visual Studio geliştirme ortamındaki proje oluştururken şablon (*template*) seçimini aşağıdaki gibi yaptık:



Bu şablon seçimi aşamasında "WPF App" (WPF Uygulaması) etiketli iki seçenek olduğunu görüyorsunuz. Bizim tercih ettiğimiz ilk seçenek Windows dışındaki işletim sistemlerinde de kullanılan .NET Core tabanlı uygulama şablonudur. Bu göreceli olarak yeni bir seçenektir. Visual Studio eski sürümlerinde tek seçenek olan "WPF App (.Net Framework)" seçeneğini tercih ederseniz uygulamanız yalnızca Windows işletim sisteminde çalışacaktır. Bu örnekleri izlerken hangisini seçtiğiniz pek de fark etmez. Ciddi kodlar yazmaya başlayacağımız ileri örneklerde, belki referans değişkenleri kullanan bazı komutlarda küçük farklar görebilirsiniz.

Devam edelim: Şablon seçinden sonraki aşamada proje adını veriyoruz ve projeyi oluşturacak sonraki aşamaya geçiyoruz.



En son aşamada hangi .NET sürümünü kullanacaksınız diye bir tercih formu daha çıkabilir. Biz bu notları yazarken varsayılan seçenek 6.0 sürümüydü ama en son sürüm olan 7.0 da tercih edebiliyorduk. Varsayılan sürümü yeterli gördük biz; siz kendi tercihinizi yapabilirsiniz.

Tüm bu aşamaları geçip projeyi oluşturduk. Karşımıza uygulama penceresinin tasarım görünümü ve onun altında bu pencerenin görünümünü düzenleyen XAML kodları çıktı:

```
<Window x:Class="Grid1.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:Grid1"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>

    </Grid>
</Window>
```

Projenin **MainWindow.**xaml adlı dosyadında yer alan bu kodların HTML kodları gibi görünmesi boşuna değildir. XAML aslında tıpkı HTML gibi, <> sembolleri içinde başlayıp </> sembolleriyle kapanan tanım blokları kullanan bir dildir. Örneğin, başlangıçtaki kalabalık tanım ile bir pencere (**Window**) oluşturulmuştur. Bu tanımın başlangıç etiketi (*tag*) ve bitişi arasında ise pencere içerisinde yer alacak görsel öğelerin tanımları vardır. Bildiğimiz sade metin olarak yazılmış bu dosyayı okuyan derleyici tanımları yapılan görsel öğelerin belirtilen özelliklerle oluşturulmasını ve belirtilen şekilde düzenlenmesini sağlayacaktır.

Günümüzde web uygulamaları veya mobil uygulamalardaki görsel arayüzleri oluşturmakta kullanılan yöntemler de benzer olduğu için, görsel uygulamaları geliştirmeye bu yoldan başlamak daha kolaylık sağlayacaktır.

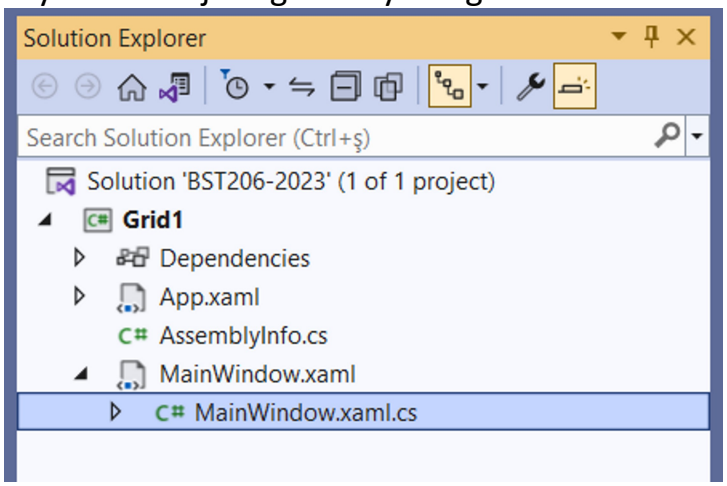
Bizim proje şablomnu olarak "WPF Uygulaması" seçmemizin nedeni de budur aslında: XAML yalnızca uygulama penceresinin görsel özelliklerini belirler ve görünüm düzenini oluşturur. Kullanıcının bu öğelerle etkileşmesini sağlayan ve bu etkileşimlere göre eylemler gerçekleştiren uygulama kodları ise ayrı bir dosyada -XAML dosyasıyle birlikte açılan- **MainWindow.xaml.cs** yer

almaktadır:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace Grid1
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
    }
}
```

Bu iki dosya aslında uygulamanın ana penceresinin (**MainWindow**) iki ayrı yüzü sayılırlar. Proje organizasyonu görünümünde de birlikte yer alırlar:



Uygulama kodlarını içeren bu dosyayı kapatıp geçebilirsiniz. Bu ilk denememizde yalnızca uygulama penceresinin görsel özellikleri ve görünüm düzeni ile uğraşacağız.

Bu özellikleri veya düzeni değiştirmek için XAML dosyasındaki kodları elle

değiştirmek yeterli olacaktır. Bu ilk deneme için elle değiştireceğimiz özellikleri sarı renkle belli ettik. Pencere başlığını (**Title**) "Ana Pencere", genişlik (**Width**) değerini "1000", yükseklik (**Height**) değerini de "650" olarak değiştirdik. Siz de o özellikleri kendi istediğiniz şekillerde değiştirmeyi deneyin; yaptığınız değişikliklerin pencere tasarım görünümünü hemen değiştirdiğini göreceksiniz.

Dilerseniz F5 kısayol tuşu ile Debug (hata ayıklama) modunda, dilerseniz de CTRL+F5 tuş kombinezonuyla Release (kullanıma hazır) modunda uygulamayı çalıştırabilirsiniz. Uygulama penceresi belirlediğiniz özelliklerle ekrana gelecektir. Hiçbir kod içermediği için henüz bir şey yapmayacaktır, ama alıştığınız diğer Windows görsel uygulamalarındaki gibi uygulama penceresini küçültüp büyültebilecek veya kenarlıklarından tutup çekerek boyutlarını değiştirebileceksiniz.

Dr. Öğretim Üyesi Hürol Aslan tarafından oluşturulmuştur.

Grid2

Thursday, March 2, 2023 19:16

İlk proje örneklerimizde WPF masaüstü uygulama pencerelerinde standart olan kafes paneli (**Grid**) tanıtmak istiyoruz. Ama yeni başlayanlar proje geliştirme aşamalarını daha iyi izlesinler diye, tanıtımı tek bir proje ile yapmayacağız. Onun yerine, önemli aşamaları ayrı projeler olarak ekleyeceğiz. Bu belgede de her aşamanın önemli yanlarını ayrı ayrı anlatacağız.

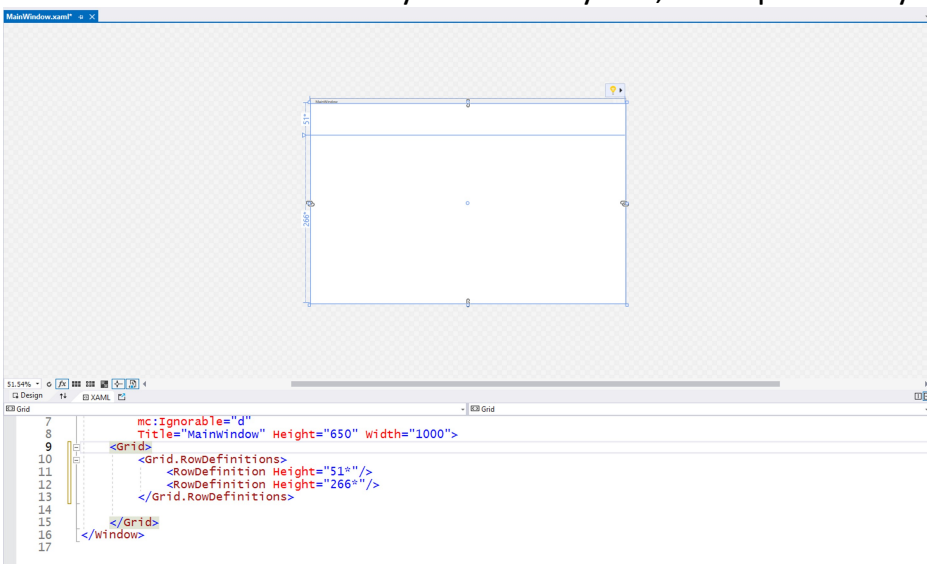
İlk projemiz Grid1'in bir kopyasını oluşturup Grid2 diye yeniden adlandırdık. Bu uygulamanın ana penceresinin görünümünü belirleyen XAML dosyasını açtık. Pencerenin içerik sunucusu (*container*) olan kafes panelin XAML blokunu

```
<Grid>

</Grid>
```

içinde herhangi bir yerde tıklayınca tasarım görünümünde pencere içindeki panel, sol ve üst kenarlarında kesikli çizgilerle gösterilen bantlar varmış gibi gözüktü.

Sol kenar bandının belli bir yerinde tıklayınca, kafes paneli iki yatay bölmeye ayırmış olduk:




```
<Grid.RowDefinitions>

</Grid.RowDefinitions>
```

bloku içinde panel bölmelerinin yükseklik oranlarını gösteren yeni tanımlar oluştuğunu görüyorsunuz.

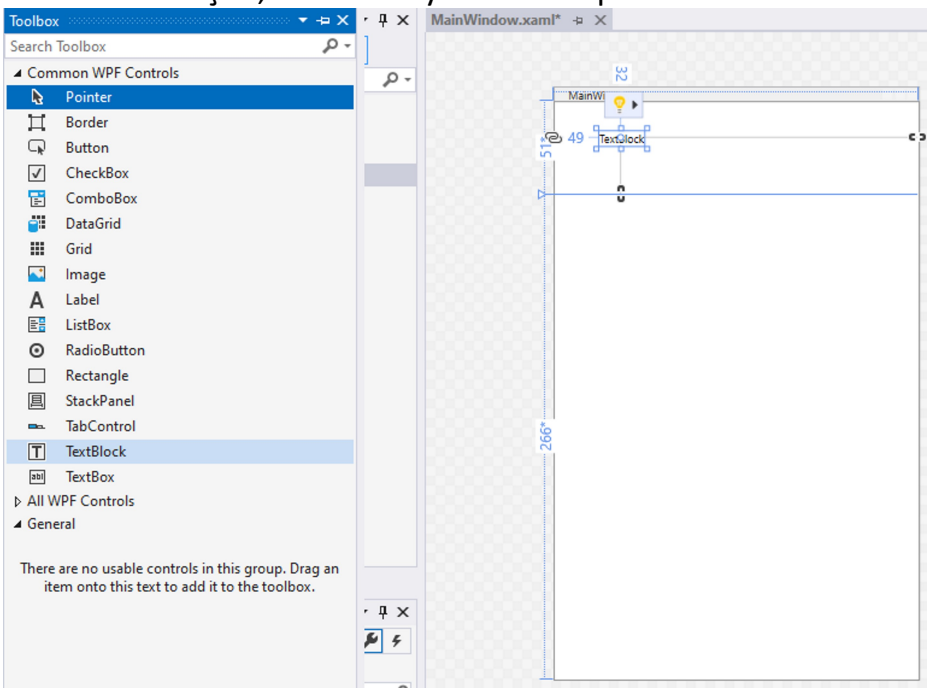
"Kafes Panel" dediğimiz **Grid** türü içerik sunucuyu (*container*) standart yapan şey işte bu bölünme özelliğidir. Birçok görsel uygulamada menü seçeneklerinin, listeleyici kontrollerin, tıkladığınız düğmelerin, içine yazı yazdığınız metin kutularının, vb., yanyana ve alt alta, bölmeli bir paneldeki gibi yerleştiklerini hatırlarsınız. Siz uygulama penceresini büyültüp küçülttükçe bu panel bölmelerinin de orantılı olarak büyüyüp küçülmesi bazı durumlarda kullanışlılık sağlayan bir görsel etki oluşturacaktır.

Kafes panel bölmelerini ciddi bir uygulamada kullanmadan önce, üst panel bölmesine bir kontrol yerleştirelim. Sonra da görünümünü ve konumunu belirleyen özelliklerle oynayalım. Bu

arada üst bölmenin pencereyle orantılı boyutlandırılmasını göreceğiz.

Öncelikle, yatay bölmelerin boyut orantısını daha iyi fark edelim diye uygulama penceresini biraz daralttık. Genişliğini (**Width**) 400 piksel yaptık.
"Piksel" demişken, aslında pek de doğru bir şey yazmadık. Normalde bilgisayarlarla bütünleşik veya ayrı kullanılan ekran boyutlarına göre, gerçek piksel ("görüntü noktacığı" mı desek?) boyutları her biri için aynı değildir. Bu nedenle, WPF için "piksel" dediğimiz şey, aslında gerçek piksel boyutu değil, standart kabul edilmiş özel bir boyut birimidir. 1 inç (2.54 cm) uzunluğun 1/96'sına eşittir.

Araç Kutusunu (*Toolbox*) açıp, orada listelenen kontrollerden metin bloğu (**TextBlock**) kontrolünü seçtik, ve sürükleyerek kafes panelin üst bölmesine yerleştirdik.



Bu kontrol içinde bir metin görüntüler, ama yalnızca görüntüler. Belki daha alışkın olduğunuz "Metin Kutusu" (TextBox) kontrolünden farklı olarak, metin girişine veya değişikliğe izin vermez.

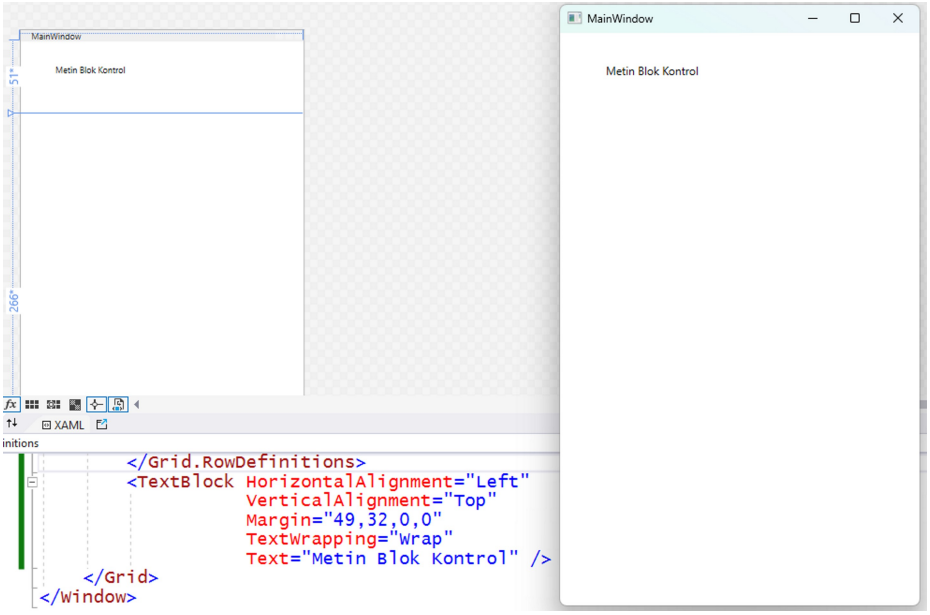
Her neyse, bu metin bloğu kafes panelin XAML kod bloğunda aşağıdaki gibi görünüyordu:

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="51*"/>
        <RowDefinition Height="266*"/>
    </Grid.RowDefinitions>
    <TextBlock HorizontalAlignment="Left"
               VerticalAlignment="Top"
               Margin="49,32,0,0"
               TextWrapping="Wrap"
               Text="TextBlock" />
</Grid>
```

Sizin denemenizdeki sonuç tam böyle değildir herhalde. Biz metin bloğu kontrolünün XAML tanımındaki özellikleri okunur olsun diye satırlara ayırdık, satırların sırasını değiştirdik ve soldan hizaladık. Siz kontrolü çekip bıraktığınızda farklı bir yere koymuşsanız, konum özellikleri de farklı olmuştur.

Kontrolün en son özelliğinden başlayalım: **Text** özelliği bloğun görüntülediği karakter dizgisidir.

Bunu istediğiniz başka bir sözcük veya cümleyle değiştirin. Biz "Metin Blok Kontrol" yazdık. Başka değişiklikler yapmadan önce de uygulamayı çalıştırıp denedik:

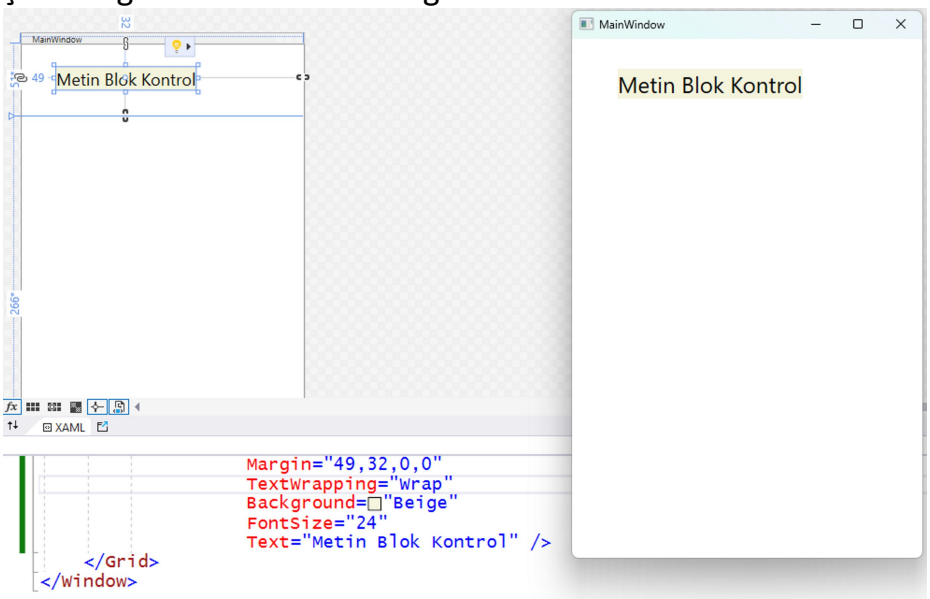


Gördüğünüz gibi, yeni belirlediğimiz metin gözüküyordu, ama hem kutunun içindeki yazı biraz küçüktü, hem de kutunun nerede başlayıp nerede bittiği belli olmuyordu.

WPF uygulamalarında hazır kullandığımız pencere ve kontrol gibi görsel öğeler için standart özellikler geçerlidir. Yazı boyutuyla ilgili bir özellik görmemiştik ama geri planda onun da bir ayarı vardı. Bu ayarı biz farklı yapalım şimdi; **TextBlock** kontrolünün XAML tanımına **FontSize** (Yazı Tipi Boyutu) özelliğini ekleyip farklı bir değer atayalım. Bir de kutunun boyutları belli olsun diye, **Background** özelliğiyle ona farklı bir geri plan rengi verelim:

```
<TextBlock HorizontalAlignment="Left"
           VerticalAlignment="Top"
           Margin="49,32,0,0"
           TextWrapping="Wrap"
           Background="Beige"
           FontSize="24"
           Text="Metin Blok Kontrol" />
```

Şimdiki görünümü daha belirgindir:



Peki bu kontrolün boyutlarını belirleyen özellikler nerede? Bu XAML tanımında genişlik (**Width**) ve yükselik (**Height**) özelliklerini görmüyoruz.
Görmüyoruz, çünkü onlar yazı boyutuna göre otomatik belirlenmiştir. Kontrol genişliği kutu içindeki metnin sığacağı kadardır. Yüksekliği de -üst ve altta biraz paylar bırakarak- karakter yükseliğine göre belirlenmiştir.

Kontrolün konumunu belirleyen özelliklere gelince, belki Windows Forms gibi başka platformlarındaki gibi sol konum (**Left**) veya üst konum (**Top**) için ayrı özellikler yoktur. Kafes panel (**Grid**) bölmeleri içine yerleştirilen kontroller kendileri bölme kenarlarına göre hizalarlar. Konumu belirleyen ölçü de bölme kenarları ile kontrol kutusu arasında kalan boşluklardır. Yani, bu örnekteki **TextBlock** kontrolü konumunu belirleyen özellik **Margin** (Kenar Boşluğu) özelliğidir. Bu özellikte dört sayısal değer görüyoruz. Bunlardan ilki sol kenar boşluğu *(left margin*)*,* ikincisi de üst kenar boşluğudur (*top margin*)*.* Biz kontrolü elle yerleştirince atanan ilk değerler yerine:

```
Margin="10,10,0,0"
```

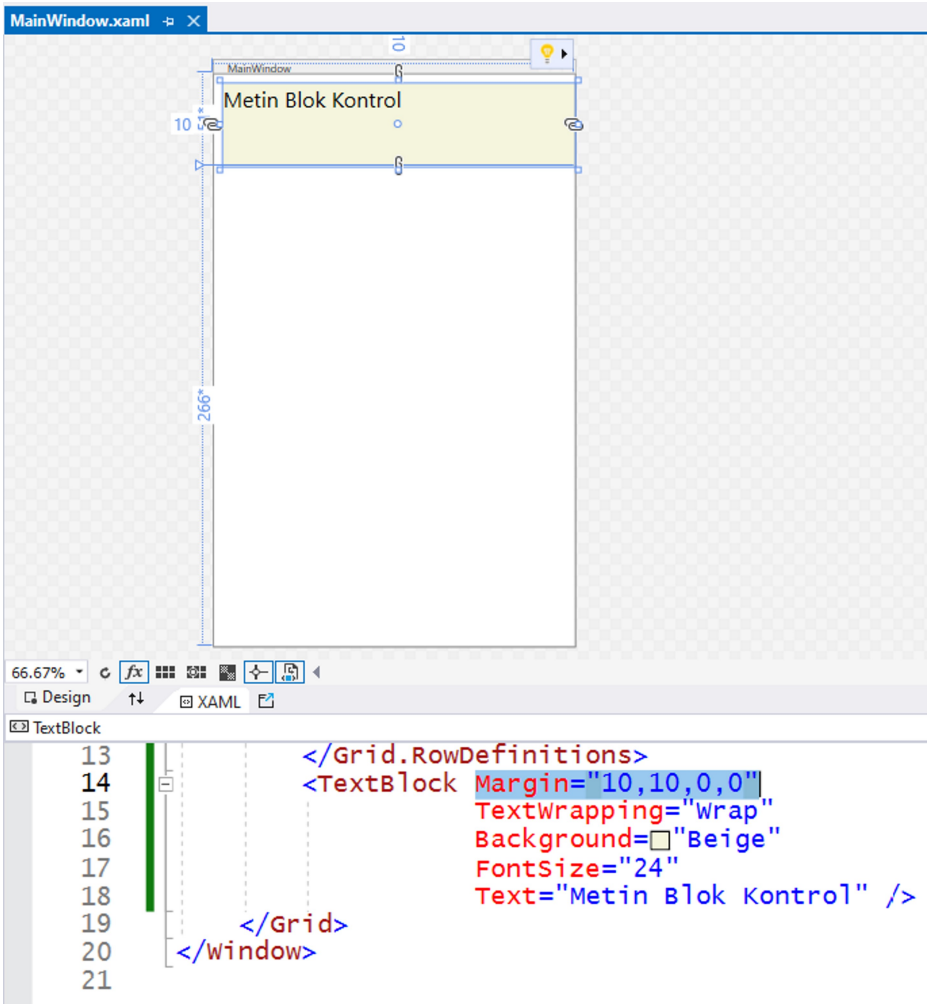
yazın, artık blok kontrolün üst bölme kenarlarına daha yakın olduğunu ve bu boşlukları koruduğunu göreceksiniz.

Bu arada, şu soru aklınıza takılmıştır belki de: **Margin** özelliğindeki son iki değer neyin nesi? Üçüncü değer sağ marjin (*right margin*) ve son değer de alt marjindir (*bottom margin*). Evet, bu iki değer sıfırdır, ama blok kontrolü ne sağa yapışmıştır, ne de alta.

Bunun nedeni kontrolün hizalanma şeklidir. Yatay hizalama şeklini belirleyen **HorizontalAlignment** özelliği **"Left"** değerini taşıyor. Yani kontrol sola hizalanmıştır. Bu nedenle de sağ marjin değeri 0 olsun, başka bir değer olsun, dikkate alınmıyor.
Ama **HorizontalAlignment="Right"** şeklinde bir değişiklik yaparsanız, kontrolün üst bölmenin sağ kenarına yapıştığını göreceksiniz.

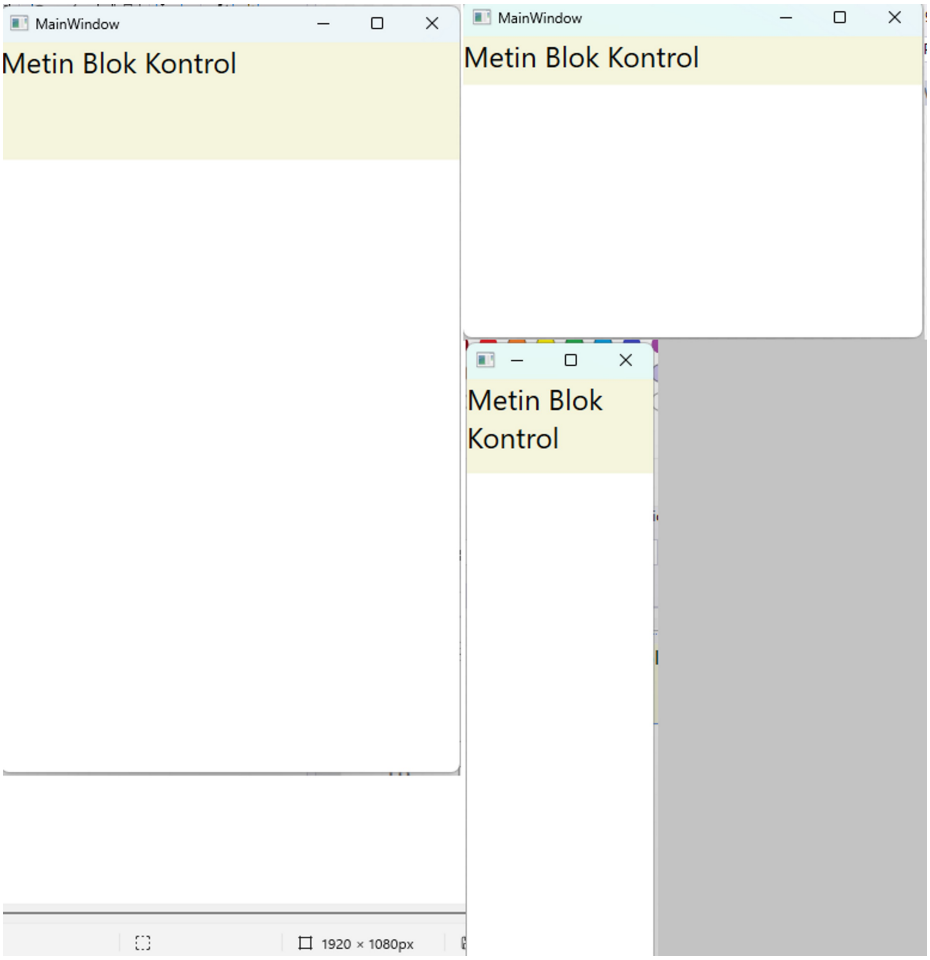
Benzer olarak, alt marjin değeri de dikkate alınmıyordu, çünkü dikey hizalama (**VerticalAlignment**) üst kenarı (**"Top"**) esas alıyordu. **VerticalAlignment="Bottom"** şeklinde bir değişiklik yaparsanız, kontrolün üst bölmenin alt kenarına yapıştığını göreceksiniz.

Peki hizalama özelliklerini devre dışı bıraksak ne olurdu? Bunu görmek için **HorizontalAlignment** ve **VerticalAlignment** özelliklerini sildik. O zaman **Margin** özelliğinin dört değerinin de etkin olduğunu gördük.



Sol ve üstten on birim boşluk varken, kutu sağ ve alt kenarlara yapışıktı. **Margin** için dört değil de, tek bir değer verince, onu da "10" yaptık, kontrolün üst bölmeye göre her taraftan 10 bitim boşluk kalacak şekilde yerleştiğini gördük.

Margin özelliğini de iptal edince, artık kontrol üst bölmeyi tamamiyle dolduruyordu. Genişliği matin uzunluğuna bağlı değildi artık, yüksekliği de yazı boyutundan bağımsızdı. Bu durumdayken uygulama penceresini büyültüp küçülttüğümüzde blok kontrolün de pencereyle orantılı boyutlandığını, genişliği pencere iç genişliği ile aynı kalırken, yüksekliği üst bölmenin pencereye orantısıyla değişiyordu.



Sağ altta resmini gösterdiğimiz son denemede **TextBlock** kontrolünün **TextWrapping** özelliğinin ne işe yaradığını fark ettik: Bu özelliğin şimdiki değeri **"Wrap"** olduğu için kutu içeriğindeki metni gerekince "alta sarıyordu" *(wrapping)*. Metin uzunluğu kutu genişliğine sığmayınca kontrol adı da bölünmüş ve bir kısmı alta kaymıştı.

Artık bu uygulama penceresindeki kafes panelin (**Grid**) yatay bölmelerinin boyut oranlarına bakmalıyız:

```
<Grid.RowDefinitions>
    <RowDefinition Height="51*"/>
    <RowDefinition Height="266*"/>
</Grid.RowDefinitions>
```

RowDefinitions öğesi kafes panelin yatay bölme (yani "satır", *row*) tanımlarını içeriyor. Satır tanımını yapan her iki **RowDefinition** öğesindeki **Height** özelliği de o satırın, yani yatay bölmenin yüksekliğini veriyor.

Ama bu yükseklikler sabit değerler değil, aslında orantılı boyutlardır. Örneğin üst bölme (ilk satır tanımı) için yükseklik 51 birim, ama yanındaki yıldız onun orantılı ya da göreceli bir ölçü olduğunu gösteriyor. Diğer bölme yüksekliği de yıldızlı verilmiş, yani o da göreceli bir ölçü. Kısacası, her iki yükseklik de aslında kafes panelin (ki o da uygulama penceresinin iç bölgesini kaplıyor) yüksekliğiyle orantılı oluyorlar. Üst bölme yüksekliği kafes panel yüksekliğinin yaklaşık %16'sı oluyor. Bu oranı 51 değerini (51 + 266) toplamına bölerek elde ettik.

Bu değerleri elle değiştirerek yatay bölme orantılarını değiştirebiliriz. Üst bölme için yüksekliği "50*" , alt bölme için de "250*" yapsak, iki bölmenin yükseklik orantıları (kafes panel

yüksekliğine göre) 1/6 ve 5/6 olacaktır. Burada göstermedik, ama siz deneyip görün. Bu yükseklik ölçüleri -yanlarında yıldız sembolü olduğu sürece- göreceli oldukları için, "50*" yerine "1*", "250*" yerine de "5*" yazsak, yükseklik orantıları yine aynı olurdu.

Ama **Grid** bölme tanımları için yükseklik ölçüsünü yıldız sembolü olmadan yazarsak, işte o zaman ilgili bölme yüksekliğini sabitlemiş oluruz. Biz üst bölme yüksekliğini "50*" yerine "50" diye değiştirip bir deneme yaptık. Uygulamayı çalıştırdığımızda, pencere yüksekliğini değiştirsek bile **TextBlock** kontrolünün yüksekliğinin aynı kaldığını gördük.