Osnove rada sa tekstom, oblik prozora i kontrola, resursi, trigeri

Da bi programer mogao da sam nacrta željeni oblik prozora, moraju se promeniti svojstva prozora koji se menja. Najpre se mora definisati svojstvo ***WindowStyle***, koje određuje kako će biti uokviren prozor, odnosno da li će imati naslovnu liniju. Vrednost se postavlja na *None,* tako da se prozoru uklanja naslovna linija.

Svojstvo ***AllowsTransparency*** određuje da li boja pozadine prozora može biti providna. U slučaju da je vrednost *true*, može se dodeliti vrednost *Transparent* svojstvu ***Background*** prozora. Svojstvo ***SizeToContent*** određuje da li će se dimenzije prozora ograničiti na sadržaj koji je postavljen na njemu. Sa željom da se iscrta oblik prozora potrebno je da sadržaj prozora ne bude u **Grid** kontroli, već u **Canvas** kontroli, koja predstavlja osnovu za crtanje ili prikaz slika.

Način da se postigne iscrtavanje je da se definiše objekat klase **Path**, koji predstavlja zatvorenu liniju. Kružnoj liniji se može definisati boja kroz svojstvo ***Stroke***, debljina linije se određuje kroz svojstvo ***StrokeThickness***. ***Fill*** svojstvo **Path** klase će definisati bojom kojom će biti obojena površina koja se formira krivom linijom, dok **Data** svojstvo određuje geometriju, odnosno način na koji će se ova putanja/linija iscrtati.

Ovo se postiže objektom tipa **PathGeometry**, u kojem se definiše crtanje kompletne figure – kroz objekat klase **PathFigure**, koji se definiše sa početnom tačkom iscrtavanja (***StartPoint***). U **WPF**-u, koordinatni sistem započinje tako što je gornji levi ugao prozora koordinatni početak, X-osa raste na desno kao i inače, dok Y-osa raste na dole. Zatvorena linija se dalje formira dodavanjem objekata klasa **LineSegment** i **ArcSegment**.

LineSegment se manifestuje kao prava linija koja se formira tako što se definiše krajnja tačka linije (***Point***), a linija počinje od prethodno navedene tačke. Sa druge strane, **ArcSegment** se definiše kroz svojstva ***Size***(određuje poluprečnike po X i Y osi, pošto se **ArcSegment** manifestuje kao elipsa), ***RotationAngle***(odgovara uglu gde kriva linije dostiže maksimum), ***IsLargeArc***(određuje da li kriva linija koja se crta zahvata ugao veći od 180 stepeni ili manji), ***SweepDirection***(govori o smeru iscrtavanja linije, da li je u smeru kretanja kazaljke na satu ili suprotno) i ***Point***(kao i kod **LineSegment**-a, krajnja tačka gde se linija završava).

Pri formiranju figure (***Data*** svojstvo klase **Path**) segmenti se navode sekvencijalno kako se nastavljaju jedan na drugi, gde se liniji zadaje krajnja tačka do koje će prostirati, a način definicije krive linije (***ArcSegment***) je objašnjen u prethodnom delu. Primer ovoga je naveden u listingu koda ispod.

```
<Path Stroke="Gray" StrokeThickness="2" Name="UIPath" >
     <Path.Fill>
        <SolidColorBrush Color="White"/>
     </Path.Fill>
     <Path.Data>
         <PathGeometry>
              <PathFigure StartPoint="50,0">
                   <LineSegment Point="0,50"/>
                                . . .
                   <ArcSegment Size="150,150" RotationAngle="180"
                  IsLargeArc="True" SweepDirection="CounterClockwise" Point="375,0" />
                   <LineSegment Point="50,0"/>
              </PathFigure>
         </PathGeometry>
      </Path.Data>
 </Path>
```

Listing 1. Primer definicije XAML koda za crtanje krive linije koja formira površinu

Ovako definisan prozor nema mogućnost, zbog nedostatka naslovne linije, prevlačenja po ekranu. Da bi se to omogućilo, može mu se dodati metoda koja će reagovati na događaj *MouseLeftButtonDown*, odnosno dok je levi taster miša pritisnut na površini objekta. U samoj implementaciji je dovoljno nad samim prozorom (***this***) pozvati metodu *DragMove()*, koja sa sobom nosi implementaciju prevlačenja prozora.

Rad sa tekstom se realizuje korišćenjem kontrole **RichTextBox**, koja, slično kao **TextBox**, omogućava unos teksta, sa razlikom da se u okviru **RichTextBox**-a mogu prikazivati sadržaji *.rtf* fajlova, što znači da se tekstu može menjati boja, veličina, font itd. Da bi bilo moguće implementirati ove funkcionalnosti, potrebno je ponuditi neki meni korisniku kako bi mogao da izabere način na koji će da promeni stil teksta u okviru **RichTextBox**-a.

Slično **Grid**-u, u **WPF**-u postoje klase koje implementiraju podelu prostora na prozoru, a jedna od njih je **DockPanel**. Koristeći objekat ove klase, korisnik definiše njegov sadržaj identično kao sa klasom **Grid**, tako što između para tagova (otvoreni, zatvoreni) postavlja druge kontrole. Kontrolama smeštenim kao sadržaj **ToolBar**-a, se dodeljuje uz koju ivicu **DockPanel**-a su "dock-ovani", odnosno poravnati.

Klasa **ToolBar** modeluje "traku" sa različitim opcijama, kao u gotovo svakom programu na operativnom sistemu Windows. Unutar **ToolBar**-a (između njegovog para tagova) se definišu kontrole koje će sa sobom nositi implementaciju opcija.

ToggleButton predstavlja klasu identičnu klasi Button, ali koja ima u sebi već skup implementacija, pa se prilikom definicije konkretnog objekta njene klase definiše koja će funkcionalnost biti korišćena. Ovo se postiže definisanjem svojstva ***Command.*** Primer vrednosti je da to bude *EditingCommands.ToggleBold*, odnosno da se klikom na to dugme pozove implementacija Bold funkcionalnosti nad selektovanim tekstom unutar **RichTextBox**-a.

Jedna korisna stvar u definiciji sadržaja **ToolBar**-a, je korišćenje klase **Separator**. Ona se navodi kao samozatvarajući tag koji modeluje uspravnu crtu kojom se razdvajaju opcije unutar **ToolBar**-a. Primer definicije ovoga je dat u listingu koda ispod.

```
<DockPanel>
  <ToolBar DockPanel.Dock="Top">
    <ToggleButton Command="EditingCommands.ToggleBold" Name="btnBold" Content="B"/>
    <Separator />
    <ComboBox Name="cmbFontFamily" Width="150"
SelectionChanged="cmbFontFamily_SelectionChanged"/>
  </ToolBar>
</DockPanel>
```

Listing 2. Primer definicije menija pomoću klasa DockPanel i ToolBar

Kada je u pitanju implementacija same funkcionalosti aplikacije koja bi odgovarala nekom "Text Editor"-u, bitne su dve stvari: funkcionalnosti koje menjaju sam stil teksta (za **ToggleButton** nije potrebna dodatna implementacija) i povratna informacija korisniku kada selektuje neki tekst, šta je sve od opcija primenjeno nad njim (na primer, kada se selektuje tekst koji je boldovan, dugme za boldovanje je uvek označeno).

Za, na primer promenu fonta, potrebno je najpre korisniku ponuditi izbor svih fontova. To se obično realizuje kroz **ComboBox**. Tom ComboBox-u najpre treba dodeliti imena svih fontova koji su dostupni na sistemu računara koji koristi. To se postiže dodelom vrednosti svojstvu ***ItemsSource*** **ComboBox**-a na način kako je definisano u listingu koda ispod.

```
cmbFontFamily.ItemsSource = Fonts.SystemFontFamilies.OrderBy(f => f.Source);
```

Listing 3. Dodela svih sistemskih fontova prema njihovom imenu

Promena fonta nekog teksta se dešava kada je željeni tekst označen (*selected*) i desi se promena u izabranoj opciji **ComboBox**-a u kojem su navedeni svi fontovi. Da bi se to postiglo, potrebno je definisati metodu koja će reagovati na događaj *SelectionChanged* **ComboBox**-a, koji se dešava čim se promeni izabrana vrednost.

U tom slučaju treba proveriti da li je išta označeno u **ComboBox**-u, a ako jeste, potrebno je promeniti vrednost svojstva ***FontFamily*** tekstu koji je označen (ako postoji takav tekst). Ako je u **XAML** kodu aplikacije dato ime samom **RichTextBox**-u, referenciranjem tog imena i pristupom njegovom svojstvu ***Selection*** dobija se označen tekst. Dodevljivanje vrednosti svojstvima se realizuje pozivom metode *ApplyPropertyValue()*, čiji su parametri samo svojstvo za koje se dodeljuje vrednost i koja mu se vrednost dodeljuje. Ovo je prikazano u listingu koda ispod.

```
private void cmbFontFamily_SelectionChanged(object sender, SelectionChangedEventArgs
e)
{
    if (cmbFontFamily.SelectedItem != null && !rtbEditor.Selection.IsEmpty)
    {
        rtbEditor.Selection.ApplyPropertyValue(Inline.FontFamilyProperty,
                                               cmbFontFamily.SelectedItem);
    }
}
```

Listing 4. Metoda kojom se menja font označenom tekstu u okviru RichTextBox-a

Da bi se korisniku dala povratna informacija o promenama stila teksta, tekst najpre mora biti označen. Promene se pokazuju nakon što se promeni selekcija unutar **RichTextBox**-a, što će odgovarati reakciji na događaj *SelectionChanged* nad samim **RichTextBox**-om.

Unutar implementacije događaja, preuzima se vrednost svojstva ***FontWeight*** (za „debljinu“ slova) i ***FontFamily*** (za ime fonta). Vrednost ***FontFamily*** svojstva se samo dodeljuje kao ***SelectedItem*** za **ComboBox** koji pokazuje izabrani font, odnosno u ovom slučaju koji font je primenjen na selektovani tekst.

Za preuzetu vrednost za „debljinu“ slova se proverava da li ima vrednost *Bold* i da li vrednost „debljine“ nije nedodeljena vrednost **DependencyProperty**-ju (predstavlja svojstvo kome se definiše vrednost kroz promene stila, DataBinding, animacije ili nasleđivanje). Ako su oba ova uslova zadovoljena (stil slova nije nedefinisan i jeste Bold) onda selektovani tekst stvarno jeste boldovan. „Logičko i“ ova dva uslova će biti indikacija da li je dugme označeno, odnosno obeleženo, kako je to slučaj u drugim aplikacijama u kojima je moguć rad sa tekstom. Ova implementacija je navedena u listingu koda ispod.

```
object temp = rtbEditor.Selection.GetPropertyValue(Inline.FontWeightProperty);
btnBold.IsChecked = (temp != DependencyProperty.UnsetValue) &&
                    (temp.Equals(FontWeights.Bold));

temp = rtbEditor.Selection.GetPropertyValue(Inline.FontFamilyProperty);
cmbFontFamily.SelectedItem = temp;
```

Listing 5. Provera vrednosti svojstava primenjenih na označenom tekstu

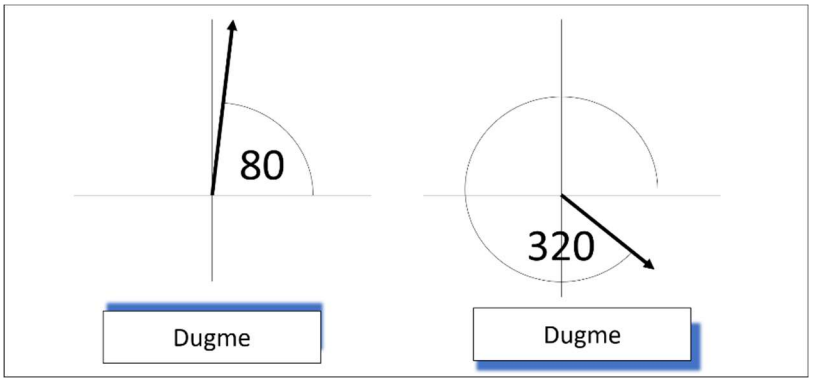
Još dva važna pojma u **WPF**-u su resursi i trigeri. Resursi predstavljaju podatke koji se definišu jednom i mogu se potom koristiti koliko god puta je programeru potrebno. Resurs može biti bilo kakav podatak i on se može definisati tako da važi za konkretnu kontrolu, deo aplikacije, ili celu aplikaciju.

Resursi mogu biti statički ili dinamički, zavisno od toga da li će tokom izvršavanja aplikacije menjati svoju vrednost ili ne. Trigeri predstavljaju alternativni način reakcije na događaje. Primer definicije resursa je prikazan u **Listingu 6**.

```
<Grid.Resources>
    <DropShadowEffect x:Key="dropMouseOverShadow" Color="LightGreen" Direction="320"
Opacity="80" ShadowDepth="10"/>
</Grid.Resources>
```

Listing 6. Definicija resursa u okviru Grid kontrole

Resursi su ovde prikazani kao dva efekta prikaza senke. Ono što je obavezno prilikom definicije resursa je njegov ključ (*Key*), koji je pandan imenu instance kontrole (*Name*) i po tome se raspoznaje i referencira. U ovom konkretnom slučaju, ***DropShadowEffect*** predstavlja efekat da se oko nekog elementa interfejsa prikaže senka. Sama senka se modeluje bojom (*Color* - dostupne su sve boje iz kolekcije klase ***Brushes***), smerom u kojem se prostire iza dugmeta (*Direction* - definiše se kao ugao koji odgovara strani kontrole koja će biti osenčena - prikazano na **Slici 1**), providnošću (*Opacity* – što je veća vrednost, boja je manje providna), i koliko je udaljena od samog dugmeta u smeru prostiranja (*ShadowDepth* - koliko je senka pomerena „dalje“ od kontrole).



Slika 1. Prikaz odnosa ugla i položaja senke prilikom definicije DropShadowEffect-a

Da bi se efekat primenio, koristiće se trigeri. Trigeri se definišu nad kontrolom nad kojom će se primeniti efekat. Da bi se pristupilo trigerima, pristupa se najpre *Style* svojstvu kontrole. Svojstvo *Style* utiče na prikaz kontrole. Način definisanja trigera koji odgovara trenutku kada se kursorom miša pređe preko dugmeta prikazan je u **Listingu 7**.

```
<Button.Style>
   <Style TargetType="{x:Type Button}">
      <Style.Triggers>
         <Trigger Property="IsMouseOver" Value="True">
            <Setter Property="Effect" Value="{StaticResource dropMouseOverShadow}"/>
         </Trigger>
      </Style.Triggers>
   </Style>
</Button.Style>
```

Listing 7. Definicija trigera kojim se primenjuje efekat

Triger, u ovom slučaju, reaguje na situaciju kada svojstvo *IsMouseOver* datog dugmeta (da li je kursor miša postavljen preko njega) dobio vrednost *true*. Ako se ovo desi, potrebno je primeniti definisani efekat. Da bi se vrednost svojstva postavila iz **XAML** koda, potrebno je koristiti tag ***<Setter>*** kojim se vrednost određenog svojstva može postaviti na željenu vrednost. Ovde se svojstvu *Effect* dodeljuje statički resurs (navodi se njegov *Key*) koji predstavlja efekat senke.

Da bi se u okviru projekta dodale slike ili drugi fajlovi kako bi se mogli koristiti, potrebno je kliknuti desnim tasterom miša na projekat u Visual Studio-ju i potom odabrati opcije Add -> Existing Item i u dijalogu za izbor fajla izabrati željeni tip fajla i potom sam fajl, koji će potom biti dodat u projekat.

Ako je na ovaj način dodata slika, možemo je, na primer, postaviti kao pozadinu prozora (u ovom slučaju kao pozadinu nacrtanog oblika prozora) tako što se dodaje tag koji određuje svojstvo *Fill* iscrtane geometrije. Njemu se dodeljuje instanca klase ***ImageBrush*** koja određuje „kojom slikom je obojen oblik“. Za putanju do slike je dovoljno navesti samo njeno ime jer se ona već nalazi u projektu. Ovo je prikazano u **Listingu 8**.

```
<Path.Fill>
    <ImageBrush ImageSource="bgimg.png"/>
</Path.Fill>
```

Listing 8. Korišćenje slike kao pozadine oblika prozora

Da bi se omogućila programska zamena slike koja je pozadina oblika, kreira se instanca krase ***ImageBrush***. Dodeljuje joj se objekat klase ***BitmapImage***, koji omogućava učitavanje slika u **XAML** kodu. Njemu se kao parametar prosleđuje putanja do slike koja se učitava (*Uniform Resource Identifier, URI*). Primer ovoga je dat u **Listingu 9**.

```
private ImageBrush img = new ImageBrush();
img.ImageSource = new BitmapImage(new Uri("..\\..\\bgimg.png", UriKind.Relative));
```

Listing 9. Programsko definisanje objekta klase ***ImageBrush***

U projektu za termin trećih vežbi ostao je i primer kako se dugme može učiniti providnim tako da se i dalje nad njim može pozvati *Click* događaj. Ovo se postiže pomoću korisnički definisanog stila (*Style*) i korišćenjem klase ***ControlTemplate*** za definiciju stila, jer će ona poništiti izgled uz definisanu providnu pozadinu, ali će ostaviti mogućnost da se na dugme može kliknuti.

Klasom ***ContentPresenter*** se definiše šta piše na kontroli (*Content*), te ako i ona ostane prazna, na kontroli neće ništa pisati. Kada se ovakav stil primeni na dugme kao resurs, dugme, pošto gubi efekat da se oboji u svetlo-plavu boju kada se preko njega pređe kursorom miša nema način da da korisniku povratnu informaciju da se na njega može kliknuti. To se može rešiti definicijom kakav će kursor biti kada se pređe preko datog dugmeta dodeljivanjem vrednosti *Hand* svojstvu *Cursor* konkretnog dugmeta, te će kada se pređe preko mesta gde stoji ovo, sada potpuno providno dugme, biti prikazan kursor u obliku ruke sa ispruženim prstom.

Sve ove funkcionalnosti su iskorišćene da se kreira primer za četvrtu nedelju vežbi koji se manifestuje kao jedan mini „Text Editor“ sa korisnički definisanim oblikom prozora. Uz to, dodat je efekat senke na dugmetu za promenu pozadinske slike glavnog prozora, dok je dugme za izlaz realizovano kao providno, tako da se njegova funkcionalnost poziva klikom na „pozadinsku sliku“.