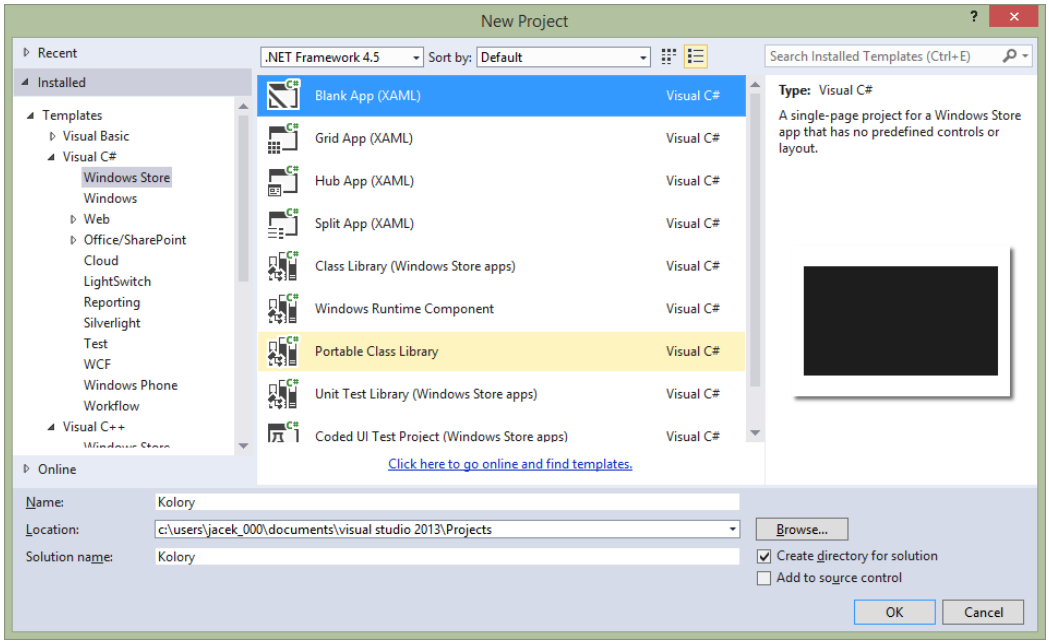
Rozdział 2.
Pierwsza aplikacja Windows Store.

Jacek Matulewski
Materiały dla Podyplomowego Studium Programowania i Zastosowania Komputerów,
sekcja Projektowanie i tworzenie aplikacji dla platformy .NET (pod patronatem Microsoft)

Czas na przejście do konkretów. Zbudujmy pierwszą aplikację Windows Store.

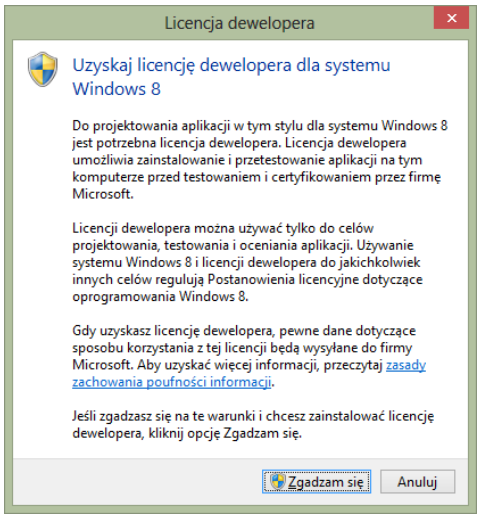
Tworzenie projektu

1. Po uruchomieniu Visual Studio 2013, z menu *File*, *New*, wybieramy pozycję *Project*.
2. W oknie dialogowym *New Project* (rysunek 2.1) w lewym panelu zawierającym drzewo szablonów projektów rozwijamy gałąź *Visual C#*, a następnie *Windows Store*.



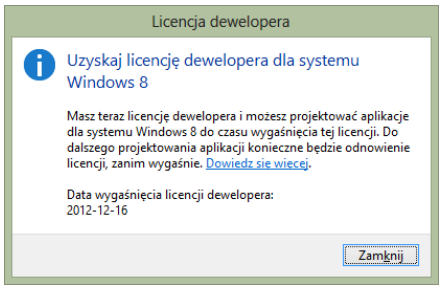
Rysunek 2.1. Tworzenie projektu aplikacji z interfejsem Modern UI

3. W liście projektów widocznej w prawym panelu zaznaczamy szablon projektu o nazwie *Blank App (XAML)*.
4. W polu *Name* wpiszmy nazwę projektu (np. „Kolory”) i kliknijmy *OK*.
5. Jeżeli to jest pierwszy projekt aplikacji z kategorii Windows Store, pojawi się propozycja zainstalowania licencji dewelopera (rysunek 2.2), która umożliwia instalowania aplikacji na lokalnym komputerze w celach rozwoju i testowania. Aby móc projektować aplikacje tego typu należy kliknąć przycisk z etykietą *Zgadzam się*. Do instalacji licencji niezbędne są uprawnienia administratora.



Rysunek 2.2. Monit o instalację licencji dewelopera

6. Następnie zostaniemy poproszeni o zalogowanie do konta Microsoft (dane Live ID), które związane będzie z licencją. Po tym licencja zostanie pobrana (rysunek 2.3). Niestety ważna będzie tylko jeden miesiąc. Przed tym okresem konieczne będzie jej przedłużenie.



Rysunek 2.3. Instalacja jest ważna miesiąc.

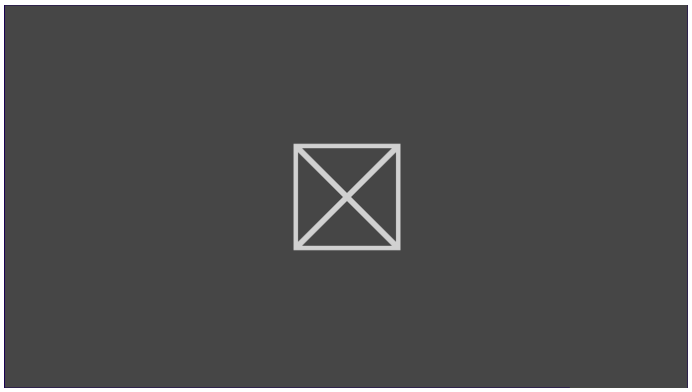
7. Po kliknięciu *Zamknij* projekt zostanie wreszcie utworzony.

Uruchamianie aplikacji na lokalnym komputerze i emulatorze tabletu

Uruchommy utworzony przed chwilą pusty projekt. Możemy zrobić to naciskając klawisz *F5*. Wówczas aplikacja zostanie w trybie debugowania. Jeżeli naciśniemy kombinację *Ctrl+F5* – uruchomiona zostanie bez debugowania.

Bez względu na to, którą opcje wybierzemy zostaniemy przełączeni do ekranu Start systemu Windows 8, pojawi się na chwilę okno sygnalizujące uruchamianie aplikacji z interfejsem Modern (rysunek 2.4), a następnie zobaczymy puste okno naszej aplikacji[1].

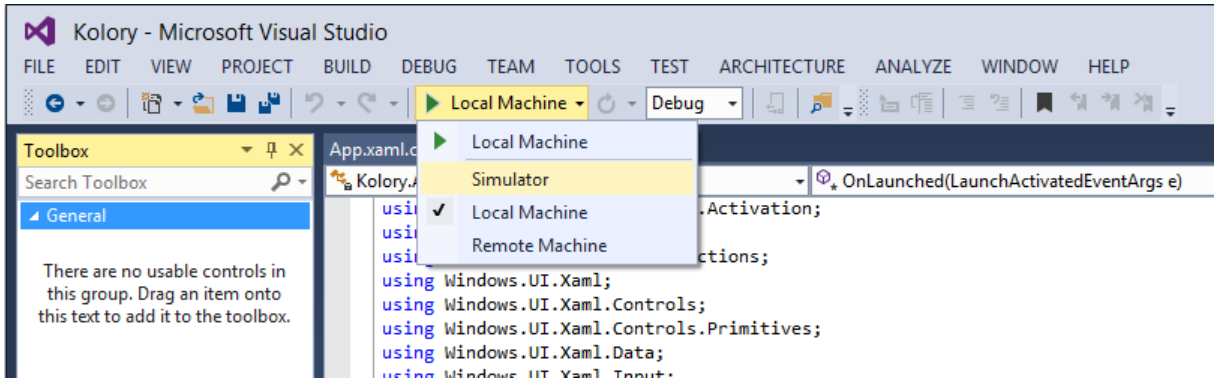
[1] W przypadku Visual Studio 2013, w trybie debugowania okno nie jest tak zupełnie puste. W górnym lewym rogu widoczne są liczby informujące o liczbie klatek odświeżania interfejsu na sekundę i poziom wykorzystania CPU przez bieżącą aplikację. W prawym górnym rogu widoczne są systemowa częstość odświeżania (60 klatek na sekundę dla monitorów LCD) i całkowite wykorzystanie CPU przez system.



Rysunek 2.4. Uruchamianie aplikacji z interfejsem Modern

Możemy zamknąć to okno klawiszami *Alt+F4*, ale i tak pozostaniemy na ekranie *Start*. Aby powrócić do pulpitu musimy przesunąć myszkę w lewy górny lub lewy dolny róg ekranu. Możemy również kliknąć kafelek z etykietą *Pulpit*.

Oprócz uruchamiania aplikacji na lokalnym komputerze, mamy możliwość uruchomienia go na innym komputerze z systemem Windows 8 i na emulatorze. Szczególnie interesująca jest ta druga opcja. Aby jej spróbować, w rozwijanej liście widocznej na rysunku 2.5 należy wybrać pozycję *Simulator*.

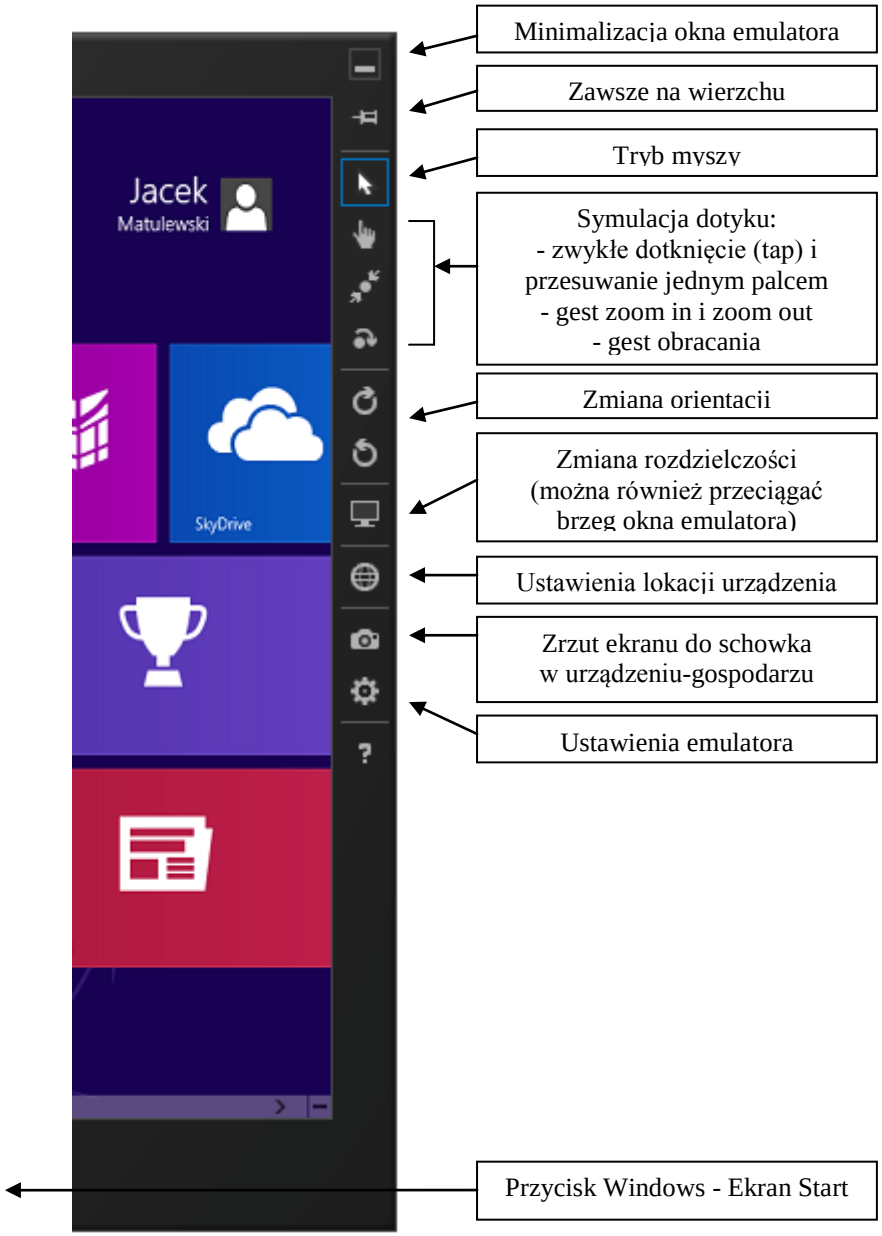


Rysunek 2.5. Wybór sposobu uruchamiania aplikacji

Jeżeli teraz uruchomimy program, pojawi się emulator tabletu, na którym uruchomiony zostanie lokalny system Windows (druga niezależna instanacja). Zobaczymy ekran logowania, zostaniemy automatycznie wlogowani na nasze konto, a następnie ekran Start i wreszcie nasza aplikacja.

Sprawdziłem, że na emulatorze można uruchomić Visual Studio 2013, ale już niemożliwe jest uruchomienie emulatora. Ponadto w macierzystym systemie może być uruchomiony tylko jeden symulator na raz.

Warto poświęcić kilka chwil na poznanie możliwości emulatora, w szczególności klawiszy widocznych po jego prawej stronie. Ich funkcje opisuje rysunek 2.6.



Rysunek 2.6. Przyciski widoczne przy prawej krawędzi okna emulatora

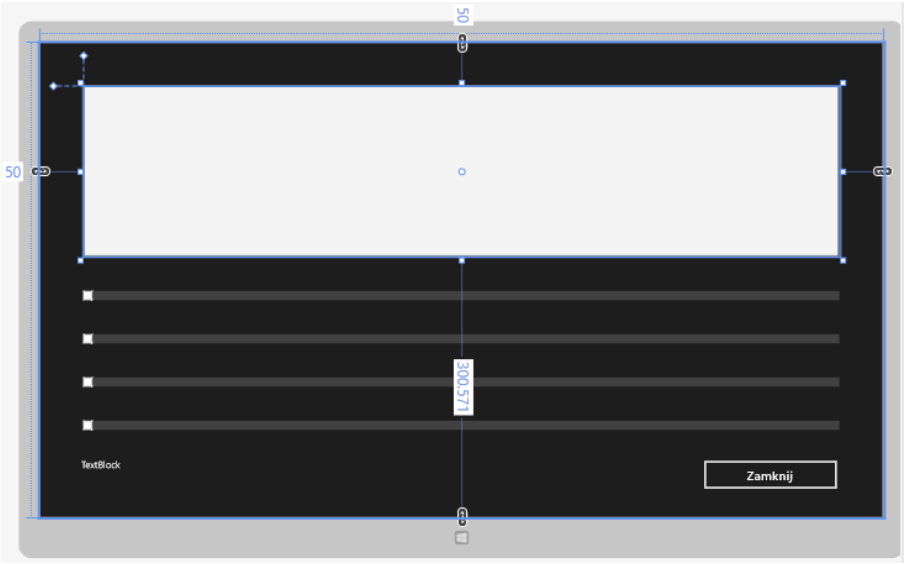
Projektowanie interfejsu

Po utworzeniu projektu, w edytorze widoczny jest kod z pliku *App.xaml*. Do tego pliku jeszcze wrócimy, a na razie zajmiemy się przygotowaniem interfejsu aplikacji, którego opis należy umieścić w pliku *MainPage.xaml*.

Nasza aplikacja ma umożliwiać wybór koloru. Czterema suwakami będziemy kontrolować składowe RGB koloru oraz kanał alfa, czyli przezroczystość (a dokładniej nieprzezroczystość). Potrzebujemy również miejsca, w którym wybrany kolor będzie prezentowany i miejsca do wyświetlenia szesnastkowego kodu koloru. Ponadto na stronie umieścimy przycisk, w którego metodzie zdarzeniowej będziemy eksperymentować z możliwościa wymuszenia zamknięcia aplikacji. A zatem:

1. Dwukrotnie kliknijmy plik *MainPage.xaml* w podoknie *Solution Explorer* widocznym z prawej strony okna Visual Studio. Zawartość tego pliku stanie się widoczna w oknie edytora.
2. W dolnej części edytora kodu widoczne są dwie zakładki: *XAML* i *Design*. Przełączmy na *Design*. Widoczny stanie się podgląd okna umieszczony w obramowaniu symbolizującym obudowę tableletu.
3. Jeżeli chcemy zobaczyć całe okno powinniśmy skorzystać z rolki myszy z jednocześnie przyciśniętym klawiszem *Ctrl*.

4. Na lewym panelu widoczne jest natomiast okno ustawień urządzenia, do którego dostosowane jest okno podglądu, zawierające orientację urządzenia, rozdzielczość ekranu, temat kolorystyczny, itd.
5. Na dole tego panelu zobaczymy trzy zakładki: widoczna w tej chwili zakładka *Device*, a oprócz tego *Server Explorer* i *Toolbox*. Przełączmy na zakładkę *Toolbox* (ang. skrzynka z narzędziami). W dwóch grupach zawiera ona kontrolki, z których można zbudować interfejs aplikacji Modern UI. Rozwińmy gałąź *Common XAML Controls*.
6. Z tej zakładki przeciągnijmy na podgląd okna kontrolkę `Rectangle`, pole tekstowe `TextBlock` (nie `TextBox`) i przycisk `Button`. Możemy wstępnie rozmieścić je myszką według wzoru na rysunku 2.1, ale w praktyce prostsza jest edycja kodu XAML. Ostatecznie powinien on wyglądać tak, jak na listingu 2.1 (proszę zwrócić na atrybuty Name).
7. Na tej zakładce nie znajdziemy jednak potrzebnych nam suwaków (kontrolka `Slider`). Są w grupie *All XAML Controls*. Przeciągnijmy je również na podgląd ekranu. Całość powinna wyglądać jak na rysunku 2.7.



Rysunek 2.7. Interfejs aplikacji. Wielkość podglądu można kontrolować rolką myszy przytrzymując klawisz Ctrl

8. Aplikacja na ekran startowy wypełnia zwykle cały ekran. Nie ma więc problemu ze zmianą rozmiaru okna. Należy jednak pamiętać, że może być wyświetlana na różnych ekranach. Warto więc zadbać o poprawne rozmieszczenie kontrolek. Jeżeli Czytelnik jest przyzwyczajony do korzystania z własności `Anchor` znanej z Windows Forms, to powinien zwrócić uwagę na klamerki widoczne po zaznaczeniu kontrolki na brzegach okna (rys. 2.7). Klikając je możemy zmienić justowanie. Linia ciągła oznacza, że kontrolka jest zakotwiczona do brzegu okna, a przerywana – że nie. Przypnijmy prostokąt do wszystkich czterech krawędzi, suwaki do bocznych i dolnej, pole tekstowe do lewej i dolnej, a przycisk do dolnej i prawej. Aby sprawdzić działanie tego mechanizmu warto uruchomić program w emulatorze, a następnie zmieniać jego rozmiary.

W efekcie kod XAML odpowiedzialny za wygląd okna powinien wyglądać jak na listingu 2.1.

XAML (ang. *eXtensible Application Markup Language*, czyt. „zamml") to nic innego jak XML, w którym poszczególne elementy opisują położenie i wygląd kontrolek. Kontrolki mogą być, podobnie jak elementy XML, wielokrotnie zagnieżdżane. Ten sposób opisu wykorzystywany jest w aplikacjach WPF, Windows Store i Windows Phone.

Listing 2.1. Kod XAML wygenerowany automatycznie przez narzędzia projektowania wizualnego

```
<Page
    x:Class="Kolory.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:Kolory"
```

```
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
        <Rectangle Fill="#FFF4F4F5" Margin="50,50,50,301" Stroke="Black"/>
        <Slider Margin="50,0,50,228" Height="51" VerticalAlignment="Bottom"/>
        <Slider Margin="50,0,50,178" Height="51" VerticalAlignment="Bottom"/>
        <Slider Margin="50,0,50,128" Height="51" VerticalAlignment="Bottom"/>
        <Slider Margin="50,0,50,78" Height="51" VerticalAlignment="Bottom"/>
        <TextBlock HorizontalAlignment="Left" Margin="50,0,0,36" TextWrapping="Wrap"
Text="TextBlock" Width="172" Height="32" VerticalAlignment="Bottom"/>
        <Button Content="Button" HorizontalAlignment="Right" Margin="0,0,50,30"
VerticalAlignment="Bottom" RenderTransformOrigin="1.339,0.244" Width="159" Height="38"/>
    </Grid>
</Page>
```

Przy bogactwie możliwości opisu interfejsu za pomocą XAML, jego edycja za pomocą wbudowanego w Visual Studio edytora jest dość niewygodna. Znacznie większe możliwości i wygodę oferuje Microsoft Blend. Można jednak dobierać wielkość i położenie kontrolek, a ponadto zakotwiczać je do krawędzi okna. Edytor umożliwia także wyrównywanie nowych kontrolek do kontrolek wcześniej umieszczonych na stronie i dopasowywanie odległości między nimi.

Nadawanie nazw kontrolkom

Zwróćmy uwagę, że wszystkie kontrolki są anonimowe. Narzędzie projektowania nie przypisuje wartości własnościom `Name` tworzonych kontrolek. Możemy uzupełnić ten brak dodając do znaczników atrybut `Name`. Porządany efekt pokazuje listing 2.2. W uzupełnianiu znaczników pominąłem znacznik `Button` – nie będziemy się do niego odwoływać z poziomu kodu.

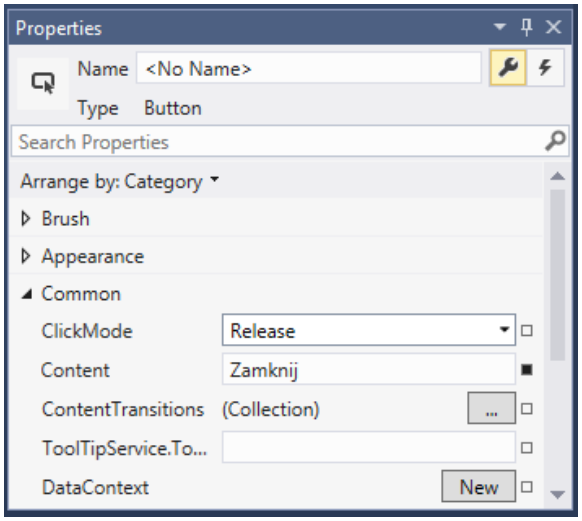
Listing 2.2. Dodane atrybuty wyróżnione zostały pogrubieniem

```
<Rectangle Name="Rectangle1" Fill="#FFF4F4F5" Margin="50,50,50,301" Stroke="Black"/>
<Slider Name="SliderR" Margin="50,0,50,228" Height="51" VerticalAlignment="Bottom"/>
<Slider Name="SliderG" Margin="50,0,50,178" Height="51" VerticalAlignment="Bottom"/>
<Slider Name="SliderB" Margin="50,0,50,128" Height="51" VerticalAlignment="Bottom"/>
<Slider Name="SliderA" Margin="50,0,50,78" Height="51" VerticalAlignment="Bottom"/>
<TextBlock Name="TextBlock1" HorizontalAlignment="Left" Margin="50,0,0,36"
TextWrapping="Wrap" Text="TextBlock" Width="172" Height="32"
VerticalAlignment="Bottom"/>
<Button HorizontalAlignment="Right" Margin="0,0,50,30" VerticalAlignment="Bottom"
RenderTransformOrigin="1.339,0.244" Width="159" Height="38" Click="Button_Click_1"/>
```

Okno własności

Zwróćmy jeszcze raz uwagę na znacznik `Button` w kodzie widocznym na listingach 2.1 i 2.2. Zawiera on między innymi atrybut Content, którego wartość to prosty łańcuch „Button” określający etykietę przycisku. Można ją zmienić oczywiście z poziomu kodu XAML, ale można zrobić to też w widoku projektowania bez zaglądania do kodu. Służy do tego podokno *Properties* (rysunek 2.8), z którego w tej książce nie będziemy zbyt często korzystać, ale które oczywiście warto Czytelnikowi przedstawić. Znajdziemy na nim wszystkie atrybuty znacznika `Button`. Także te, które mają wartości domyślne i nie pojawiają się w kodzie. Są one pogrupowane. Nas interesuje grupa *Common*, którą należy rozwinąć. W niej znajduje się wspomniany atrybut `Content`. Pole tekstowe z prawej strony pozwala na zmianę etykiety przycisku. Zmieńmy ją na „Zamknij”.

W ten sam sposób zmieńmy również wielkość czionki kontrolki `TextBlock`. Ustawmy ją równą 14 px. Odpowiada to atrybutowi `FontSize` znacznika `TextBlock` w kodzie XAML.



Rysunek 2.8. Okno własności w przypadku projektów WPF i Modern UI traci nieco na znaczeniu

Korzystając z okna własności można również zmieniać wspólne własności grupy kontrolek. Zaznaczmy dla przykładu wszystkie cztery suwaki (przytrzymując klawisz *Ctrl*) i w podoknie własności zmieńmy własność `Maximum` ich wszystkich na 255. Możemy również zmienić wartość własności `TickFrequency` na 15.

Wiązanie metod ze zdarzeniami domyślnymi w widoku projektowania

Po zaprojektowaniu interfejsu aplikacji, czas na przystąpienie do zdefiniowania w jaki sposób będzie ona reagować na działania użytkownika. Zacznijmy od kliknięcia przycisku. W przypadku przycisku kliknięcie jest zdarzeniem domyślnym. Najprościej metodę do takiego zdarzenia możemy utworzyć klikając dwukrotnie kontrolkę, w tym przypadku przycisk.

Gdy to zrobimy, zostaniemy przeniesieni do pliku *MainPage.xaml.cs* stowarzyszonego z plikiem XAML i zawierający kod C#. W pliku tym utworzona została metoda `Button_Click_1`, w której umieśćmy kod widoczny na listingu 2.3.

W tej książce będziemy programować w języku C#. To nie jest jedyna możliwość w przypadku aplikacji Windows Store. Alternatywą jest Visual Basic. Aplikacje tego typu można również tworzyć w C++. W tym przypadku szczególnie łatwe jest użycie DirectX. Można również korzystać z JavaScript.

Listing 2.3. Zamykanie aplikacji

```
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    App.Current.Exit();
}
```

W metodzie tej użyliśmy statycznej własności `Current` zdefiniowanej w klasie `App`. Klasa ta reprezentuje aplikację, a `Current` zwraca jej bieżącą instancję. Na jej rzecz wywołaliśmy metodę `Exit`, która natychmiast kończy działanie aplikacji.

Aplikacja może być również zamknięta gestem. Należy kursor przesunąć do górnej krawędzi ekranu. Kursor zmieni kształt na dłoń. Następnie należy „złapać” okno i przeciągnąć w dół do dolnej krawędzi ekranu.

Do zamknięcia aplikacji nie należy używać metody `Window.Current.Close` – nie jest dozwolone zamykanie głównego okna aplikacji. Prawdę mówiąc w ogóle pomysł, aby z interfejsu aplikacji umożliwiać takie brutalne jej zamknięcie jest dalekie od filozofii WinRT, w którym cyklem zycia aplikacji zarządza system.

Wiązanie metod ze zdarzeniami z poziomu kodu XAML

Metody związane z suwakami stworzymy z poziomu kodu XAML. W tym celu wszystkie znaczniki `Slider` w kodzie XAML uzupełniamy o atrybut `ValueChanged="Sliders_ValueChanged"`. Wiąże on zdarzenia `ValueChanged` wszystkich trzech suwaków z nieistniejącą jeszcze metodą `Sliders_ValueChanged`. Aby zdefiniować tą metodę wróćmy do pliku *MainPage.xaml.cs* i zdefiniujmy w niej metodę widoczną na listingu 2.4. Metoda ta korzysta z pędzli, które zdefiniowane są w przestrzeni nazw `Windows.UI.Xaml.Media` (domyślnie zadeklarowanej w nagłówku pliku). Klasa `Color` użyta w poniższych metodach zdefiniowana jest w przestrzeni nazw `Windows.UI`, której nie ma liście poleceń using na początku pliki. Możemy ją dopisać lub stworzyć alias: `using Color = Windows.UI.Color;`

Listing 2.4. Metoda reagująca na zmiane pozycji suwaków i metody przygotowujące łańcuchy z opisem składowych RGB koloru

```
private Color KolorProstokąta
{
    get
    {
        return (Rectangle1.Fill as SolidColorBrush).Color;
    }
    set
    {
        (Rectangle1.Fill as SolidColorBrush).Color = value;
    }
}

private static string SkładoweKoloruDec(Color kolor)
{
    return "R=" + kolor.R.ToString() + ", G=" + kolor.G.ToString() + ", B=" +
kolor.B.ToString() + ", A=" + kolor.A.ToString();
}

private static string SkładoweKoloruHex(Color kolor)
{
    return "#" + kolor.R.ToString("X2") + kolor.G.ToString("X2") +
kolor.B.ToString("X2") + ", " + kolor.A.ToString("X2");
}

private void Sliders_ValueChanged(object sender, RangeBaseValueChangedEventArgs e)
{
    Color kolor = new Color()
    {
        R = (byte)SliderR.Value,
        G = (byte)SliderG.Value,
        B = (byte)SliderB.Value,
        A = (byte)SliderA.Value
    };
    KolorProstokąta = kolor;

    TextBlock1.Text = SkładoweKoloruDec(kolor) + "\n";
    TextBlock1.Text += "HEX: " + SkładoweKoloruHex(kolor);
```

```
}
```

Na listingu 2.4 oprócz zasadniczej metody `Sliders_ValueChanged` widoczne są również dwie metody pomocnicze przygotowujące opisy koloru oraz własność, która ułatwia zmianę koloru prostokąta.

Istnieje również możliwość wiązania zdarzeń z metodami z poziomu okna *Properties*. Aby zobaczyć pełną listę zdarzeń zaznaczonej kontrolki należy, będąc w widoku projektowania, zaznaczyć kontrolkę, a w podoknie *Properties* zmienić zakładkę na *Event handlers for selected element* (ikona błyskawicy).

Uzgodnienie początkowego stanu okna

Aby uzgodnić położenie suwaków z kolorem prostokąta do konstruktora klasy `MainPage` dodajmy kod z listingu 2.5.

Listing 2.5. Konstruktor klasy opisującej stronę

```
public MainPage()
{
    this.InitializeComponent();

    Color kolor = KolorProstokąta;
    SliderR.Value = kolor.R;
    SliderG.Value = kolor.G;
    SliderB.Value = kolor.B;
    SliderA.Value = kolor.A;
    TextBlock1.Text = SkładoweKoloruDec(kolor) + "\n";
    TextBlock1.Text += "HEX: " + SkładoweKoloruHex(kolor);
}
```

Teraz po uruchomieniu aplikacja powinna wyglądać jak na rysunku 2.9.

Rysunek 2.9. Wygląd interfejsu po uruchomieniu aplikacji

Zwróćmy uwagę, że każdej z czterech obecnych w konstruktorze operacji przypisania wartości poszczególnym suwakom towarzyszy wywołanie metody zdarzeniowej `Sliders_ValueChanged`. To nie jest oczywiście

porządane zachowanie aplikacji, choć użytkownik tego nie zauważy. Kontrolka Slider nie posiada niestety zdarzenia opisującego przesuwanie suwaka, ale nie wywoływanego, gdy wartość zmieniana jest z kodu (odpowiednika zdarzenia `Scroll` kontrolki `TrackBar` z biblioteki Windows Forms). Możemy jednak łatwo tego uniknąć definiując pole klasy `MainPage` informujące o tym, czy inicjacja interfejsu została już zakończona. Zainicjujmy ją wartością false i wartość tę zmieńmy w konstruktorze po wykonaniu wszystkich instrukcji (listing 2.6).

Listing 2.6. Flaga podnoszona po zakończeniu inicjacji interfejsu

```
public sealed partial class MainPage : Page
{
    private bool inicjacjaZakonczona = false;

    public MainPage()
    {
        this.InitializeComponent();

        ...

        inicjacjaZakonczona = true;
    }
```

Teraz wystarczy na początku metody `Sliders_ValueChanged` sprawdzać, czy ta flaga nie została podniesiona:

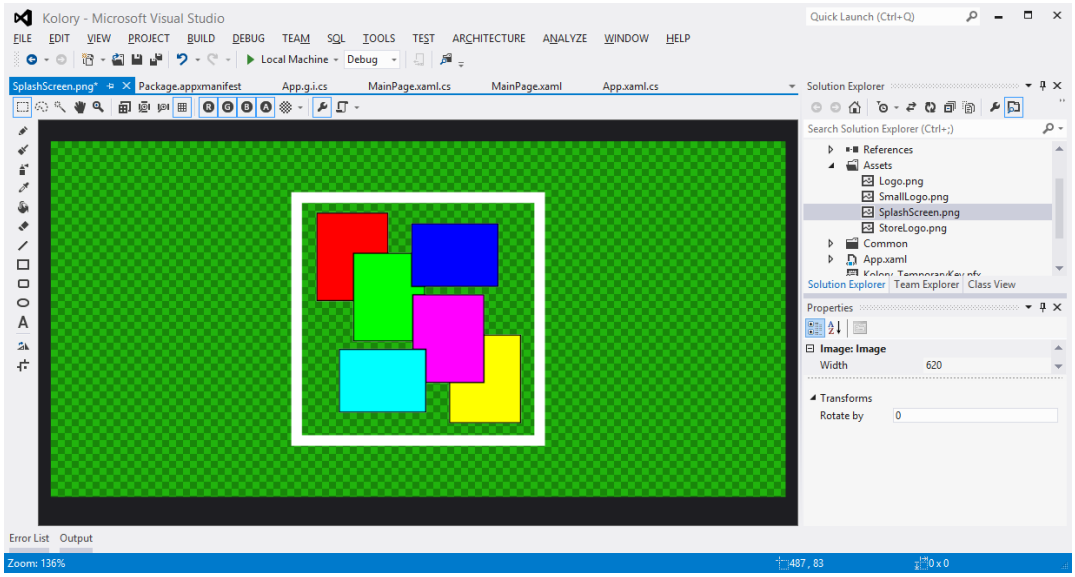
```
private void Sliders_ValueChanged(object sender, RangeBaseValueChangedEventArgs e)
{
    if (!inicjacjaZakonczona) return;
    ...
```

Logo aplikacji

W trakcie ładowania aplikacji widzimy okno powitalne zawierające coś na kształt logo aplikacji (rysunek 2.4). Warto zastąpić domyślne logo taki, które będzie kojarzyć się z przeznaczeniem aplikacji. Użytkownik zyska w ten sposób pewność, że kliknął prawidłowy kafelek.

W podoknie *Solution Explorer* rozwińmy zakładkę *Assets*. Zobaczymy cztery pliku: pierwsze zawiera logo aplikacji (*Logo.scale-100.png*, 150x150 pikseli), małe logo (*SmallLogo.scale-100.png*, 30x30 pikseli), logo z ekranu powitalnego (*SplashScreen.scale-100.png*, 620x300 pikseli) i logo aplikacji w sklepie Windows Store (*StoreLogo.scale-100.png*, 50x50 pikseli). Poza nimi można również użyć szerokiego logo (310x150 pikseli), ale to zrobimy nieco później.

Zmienimy plik *SplashScreen.scale-100.png*. Możemy do tego użyć programu wbudowanego w Visual Studio – wystarczy dwukrotnie kliknąć ten plik. Logo aplikacji na ekran startowy jest zwykle oszczędne w wyrazie. Zwykle bez kolorów, białe – jest piktogramem wyrażającym funkcję programu. Ale nie ma takiego obowiązku. Ja jako ikony aplikacji użyłem zestawu kolorowych prostokątów (rysunek 2.10).



Rysunek 2.10. Projektowanie logo aplikacji

Jeżeli zmienimy logo na ekranie powitalnym, warto zmienić również logo aplikacji widoczne na ekranie Start. Odpowiada za nie obraz w pliku *Logo.scale-100.png*. Najlepiej byłoby oczywiście, gdyby to był ten sam obraz z uwzględnieniem różnicy rozmiaru (por. rysunek 2.11). Analogicznie zmieńmy pozostałe rysunki.

Rysunek 2.11. Ekran Start z aplikacją Kolory

Jeżeli zamiast modyfikować pliki z katalogu *Assets*, chcemy użyć innych – możemy je wskazać w pliku manifestu *Package.appxmanifest*, na zakładce *Application UI*. Tam możemy również wskazać język domyślny aplikacji (zmieńmy go na *pl-PL*).

W przypadku Visual Studio 2013, inaczej niż w poprzedniej wersji, kafelek projektowanej aplikacji nie jest automatycznie przypinany do ekranu *Start*. Można go tam jednak łatwo umieścić. W tym celu przejdźmy do ekranu Start i wpiszmy „Kolory”. Pojawi się lista pasujących aplikacji. Wystarczy na właściwej aplikacji kliknąć prawym klawiszem myszy i z menu kontekstowego wybrać *Przypnij do ekranu startowego*.

Kolorystyka

W pliku manifestu, na zakładce *Visual Assets* możemy również określić kolor tła kafelka i okna powitalnego. W obu przypadkach użyłem zieleni, ale dla okna powitalnego ciemnej zieleni (*#002200*), a dla kafelka nieco jaśniejszej (*#007700*). Aby ustawic kolor tła kafelka należy zaznaczyć pozycję *Tile Images and Logos* w drzewie

widocznym z lewej strony podokna i zmienić wartość w polu edycyjnym *Background color* na *#007700*. Następnie zaznaczamy pozycję Splash screen i w polu edycyjnym wpisujemy *#002200*.

Aby zmienić kolor samego okna aplikacji, należy wrócić do widoku projektowania (zakładka *MainPage.xaml*) i zaznaczając po kolei kontrolki zmieniać ich własności `Foreground` i `Background` za pomocą podokna *Properties* lub bezpośrednio edytując kod XAML.

Dla przykładu zaznaczmy siatkę tła (znacznik `Grid`). W grupie własności *Brush*, w podoknie *Properties* odnajdźmy `Background` i rozwińmy listę związaną z tą pozycją. Wybierzmy *Convert to Local Value*. Dzięki temu będziemy mogli bezpośrednio wskazać kolor tła. Ja wybrałem kolor o składowych R = 0, G = 34, B = 0, A = 100%, czyli ciemnozielony jak w oknie powitalnym. Natomiast kolory suwaków (własność `Foreground`) zmieniłem tak, aby odpowiadały składowym, za których zmianę odpowiadają.

W efekcie kod XAML odpowiadający za zasadniczą część interfejsu wyglądać powinien tak, jak pokazuje listingu 2.7.

Listing 2.7. Zmiana kolorów kontrolek

```
<Grid Background="#FF002200">

    <Rectangle Name="Rectangle1" Fill="#FFF4F4F5" Margin="50,50,50,301" Stroke="Black"/>

    <Slider Name="SliderR" Margin="50,0,50,228" Height="51" VerticalAlignment="Bottom"
ValueChanged="Sliders_ValueChanged" Maximum="255" TickFrequency="15" Foreground="Red"/>

    <Slider Name="SliderG" Margin="50,0,50,178" Height="51" VerticalAlignment="Bottom"
ValueChanged="Sliders_ValueChanged" Maximum="255" TickFrequency="15" Foreground="Lime"/>

    <Slider Name="SliderB" Margin="50,0,50,128" Height="51" VerticalAlignment="Bottom"
ValueChanged="Sliders_ValueChanged" Maximum="255" TickFrequency="15" Foreground="Blue"/>

    <Slider Name="SliderA" Margin="50,0,50,78" Height="51" VerticalAlignment="Bottom"
ValueChanged="Sliders_ValueChanged" Maximum="255" TickFrequency="15"
Foreground="White"/>

    <TextBlock Name="TextBlock1" HorizontalAlignment="Left" Margin="50,0,0,36"
TextWrapping="Wrap" Text="TextBlock" Width="424" Height="32" VerticalAlignment="Bottom"
FontSize="14"/>

    <Button Content="Zamknij" HorizontalAlignment="Right" Margin="0,0,50,30"
VerticalAlignment="Bottom" RenderTransformOrigin="1.339,0.244" Width="159" Height="38"
Click="Button_Click_1"/>

</Grid>
```

Po uruchomieniu przekonamy się, że kolory suwaków rzeczywiście zmieniły sie na czerwony, zielony i niebieski, ale po najechaniu na nie myszką i tak stają się jasnofioletowe. To nie wygląda najlepiej. To jest jednak trudne do zmiany bez użycia Microsoft Blend lub znacznego rozbudowania znacznika odpowiadającego za pędziel używany do rysowania tła suwaków.