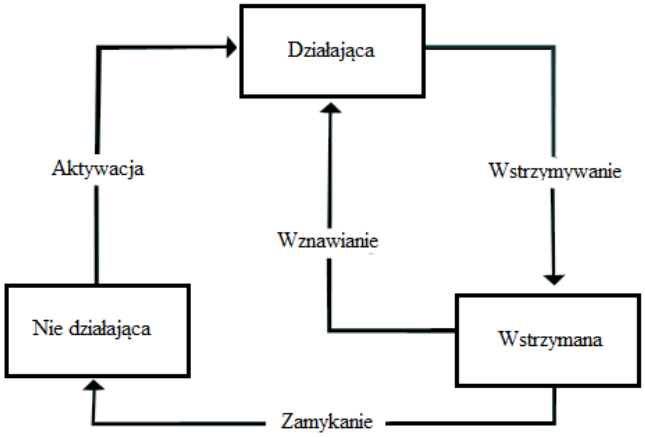
Rozdział 3.
Zapisywanie stanu aplikacji w ustawieniach lokalnych

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

Cykl życia aplikacji. Wybór momentu zapisania stanu

W przypadku aplikacji na ekran Start w systemie Windows 8 (aplikacje Modern UI) podlegają zarządzaniu, jaki znamy z systemów mobilnych. To nie użytkownik, a system decyduje kiedy powinny zostać zamknięte (w tym sensie nasz przycisk *Zamknij*, który dodaliśmy do aplikacji odbiega od standardów). System może również wstrzymać aplikację, jeżeli nie jest widoczna na ekranie, a użytkownik otwiera nowe aplikacje lub komputer przechodzi w stan niższego poboru energii. Wstrzymana aplikacja nadal znajduje się w pamięci, więc jej wznowienie jest szybkie. Dzięki przeniesieniu konroli nad cyklem życia aplikacji z rąk użytkownika do systemu, ten może sprawniej zarządzać pamięcią i użyciem procesora. Schemat cyklu życia aplikacji Modern UI widoczny jest na rysunku 3.1.



Rysunek 3.1. Cykl życia aplikacji Modern UI (tłumaczenie rysunku z MSDN)
(zob. http://msdn.microsoft.com/en-us/library/windows/apps/hh986968.aspx)

Wstrzymanie i wznowienie aplikacji zasadniczo nie wymaga od programisty żadnych działań. Problemem jest natomiast możliwość zamknięcia aplikacji. Aby nie traciła ona swojego stanu, programista powinien zadbać, aby aplikacja potrafiła przechować ustawienia i zawartość kontrolek, lub jeszcze lepiej stan modelu (dane), na bazie którego tworzony jest widoczny w interfejsie widok (stan kontrolek). W naszej aplikacji będą to cztery liczby odpowiadające składowym R, G, B i A koloru lub po prostu obiekt typu `Color`.

Co ciekawe zamykana aplikacja (bez względu na to, czy użyjemy dodanego przez nas przycisku, kombinacji klawiszy *Alt+F4*, czy gestu zamykającego aplikację), zostanie ona wpierw wstrzymana na 10 sekund, a dopiero potem zamknięta. Aplikacja nie jest powiadamiana o zamknięciu, a jedynie o wstrzymaniu. Zatem już wtedy powinna zadbać o zachowanie swojego stanu. I ma na to tylko 5 sekund.

Tyle teorii. W praktyce również zdarzenie towarzyszące wstrzymaniu aplikacji nie jest dobrym miejscem na zapisywanie danych. Raz, że czas od wstrzymania aplikacji do jej zamknięcia jest krótki, a dwa – nie ma gwarancji, że metoda `App.onSuspending` zostanie w ogóle wywołana. Dlatego warto ważne dane zapisać już wcześniej. Można do tego wykorzystać np. zdarzenie `VisibilityChanged` okna – wówczas dane zostaną zapisane w momencie, gdy okno aplikacji znika lub pojawia się z ekranu.

Wiemy zatem kiedy mamy zapisywać dane. Odczytywać będziemy je przy tworzeniu strony, aby stan aplikacji był odtwarzany nie tylko w razie wstrzymania, ale również po zamknięciu w kolejnej instancji. Pozostaje zatem ustalić gdzie mają być zapisywane dane, i co tak właściwie mamy przechowywać. Na ostatnie pytanie najłatwiej odpowiedzieć – stan naszej aplikacji określa kolor prostokąta. Od tego koloru, a konkretnie od jego składowych R, G, B i A zależą również pozycje suwaków.

Do przechowywania danych użyjemy natomiast mechanizmu ustawień lokalnych. Implementuje go klasa `Windows.Storage.ApplicationData.Current.LocalSettings` typu `ApplicationDataContainer`. Pozwala on na przechowywanie danych w spersonalizowanym folderze danych aplikacji, podobnie jak to ma miejsce w przypadku mechanizmu ustawień z aplikacji desktopowych Windows Forms i WPF. W przypadku tego projektu Visual Studio stworzyło plik binarny *Settings.dat* w katalogu *c:\Users\jacek_000\AppData\Local\Packages\9648859e-db68-477e-a389-743c15e25923_n88mjzycwkygy\Settings*.

Oprócz ustawień lokalnych (tj. ograniczonych do tego jednego komputera), aplikacja może również przechowywać ustawienia wspólne dla tej aplikacji i tego użytkownika na wielu komputerach (katalog `Windows.Storage.ApplicationData.Current.RoamingSettings`). Dane są wówczas przechowywane w chmurze. Więcej o tym w następnym rozdziale. **** Przekazywanie ustawień nie zawsze działa.

Przygotujmy klasę statyczną `Ustawienia`, która w oparciu o ustawienia lokalne pozwoli na przechowanie i odtworzenie stanu aplikacji (koloru). Pokazuje ją listing 3.1. Zwróćmy uwagę, że przechowywanie danych polega na zapisywaniu ich do kolekcji `Values` – nie ma w tym zatem niczego skomplikowanego.

Listing 3.1. Klasa ustawień. Może być zdefiniowana gdziekolwiek, nawet jako klasa zagnieżdżona klasy Kolory.MainPage, ale ja umieściłem ją w osobnym pliku Ustawienia.cs.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Color = Windows.UI.Color;

namespace Kolory
{
    static class Ustawienia
    {
        private static Windows.Storage.ApplicationDataContainer ustawienia =
Windows.Storage.ApplicationData.Current.LocalSettings;

        public static bool CzyObecne()
        {
            return
                ustawienia.Values.Keys.Contains("kolor_R") &&
                ustawienia.Values.Keys.Contains("kolor_G") &&
                ustawienia.Values.Keys.Contains("kolor_B") &&
                ustawienia.Values.Keys.Contains("kolor_A");
        }
```

```
        public static Color Czytaj()
        {
            Color kolor;
            kolor.R = (byte)ustawienia.Values["kolor_R"];
            kolor.G = (byte)ustawienia.Values["kolor_G"];
            kolor.B = (byte)ustawienia.Values["kolor_B"];
            kolor.A = (byte)ustawienia.Values["kolor_A"];
            return kolor;
        }

        public static void Zapisz(Color kolor)
        {
            ustawienia.Values["kolor_R"] = kolor.R;
            ustawienia.Values["kolor_G"] = kolor.G;
            ustawienia.Values["kolor_B"] = kolor.B;
            ustawienia.Values["kolor_A"] = kolor.A;
        }

        public static void Usun()
        {
            ustawienia.Values.Remove("kolor_R");
            ustawienia.Values.Remove("kolor_G");
            ustawienia.Values.Remove("kolor_B");
            ustawienia.Values.Remove("kolor_A");
        }
    }
}
```

Wywołajmy metodę `Czytaj` w konstruktorze klasy `MainPage`, a `Zapisz` w metodzie związanej ze zdarzeniem zmiany widoczności okna. Wiązanie metody ze zdarzeniem umieściliśmy również w konstruktorze klasy `MainPage`. Listing 3.2 pokazuje zarówno konstruktor, jak i metodę zdarzeniową. W efekcie aplikacja po ponownym uruchomieniu będzie znajdowała się w stanie, w jakim ją opuścimy.

Listing 3.2. Odtwarzanie stanu aplikacji

```
using Color = Windows.UI.Color;

namespace Kolory
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame.
    /// </summary>
    public sealed partial class MainPage : Page
    {
        private bool inicjacjaZakonczona = false;

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

```
            if (Ustawienia.CzyObecne()) KolorProstokąta = Ustawienia.Czytaj();

            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);

            Window.Current.VisibilityChanged += Current_VisibilityChanged;

            inicjacjaZakonczona = true;
        }

        void Current_VisibilityChanged(object sender,
Windows.UI.Core.VisibilityChangedEventArgs e)
        {
            if (!e.Visible) Ustawienia.Zapisz(KolorProstokąta);
        }
```

O zapisaniu danych należy również pamiętać po kliknięciu przycisku, przed wywołaniem metody `App.Current.Exit` (listing 3.3).

Listing 3.3. Zapisywanie ustawień przed wymuszeniem zamknięcia aplikacji

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

Model

Możemy pójść o krok dalej. Do klasy `Ustawienia` dodajmy statyczną składową przechowujące aktualną wartość koloru ustalonego za pomocą suwaków i prezentowanego w prostokącie. W ten sposób klasa ta stworzy w pełni funkcjonalną warstwę danych.

1. Zacznijmy od dodania do klasy `Ustawienia` prywatnego pola `kolor` i udostępniającej go własności `Kolor` (listing 3.4). Pole będzie typu `Nullable<Color>` (co w skrócie można zapisać jako `Color?`) i zainicjowane wartością `null` świadczącą o tym, że jego wartość nie została jeszcze odczytana z ustawień. W sekcji `get` własności, której kod wykonywany jest w momencie odczytywania wartości własności, sprawdzamy czy wartość została już odczytana z ustawień i jeżeli nie, to to robimy. W ten sposób wartość zostanie zainicjowana automatycznie przy pierwszym odczytaniu.

Listing 3.4. Przekształcania klasy pomocniczej służącej do przechowywania ustawień w model tworzący warstwę danych aplikacji

```
static class Ustawienia
{
    private static Windows.Storage.ApplicationDataContainer ustawienia =
        Windows.Storage.ApplicationData.Current.LocalSettings;

    private static Color kolorDomyslny = Color.FromArgb(255,0,100,0);
```

```
    private static Color? kolor = null;

    public static Color Kolor
    {
        get
        {
            if (!kolor.HasValue)
            {
                if (CzyObecne()) Czytaj();
                else kolor=kolorDomyslny;
            }
            return kolor.Value;
        }
        set
        {
            kolor = value;
        }
    }

    ...
```

2. Przy takim scenariuszu metody `CzyObecne` i `Czytaj` mogą zostać ukryte (ich modyfikator ustawiamy na `private`). Ponadto metoda `Czytaj` wymaga dodatkowym modyfikacji. Zamiast zwracać kolor będzie modyfikować stan ustawień (listing 3.5).

Listing 3.5. Upraszczanie interfejsu klasy Ustawienia

```
private static void Czytaj()
{
    Color kolor;
    kolor.R = (byte)ustawienia.Values["kolor_R"];
    kolor.G = (byte)ustawienia.Values["kolor_G"];
    kolor.B = (byte)ustawienia.Values["kolor_B"];
    kolor.A = (byte)ustawienia.Values["kolor_A"];
    Ustawienia.kolor = kolor;
}
```

3. Zmieniamy również metodę `Zapisz` tak, aby do ustawień przesyłana była wartość z pola `kolor` (w tej wersji metoda pozbawiona jest argumentu, zob. listing 3.6).

Listing 3.6. Drobne modyfikacje w metodzie `Zapisz`

```
public static void Zapisz()
{
    if (kolor.HasValue)
    {
        ustawienia.Values["kolor_R"] = kolor.Value.R;
        ustawienia.Values["kolor_G"] = kolor.Value.G;
        ustawienia.Values["kolor_B"] = kolor.Value.B;
        ustawienia.Values["kolor_A"] = kolor.Value.A;
    }
}
```

4. Możemy teraz przejść do kodu klasy `MainPage`. Zmodyfikujmy konstruktor zgodnie ze wzorem z listingu 3.7.

Listing 3.7. Zmiany w klasie strony

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

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

    Window.Current.VisibilityChanged += Current_VisibilityChanged;

    inicjacjaZakonczona = true;
}

void Current_VisibilityChanged(object sender, Windows.UI.Core.VisibilityChangedEventArgs
e)
{
    if (!e.Visible) Ustawienia.Zapisz(KolorProstokąta);
}

...

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

5. No i na koniec zmodyfikujmy metodę zdarzeniową związaną ze zmianą wartości suwaków zgodnie ze wzorem z listingu 3.8.

Listing 3.8. Synchronizacja koloru prostokąta i stanu modelu

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

    Ustawienia.Kolor = new Color()
    {
        R = (byte)SliderR.Value,
        G = (byte)SliderG.Value,
        B = (byte)SliderB.Value,
        A = (byte)SliderA.Value
```

```
        };
        KolorProstokąta = Ustawienia.Kolor;

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

Uwaga! O ile użycie pola `czyZainicjowane` było wcześniej podyktowane tak naprawdę tylko względami estetycznymi, to teraz jest ono konieczne do prawidłowego działania aplikacji. Bez tego `Ustawienia.Kolory` byłyby niewłaściwie zmieniane przed właściwym uruchomieniem programu.