KAPITOLA 4

Základní principy vývoje aplikací pro Windows Store

V této kapitole:
- Základní principy versus pokročilá témata
- Životní cyklus aplikace
- Asynchronní programování
- Dynamické dlaždice
- Komunikace aplikace prostřednictvím oznámení
- Unifikace funkcí ovládacích tlačítek
- Nastavení parametrů aplikace
- Implementace jednotného vyhledávání
- Panel aplikací
- Test připojení k Internetu
- Aplikační manifest
- Rekapitulace kapitoly
- Kontrolní otázky

Základní principy versus pokročilá témata

Název této kapitoly vyžaduje ve vztahu k jejímu obsahu krátké vysvětlení. Řadě začátečníků se bude zdát, že obsahuje některá pokročilá témata, která se hodí do kapitoly 8. Princip členění je takový, že za základní považujeme jednak implementaci základních funkcionalit, jako je například panel aplikací, nastavování, vyhledávání, a jednak témata, bez kterých se vaše aplikace neobejde.

Jinak řečeno, pokud nebude mít určitou funkcionalitu, například implementované nastavení a v něm deklarovanou ochranu údajů, případně deklarovanou skutečnost, že aplikace žádné údaje od uživatele nesbírá a neukládá, aplikaci vám jednoduše ve Windows Store neschválí.

Životní cyklus aplikace

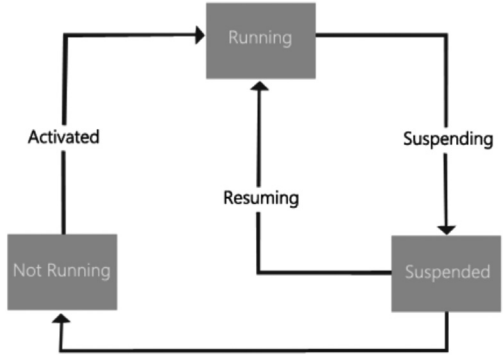
Během stále nekončícího pontifikátu aplikací Win32 na našich desktopech se vžil určitý postup životního cyklu aplikace a jejího běhu, který byl řízen událostmi. Multitasking se považoval za

samozřejmost a aplikace běžela v okně, které mohlo mít libovolný rozměr, okna se mohla navzájem překrývat a uživatel mohl kteroukoliv aplikaci kdykoliv minimalizovat na hlavní panel. Bez ohledu na to, co se s oknem aplikace navenek dělo, její kód mohl stále naplno pracovat.

Životní cyklus nových aplikací pro Windows 8.1 je diametrálně odlišný. Aplikace může po spuštění běžet na popředí buď sama, přičemž má k dispozici celou obrazovku, nebo může plochu obrazovky sdílet s dalšími aplikacemi. Uživatel však může kdykoliv přejít na úvodní obrazovku s dynamickými dlaždicemi, případně se přepnout do jiné aplikace. Co se stane s aplikací v takovém případě? Aplikace může být v jednom ze tří stavů, úmyslně je uvádíme v originální terminologii:

- Running
- Suspended
- NotRunning

Přechody mezi stavy názorně ukazuje diagram na obrázku.



Obrázek 4.1: Životní cyklus aplikace pro Windows Store

Uživateli se tento životní cyklus navenek jeví takto:

- Systém pozastaví aplikaci, kdykoliv se uživatel přepne na jinou aplikaci nebo na úvodní obrazovku.
- Systém obnoví aplikaci, když se uživatel přepne zpět.

Po obnově bude obsah proměnných a datových struktur stejný jako před pozastavením aplikace. Systém obnoví aplikaci přesně tam, kde byla přerušena, takže se to celé navenek jeví, jako kdyby byla mezitím spuštěná na pozadí. Aby aplikace takto fungovala, je nutný kód pro obsluhu událostí přechodů mezi stavy. Tento kód, přesněji těla procedur, je součástí šablon projektů. Pro každou aplikaci je potřeba definovat chování v jednotlivých režimech, hlavně podle toho, jak pracuje s údaji.

Stav „Running“

Do tohoto z hlediska funkcionality klíčového stavu se může aplikace dostat ze dvou výchozích stavů. Buď se reaktivuje ze stavu *Suspended*, tedy aplikace už byla předtím spuštěná, nebo se

zavede do WinRT a spustí. Tento proces nastane tehdy, když aplikace ještě nebyla spuštěna nebo byla ve stavu *Suspended* odstraněna z paměti, například kvůli nedostatku paměťové kapacity.

Při spouštění aplikace se zobrazí úvodní obrazovka (anglicky splash screen), kterou uživatel vidí během inicializace hlavních úloh, které tvoří aplikaci. Měla by být zobrazena jen co nejkratší čas nezbytně nutný k inicializaci aplikace. Podle doporučení Microsoftu by pro aplikace Modern UI ve Windows 8.1 neměla doba inicializace přesáhnout 5 sekund. Jakmile jsou všechny hlavní úlohy inicializované, úvodní obrazovka se zavře a zobrazí se uživatelské rozhraní aplikace. Stav aplikace se změní z *NotRunning* na *Running*. Ve stavu *Running* realizuje aplikace činnosti, ke kterým byla vytvořena, včetně interakce s uživatelem.

Stav „Suspended"

Když uživatel přesune aplikaci na pozadí, například přejde na úvodní obrazovku, případně se gestem „swipe" (nájezd prstem dovnitř obrazovky přes její okraj z pravé strany) přepíná mezi aplikacemi, nebo zařízení, na kterém aplikace běží, přejde do úsporného stavu, aplikace se sama pozastaví. Přejde do stavu *Suspended*.

Mezi stavem *Running* a *Suspended* nastane přechodový stav, kdy je aplikace už na pozadí, systém ale ještě 10 sekund čeká, aby se ubezpečil, že uživatel nechce přejít zpět do aplikace. Stává se to při přepínání mezi aplikacemi a je určitá pravděpodobnost, že se uživatel z jiného programu, případně úvodní obrazovky, přepne znovu do vaší aplikace. Proto je po dobu 10 sekund program na pozadí brán jako běžící, nedochází k jeho pozastavení a po přepnutí je náběh okamžitý.

Až po uplynutí tohoto časového intervalu systém automaticky přesune aplikaci do úsporného režimu. Aplikace ve stavu *Suspended* je sice zavedena v paměti, v tomto stavu ale nemůže vykonávat žádnou činnost. Během přechodu může aplikace například uložit neuložená data do úložného zařízení. Na toto uložení dat má maximálně 5 sekund.

Podle toho, kolik má systém k dispozici paměťové kapacity, se snaží udržet pozastavené aplikace v paměti, čímž zabezpečí, že uživatel může rychle přejít zpět do aplikace. Aplikaci je možné ukončit v místní nabídce nebo jejím uchopením v horní části a „shozením" dolů. Aplikace se nejprve suspenduje a potom ukončí. Jinak zůstává aplikace uložená v paměti tak dlouho, dokud má systém k dispozici dostatek paměti. Pokud začíná být volné paměti nedostatek, systém na tento stav reaguje tak, že automaticky odstraní nepoužívané aplikace z paměti. Systém aplikaci neoznámí, že ji ukončuje, proto je potřeba zpracovat neuložená data během přechodu do stavu *Suspended*.

Aplikace má k dispozici událost `Suspending`, která je generovaná těsně před tím, než je program přerušen. V obsluze události je potřeba uložit všechny údaje, se kterými aplikace aktuálně pracuje.

Poznámka: Zmíněných 5 sekund, které má aplikace k dispozici na uložení údajů, nemusí u velkých objemů stačit. Správně navržená aplikace by měla údaje ukládat nepřetržitě za běhu aplikace, nejlépe tehdy, když dochází k jejich změně, a v obsluze události `OnSuspending` uloží jen aktuální stav uživatelského rozhraní.

Doporučuje se uvolnit i všechny výhradně alokované systémové zdroje. O tom, že 5 sekund na přechod do stavu *Suspended* uplynulo, se aplikace už nedozví a nemá žádnou možnost, jak tento čas prodloužit, a nedokáže ani přechodu do stavu *Suspended* nijak zabránit. Pokud aplikace zpracovává údaje i po uplynutí tohoto časového intervalu, operační systém se domnívá, že aplikace přestala reagovat, a explicitně ji ukončí.

Jaký je další osud aplikace ve stavu *Suspended*? Po delší době spuštění operačního systému, kdy uživatel podle potřeby přechází mezi aplikacemi, nastane stav, kdy je v paměti zavedených několik aplikací, uživatel ale aktuálně pracuje jen s jednou z nich. Operační systém nechává v tomto stavu tolik aplikací, kolik mu operační paměť umožňuje. Pokud dojde k nedostatku alokovatelné paměti, operační systém ukončí běh některých aplikací ve stavu *Suspended*. Ani o této aktivitě není žádná aplikace žádným způsobem informována.

Přechodový stav „Resuming"

Aplikace se přenese z paměti do popředí. Během přechodu by měla načítat údaje, které byly uloženy během pozastavení. Stav aplikace se změní z pozastaveného *Suspended* na běžící *Running*. Aplikace by měla pokračovat v činnosti tam, kde přestala předtím, než byla pozastavena. Při obnovení aplikace je generovaná událost `Resuming`, kterou je potřeba vhodně obsloužit, načíst uložené údaje a obnovit stav objektů.

U aplikací, které pracují s údaji z Internetu, je potřeba jejich aktualizace. Aplikace totiž může být pozastavena i několik hodin, či dokonce dní, pokud se notebook nevypíná, ale pouze klasicky uspí. Nové tablety navržené pro Windows 8.1 mají k dispozici i režim *Connected Standby*. V něm je možné uspat celé zařízení s výjimkou aplikací a procesů, které potřebují běžet na pozadí. Tyto mají k dispozici připojení k síti, aby mohly reagovat na podněty zvenku. Po zpracování podnětu se zařízení opět vrátí do režimu spánku. To však neplatí pro aplikace ve stavu *Suspended*, které jsou skutečně pozastavené.

Ukončení aplikace

Tato část je i není chytákem. Psali jsme, že cyklus ukončování aplikací řídí operační systém. Uživatelé nemusí explicitně ukončovat aplikace a měli by přenechat tuto činnost operačnímu systému. Uživatel napříč tomu má možnost kteroukoliv aplikaci ukončit například pomocí příslušného gesta.

Upozornění: Samotná aplikace **nesmí** mít žádný ovládací prvek na svoje ukončení. V opačném případě nesplňuje základní požadavky na aplikace Modern UI a neprojde certifikačním a schvalovacím procesem pro Windows Store.

Pokud se aplikace ukončí sama od sebe, Windows 8.1 ji bude považovat za ukončenou chybovým stavem, jinak řečeno havarovanou.

I při explicitním ukončení aplikace uživatelem přes ovládací prvky operačního systému je nejprve navozen stav *Suspended* a aplikace si může uložit údaje. Až potom je ukončena, čili ve stavu *NotRunning*.

Když jsme zmínili havárii aplikace, Windows 8.1 bere v úvahu i tento stav. K vnějším příznakům tohoto stavu patří například nemožnost interakce s uživatelem ve stavu, kdy je interakce vyžadována. Například v případě karetní hry se čeká na přesun nebo odkrytí karty, ale uživatel nemůže tuto akci provést, jelikož aplikace nereaguje na gesta ani na alternativní ovládání pomocí klávesnice a myši. Podle politik pro Windows 8 by se aplikace po identifikaci takovéhoto stavu měla urychleně dostat do výchozího místa operačního systému, tedy na úvodní obrazovku, a pokud je to povoleno, odešle aplikace údaje o chybě Microsoftu. Ten je odevzdá vývojáři, aby chybu urychleně odstranil.

Příklad řízení údajů během životního cyklu aplikace

Nejlépe vysvětlíme manipulaci s údaji během životního cyklu aplikace na příkladu jednoduché aplikace, která bude ukládat 4 textové údaje (jméno, město, zvíře, věc).

Poznámka: Ukládání údajů je podrobně vysvětleno v osmé kapitole.

Vytvoříme projekt aplikace XAML C#. Kód XAML uživatelského rozhraní bude:

```
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
  <StackPanel Orientation="Vertical" Margin="120,120,80,60">
    <TextBlock Text="Meno"></TextBlock>
    <TextBox x:Name="tbMeno"  Width="200" HorizontalAlignment="Left"
      KeyUp="tbMeno_KeyUp" />
    <TextBlock Text="Mesto"></TextBlock>
    <TextBox x:Name="tbMesto" Width="200" HorizontalAlignment="Left"
      KeyUp="tbMesto_KeyUp"  />
    <TextBlock Text="Zviera"></TextBlock>
    <TextBox x:Name="tbZviera" Width="200" HorizontalAlignment="Left"
      KeyUp="tbZviera_KeyUp" />
    <TextBlock Text="Vec"></TextBlock>
    <TextBox x:Name="tbVec" Width="200" HorizontalAlignment="Left"
      KeyUp="tbVec_KeyUp" />
  </StackPanel>
</Grid>
```

V souboru *App.xaml.cs* přidejte potřebné deklarace jmenných prostorů a vytvořte třídu pro údaje, které se budou ukládat:

```
using System.Runtime.Serialization;
using System.Threading.Tasks;
using Windows.Storage;
using Windows.Storage.Streams;

public class Udaje
{
  public string meno { get; set; }
  public string mesto { get; set; }
  public string zviera { get; set; }
  public string vec { get; set; }
  public bool JeHodnota { get; set; }
}
```

Třída s údaji bude deklarována a vytvořena v konstruktoru třídy `App` v souboru *App.xaml.cs*.

```
sealed partial class App : Application
{
  static public Udaje data = new Udaje();
  private const string filename = "UlozeneUdaje.mmz";
...
```

Poznámka: Přestože budeme údaje ukládat ve formátu XML, úmyslně jsme použili vlastní příponu MMZ, jelikož jak uvidíte později, budeme muset pro aplikaci zaregistrovat příponu souborů, ve kterých ukládá údaje.

Doporučuje se ukládat údaje stále za běhu aplikace a v události `OnSuspending` uložit jen aktuální stav uživatelského rozhraní. Je potřeba najít vhodnou příležitost k průběžnému ukládání údajů, například po jejich potvrzení uživatelem, případně v okamžiku, když uživatel dokončil zadávání údajů v jednom ovládacím prvku, například v editačním poli vytvořeném pomocí prvku `TextBox`, případně pokud zadáváme delší údaje a předpokládáme, že se uživatel občas přepne do jiné aplikace, je rozumné zadávané údaje průběžně ukládat po napsání každého znaku.

Poznámka: Přepnutí do jiné aplikace během psaní delšího textu, například e-mailové zprávy, je ve Windows 8 mnohem pravděpodobnější, než byste očekávali. Například přijde oznámení ze sociální sítě a uživatel se na chvíli přepne do příslušné aplikace, aby na zprávu operativně reagoval.

V našem příkladu je proto v každém prvku `TextBox` obsluha události `KeyUp`. Uvádíme kód (v souboru *MainPage.xaml.cs*) jen pro jeden atribut.

```
private void tbMeno_KeyUp(object sender, KeyRoutedEventArgse)
{
  App.data.JeHodnota = tbMeno.Text.Length > 0 || tbMeno.Text.Length > 0;
  App.data.meno = tbMeno.Text;
}
```

V konstruktoru třídy `App` je už registrovaná obsluha události přechodu do stavu *Suspended*.

```
public App()
{
  this.InitializeComponent();
  this.Suspending += OnSuspending;
}
```

Máte k dispozici připravené tělo procedury, ve kterém je pomocí komentáře `TODO` naznačeno, kam je potřeba vložit kód na ukládání údajů.

```
private void OnSuspending(object sender, SuspendingEventArgs e)
{
   var deferral = e.SuspendingOperation.GetDeferral();
  //TODO: Save application state and stop any background activity

  deferral.Complete();
}
```

Jelikož se pro akce, které trvají, případně mohou trvat, určitý čas, typicky nad 50 milisekund, používají asynchronní metody, i v našem příkladu použijeme asynchronní uložení. Všimněte si změny deklarace metody z `private` na `public async`.

```
public async void OnSuspending(object sender, SuspendingEventArgs e)
{
  var deferral = e.SuspendingOperation.GetDeferral();
  //TODO: Save application state and stop any background activity
  await AsynchronneUlozenie();
  deferral.Complete();
}
```

Kód uložení údajů bude v samostatné metodě.

```
async static public Task AsynchronneUlozenie()
{
  StorageFile file = await
    ApplicationData.Current.LocalFolder.CreateFileAsync(filename,
    CreationCollisionOption.ReplaceExisting);
    IRandomAccessStream raStream = await
    file.OpenAsync(FileAccessMode.ReadWrite);
  using (IOutputStream outStream = raStream.GetOutputStreamAt(0))
  {
    //serializácia.
    DataContractSerializer serializer = new
      DataContractSerializer(typeof(Udaje));
    serializer.WriteObject(outStream.AsStreamForWrite(), data);
    await outStream.FlushAsync();
  }
}
```

Uložené údaje je potřeba při opětovném přepnutí se do aplikace načíst. Při zavedení aplikace se volá metoda `OnLaunched`. Pokud aplikace ukládá údaje, je v této metodě potřeba otestovat, zda se aplikace vrací ze stavu *Suspended*, nebo se spouští nanovo. Opět máte komentářem v kostře metody naznačeno, kam je potřeba kód načítání umístit. I v tomto případě je potřeba přidat do deklarace metody `async`.

Tip: Komentáře implicitně umístěné vývojovým prostředím v této proceduře jsou velmi poučné, doporučujeme je prostudovat.

```
protected async override void OnLaunched(LaunchActivatedEventArgs args)
{
  Frame rootFrame = Window.Current.Content as Frame;

  // Do not repeat app initialization when the Window already has content,
  // just ensure that the window is active
  if (rootFrame == null)
  {
    // Create a Frame to act as the navigation context and navigate
    // to the first page
    rootFrame = new Frame();
```

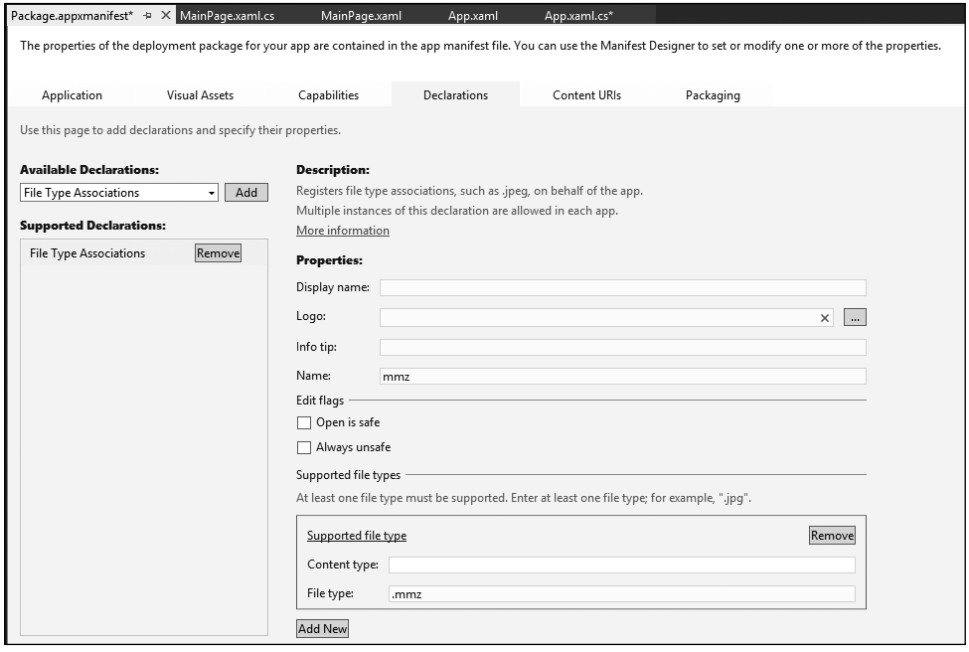
```
    if (args.PreviousExecutionState ==
      ApplicationExecutionState.Terminated)
    {
      //TODO: Load state from previously suspended application
      await AsynchronneObnovenie();
    }

    // Place the frame in the current Window
    Window.Current.Content = rootFrame;
  }

  if (rootFrame.Content == null)
  {
    // When the navigation stack isn't restored navigate to the first page,
    // configuring the new page by passing required information
    // as a navigation parameter
    if (!rootFrame.Navigate(typeof(MainPage), args.Arguments))
    {
      throw new Exception("Failed to create initial page");
    }
  }
  // Ensure the current window is active
  Window.Current.Activate();
}
```

Načítání údajů je také asynchronní. Kód je v samostatné metodě.

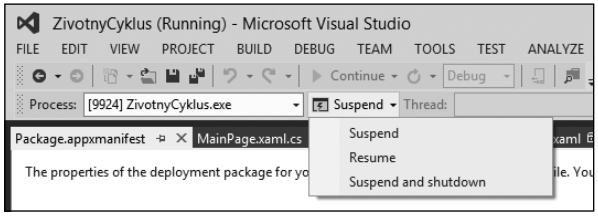
```
static async public Task AsynchronneObnovenie()
{
  StorageFile file = await
    ApplicationData.Current.LocalFolder.GetFileAsync(filename);
  if (file == null) return;
  IInputStream inStream = await file.OpenSequentialReadAsync();
  DataContractSerializer serializer =
    new DataContractSerializer(typeof(Udaje));
  data = (Udaje)serializer.ReadObject(inStream.AsStreamForRead());
}
```

Otevřete aplikační manifest a na kartě **Declarations** přiřaďte aplikaci typ souboru, v našem případě MMZ. Přejděte tedy na tuto kartu a přidejte **File Type Associations**.

Nyní můžete aplikaci vyzkoušet. Stav *Suspended* je možné po spuštění aplikace v simulátoru navodit pomocí nabídky **Suspend** na panelu nástrojů vývojového prostředí.



Obrázek 4.2: Zaregistrování typu souboru



Obrázek 4.3: Navození stavu Suspended a obnovení aplikace

Asynchronní programování

Aplikace s novým uživatelským rozhraním pro Windows 8.1 využívají asynchronní programování v mnohem větším měřítku, než bylo zvykem u klasických desktopových aplikací pro předchozí verze operačních systémů (případně pro desktopové aplikace ve Windows 8.1). Smyslem asynchronního programování je zabránění zpomalení, či dokonce přerušení odezvy na povely od uživatele bez ohledu na to, zda jsou zadávané dotykovými gesty, pomocí myši či klávesnice. Jinými slovy, aplikace se vůči uživateli nesmí jevit jako nečinná.

Například aplikace, která stahuje informace z Internetu, může strávit několik sekund čekáním na požadované informace. Při použití synchronní metody pro toto vlákno je aplikace blokovaná, dokud tato metoda nenačte údaje. Aplikace nebude reagovat na interakci s uživatelem, což

může způsobit nespokojenost uživatele, případně až pocit, že aplikace „zamrzla". Mnohem lepší je použít asynchronní přístup, kde aplikace, zatímco čeká na dokončení asynchronní operace, běží a reaguje na uživatelské rozhraní.

Typickými příklady pro asynchronní volání Windows Runtime API jsou:

- Odesílání a přijímání údajů do/z Internetu
- Zobrazení oznámení
- Práce se soubory

Každý z podporovaných programovacích jazyků řeší asynchronní operace jinak.

Programovací jazyk	Asynchronní operace
JavaScript	Objekty `promise`, funkce `then`
C#	Objekty `future`, operátor `await`
Microsoft Visual Basic .NET	Objekty `future`, operátor `await`
Microsoft Visual C++	Třída `task`, metoda `then`

Poznámka: Podle doporučení pro aplikace Modern UI ve Windows 8.1 by každá akce, jejíž trvání přesáhne 50 milisekund, měla být naprogramovaná jako asynchronní. Toto pravidlo je uplatněno i v knihovně tříd (BCL), kde je hodně metod asynchronních, přičemž jejich synchronní ekvivalenty vůbec neexistují.

Dynamické dlaždice

Dlaždice v novém prezentačním rozhraní Windows 8.1 umožňuje uživateli spustit aplikaci nebo se přepnout do už spuštěné aplikace. Na rozdíl od jiných platforem to nejsou jen statické ikony, ale dokážou v souladu s účelem příslušné aplikace, kterou na domovské obrazovce zastupují, dynamicky zobrazovat rozmanitý informační, případně ilustrační, obsah, a to i tehdy, když aplikace není spuštěna.

Poznámka: Dynamické dlaždice jsou díky rozšířené funkcionalitě vděčným objektem pro designéry a copywritery. Plní úlohu výkladní skříně aplikace. Jejich úlohou je nalákat uživatele, aby aplikaci spustil, případně se do ní vrátil.

Princip fungování dynamických dlaždic

Po přečtení části o životním cyklu aplikací jste se dozvěděli, že aplikace ve stavech *Suspended* ani *Terminated* nemá přidělovány žádné systémové zdroje, a nemůže tedy vykonávat žádnou činnost. V tomto kontextu se může na první pohled fungování dynamických dlaždic jevit jako záhada. Jak dokáže aplikace dynamicky obnovovat texty a obrázky na dynamických dlaždicích?

Nezávisle na tom, zda je dlaždice čtvercová (150 × 150 pixelů), velká čtvercová (310 × 310 pixelů) nebo obdélníková (310 × 150 pixelů), pokud je připnutá na úvodní obrazovku, může být

dynamická. To znamená, že může zobrazovat adekvátní obsah, nejen textový, ale i obrázky ve formátech JPEG nebo PNG maximálně do velikosti 150 kilobajtů. Předdefinovanou sekvenci obrázků můžete využít k animaci. Statické jsou jedině ikony o rozměrech 32 × 32 pixelů, které se zobrazí například na obrazovce při vyhledávání aplikací. Standardně se zobrazuje jen nejnovější aktualizace, lze ale i cyklicky zobrazovat nejnovějších 5 notifikací (Notification Queuing).

Poznámka: Připomínáme jednu důležitou věc. Z hlediska navázání interakce je dlaždice pasivní prvek, to znamená, že obsah, který zobrazuje, uvidí uživatel pouze tehdy, pokud se přepne na úvodní obrazovku. Pokud potřebuje aplikace něco uživateli operativně oznámit, má pro tento účel k dispozici jiné výrazové prostředky.

Obrázek 4.4: Dynamické dlaždice mohou zobrazovat textové a obrazové informace

Dynamické dlaždice z pohledu designéra

Bez ohledu na to, podle jaké šablony vytvoříte aplikaci pro Windows Store, má v aplikačním manifestu předdefinovanou čtvercovou i obdélníkovou dlaždici a také i malou dlaždici (SmallLogo), která se bude v operačním systému zobrazovat v seznamu aplikací, výsledcích vyhledávání a podobně. Přiřazení statických obrázků k jednotlivým velikostem dlaždic se realizuje v aplikačním manifestu (soubor *Package.appxmanifest*), který se ve Visual Studiu implicitně otevírá ve vizuálním návrhovém okně **Manifest Designer**.

Všimněte si, že je možné přiřadit více obrázků v různých měřítcích. Jelikož aplikace může být spouštěna na různých zařízeních, která mají různá rozlišení i velikost displeje, všechny grafické prvky tvořící vizuál aplikace mohou být ve složce *Assets* v různých velikostech, jejichž měřítko je vyjádřeno procentuálně. Konkrétně se jedná o 80, 100, 140 a 180 procent základního rozměru.