

**Příloha 1:**

# **Microsoft Silverlight 2.0**

**Příručka pro začínající programátory**

**Jiří Kolda**

2009

# Obsah

<b>1. ÚVOD.....</b>	<b>4</b>
1.1. CO JE SILVERLIGHT?.....	4
1.2. SILVERLIGHT VS. ASP.NET.....	5
1.3. SILVERLIGHT VS. ADOBE FLASH.....	6
1.4. HISTORIE.....	9
1.5. POUŽITÍ TECHNOLOGIE.....	10
1.6. POTŘEBNÉ ZNALOSTI.....	12
1.7. PODPORA PROHLÍŽEČŮ.....	13
<b>2. XAML.....</b>	<b>15</b>
2.1. POPIS ELEMENTŮ POUŽITELNÝCH V SILVERLIGHT 2.0.....	17
2.2. KOŘENOVÝ ELEMENT.....	17
2.3. LAYOUT - ROZVRŽENÍ KOMPONENT NA STRÁNCE.....	17
2.4. ZÁKLADNÍ KOMPONENTY.....	22
2.5. GRAFICKÝ DESIGN.....	26
2.6. STYLY, ŠABLONY.....	30
2.7. ANIMACE.....	34
2.8. MÉDIA.....	42
2.9. DATA.....	44
<b>3. C#.....</b>	<b>48</b>
3.1. PROPOJENÍ C# A XAML.....	48

<b>4.</b>	<b>VÝVOJOVÁ PROSTŘEDÍ.....</b>	<b>51</b>
4.1.	MICROSOFT VISUAL STUDIO 2008.....	51
4.2.	MICROSOFT BLEND 2.....	54
<b>5.</b>	<b>PRVNÍ APLIKACE V SILVERLIGHT 2.0.....</b>	<b>56</b>
5.1.	STRUČNÝ POPIS PRÁCE S VISUAL STUDIO 2008.....	56
5.2.	ROZVRŽENÍ A ADRESÁŘOVÁ STRUKTURA.....	59
5.3.	SOUBOR ".XAP" .....	60
5.4.	PUBLIKOVÁNÍ SILVERLIGHT APLIKACE.....	60

# 1 Úvod

S příchodem operačního systému Windows Vista představila společnost Microsoft několik nových technologií. Mezi ně patří například Windows Presentation Foundation (WPF), Windows Communication Foundation (WCF) nebo Windows Workflow Foundation (WWF). Hlavním cílem technologie WPF je zjednodušení tvorby grafického vzhledu aplikací a oddělení grafického návrhu od aplikační logiky. Grafický design je zde založen na jazyku XAML (o němž pojednává druhá kapitola), který je založen na XML technologii, a programování v něm je podle mého mínění velmi jednoduché a přehledné.

Po zavedení technologie WPF začala společnost Microsoft vyvíjet technologii s pracovním názvem WPF/E, jejímž účelem byla možnost zobrazení WPF aplikací v prohlížeči Internet Explorer. Později se název technologie změnil z WPF/E na Silverlight, a zrodila se technologie, která není jakýmsi WPF emulátorem, nýbrž je schopna pomocí speciálního pluginu využívat XAML knihoven k tvorbě webových aplikací.

## 1.1 Co je Silverlight?

Microsoft Silverlight je multiplatformní implementace .NET Frameworku určená pro tvorbu aplikací RIA a mediálních prezentací na webu. Silverlight je současně také tzv. "cross-browser" technologií, což znamená, že by měl být podporován všemi verzemi všech prohlížečů dostupných v době představení technologie. Tato vlastnost je samozřejmě limitována tím, že Silverlight je úzce spojen s operačním systémem Windows, takže na konkurenčních operačních systémech již není podpora zaručena.

## 1.2 Silverlight vs. ASP.NET

Kdo někdy vytvářel web pomocí ASP.NET (dále jen ASP), ten při seznámení se s technologií Silverlight ihned pozná několik podobností. Přirozeně je to proto, že obě technologie využívají platformu .NET - Silverlight využívá .NET Framework 3.0 a 3.5. V podstatě je možno se na Silverlight dívat jako na jistou odnož ASP technologie, přičemž můžeme použít následující analogii: stejně jako systém Windows Forms (Windows XP a starší) přešel na Windows Presentation Foundation (Windows Vista, budoucí Windows Seven), tak z technologie ASP, využívající grafické knihovny Windows Forms, vznikla technologie Silverlight, která využívá grafické knihovny WPF.

Jaké jsou tedy rozdíly mezi technologiemi ASP a MS Silverlight? Ten hlavní spočívá ve způsobu komunikace uživatele se serverem. V ASP uživatel posílá na server požadavky na jednotlivé stránky a na serveru je, aby vyrenderoval každou vyžádanou stránku podle potřeb uživatele a odeslal mu ji jako celek, zatímco v Silverlight uživatel pracuje s jednou celistvou aplikací, přes kterou odesílá na server požadavky pouze o dílčí prostředky. Server s těmito požadavky nemusí nijak manipulovat, pouze odešle uživateli příslušná data a Silverlight aplikace je zpracuje sama. Z toho vyplývá, že v Silverlight je naprostá většina dat zpracovávána na straně uživatele, zatímco v ASP na serveru.

### 1.3 Silverlight vs. Adobe Flash

Jedním z důvodů vzniku technologie Silverlight byla i přímá konkurence technologii Adobe Flash (dále Flash) ze strany společnosti Microsoft.

Největší rozdíl nalezneme nejspíše v systému animace. Flash poskytuje tzv. frame-by-frame animaci. To znamená, že pro každou změnu stavu určitého objektu na scéně za jednotku času je vypočítáno, jak přesně bude daný objekt vypadat na všech snímcích, které tuto změnu zobrazují. Bohužel to znamená, že má-li uživatel pomalejší počítač, pak může zobrazení každého jednotlivého snímku trvat delší dobu, než by mělo, a výsledkem je prodloužení času, který je na animaci vyhrazen. Oproti tomu Silverlight pracuje na principu stavové animace. Tento princip vyžaduje pouze určení počátečního a finálního stavu objektu a času, během kterého má být změna provedena. Silverlight sám dokáže zjistit, zda bude výkon počítače na zobrazení stačit, a přizpůsobit příslušné mezistavy objektu tomu, aby změna opravdu proběhla v zadaném čase.

Obecně by se dalo říci, že technologie Silverlight je v mnohém jednodušší a pro vývojáře příjemnější než Flash, jelikož hodně práce udělá za něj. Oproti tomu technologie Flash svou volností (většinu práce musí obstarat sám vývojář) umožňuje vytvářet graficky i funkčně unikátní aplikace, zatímco Silverlight je odkázán na grafické možnosti jazyka XAML.

Můj názor je jakousi zlatou střední cestou. Pro grafickou stránku webové aplikace bych jednoznačně zvolil technologii Adobe Flash a pro programovou část bych použil Silverlight. Přesto však nepochybuji, že se vždy najde někdo, kdo bude tvrdit, že jemu se lépe programuje ve Flashi, a zároveň někdo, komu více vyhovují grafické knihovny jazyka XAML.

Naprosto stežejním předmětem porovnání je také podpora video formátů. Technologie Silverlight totiž nabídla už v první verzi možnost přehrávání videa v HD kvalitě, zatímco Flash tuto vlastnost neměl. Konkrétním kodekovým standardem, který umožňuje technologii Silverlight přehrávat HD video je VC-1. Dnes již společnost Adobe odpověděla na tuto novinku tím, že do Flash

Playeru začlenila podporu standardu H.264, také umožňujícího přehrávání videa v HD rozlišení. V několika následujících odstavcích popisují rozdíly, výhody a nevýhody jednotlivých standardů.

Výhodou VC-1 je, že umožňuje přímo kódovat prokládané video. H.264 tuto schopnost nemá, nýbrž vyžaduje, aby před kódováním bylo prokládané video převedeno na progresivní. Tento postup pak samozřejmě vyžaduje více času i práce při manipulaci s videem.

Další zásadní vlastností VC-1 je to, že pro přehrávání (resp. dekódování) videa není zapotřebí mnoho výpočetního výkonu. Dá se říci, že za tímto účelem byl standard VC-1 společností Microsoft vyvíjen. Ruku v ruce s odlehčeným dekódováním jde ovšem větší výpočetní náročnost při kódování videa. Zde VC-1 prohrává pomyslný souboj s H.264, který má o něco nižší nároky na kódování, ale vyšší na dekódování.

Co se týče kvality obrazu, zde se názory liší. Podle mého soudu jsou oba standardy schopny nabídnout velmi vysokou kvalitu obrazu. Jak již bylo řečeno, VC-1 se snaží o co největší kvalitu při co nejnižší výpočetní náročnosti při dekódování. Oproti tomu H.264 byl navržen hlavně pro široké využití v nejrůznějších oblastech, má mnoho profilů s různou kvalitou obrazu, kompresním poměrem a výpočetní náročností.

Na závěr je třeba také zmínit podporu obou standardů na koncových zařízeních. Standard VC-1 vyhovuje hlavně zařízením s malým výkonem kvůli odlehčenému dekódování. Je však třeba říci že tento standard ještě není příliš rozšířen a je v současné době podporován především v HD-Ready televizorech a DVD přehrávačích. Oproti tomu H.264 je již dnes velmi rozšířen, především na produktech společnosti Apple, Xbox 360, PSP či na novějších mobilních telefonech Nokia.

Vzhledem k tomu, že Silverlight má přímo v runtime pluginu zabudován kodek WMV9 (nejpopulárnější implementaci standardu VC-1), není třeba stahovat k zobrazení videí žádné další knihovny či moduly.

Velmi zajímavou službou, kterou společnost Microsoft poskytuje, je tzv. Silverlight Streaming. Jedná se o bezplatný hosting pro Silverlight aplikace, ale především také pro videa v HD kvalitě. Na stránkách *silverlight.live.com* má každý registrovaný uživatel k dispozici až 10GB prostoru k hostování aplikací nebo ukládání videosouborů.



## 1.4 Historie

Název "Silverlight" jako takový se poprvé objevil v roce 2007 na konferenci MIX07. Předtím se o této technologii vědělo jen sporadicky a měla označení WPF/E (Windows Presentation Foundation/Everywhere). Již od začátku vývoje bylo hlavním cílem technologie přenést funkcionalitu Windows do webového prohlížeče a umožnit tak použití nových technologií při vytváření webových prezentací. Vzhledem k lepší orientaci zde nebudu odkazovat na dřívější verze jako na WPF/E, ale raději i u těchto verzí budu mluvit o technologii Silverlight.

První verzí byl, jak již označení napovídá, Silverlight 1.0 (dále SL1.0). Tato verze umožňovala použít pouze několik málo elementů jazyka XAML. Velmi důležitým faktem je, že veškerá interaktivita byla zajišťována Javascriptem. Propojení XAML a Javascriptu bylo přímé, tedy skripty nebyly kompilované. SL1.0 neobsahoval žádné "layout komponenty", hlavním prvkem byl element <Canvas> a všechny ostatní komponenty se musely umisťovat absolutně pomocí souřadnic. Grafický design byl zajištěn pomocí 2D geometrických tvarů, element <Storyboard> umožňoval jednoduché animace a také bylo možno zobrazovat obrázky a video. Velmi zajímavým počinem bylo použití tzv. "štetců" [brush], díky nimž šlo vyplnit obrazce buď jednou barvou, lineární interpolací více barev, obrázkem či dokonce videem.

Jakousi "mezi-verzí" byl Silverlight 1.1, vydaný velmi brzy po uvedení verze 1.0. Hlavním aspektem této verze byl "multiplatformní CLR". Šlo o spojení technologie Silverlight s platformou .NET, což velmi rozšířilo využití technologie a usnadnilo vývoj internetových aplikací.

Po několika měsících, těsně před zmiňovanou konferencí MIX v roce 2007, společnost Microsoft oznámila, že Silverlight 1.1 bude vydán jako samostatná verze Silverlight 2.0, a to z důvodu velkého počtu nových vylepšení. Verze 2.0 již obsahuje balík základních ovládacích prvků a umožňuje síťovou komunikaci za použití WCF. Zároveň se rapidně zvýšil počet podporovaných XAML elementů. Hlavním rysem však je fakt, že interaktivitu lze zajistit všemi

běžnými .NET jazyky (C#, Visual Basic) společně s jazyky JavaScript a IronPhyton.

### 1.5 Použití technologie

V této části se zmíním o hlavních výhodách a nevýhodách, které technologie Silverlight obnáší a následně popíši, jak se s technologií pracuje a jak je podporována v nejpoužívanějších prohlížečích.

#### *Výhody*

Nespornou výhodou technologie Silverlight je spojení s platformou .NET. Díky tomu lze totiž využít hned několika programovacích jazyků, je k dispozici velmi široká nabídka knihoven a komponent a to vše se používá velmi podobně, jako kdybychom vytvářeli Windows aplikace.

Další velkou výhodou je, že Silverlight je cross-browser multiplatformní technologií. Je tedy podporována téměř všemi prohlížeči (po instalaci pluginu) a je tedy velmi snadno použitelná. Dá se říci, že jakmile vytvoříte Silverlight aplikaci a umístíte ji na web, měla by se ve všech prohlížečích chovat naprosto stejně a měla by i stejně vypadat.

K Silverlight existuje celá řada volně stažitelných komponent, ačkoli hlavní výhodou není to, že si lze spoustu komponent stáhnout, ale samotný fakt, že je velmi jednoduché si vlastní komponenty vytvářet pomocí jazyka XAML. V další kapitole přímo popíši jak.

Na závěr je třeba jako výhodu zmínit i fakt, že v současní době je vyvíjena technologie Moonlight jakožto open-source implementace technologie Silverlight. Technologie Moonlight si dává za cíl umožnit spouštění a vývoj technologie Silverlight na operačním systému Linux a ostatních Unixových OS. Zatím je k dispozici verze 1.0, která implementuje Silverlight 1.0, v budoucnu se počítá s verzí 2.0 (v současné době existuje pouze alfa verze), která bude implementovat Silverlight 2.0.

### *Nevýhody*

Je pravdou, že Silverlight přichází trochu pozdě, tedy alespoň co se týče konkurence pro Adobe Flash. Flash je v současné době již mnohokrát vylepšenou, stabilní a všudypřítomnou technologií a Silverlight bude mít co dělat, aby tento předstih dohnal.

Mezi nevýhody dále podle mého názoru patří také nástroje pro návrh grafického designu Silverlight aplikací - Expression Blend a Expression Design. Tyto nástroje jsem vyzkoušel a z mého pohledu se zdaleka nemohou vyrovnat nejčastěji používanému nástroji Adobe Photoshop.

Velkou nevýhodou technologie Silverlight je fakt, že neexistuje podpora mobilních zařízení (řeč je o SL 2.0). Společnost Microsoft ještě před nedávnem tvrdila, že Silverlight bude podporován na poslední verzi Windows Mobile, ale zatím podpora stále chybí.

Vzhledem k tomu, že pro vytváření Silverlight aplikací je třeba mít MS Visual Studio, je třeba počítat s tím, že zatím neexistuje vhodná a stabilní cesta, jak tento software spustit na jiném operačním systému než MS Windows.

Na závěr bych uvedl fakt týkající se technologie Silverlight, který je podle mého názoru velmi problémový. Tím je naprostá zpětná nekompatibilita pluginu pro zobrazení Silverlight aplikací v prohlížeči. To znamená, že pro správné zobrazení aplikace vytvořené například ve verzi SL 1.1 musí mít uživatel nainstalovanou právě tu samou verzi. Problém nastane ve chvíli, kdy má uživatel nainstalovanou verzi novější a přesto se mu aplikace nezobrazí. V tomto směru tedy Silverlight zatím nemůže konkurovat ostatním technologiím (např. Adobe Flash).

## 1.6 Potřebné znalosti

Jak jsem již uvedl výše, technologie Silverlight je založena na kombinaci jazyka XAML se standardními .NET jazyky. Již z této základní charakteristiky vyplívá, že hlavní potřebnou dovedností je znalost jazyka XAML. Tento programovací jazyk zajišťuje grafický design aplikace. Co se týče interaktivity aplikace, k tomu již můžeme použít jednu z více možných alternativ. Já pracuji s jazykem C# a tento bude použit i při vysvětlování jednotlivých příkladů v této publikaci. Kromě tohoto jazyka lze použít jazyky Visual Basic, JScript, IronRuby a IronPhyton.

Pokud uživatele zajímá tato technologie a má v úmyslu s její pomocí vyvíjet internetové aplikace, pak nejnazší cesta povede přes technologii ASP.NET, která je technologii Silverlight nejpodobnější. Pokud již uživatel ASP.NET ovládá, pak stačí jen přejít na principy WPF, WCF a dalších nových technologií společnosti Microsoft. Pak již Silverlight nebude dělat žádné problémy.

### 1.7 Podpora prohlížečů

V následující tabulce je znázorněno, jaké prohlížeče a na jakých operačních systémech podporují technologii Silverlight. Osobně jsem otestoval funkcionality technologie na prohlížečích Internet Explorer 7, Mozilla Firefox 3.0.6, Google Chrome, Opera 9.63, Netscape Navigator 9 a Safari 3.2.2, a to za použití operačního systému Windows XP Professional SP3. Jelikož není možné mít nainstalované zároveň dvě verze Silverlight, testoval jsem pouze verzi 2.0, neboť právě touto verzí se zabývá celá tato práce. Následující tabulka zobrazuje výsledky testování.

OS	IE7	Firefox 3	Chrome	Opera 9.63	Netscape 9	Safari
Windows XP	ANO	ANO	ANO	ANO	NE	ANO
Windows Vista	ANO	ANO	ANO	ANO	NE	ANO
Mac OS X	NE	ANO	NE	NE	NE	ANO

Během testování se zobrazoval Silverlight ve všech podporovaných prohlížečích stejně a pracoval stabilně. Velmi zásadním problémem však zůstává zmiňovaná zpětná nekompatibilita pluginu. Ve všech prohlížečích nastávaly problémy v případech, kdy zobrazovaná aplikace byla vytvořena v jiné verzi Silverlight, než jaká je nainstalována v prohlížeči. V prohlížečích Internet Explorer a Mozilla Firefox se při zobrazení starších verzí (1.0, 1.1) zobrazilo upozornění, že Silverlight plugin není nainstalován a při následné instalaci instalátor vypsal, že je již nainstalována novější verze, tudíž instalace neproběhne. Safari nezobrazoval starší verze vůbec a Chrome nesprávně. Jediným možným řešením je instalovat všechny verze Silverlight postupně od

nejstarší po nejnovější, aby se každá verze zapsala do prohlížečů. Pak se všechny verze zobrazují bez problémů.

Testování neproběhlo na žádných distribucích Linuxu, jelikož zde Silverlight jako takový vůbec není podporován. Pro OS Linux je vyvíjen systém Moonlight, jehož cílem je zobrazovat a vyvíjet Silverlight aplikace na tomto operačním systému.

## 2 XAML

### Co je to XAML?

Extensible Application Markup Language (XAML) byl představen současně s technologií WPF, na které je založen systém Windows Vista a je na ní stavěn i operační systém Windows Seven.

XAML je deklarativní programovací jazyk, syntakticky vycházející z jazyka XML. Skládá se tedy s jednotlivých elementů, jejichž názvy ("tagy") se zapisují v ostrých závorkách. Tagy se vyskytují buď samostatně - tzv. "self-closing", tedy "samouzavírací" nebo ve dvojicích, kde jeden je počáteční a druhý je uzavírací. Pokud se nějaké elementy vyskytují mezi počátečním a uzavíracím tagem jiného elementu, pak tvrdíme, že tyto elementy jsou "potomci" či "subelementy" onoho nadřazeného elementu, jenž je jejich "rodičem".

Velmi užitečnou vlastností jazyka XAML je fakt, že naprostou většinu parametrů jednotlivých elementů lze použít jako jejich subelementy. Např. chceme-li nastavit barvu pozadí obdélníku, můžeme to udělat takto:

```
<Rectangle Fill="Blue"/>
```

Nebo takto:

```
<Rectangle>
  <Rectangle.Fill>
    <SolidColorBrush Color="Blue"/>
  </Rectangle.Fill>
</Rectangle>
```

Díky tomuto způsobu zadávání parametrů je možné vytvořit mnohem komplexnější styl pozadí (lineární interpolace, obrázek, videoklip, atd.) velmi jednoduše a přehledně. Stejným způsobem lze např. na tlačítko umístit místo základního popisku obrázek nebo jakýkoli jiný objekt, popř. i více objektů najednou. Více o grafickém designu vysvětlím dále.

Jazyk XAML je nedílnou součástí technologie Silverlight, jelikož s jeho pomocí se vytváří grafický design Silverlight aplikací (stejně jako v případě WPF). Jak již bylo zmíněno v předchozí kapitole, technologie Silverlight umožňuje používat pouze podmnožinu všech XAML elementů využívaných ve WPF. V tomto oddílu popíše všechny důležité elementy a jejich parametry a na několika příkladech ukáží, jak je tento jazyk koncipován a jak se používá.



## 2.1 Popis elementů použitelných v Silverlight 2.0

Všechny elementy jazyka XAML mají dva základní typy parametrů, které můžeme rozdělit do dvou hlavních skupin. Jsou to parametry označující grafický vzhled (vlastnosti) tlačítka a parametry určující jeho chování (události). Události jsou ve své podstatě odkazy na metody, které jsou zavolány po určité události. Např. "Je-li stisknuto tlačítko, zavolej metodu Button\_Click".

## 2.2 Kořenový element

### <UserControl>

Element <UserControl> je vždy nejnadřazenějším elementem na stránce a obsahuje všechny ostatní elementy. Zpravidla je vytvořen automaticky hned po vytvoření projektu. Na jedné stránce musí být stejně jako v XML pouze jeden hlavní element. To znamená, že element <UserControl> se může vyskytovat v každém XAML souboru pouze jednou. Důležité také je, že za ukončovacím tagem </UserControl> již nesmí nic následovat.

Element <UserControl> má řadu parametrů shodnou s dalšími elementy (např. výška, šířka, barva pozadí, atd.), ale hlavní parametr, který uvedu, je "xmlns". Díky tomuto parametru můžeme vkládat do XAML souboru reference na nejrůznější namespace .NET Frameworku, zejména pak na namespace obsahující definice elementů používaných v Silverlight.

## 2.3 Layout - rozvržení komponent na stránce

Tato skupina elementů slouží k uspořádání a rozvržení komponent na scéně aplikace. Můžeme pomocí nich umisťovat objekty buď absolutně (souřadnicemi X a Y), nebo relativně (nastavováním vzdáleností mezi jednotlivými objekty, atd.). Do této skupiny patří elementy <Canvas>, <StackPanel> a <Grid>.

### <Canvas>

Tento element je nejjednodušší z tří výše zmíněných. Jde o jakousi pracovní plochu, na kterou je možné umisťovat objekty pomocí souřadnic. Tento postup umisťování je sice velmi jednoduchý, ale zároveň velmi pracný. Element <Canvas> nemá žádné specifické parametry, díky kterým bychom mohli jednoduše měnit uspořádání objektů. Následující kód ukazuje, jakým způsobem umístit tlačítko na stránce za pomoci elementu <Canvas>:

```
<Canvas Height="200" Width="300" Background="White">  
  <Button Width="100" Height="30" Content="Click" Canvas.Left="30"  
    Canvas.Top="50" />  
</Canvas>
```

Tlačítko bude vzdáleno 30px od levého a 50px od horního okraje scény. Výsledek bude vypadat takto:



### <StackPanel>

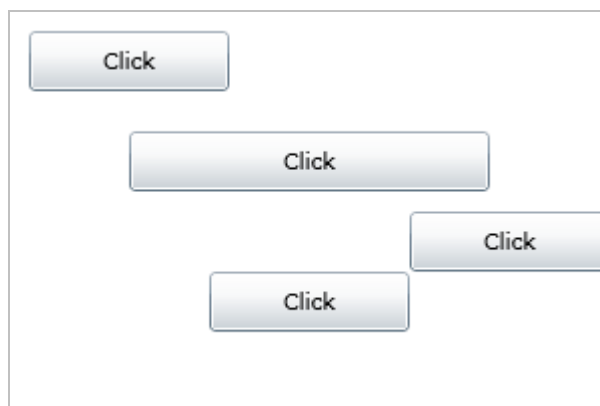
Jak je vidět, použití elementu <Canvas> je velmi neefektivní, protože každému objektu, který je elementu přidružený, musíme zadávat zvlášť souřadnice. Tuto nevýhodu řeší element <StackPanel>. Jeho úkolem je řazení objektů za sebou vodorovně nebo svisle. Díky této možnosti můžeme velmi jednoduše zobrazovat například položky menu nebo seznam obrázků, aniž bychom museli každému jednotlivému objektu zadávat přesnou pozici. Stačí pouze u jednotlivých objektů použít parametr "Margin", což je údaj v pixelech, říkájící, jak daleko od objektu může být jiný objekt. Jak je uvedeno výše,

<StackPanel> umožňuje řadit objekty vodorovně nebo svisle. O volbu směru se stará parametr "Orientation", který může nabývat hodnoty "Horizontal" nebo "Vertical". Pakliže zvolíme u elementu <StackPanel> orientaci vertikálně, můžeme u jednotlivých objektů zvolit čtyři možná zarovnání pomocí parametru "HorizontalAlignment" - Left, Center, Right a Stretch (= Roztáhnout). Použijeme-li pro StackPanel orientaci horizontální, můžeme u jednotlivých objektů nastavit zarovnání analogicky pomocí parametru "VerticalAlignment", který může nabývat hodnot Top, Center, Bottom a opět Stretch.

V následující ukázce kódu jsou nastaveny u objektů různé hodnoty parametru "Margin" a "HorizontalAlignment":

```
<StackPanel Background="White" Orientation="Vertical">  
  <Button Width="100" Height="30" Content="Click"  
    Margin="10" HorizontalAlignment="Left" />  
  <Button Height="30" Content="Click" Margin="60,10"  
    HorizontalAlignment="Stretch" />  
  <Button Width="100" Height="30" Content="Click"  
    Margin="0" HorizontalAlignment="Right" />  
  <Button Width="100" Height="30" Content="Click"  
    Margin="0" />  
</StackPanel>
```

Výsledek vypadá takto:



Jak je vidět, parametr "Margin" můžeme použít několika způsoby. Pokud použijeme jednu číselnou hodnotu, bude nastaven příslušný odstup na všech stranách. Pokud ale použijeme dvě číselné hodnoty oddělené čárkou, pak první hodnota vyjadřuje vodorovný a druhá hodnota svislý odstup. Pokud by nám ani

toto nestačilo, můžeme zadat i čtyři číselné hodnoty oddělené čárkami, kterými zadáme odstup z každé strany zvlášť. Pokud u některého z objektů vynecháme parametr "HorizontalAlignment", pak bude automaticky použita hodnota "Stretch". Pokud má ovšem objekt zároveň nastavenou šířku, má tento parametr před roztažným přednost a objekt se zarovná na střed, tedy bude použita hodnota "Center".

### <Grid>

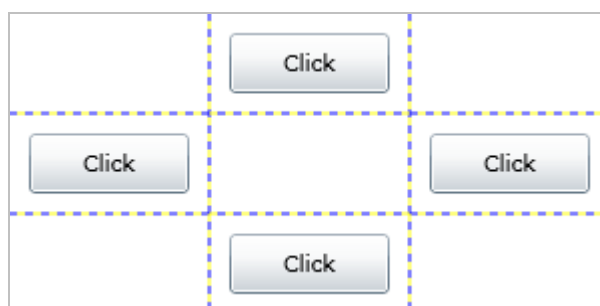
Asi nejmocnějším nástrojem pro rozmisťování objektů na scéně je element <Grid>. Tento element totiž umožňuje rozdělovat scénu na řádky a sloupce a vytvářet tak síť buněk, do kterých je možné umisťovat jednotlivé objekty. Použití elementu <Grid> je již relativně složité. Proto tentokrát zvolím opačný postup. Nejdříve ukáži příklad kódu a teprve poté na něm vysvětlím jednotlivé části a nastavení. Zde je tedy ukázka kódu:

```
<Grid Background="White" ShowGridLines="True">
  <Grid.RowDefinitions>
    <RowDefinition Height="50"/>
    <RowDefinition Height="50"/>
    <RowDefinition Height="50"/>
  </Grid.RowDefinitions>

  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="100"/>
    <ColumnDefinition Width="100"/>
    <ColumnDefinition Width="100"/>
  </Grid.ColumnDefinitions>

  <Button Width="80" Height="30" Content="Click" Grid.Column="0" Grid.Row="1" />
  <Button Width="80" Height="30" Content="Click" Grid.Column="1" Grid.Row="0" />
  <Button Width="80" Height="30" Content="Click" Grid.Column="1" Grid.Row="2" />
  <Button Width="80" Height="30" Content="Click" Grid.Column="2" Grid.Row="1" />
</Grid>
```

Takto bude vypadat výsledek:



Jak je vidět na obrázku, scéna je rozdělena na tři řádky a tři sloupce. Toto rozvržení je třeba prvně nadefinovat. Právě k tomu slouží dva subelementy elementu Grid - "Grid.ColumnDefinitions" a "Grid.RowDefinitions". Z kódu je jistě naprosto zřejmé, jakým způsobem se deklarují jednotlivé řádky a sloupce. Každý nadeklarovaný řádek a sloupec automaticky dostane přidělen index (začínající nulou), díky kterému je pak možno se na něj odkázat. To je vidět u jednotlivých tlačítek, která mají dva parametry - "Grid.Column" a "Grid.Row". Jejich číselná hodnota vyjadřuje, v jakém sloupci a řádku se jednotlivá tlačítka nacházejí.

Použití elementu <Grid> je velmi praktické, ale pro rozmístování jednotlivých objektů, jako jsou tlačítka nebo textová pole na scéně, se příliš nehodí. Nejčastěji se používá pro rozdělení scény na několik částí, v nichž se pak rozmisťují jednotlivé komponenty pomocí elementů <StackPanel> a <Canvas>.

## 2.4 Základní komponenty

V předchozích několika odstavcích jsem uvedl některé základní komponenty jako tlačítko nebo textové pole. V následujícím oddíle popíši nejdůležitější z nich a ukáži způsob jejich použití.

### <Button>

Tlačítko je snad nejzákladnější interaktivní prvek, který je možno použít. Mezi jeho klíčové vlastnosti patří "Content", určující popis tlačítka, či např. "Cursor", udávající, na co se změní kurzor po najetí nad tlačítko. Asi nejčastěji používanou tzv. událostí je "Click", jejíž hodnota udává název metody, která je vykonána po stisknutí tlačítka. Následují dvě ukázky kódu, na kterých je ukázáno hlavně použití subelementu <Button.Content>, díky čemuž je možno radikálně měnit vzhled tlačítka.

```
<Button Width="80" Height="30" Content="Click" Cursor="Hand"
        IsEnabled="True" Click="Button_Click" />
```

Takto vypadá základní tag. Je zde nastavena šířka, výška, popis tlačítka "Click", změna kurzoru na "Hand", tlačítko je aktivní a po jeho kliknutí se zavolá metoda "Button\_Click".



```
<Button Width="80" Height="30" Cursor="Hand" IsEnabled="True"
        Click="Button_Click" >
  <Button.Content>
    <StackPanel Orientation="Horizontal">
      <Ellipse Fill="Red" Height="10" Width="10"/>
      <Ellipse Fill="Orange" Height="10" Width="10" Margin="3,0"/>
      <Ellipse Fill="Green" Height="10" Width="10"/>
      <TextBlock Text="GO!" Margin="3,0,0,0" Foreground="Green" />
    </StackPanel>
  </Button.Content>
</Button>
```

Zde je použit subelement <Button.Content> jako náhrada za parametr "Content". Jak je vidět na obrázku napravo, použití



subelementu místo parametru umožnilo přidání dalších objektů do popisku tlačítka, v tomto případě jsou to tři kolečka v barvách semaforu (více o elementu <Ellipse> naleznete dále). Důležité je použití elementu <StackPanel>, jelikož XAML umožňuje vložit za parametr "Content" jen jeden

element. Pokud bychom napsali cokoli za uzavírací tag elementu <StackPanel>, debugger nahlásí syntaktickou chybu.

### <TextBlock>

Element <TextBlock> vyjadřuje statické nepřepisovatelné textové pole. S jeho pomocí se dají vytvářet nadpisy, popisky objektů, atd. Nastavením jeho mnoha různých parametrů můžeme formátovat zobrazovaný text, určovat výšku řádků v odstavci nebo zarovnávat text běžnými způsoby. Následující příklad ukazuje základní použití a nastavení všech důležitých parametrů.

```
<TextBlock Text="Hello World!" Foreground="Black" FontFamily="Arial"
          FontSize="20" FontWeight="Bold" />
```

Výsledek tohoto zápisu vidíme na obrázku napravo. Jelikož není přímo nastavena velikost textového pole, zvětšuje se automaticky v závislosti na textu. K této ilustraci byla nastavena



**Hello World!**

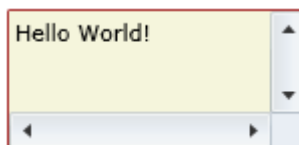
pozice textového pole na scéně pomocí souřadnic, ale z důvodu zachování přehlednosti není toto nastavení vidět v ukázce kódu.

### <TextBox>

Oproti statickému elementu <TextBlock> je tento element přímo určen k tomu, aby do něj uživatel byl schopen zadat text. Nejedná se tedy z vizuálního hlediska pouze o text samotný, ale je to rámeček, kterému je možné nastavit pozadí, barvu a šířku okraje, atd. Zároveň má parametry jako např. "SelectedText" nebo "VerticalScrollBarVisibility", které uchovávají užitečné informace o textu v textovém poli a grafické stránce komponenty. Samozřejmě nechybí ani možnost formátovat text a zarovnávat odstavce, jako tomu bylo u předchozího elementu.

```
<TextBox Height="70" Width="150" Background="Beige" BorderBrush="Brown"
VerticalScrollBarVisibility="Visible" HorizontalScrollBarVisibility="Visible"
Text="Hello World!" />
```

Na tomto příkladě je vidět několik základních nastavení, která `<TextBox>` umožňuje. Výsledek tohoto zápisu je zde:

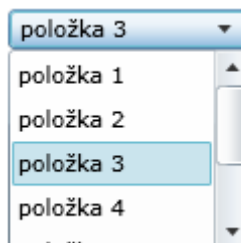


### `<ComboBox>`

Dalším velmi často používaným prvkem je `<ComboBox>`, který slouží především k výběru jedné z více položek. Element `<ComboBox>` obsahuje opět řadu parametrů ovlivňujících formát zobrazovaných položek. Mezi ně patří zejména formát textu, barva pozadí a okraje, atd. Kromě těchto všeobecných parametrů můžeme nastavovat také např. délku zobrazovaného seznamu, aktuálně označenou položku nebo můžeme sledovat, zda je tzv. roztažen seznam položek. Jelikož samotný jazyk XAML slouží hlavně pro grafický návrh scény, neumí přímo pracovat se seznamy. O mnoho jednodušší je tedy přiřadit do prvku `<ComboBox>` seznam položek právě kódově. To je i případ následující ukázky:

```
<ComboBox Width="120" MaxDropDownHeight="100" />
```

Jak je vidět, jazyk XAML je zde použit pouze pro nastavení vzhledu prvku. Po přidání seznamu může komponenta vypadat následovně (postup pro přidání seznamu zde není vysvětlen, neboť tato problematika zasahuje až do třetí kapitoly):





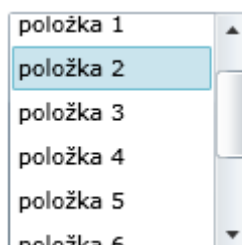
Na obrázku je komponenta ve stavu "roztažena". To znamená, že došlo ke stisku horního tlačítka a zobrazil se seznam položek. Výšku tohoto zobrazení určuje v ukázce použitý parametr "MaxDropDownHeight" udávaný v pixelech. Element `<ComboBox>`, stejně jako další prvky zobrazující seznamy, jsou většinou používány ve spojení se šablonami. Tento způsob utváření grafického vzhledu je vysvětlen dále.

### **<ListBox>**

Poslední ze základních komponent, které zde popíši, je prvek `<ListBox>`. Jak již jeho pojmenování naznačuje, jde o prvek zobrazující nějaký seznam. Většina jeho parametrů je naprosto stejná jako u elementu `<ComboBox>`, popsaném výše. Kromě základních formátovacích parametrů jsou tu navíc jen parametry zarovnání položek svisle a vodorovně a dále parametry určující viditelnost svislého a vodorovného posuvníku.

```
<ListBox Width="120" Height="120" />
```

Na ukázce kódu je opět vidět pouze nastavení velikosti komponenty. Po přidání seznamu položek bude `<ListBox>` vypadat takto:



Vertikální posuvník se automaticky zviditelnil, jelikož délka seznamu překročila zadanou výšku.

## 2.5 Grafický design

Součástí jazyka XAML jsou kromě základních komponent také jednoduché grafické prvky, díky kterým je možné vytvářet grafický design scény. Mezi tyto prvky patří úsečka, elipsa, obdélník, okraj a cesta. Tyto komponenty jsou čistě grafické, ale to neznamená, že je nemůžeme použít i jako interaktivní prvky. Ve skutečnosti slouží právě k tomu, že jimi pomocí šablon zcela přetváříme vzhled základních komponent. Tímto postupem se můžeme zcela oprostit od klasického vzhledu tlačítek, textových polí a ostatních komponent. Tomuto tématu je věnován následující oddíl, na následujících několika stranách si ukážeme základní použití zmíněných grafických prvků na scéně.

### <Line>

Úplně nejzákladnějším grafickým prvkem je úsečka. Jde v podstatě o spojnicí dvou bodů o zadaných souřadnicích. Její vzhled můžeme modifikovat pomocí několika parametrů jako např. šířka štětce, styl čáry (plná nebo přerušovaná), samozřejmě barva nebo tvar zakončení. Nejdůležitější dvojicí parametrů jsou parametry "X1", "Y1", "X2" a "Y2", které udávají souřadnice počátečního a koncového bodu úsečky v pixelech.

```
<Line X1="5" Y1="5" X2="100" Y2="75" StrokeDashArray="2"
      StrokeThickness="5" Stroke="Red" />
```

Tento příklad použití definuje červenou přerušovanou úsečku vedoucí z bodu [5,5] do bodu [100,75]. Výsledek vidíme vpravo na obrázku.



### <Rectangle>, <Ellipse>

Dalšími základními tvary sloužícími pro tvorbu grafického designu scény jsou elementy <Rectangle> a <Ellipse>. Element <Rectangle> představuje obdélník, u kterého můžeme nastavovat barvu nebo styl výplně, šířku a barvu okrajů nebo např. zaoblení rohů. Klasický zápis kódu vypadá takto:

```
<Rectangle Fill="Gray" Stroke="Black" StrokeThickness="2" Height="50" Width="100"
  RadiusX="5" RadiusY="5" />
```

Výsledkem tohoto zápisu je šedivý obdélník o velikosti 100 x 50 pixelů s černým okrajem a zaoblením rohů s poloměrem 5 pixelů - viz. obrázek vpravo.



Element <Ellipse> má vesměs stejné parametry jako element <Rectangle>, s tím rozdílem, že po zadání rozměrů a dalších parametrů se vykreslí elipsa dané velikosti. Následuje opět ukázka zápisu a výsledné zobrazení.

```
<Ellipse Width="100" Height="50" Fill="Gray" Stroke="Black" StrokeThickness="2" />
```



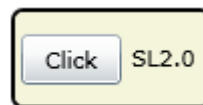
### <Border>

Element <Border> je obdobou elementu <Rectangle>. Jediným rozdílem oproti tomuto elementu je několik parametrů sloužících ke správě elementů obsažených uvnitř elementu <Border>. Tento element je totiž primárně určen k tomu, aby byl do něj vkládán nějaký obsah. Jak již jeho pojmenování napovídá, jde tedy o jakési ohraničení, které může obsahovat další prvky na scéně. Mezi zmíněné parametry spravující obsah elementu <Border> patří např. parametr "Child", ve kterém je uchováván odkaz na jeho "potomka". Je důležité zmínit, že element <Border> může obsahovat pouze jeden subelement (s výjimkou svých vlastních parametrů použitých jako subelementy). Chceme-li do elementu <Border> vložit více prvků, pak je třeba je ještě uzavřít do některého z elementů zajišťujících rozvržení objektů na scéně, jako je např. <StackPanel>.

Na následujícím příkladě je ukázáno použití elementu <Border> společně s elementem <StackPanel>, do něhož je uzavřeno tlačítko a popisek.

```
<Border Width="100" Height="50" Background="Beige" BorderBrush="Black"
  BorderThickness="2" CornerRadius="5">
  <StackPanel Orientation="Horizontal" Margin="3,0">
    <Button Content="Click" Height="25" Width="50" />
    <TextBlock Text="SL2.0" VerticalAlignment="Center" Margin="5,0" />
  </StackPanel>
</Border>
```

Výsledné zobrazení vypadá takto:



### <Path>

Posledním a nejsložitějším grafickým prvkem je "cesta". Pomocí tohoto prvku můžeme kreslit na scénu složitější geometrické tvary. Existují dva způsoby, jak popsat složitější geometrické obrazce. Jedním z nich je velmi jednoduchý, avšak dosti nepřehledný, programovací jazyk, který se vepisuje přímo jako parametr "Data" elementu <Path>. Jeho syntaxe je velmi dobře zdokumentována na webu<sup>1)</sup>, nebudu ho zde proto příliš detailně popisovat. Uvedu jen příklad tvaru, na kterém popíši jednotlivé zákonitosti.

```
<Path Stroke="Black" Fill="Green" Data="M 10,100 C 10,300 200,-200 200,100 Z" />
```

Tato ukázka definuje tvar přímo pomocí parametru "Data". Písmeno "M" značí počáteční bod křivky. Křivka tedy začíná na souřadnicích [10,100]. Následuje příkaz "C", který určuje, že z počátečního bodu povede Bezierova křivka. Za tímto příkazem se nachází tři dvojice souřadnic. První dvě dvojice popisují kontrolní body Bezierovy křivky a poslední dvojice určuje koncový bod. Poslední příkaz je písmeno "Z", značící, že celá křivka je uzavřená, tedy že z koncového bodu Bezierovy křivky vede spojnice k počátečnímu bodu. Kromě této definice tvaru obsahuje element <Path> ještě parametry "Stroke" - barva čáry a "Fill" - barva výplně.



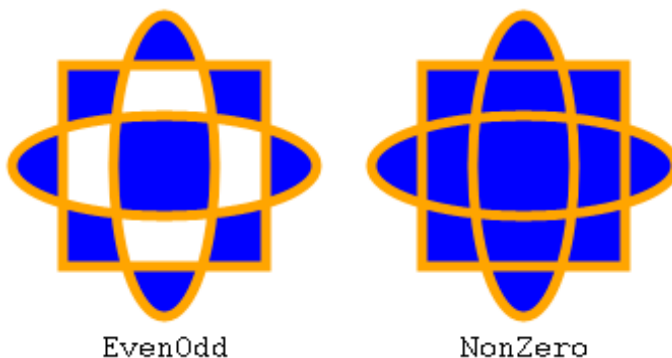
Druhou možností, jak definovat složitější tvary pomocí elementu <Path>, je vnoření několika jednoduchých tvarů do subelementu <Path.Data>. Takto je možné seskupovat více tvarů na scéně a přistupovat k nim jako k jednomu objektu.

```

<Path Margin="10" Fill="Blue" Stroke="Orange" StrokeThickness="5">
  <Path.Data>
    <GeometryGroup FillRule="EvenOdd">
      <RectangleGeometry Rect="0,0 100,100" />
      <EllipseGeometry Center="50,50" RadiusX="75" RadiusY="25" />
      <EllipseGeometry Center="50,50" RadiusX="25" RadiusY="75" />
    </GeometryGroup>
  </Path.Data>
</Path>

```

Na tomto příkladě je ukázáno vytváření složitějších tvarů pomocí subelementu <GeometryGroup>, který je vložen do subelementu <Path.Data>. <GeometryGroup> slouží jako skupina geometrických obrazců, které se posléze tváří jako jeden objekt na scéně. V našem případě jsme spojili jeden obdélník (<RectangleGeometry>) a dvě elipsy (<EllipseGeometry>). Element <GeometryGroup> má specifický velmi užitečný parametr "FillRule", který udává způsob vyplňování dílčích tvarů. Pokud je hodnota tohoto parametru nastavena na "EvenOdd", pak se spolu sousedící plochy vyplňují střídavě. Je-li hodnota nastavena na "NonZero", pak je vyplněna celá plocha výsledného obrazce stejnou barvou nastavenou v parametru "Fill" elementu <Path>. Výsledný tvar odpovídající příkladu vypadá následovně. Na obrázku je ukázán rozdíl v hodnotě parametru "FillRule".



## 2.6 Styly, Šablony

Nyní je třeba vysvětlit rozdíl mezi dvěma základními XAML soubory v projektu. Jsou to soubory "page.xaml" a "app.xaml". Vše, co jsem zatím popisoval z jazyka XAML se týkalo prvního z obou jmenovaných. Soubor "page.xaml" je určen pro správu všech prvků na scéně. Do něj tedy vepisujeme všechno, co na scéně má být zobrazeno. Naopak soubor "app.xaml" je určen kromě jiného hlavně ke správě pozadí projektu. Tímto pozadím jsou míněny právě definice stylů a šablon, které určují vzhled jednotlivých komponent, můžeme sem zařadit i předdefinované animace, vůbec všechna nastavení týkající se celé Silverlight aplikace. Tomuto souboru definic se také říká "statické zdroje". Ze všech těchto globálních nastavení jsou nejčastěji používané právě styly, šablony a animace. V této části je popsáno, jak vytvářet styly a šablony komponent.

### <Style>

Element <Style> je velmi mocným prvkem sloužícím k tvorbě vizuální podoby komponent. Jeho použití je velmi podobné technologii CSS, ve které jsou definice barev, formátu písem a další parametry objektů na stránce uchovávány v samostatném souboru nebo v oddělené části zdrojového kódu a objekty na stránce se na tyto definice odkazují pomocí zvoleného označení. Tento způsob umožňuje velmi jednoduché a přehledné psaní zdrojového kódu, protože autor nemusí každému dalšímu prvku na stránce definovat vzhled od základu.

V následujícím příkladě použijeme element <Rectangle>, konkrétně to bude šedivý obdélník ukázaný v příkladu výše.

```
<Rectangle Fill="Gray" Stroke="Black" StrokeThickness="2" Height="50"
          Width="100" RadiusX="5" RadiusY="5" />
```

Z ukázky kódu je jasně patrné, že pro zobrazení tohoto obdélníku je potřeba spousta parametrů a bylo by velmi pracné zapisovat je všechny znova, pokud bychom chtěli vytvořit další



obdélník o stejných rozměrech, jen např. s jinou barvou výplně. Abychom se tomuto pracnému zapisování vyhnuli, použijeme element <Style> a vytvoříme si vlastní styl. Zápis tohoto stylu bude vypadat takto (připomínám, že element <Style> bude v souboru "app.xaml"):

```
<Style x:Key="rec" TargetType="Rectangle">
  <Setter Property="Height" Value="50" />
  <Setter Property="Width" Value="100" />
  <Setter Property="Stroke" Value="Black" />
  <Setter Property="StrokeThickness" Value="2" />
  <Setter Property="RadiusX" Value="5" />
  <Setter Property="RadiusY" Value="5" />
</Style>
```

Element <Style> má dva velmi důležité parametry. První z nich je "x:Key", jehož hodnota udává pojmenování stylu pro pozdější použití u komponenty na scéně. Hodnota druhého parametru, "TargetType", říká, jakému typu objektu je styl určen. V našem případě je tedy tato hodnota rovna "Rectangle". V těle elementu <Style> se nachází několik subelementů <Setter>. Tento element slouží k uchování definic jednotlivých parametrů objektu. Mezi jeho parametry musí vždy patřit dvojice parametrů "Property" a "Value", z nichž "Property" označuje název parametru cílového objektu a "Value" uchovává hodnotu tohoto parametru.

Poté, co jsme vytvořili styl s názvem "rec", stačí místo dlouhého výpisu parametrů v elementu <Rectangle> použít parametr "Style" následujícím způsobem:

```
<Rectangle Style="{StaticResource rec}" Fill="Red" />
```

Jak je vidět na zdrojovém kódu, jako hodnotu parametru "Style" nestačí zapsat pouze název vytvořeného stylu. Je třeba použít klíčové slovo "StaticResource" spolu s názvem stylu a to vše ve složených závorkách. Debugger pak z tohoto zápisu při překladu přečte informaci, že existuje styl s názvem "rec" umístěný v statických zdrojích aplikace - tedy v souboru "app.xaml". Do elementu <Rectangle> ještě doplníme parametr "Fill" a

nastavíme barvu pozadí na červenou a výsledkem bude obdélník totožný s původním, jen se bude lišit barva pozadí (viz. obrázek vpravo).



### <Template>

Velmi užitečnou a hojně využívanou technikou je tvorba šablon, které nahrazují klasický vzhled základních komponent. Vytvářením šablon si můžeme velmi ulehčit práci, protože relativně jednoduchým způsobem získáme např. tlačítko, které se bude chovat naprosto adekvátním způsobem a bude mít všechny klasické parametry tlačítka, ovšem bude vypadat tak, jak to my nadefinujeme.

Následující příklad ukazuje vytvoření šablony pro tlačítko. Místo klasického obdélníkového vzhledu jej bude představovat kruh vyplněný radiálním gradientem přecházejícím z bílé do červené barvy.

```
<Style x:Key="butTemp" TargetType="Button">
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate>

        <Grid>
          <Ellipse Height="{TemplateBinding Height}"
                  Width="{TemplateBinding Width}" Stroke="Black"
                  StrokeThickness="1">
            <Ellipse.Fill>
              <RadialGradientBrush GradientOrigin="0.3,0.3">
                <GradientStop Color="White" Offset="0"/>
                <GradientStop Color="Red" Offset="1"/>
              </RadialGradientBrush>
            </Ellipse.Fill>
          </Ellipse>

          <ContentPresenter Content="{TemplateBinding Content}"
                            HorizontalAlignment="Center"
                            VerticalAlignment="Center" />
        </Grid>

      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>
```

Jak vidíme, šablona se vytváří stejně jako jakýkoli styl. V elementu <Setter> stačí zadat za parametr "Property" hodnotu "Template" a parametr "Value" rozepsat jako jeho subelement. Element <ControlTemplate>, jenž se nachází uvnitř elementu <Setter.Value> určuje, že půjde o jakousi šablonu



komponenty. Cokoli, co se pak nachází uvnitř tohoto elementu, udává grafickou podobu tlačítka. V tomto příkladě je pro grafickou podobu tlačítka použit element <Ellipse> spolu s elementem <ContentPresenter>. Jelikož element <ControlTemplate> může obsahovat pouze jeden subelement, jsou tyto dva prvky uzavřeny do elementu <Grid>. Velmi důležité jsou v tomto příkladě parametry "Height" a "Width" elementu <Ellipse> a parametr "Content" elementu <ContentPresenter>. Hodnoty těchto parametrů jsou složeny ze dvou částí, z nichž první je klíčové slovo "TemplateBinding" a druhou část tvoří název daného parametru. Tento způsob zápisu se používá tehdy, kdy chceme, aby hodnota parametru v šabloně byla převzata z hodnoty parametru přímo zadaného v souboru "page.xaml".

Po vytvoření šablony je již třeba pouze vytvořit na scéně objekt, který bude na šablonu odkazovat. Zápis v souboru "page.xaml" bude vypadat takto:

```
<Button Content="Click" Height="50" Width="50" Style="{StaticResource butTemp}" />
```

Takto popsané tlačítko bude vypadat jako na obrázku vpravo. Hodnoty šířky, výšky a obsahu tlačítka se načetly do šablony a vznikl kruh o rozměrech 50x50 pixelů a popisek "Click" v jejím středu. Právě popisek na tlačítku je zajišťován prvkem <ContentPresenter>. Pokud bychom chtěli místo pouhého popisku více objektů uvnitř tlačítka, budeme postupovat stejným způsobem, jako je vysvětleno výše v části "Základní komponenty" v oddílu <Button>.



Pomocí šablon můžeme docílit obrovských změn v grafickém vzhledu scény. Nejen, že můžeme přepisovat vzhled základních komponentů, ale je možné například vytvořit i šablonu pro element <ListBoxItem>, který v komponentě <ListBox> vyjadřuje jednu položku, a touto šablonou docílit komplexního zobrazení včetně obrázků, různých grafických prvků, atd.

## 2.7 Animace

V Silverlight existuje několik způsobů animace. Jedná se o animaci barevnou, tvarovou, bodovou, nebo o animaci složitější - po klíčových snímcích. Hned na úvod je velmi důležité říci, jakým způsobem jsou animace v technologii Silverlight prováděny. Ve světě počítačové grafiky existují dva základní typy animací. Jsou to tzv. "frame-by-frame" animace a animace "stavové". Příkladem pro frame-by-frame animace může být technologie Adobe Flash, která je na tomto způsobu animace postavena. Výhodou tohoto principu je jednoduchost vývoje, jelikož v podstatě stačí vytvořit několik snímků na časové ose a do každého vložit nějakou grafiku. Po spuštění animace jsou pak tyto snímky zobrazovány postupně jeden po druhém. Technologie Silverlight je postavena na animaci "stavové". Tato technika animace funguje tak, že definujeme počáteční a konečný stav a časový rozdíl mezi nimi, např. "Změna barvy z červené na zelenou za 5 sekund". Silverlight pak nemusí střídat jednotlivé klíčové snímky, což pro vývojáře aplikace znamená, že nemusí tyto snímky vytvářet, stačí pouze definovat objekty a jejich chování v čase. Následující část detailně popisuje jednotlivé techniky animování objektů na scéně pomocí elementu <Storyboard>.

### <Storyboard>

Element <Storyboard> je hlavním prostředkem pro vytváření animací v Silverlight. Způsob jeho použití záleží na situaci a rozhodnutí programátora. Můžeme jej připojit ke každému objektu na scéně zvlášť nebo můžeme vytvořit jeden typ animace společný pro více objektů. Také je možné jej zapisovat jak do souboru "page.xaml", tak do "app.xaml". V následujících příkladech bude použit element <Storyboard> s různými typy objektů a bude vysvětleno a ukázáno, jaké vlastnosti objektů je možné animovat. Více informací o animaci naleznete v té části této publikace, ve které je popisována praktická část, kde je na konkrétních příkladech vysvětlen princip jednotlivých animačních technik.

V Silverlight je třeba v každé vytvářené animaci předem definovat, co se bude animovat. Toto je zajišťováno pomocí speciálních elementů vkládaných do elementu <Storyboard>. Mezi tyto elementy patří <ColorAnimation>, <DoubleAnimation> a <PointAnimation>. Každý z těchto elementů se stará o nějaký způsob animace objektu a každý můžeme popsat i pomocí klíčových snímků (viz. dále).

### <ColorAnimation>

Element <ColorAnimation> slouží k animování všech parametrů, které mají něco společného s barvou. Typickým příkladem je výplň nebo barva okraje grafických prvků. Následující příklad ukazuje, jak docílit plynulé změny barvy výplně obdélníku po najetí kurzoru.

```
<Rectangle Style="{StaticResource rec}" MouseEnter="Rectangle_MouseEnter"
    MouseLeave="Rectangle_MouseLeave">
  <Rectangle.Fill>
    <SolidColorBrush x:Name="fill" Color="Red" />
  </Rectangle.Fill>
  <Rectangle.Resources>
    <Storyboard x:Name="colorTransform">
      <ColorAnimation BeginTime="00:00:00" Storyboard.TargetName="fill"
        Storyboard.TargetProperty="Color" Duration="00:00:01"
        From="Red" To="Yellow" RepeatBehavior="Forever"
        AutoReverse="True" />
    </Storyboard>
  </Rectangle.Resources>
</Rectangle>
```

V této ukázce je použit obdélník z předchozích příkladů. Je na něm použit styl s názvem "rec". Velmi důležité je, že parametr "Fill" je rozepsán jako subelement a jako výplň je použit element <SolidColorBrush>, který přiřazuje obdélníku jednodílnou červenou barvu. Tento způsob zápisu je důležitý, neboť abychom mohli animovat výplň obdélníku, musíme tuto výplň jednoznačně identifikovat. V našem případě je toto zaručeno tím, že element <SolidColorBrush> uvnitř elementu <Rectangle.Fill> má definován parametr "x.Name" s hodnotou "fill". Tato hodnota reprezentuje jméno výplně a na toto jméno se posléze odkazuje právě element <ColorAnimation>.

Tedy tedy k samotné definici animace. Jak je již uvedeno výše, element <ColorAnimation> je vložen do elementu <Storyboard>, který je definován

jako zdroj obdélníku, tedy <Rectangle.Resources>. Element <ColorAnimation> má mnoho parametrů, na kterých je animace přímo závislá. Mezi ně patří např. dvojice "From" a "To", které udávají původní a konečnou barvu, "BeginTime" a "Duration", které obsahují informace o okamžiku spuštění a době trvání animace a v neposlední řadě právě parametry odkazující na prvek, který se má animovat. Těmi jsou "Storyboard.TargetName" (jméno prvku, který bude animován) a "Storyboard.TargetProperty" (parametr animovaného prvku, jehož hodnota se bude měnit). V příkladech jsou použity také parametry "RepeatBehavior", který udává, zda se bude animace neustále opakovat, a "AutoReverse", jehož hodnota "True" říká, že po změně z počáteční barvy na koncovou se má barva opět změnit na počáteční.

Nakonec je potřeba vysvětlit způsob spuštění animace. V Silverlight se ke spuštění a manipulaci s animacemi používají nejčastěji handlers elementů. Chceme-li například, aby se animace spustila ihned po načtení scény, použijeme u animovaného elementu handler "Loaded". V našem příkladě jsou použity handlers "MouseEnter" a "MouseLeave". První jmenovaný zajistí spuštění animace, když nad obdélník najede kurzor, druhý jmenovaný animaci zastaví, když kurzor z obdélníku vyjede. Tím, jak je napojen C# kód na handlers v kódu XAML, se více zabývá třetí kapitola. Prozatím pro pochopení stačí, když si řekneme, že metody, která zajišťují spuštění a zastavení animace vypadají následovně:

```
private void Rectangle_MouseEnter(object sender, MouseEventArgs e)
{
    colorTransform.Begin();
}

private void Rectangle_MouseLeave(object sender, MouseEventArgs e)
{
    colorTransform.Stop();
}
```

Všimněme si názvů metod - "Rectangle\_MouseEnter" a "Rectangle\_MouseLeave". Tyto názvy jsou uvedeny jako hodnoty handlerů v XAML kódu. V těchto metodách je použit název našeho elementu <Storyboard>, tedy "colorTransform". Díky tomuto jménu se můžeme na

element odkazovat a využívat jeho metod. V našem případě to jsou metody Begin() a Stop(). Tyto metody, jak již jejich název napovídá, obstarávají spuštění a zastavení animace. Kromě těchto dvou metod umožňuje Silverlight ještě animaci pozastavit metodou Pause() a v animaci pokračovat metodou Resume(). V poslední kapitole je na rozsáhlejších příkladech názorně ukázáno jak všechny čtyři metody fungují a co přesně se s animací po jejich zavolání stane. Tyto metody jsou přímo spjaty s elementem <Storyboard>. Nezáleží tedy jaký druh animace vytváříme, názvy a použití metod je pro všechny stejné.

### <DoubleAnimation>

Chceme-li animovat prvek změnou parametru, jehož hodnota je typu "Double", tedy reálné číslo, můžeme použít element <DoubleAnimation>. S jeho pomocí můžeme měnit například rozměry objektů, jejich průhlednost nebo pozici na scéně. Jelikož je jeho použití velmi podobné použití předchozímu, uvedu zde pouze příklad, na kterém popíši jednotlivé odlišnosti.

```
<Rectangle x:Name="rect" Style="{StaticResource rec}" Fill="Red"
  MouseEnter="Rectangle_MouseEnter" MouseLeave="Rectangle_MouseLeave">
  <Rectangle.Resources>
    <Storyboard x:Name="xTransform">
      <DoubleAnimation BeginTime="00:00:00" Storyboard.TargetName="rect"
        Storyboard.TargetProperty="Width" Duration="00:00:01"
        From="100" To="200" RepeatBehavior="Forever"
        AutoReverse="True" />
    </Storyboard>
  </Rectangle.Resources>
</Rectangle>
```

Jak je vidět, tentokrát nebylo třeba rozepisovat parametr "Fill", jelikož je animována šířka obdélníku. Aby bylo možné šířku, tedy parametr "Width", animovat, je třeba opět pojmenovat element, ve kterém se parametr nachází - tedy samotný obdélník (x:Name="rect"). Samotná animace je definována naprosto stejně jako předešlá animace výplně. Opět zde vidíme parametry "From" a "To", přičemž tentokrát jejich hodnoty vyjadřují šířku obdélníku v pixelech. Animace se bude opět opakovat neustále a šířka se bude vracet na původní hodnotu (viz. "RepeatBehavior" a "AutoReverse"). Jediný zásadnější rozdíl je tedy v odkazu na element (Storyboard.TargetName="rect")

a především v odkazu na námi animovaný parametr (Storyboard.TargetProperty="Width").

Spuštění a zastavení animace je zajištěno opět stejným způsobem jako v předešlém příkladě, tedy za pomoci handlerů "MouseEnter" a "MouseLeave". Výsledkem tohoto příkladu bude tedy červený obdélník o rozměrech 100x50 pixelů, který se po najetí kurzorem nad jeho plochu plynule roztáhne na šířku 200 pixelů a zpět, přičemž dokud neopustíme plochu obdélníku, bude se neustále roztahovat a ztahovat, jak je znázorněno na obrázku. Zajímavým faktem také je, že aktivní plocha obdélníku se rozšiřuje s jeho šířkou.



#### <PointAnimation>

Posledním typem animace je animace bodu, tedy parametrů, jejichž hodnota je typu "Point". Tento typ je vždy reprezentován dvěma číselnými hodnotami, vyjadřujícími souřadnice X a Y. Parametrem, jehož hodnotou je "bod", může být například střed elipsy, přesněji elementu <EllipseGeometry> vytvořené v prvku <Path>, nebo obrysový bod křivky. Použití tohoto typu animace je opět naprosto shodné s předchozími dvěma případy, uvedu tedy opět jen příklad.

```
<Path Fill="Red" Stroke="Black" StrokeThickness="2"
      MouseEnter="Rectangle_MouseEnter" MouseLeave="Rectangle_MouseLeave">
  <Path.Resources>
    <Storyboard x:Name="pTransform">
      <PointAnimation BeginTime="00:00:00" Duration="00:00:0.2"
                      Storyboard.TargetName="eli"
                      Storyboard.TargetProperty="Center"
                      From="50,50" To="50,40" RepeatBehavior="Forever"
                      AutoReverse="True" />
    </Storyboard>
  </Path.Resources>
  <Path.Data>
    <GeometryGroup>
      <EllipseGeometry x:Name="eli" Center="50,50" RadiusX="20"
                      RadiusY="20" />
    </GeometryGroup>
  </Path.Data>
</Path>
```

Použití elementu `<EllipseGeometry>` jako reprezentace elipsy je nezbytné, jelikož klasický prvek `<Ellipse>` je definován pomocí výšky a šířky, z čehož vyplývá, že nemá pro nás stěžejní parametr "Center", jehož hodnota je právě "bod". Animace, která je v tomto příkladě definována, má za úkol plynule rozpohybovat elipsu o deset pixelů nahoru a zpět a tento pohyb neustále opakovat. Zásadními parametry elementu `<PointAnimation>` jsou v tomto případě parametry "From" a "To", jejichž hodnoty jsou složeny z výše zmíněné dvojice hodnot vyjadřujících souřadnice X a Y. Můžeme si dále povšimnout i časové hodnoty parametru "Duration", která je nastavena na 0,2 sekundy. Tato hodnota je nastavena úmyslně pro ukázkou toho, že doba animace nemusí být udávána v celých sekundách, ale i v menších časových údajích.

### **Animace po klíčových snímcích**

K vytváření složitějších animací slouží tzv. "klíčové snímky". Tyto snímky umožňují zřetěžit několik animací těsně za sebe. Princip a použití klíčových snímků je založen na tom, že uživatel nadefinuje hodnoty animovaného parametru v několika po sobě jdoucích časech. Silverlight pak jede po pomyslné časové ose a podle zadaných pravidel (lineárně nebo po křivce) přechází z jedné hodnoty na druhou. Animaci po klíčových snímcích je možno definovat pro kterýkoli typ animace. Příslušné tagy se nazývají

`<ColorAnimationUsingKeyFrames>`,  
`<DoubleAnimationUsingKeyFrames>`,  
`<PointAnimationUsingKeyFrames>`.

Pro tyto typy animací lze definovat tři možné způsoby interpolace mezi jednotlivými hodnotami - "Linear", "Spline" a "Discrete". První jmenovaný způsob zajistí lineární interpolaci, druhý jmenovaný interpolaci po křivce a třetí způsob hodnoty neinterpoluje, nýbrž je v daný okamžik zamění. Kromě tří zmiňovaných typů animace po klíčových snímcích existuje ještě jeden typ. Tím je tzv. objektová animace - `<ObjectAnimationUsingKeyFrames>`. Tento typ

animace nepodporuje žádný způsob interpolace mezi hodnotami. Umožňuje pouze okamžité změny objektových parametrů na definovaných klíčových snímcích. Nejedná se tedy o animaci jako takovou, ale díky tomuto prvku můžeme kaskádovitě měnit hodnoty komplexních parametrů. Např. můžeme velmi jednoduchým způsobem měnit pozadí jakéhokoli objektu z jedné barvy přes barevný přechod na obrázek atd. Připomínám, že tyto změny však nikdy nejsou plynulé, nýbrž dochází k záměnám hodnot v definovaných časech.

Následující příklad ukazuje použití animace po klíčových snímcích na plynulé změně výšky obdélníku s použitím tří klíčových snímků.

```
<Rectangle x:Name="rect" Style="{StaticResource rec}" Fill="Red"
  MouseEnter="Rectangle_MouseEnter" MouseLeave="Rectangle_MouseLeave">
  <Rectangle.Resources>
    <Storyboard x:Name="pTransform">
      <DoubleAnimationUsingKeyFrames Duration="00:00:1.2"
        Storyboard.TargetName="rect"
        Storyboard.TargetProperty="Height">
        <SplineDoubleKeyFrame KeySpline="0.5,0 1,0.5" KeyTime="00:00:01"
          Value="200"/>
        <LinearDoubleKeyFrame KeyTime="00:00:1.1" Value="190"/>
        <LinearDoubleKeyFrame KeyTime="00:00:1.2" Value="200"/>
      </DoubleAnimationUsingKeyFrames>
    </Storyboard>
  </Rectangle.Resources>
</Rectangle>
```

Na ukázce kódu je vidět, že v elementu `<DoubleAnimationUsingKeyFrames>` definujeme pouze celkovou dobu trvání animace a odkazy na název objektu a jeho animovaného parametru. Do tohoto elementu jsou pak uzavřeny tři klíčové snímky, z nichž první zajišťuje interpolaci hodnot po křivce a další dva interpolaci lineární. Každému klíčovému snímku je přidělen čas a hodnota, které má animovaný parametr v onen čas dosáhnout. Zde tedy vidíme, že nejprve se obdélník za dobu jedné sekundy roztáhne na výšku 200 pixelů, poté se za desetinu sekundy jeho výška zmenší na 190 px a nakonec se hodnota za stejnou dobu vrátí na 200 px.



Pomocí klíčových snímků lze dosáhnout již velmi propracovaných animací. Mnohem lepší výsledky zaručuje také možnost kombinovat různé typy animací najednou. Stačí pouze vložit do elementu <Storyboard> více animačních prvků a výsledkem bude jejich kombinace. Například následující příklad zajistí nejen roztažení obdélníka do šířky, ale i změnu jeho barvy.

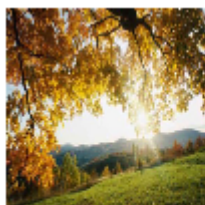
```
<Rectangle x:Name="rect" Style="{StaticResource rec}"
  MouseEnter="Rectangle_MouseEnter" MouseLeave="Rectangle_MouseLeave">
  <Rectangle.Fill>
    <SolidColorBrush x:Name="fill" Color="Red" />
  </Rectangle.Fill>
  <Rectangle.Resources>
    <Storyboard x:Name="xTransform">
      <DoubleAnimation BeginTime="00:00:00" Storyboard.TargetName="rect"
        Storyboard.TargetProperty="Width" Duration="00:00:01"
        From="100" To="200" RepeatBehavior="Forever"
        AutoReverse="True" />
      <ColorAnimation BeginTime="00:00:00" Storyboard.TargetName="fill"
        Storyboard.TargetProperty="Color" Duration="00:00:01"
        From="Red" To="Orange" RepeatBehavior="Forever"
        AutoReverse="True"/>
    </Storyboard>
  </Rectangle.Resources>
</Rectangle>
```

## 2.8 Média

Technologie Silverlight samozřejmě podporuje celou řadu mediálních prvků. Mohou to být jak obrázky, tak zvuk a video. Jsou podporovány všechny klasické obrazové formáty (JPEG, BMP, PNG, GIF, ...), zvuk ve formátu MP3 a WMA a video ve formátu WMV a VC-1. Rovněž umožňuje přehrávat video soubory v HD kvalitě. Tato část se bude věnovat několika mediálním elementům a rovněž způsobům, jak s mediálními soubory nakládat a co vše s nimi dělat.

### <Image>

Element <Image> je prvním ze dvou zde zmiňovaných mediálních elementů a slouží k zobrazení a manipulaci s externě načtenými obrázky. Jeho stěžejními parametry jsou "Source" a "Stretch", z nichž hodnota prvního udává cestu k cílovému souboru. Bližší informace o umístění souborů je možno nalézt v páté kapitole. Druhý parametr může nabývat hodnot "Fill", "None", "Uniform" a "UniformToFill" a tyto hodnoty určují, jakým způsobem bude obrázek zobrazen. Následující obrázek názorně ukazuje rozdíl mezi těmito hodnotami.



Fill



None



Uniform



UniformToFill

### <MediaElement>

S využitím prvku <MediaElement> můžeme načítat do Silverlight aplikace libovolné mediální soubory. Tento element se nejčastěji používá k načítání videa, ale slouží také k načítání zvuku (pokliže není načítán programově). Mezi jeho parametry patří také "Source" a "Stretch", jako v případě elementu

<Image>, ale také několik dalších velmi užitečných parametrů a vlastností, které ovlivňují hlasitost a rozložení zvuku nebo samotné přehrávání mediálního souboru. <MediaElement> má rovněž celou řadu handlerů definovaných pro události spojené s načítáním a přehráváním, které umožňují velmi jednoduché programování tzv. "preloaderů" (indikátorů načítání souboru) a ovládacích prvků.

Obrazové a video soubory lze používat v Silverlight ještě jedním způsobem. Každý element vlastní parametr Fill nebo Background totiž může mít jako výplň přiřazen obrázek či video. K tomu slouží tzv. "Brush-elementy". Již jsem se několikrát zmiňoval o rozepsání parametru "Fill" jako subelementu daného elementu. Takto rozepsaný parametr výplně může obsahovat několik typů subelementů, z nichž dva jsou z mediálního hlediska velmi užitečné. Jsou jimi prvky <ImageBrush> a <VideoBrush>. První jmenovaný umožňuje přiřadit pozadí prvku obrázek, druhý jmenovaný videoklip. Následující příklad ukazuje jak je tohoto docíleno.

```
<MediaElement x:Name="mediaElement" Source="helloWorld.wmv" Opacity="0.0"/>
<Ellipse Width="150" Height="150" Stroke="Black" StrokeThickness="2">
  <Ellipse.Fill>
    <ImageBrush ImageSource="sunset.jpg" Stretch="Fill"/>
  </Ellipse.Fill>
</Ellipse>
<Ellipse Width="200" Height="150" Stroke="Black" StrokeThickness="2">
  <Ellipse.Fill>
    <VideoBrush SourceName="mediaElement" Stretch="Uniform"/>
  </Ellipse.Fill>
</Ellipse>
```

V tomto příkladě je použit jeden <MediaElement> a dvě elipsy. První elipsa má za pozadí obrázek a druhá videoklip. <MediaElement> zde slouží k načtení videoklipu a je na něj odkázána výplň druhé elipsy. Jelikož je to však samostatný objekt, zobrazil by se zároveň s elipsami, což v tomto případě není žádoucí. Proto je jeho parametr "Opacity" (= viditelnost) nastaven na hodnotu "0.0", což jej zneviditelní. Výsledkem tohoto zdrojového kódu je následující zobrazení.



<ImageBrush>



<VideoBrush>

## 2.9 Data

Jakožto .NET technologie Silverlight samozřejmě podporuje také práci s daty. Celá problematika připojení k databázovému serveru je názorně ukázána na posledním projektu praktické části, proto se zde budu věnovat pouze způsobům zobrazení načtených dat.

Nejjednodušší způsob, jak zobrazit data z databáze je použití elementu <DataGrid>. Není to samozřejmě jedinná možnost, především díky šablonám, s jejichž pomocí lze velmi jednoduše nadefinovat vzhled a rozvržení jednoho řádku v komponentě <ListBox> a tím mnohonásobně rozšířit oblast jeho využití.

### <DataGrid>

Komponenta <DataGrid> je přímo určená zobrazování dat, a to prostřednictvím tabulky. Je schopna automaticky generovat sloupce a přirozeně řádky podle formátu a množství načtených dat, formátovat sloupce podle typu dat (text, boolean, datum, ...) a umožňuje uživateli interaktivně měnit jejich pořadí, šířku a seřazovat podle nich data vzestupně či sestupně.

Element <DataGrid> má celou řadu parametrů a vlastností, které více či méně ovlivňují grafické vzezření komponenty. Samozřejmostí jsou rozměrové parametry, barva pozadí a okrajů nebo zobrazení horizontálního a vertikálního posuvníku. Dále je možné nastavit šířku sloupců, výšku řádků nebo formát zobrazovaného písma. Velmi zajímavou možností je také nastavení různých

barev pozadí pro sudé a liché řádky. Pokud by programátorovi tyto parametry nestačily, je možné definovat šablony buď pro řádky nebo i pro jednotlivé buňky. Velmi užitečné může být i využití tzv. "detailů řádku" ("RowDetails"), které se používají v případě, kdy nechceme nutně zobrazovat všechny údaje najednou. Např. máme-li databázi zákazníků a o každém zákazníkovi je v ní uloženo mnoho údajů, můžeme nastavit <DataGrid> tak, že bude zobrazovat pouze jejich jména a teprve po označení příslušného řádku tabulky se onen řádek rozšíří a zobrazí se všechny informace o vybraném zákazníkovi. Kromě parametrů ovlivňujících vzhled komponenty můžeme nastavovat např. způsob označování řádků/buněk, nebo rozhodovat o tom, zda bude uživateli povoleno manipulovat se sloupci a řádky.

Vzhledem k tomu, že předmětem této kapitoly není vysvětlovat princip napojení na databázový server, nebudu podávat bližší vysvětlení k následující ukázkce. Tato ukáзка slouží pouze k předvedení grafického vzhledu prvku <DataGrid> po nastavení zmíněných parametrů.

```
<data:DataGrid x:Name="dataGrid" Loaded="dataGrid_Loaded"
AutoGenerateColumns="True" Width="600" Height="200"
AlternatingRowBackground="Beige" RowBackground="White"
BorderBrush="Brown" />
```

id	text	t
1	Žil jste někdy v jiné zemi po dobu delší než 6 měsíců?	1 ▲
2	Primární starost člověka by se měla týkat rodiny a blízkých přátel, členů širší rodiny nevymáje.	2
3	Kolik evropských zemí jste navštívil od svých 16 let (kromě průjezdů)?	3
4	Které TŘI evropské státy jsou Vám nejbližší?	4
5	Toto je 1. pokusná otázka patřící k Okruhu 5. Líbí se vám?	1
6	Toto je 2. pokusná otázka patřící k Okruhu 5. Jak se Vám líbí?	2
8	tradaaaa	7 ▼

### **<ListBox>**

Jak jsem již uvedl výše, velmi dobrou alternativou elementu <DataGrid> je použití komponenty <ListBox> spolu se šablonami. Tento způsob zobrazování dat je sice o něco složitější než předchozí, jelikož zde nemáme k dispozici

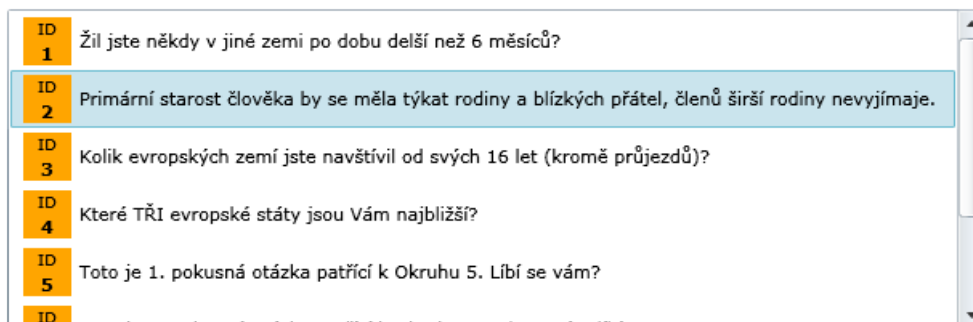
velké množství již definovaných parametrů a musíme si vše definovat sami, ale na druhou stranu nám dává mnohem větší volnost v rozvržení jednotlivých textových polí a jiných prvků, které posléze zobrazují načtená data.

Celý princip je založen na vytvoření šablony pro položky v seznamu. Místo pouhého textového popisku můžeme do těla položky vložit jakékoli grafické prvky a rozvrhnout jejich rozmístění pomocí elementů <Grid> nebo <StackPanel>. Následující příklad ukazuje definici jednoduché šablony pro položky elementu <ListBox>.

```
<ListBox x:Name="dataList" Width="600" Height="200">
  <ListBox.ItemTemplate>
    <DataTemplate>
      <StackPanel Orientation="Horizontal">
        <StackPanel Margin="5,0" VerticalAlignment="Center"
          Orientation="Vertical" Width="30" Height="30"
          Background="Orange">
          <TextBlock Text="ID" HorizontalAlignment="Center"
            FontSize="10" />
          <TextBlock Text="{Binding id}"
            HorizontalAlignment="Center"
            FontWeight="Bold" />
        </StackPanel>
        <TextBlock Text="{Binding text}" VerticalAlignment="Center" />
      </StackPanel>
    </DataTemplate>
  </ListBox.ItemTemplate>
</ListBox>
```

V tomto příkladě je definice vnořena přímo do elementu <ListBox>. Alternativou by samozřejmě bylo vytvořit šablonu v souboru "app.xaml" a odkázat se na ní pomocí parametru "Style". Subelementem zastřešujícím samotnou definici je <ListBox.ItemTemplate>. V něm vidíme další subelement <DataTemplate>, který se používá ve všech případech, kdy chceme zobrazovat nějakým způsobem načtená externí data. Veškerý obsah tohoto elementu udává již samotný design položky. Vidíme zde použitý horizontální <StackPanel>, obsahující jeden vertikální <StackPanel>, uvnitř tohoto panelu se nachází další dva elementy <TextBlock>, z nichž první má pevně nastaven parametr Text="ID". Stežejní jsou v této ukázce hodnoty druhého a třetího elementu <TextBlock> (umístěného za vertikální panel). Jejich struktura je velmi podobná odkazům na styly s tím rozdílem, že v tomto případě je použito klíčové slovo "Binding". Výraz následující za tímto klíčovým slovem je

odkazem na název sloupce načítané tabulky, jehož hodnota má být na příslušném místě zobrazena. To znamená, že do druhého textového pole budou načítána data ze sloupce s názvem "id" a ve třetím poli budou data ze sloupce "text". Výsledkem tohoto zdrojového kódu bude po načtení dat z databáze následující zobrazení.



Z této ukázky je patrné, že s pomocí šablon lze velmi jednoduše nastavit, jakým způsobem budou načtená data zobrazována. Dodejme ještě, že nemusíme skončit pouze u vytváření šablon pro položky seznamu, ale můžeme kompletně přepracovat vzhled samotné komponenty <ListBox> a vyhnout se tak základnímu obdélníkovému zobrazení.

## 3 C#

### 3.1 Propojení C# a XAML:

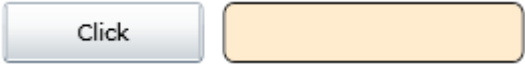
Technologie Silverlight vychází ze zaběhlé architektury používané jak ve Windows Forms, tak v ASP.NET a jiných .NET technologiích. Základem této architektury je, že každou scénu (ať už se jedná o internetovou stránku nebo a okno aplikace) tvoří dva spolu propojené soubory. Prvním je tzv. "designer" a druhým "code-behind". V technologii Silverlight je toto rozdělení zachováno tak, že XAML soubor je designerem a jako code-behind slouží soubor obsahující zdrojový kód napsaný v jenom z programovacích jazyků zmíněných v první kapitole. V případě kompilovaných jazyků je jejich činnost zajišťována systémem CLR (Common Language Runtime), u skriptovacích jazyků je běh zajištěn pomocí DLR (Dynamic Language Runtime), který překládá zdrojové kódy za běhu aplikace. Jak je uvedeno v úvodní kapitole této publikace, mně nejbližším programovacím jazykem je C# a tento jazyk také budu popisovat v této části.

Nejdůležitějším faktem pro pochopení vztahu XAML a code-behind souborů je, že všechny elementy používané v XAML souborech jsou zároveň C# třídami, ze kterých je možné dynamicky vytvářet objekty a využívat u nich stejných parametrů, vlastností a handlerů, jako kdybychom je definovali přímo v XAML. Díky tomuto vzájemnému propojení je také možné odkazovat se na jednotlivé objekty na scéně přímo z code-behind souboru. Typickým příkladem, který již byl uveden v části zabývající se animací, je spouštění animace definované v XAML příkazem zapsaným v code-behind souboru. Tento příkaz je možné uskutečnit právě proto, že z pohledu code-behind souboru "existuje na scéně objekt třídy StoryBoard", který má definovatelné parametry, handlers a především metody, které slouží k manipulaci s animací.



Ve chvíli, kdy v XAML souboru přiřadíme handleru určitého objektu nějakou metodu, je tato metoda vytvořena v code-behind souboru. Jakýkoli kód zapsaný do těla metody je posléze vykonán, jakmile nastane událost, kterou onen handler sleduje. Následující příklad názorně ukazuje, jakým způsobem pracuje propojení designeru a code-behind souboru.

```
<StackPanel Orientation="Horizontal">
  <Button Width="100" Height="30" Content="Click" Click="Button_Click"
    Margin="0,0,10,0" HorizontalAlignment="Left" />
  <Border Background="BlanchedAlmond" Width="150" Height="30"
    BorderBrush="Black" BorderThickness="1"
    CornerRadius="5" HorizontalAlignment="Left">
    <TextBlock x:Name="label" VerticalAlignment="Center"
      HorizontalAlignment="Center" />
  </Border>
</StackPanel>
```

Tento zdrojový kód definuje strukturu, jejíž hlavním layout-prvkem je `<StackPanel>`. Uvnitř tohoto elementu jsou umístěny elementy `<Button>` a `<TextBlock>`. Druhý jmenovaný je ještě ohraničen prvkem `<Border>`, aby byl na scéně vidět (prázdný `<TextBlock>` je v základu neviditelný, jelikož nemá žádný okraj ani výplň).  Výsledné zobrazení vypadá jako na obrázku vpravo.

V tomto příkladě jsou nejdůležitějšími dvěma prvky parametr "x:Name" elementu `<TextBlock>` a handler "Click" elementu `<Button>`. Tento handler sleduje událost kliknutí na tlačítko a jakmile tato událost nastane, zavolá metodu "Button\_Click", která je deklarována a zapsána v code-behind souboru. Její zdrojový kód je velmi jednoduchý a vypadá takto:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    label.Text = "Hello World!";
}
```

Jediný příkaz, který metoda vykonává je ten, že prvku na scéně se jménem "label" přiřadí do parametru "Text" řetězec "Hello World!". A jelikož tím prvkem na scéně se jménem "label" je právě `<TextBlock>`, výsledkem této metody je následující zobrazení. Na scéně se objeví nápis "Hello World!".



Můžeme si všimnout, že na obrázku je modře zbarvený okraj tlačítka. To znamená, že tlačítko je označeno - tedy, že na něj bylo kliknuto.

V této kapitole jsem vysvětlil, jakým způsobem jsou propojeny soubory designer a code-behind. Záměrně zde uvádím pouze náležitosti, které je třeba znát ke správnému pochopení tématu, a nerozvádím zde samotný jazyk C#. Popis jazyka C# by byl totiž velice náročný a dlouhý a v této práci na něj není místo. Pro bližší popis jazyka odkazuji čtenáře na některou z mnoha publikací<sup>[1]</sup> zabývajících se detailně jeho syntaxí a nebo na web MSDN<sup>[2]</sup> společnosti Microsoft, kde se nachází kompletní referenční příručka jazyka C# a mnoho jiných užitečných informací i o samotné technologii Silverlight.

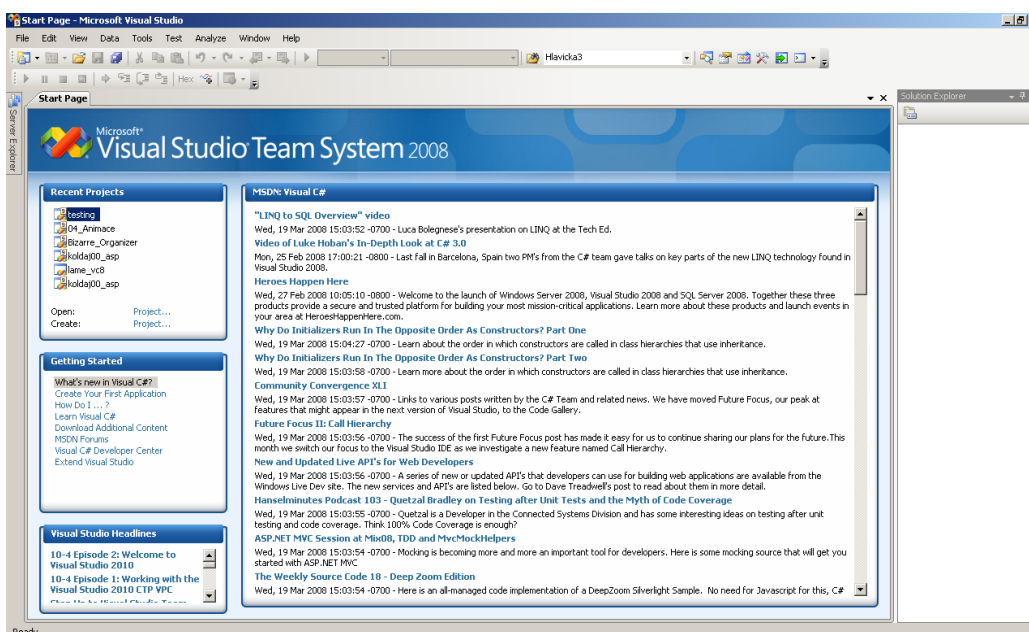
[1] SHARP, John, JAGGER, Jon. *Microsoft Visual C# .NET Krok za krokem*

[2] [http://msdn.microsoft.com/cs-cz/library/kx37x362\(en-us\).aspx](http://msdn.microsoft.com/cs-cz/library/kx37x362(en-us).aspx)

## 4 Vývojová prostředí

V této kapitole popíšeme dvě vývojová prostředí, s jejichž pomocí lze vytvářet Silverlight aplikace. Prvním z nich je Microsoft Visual Studio 2008 (dále Visual Studio) a druhým je Microsoft Blend 2, který je součástí balíčku Microsoft Expression. Vzhledem k rozsáhlosti obou aplikací zde popíšeme pouze některé jejich základní vlastnosti a přednosti, jelikož podobně jako v případě jazyka C# existují mnohé rozsáhlejší a detailnější publikace zabývající se touto problematikou a kompletní popis těchto aplikací není cílem této práce.

### 4.1 Microsoft Visual Studio 2008



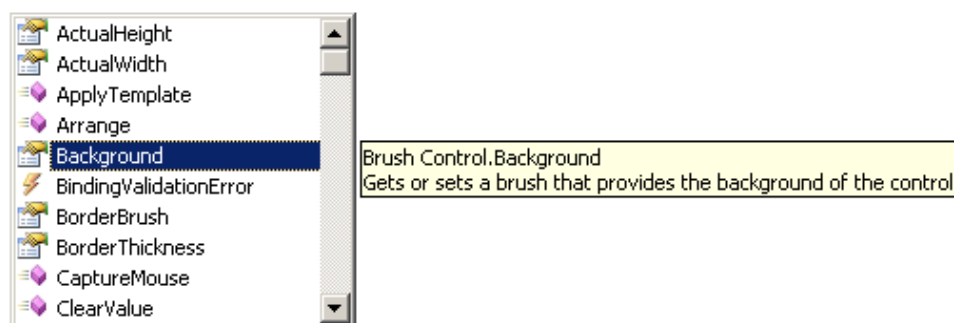
Visual Studio společnosti Microsoft je velmi sofistikovaný a široce použitelný nástroj pro vývoj veškerých .NET aplikací. Aby bylo možné využít tento nástroj i pro vytváření Silverlight aplikací, je třeba do něj nainstalovat rozšíření s názvem *Microsoft Silverlight 2 Tools for Visual Studio 2008*. Toto rozšíření nainstaluje kompletní Silverlight SDK (Software Development Kit), ve kterém se nachází všechny potřebné knihovny, dále pak Silverlight plugin

do prohlížečů, aby v nich bylo možno zobrazovat hotové aplikace a nakonec připojí k Visual Studiu několik projektových šablon, které slouží ke snadnému vytváření nových Silverlight projektů.

Všechny příklady a praktická řešení spojené s touto publikací byly vytvořeny ve Visual Studiu Team System 2008 SP1 za použití .NET Framework 3.5 SP1.

Hlavní předností Visual Studia je v něm zabudovaný systém Intellisense. Tento systém slouží k automatickému dokončování zapisovaných příkazů v kódu a zároveň navrhuje programátorovi další možná klíčová slova, která je možno v určitých situacích použít. Na následujícím obrázku je znázorněno, jak systém intellisense vypadá.

```
Button but = new Button();  
but.Back
```



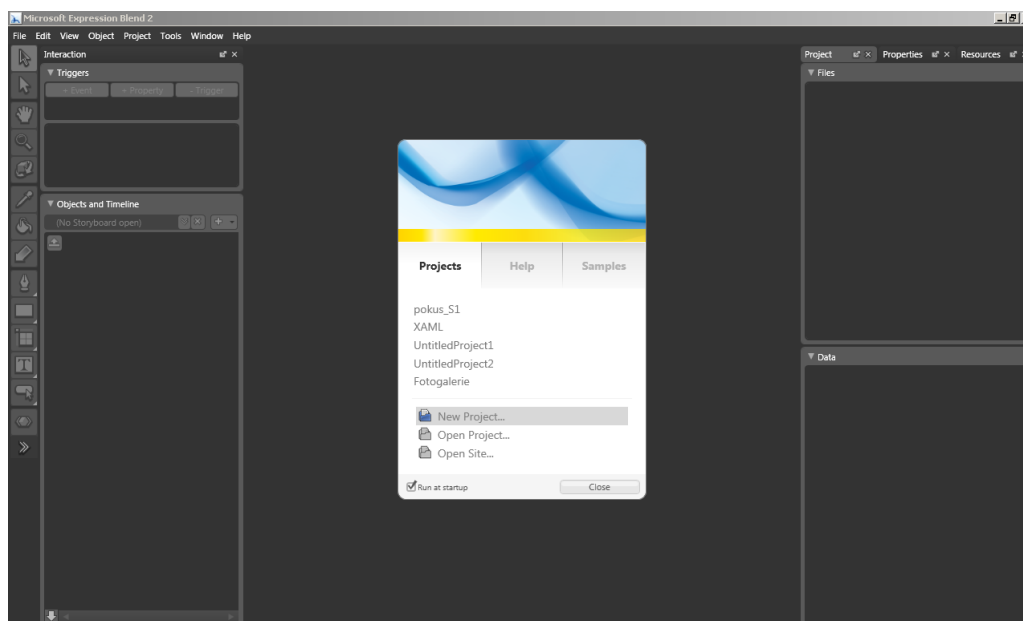
Jak je z obrázku patrné, systém intellisense je schopen automaticky dokončovat jednotlivé výrazy a zároveň slouží jako referenční příručka používaného jazyka. Díky tomuto systému je programování ve Visual Studiu velmi rychlé a často i jednoduché.

Velmi užitečnou pomůckou je také možnost použití zmíněných projektových šablon. Tyto šablony se starají o to, aby programátor nemusel vytvářet postupně všechny zdrojové soubory, které jsou potřeba v projektech. Visual Studio tyto soubory generuje automaticky (více informací o adresářové struktuře naleznete v kapitole 5) a opět tím vývojáři výrazně urychluje práci.

Jedinou, avšak docela podstatnou nevýhodou je fakt, že Visual Studio neumožňuje přímou tvorbu grafického designu pomocí kreslicích nástrojů

(jako je tomu např. v Adobe Flash). Grafik je zde tedy odkázán na ruční psaní celého XAML kódu a k dispozici má pouze statický náhled, naprosto neinteraktivní a neměnný.

## 4.2 Microsoft Blend 2



Microsoft Blend 2 (dále Blend) je oproti Visual Studiu přímo zaměřen na jednoduchou tvorbu grafiky s animací pomocí kreslících nástrojů. Samozřejmě je zde možnost zapisovat XAML kód ručně, ale zároveň jsou zde kreslící nástroje schopné generovat kód automaticky.

Tento vývojový nástroj je reakcí na technologii Adobe Flash, ve které je také možné definovat grafický vzhled scény pomocí velmi kvalitních a užitečných kreslících nástrojů. Blend umožňuje kreslit jakékoliv tvary, vkládat na scénu různé základní komponenty a jednoduše manipulovat s jejich parametry (např. rozměry a pozice na scéně). Zároveň umožňuje pomocí automatického "nahrávání" scény zachycovat uživatelem prováděné změny a posléze vytvářet příslušné animace.

Blend také umožňuje programovat code-behind soubory v .NET jednotlivých jazycích, avšak ačkoli i zde je zaveden systém intellisense, není natolik sofistikovaný jako v případě Visual Studia a programování je zde o něco složitější a těžkopádné. Tuto nevýhodu však plně nahrazuje skutečnost, že přímo z Blendu lze spustit Visual Studio, ve kterém se otevře stejný projekt

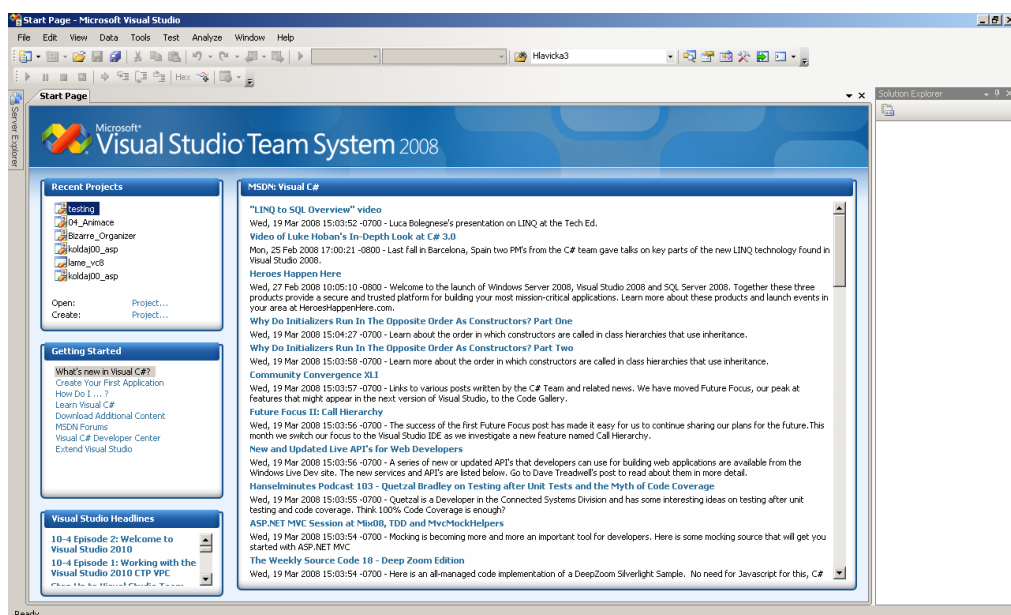
a následně je možno používat obě vývojová prostředí najednou. Každá úprava zdrojových kódů je pak sledována oběma aplikacemi a při přepnutí mezi nimi vždy dochází k synchronizaci projektů. Z důvodu této jednoduché propojitelnosti se velmi často používá Blend pro grafickou část aplikace a Visual Studio pro programovou část.

## 5 První aplikace v Silverlight 2.0

Tato kapitola se blíže zabývá procesem vytvoření Silverlight projektu v prostředí Microsoft Visual Studio 2008, je v ní popisována adresářová struktura Silverlight aplikace a na závěr je zde vysvětlena funkce tzv. ".xap" souboru a proces publikování Silverlight aplikací na webu.

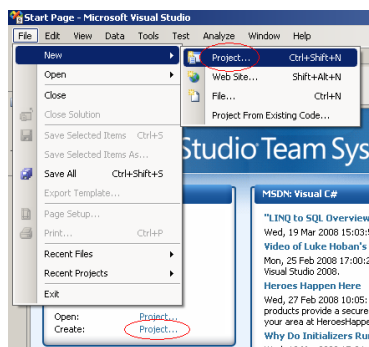
### 5.1 Stručný popis práce s MS Visual Studio 2008:

Tato část ukazuje na jednoduchém příkladě vytvoření nového projektu ve Visual Studiu a definici jednoduché aplikační logiky. Po spuštění Visual Studia se zobrazí startovní stránka:



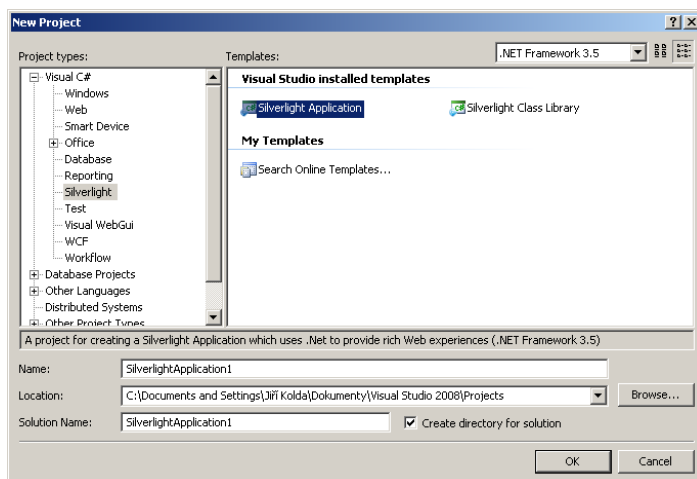
Startovní stránka se skládá z několika částí. Největší část s názvem "MSDN: Visual C#" obsahuje odkazy na online články týkající se programování v jazyce C#. Nalevo od tohoto bloku se nachází tři menší okna, z nichž první shora "Recent Projects" umožňuje znovu otevírat naposledy otevřené projekty a vytvářet projekty nové. Pod ním se nachází okno s několika odkazy na články obsažené v nápovědě MSDN. Tyto články obsahují informace vhodné pro začínající programátory. Vespod je pak poslední okno obsahující odkazy na články týkající se samotného Visual Studia.



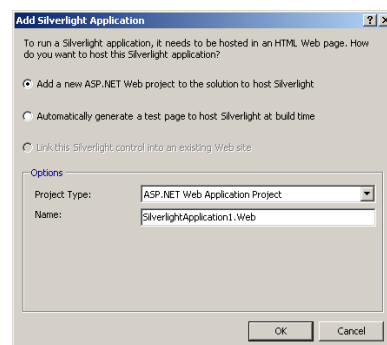


K vytvoření nového projektu vedou dvě cesty. Buďto můžeme kliknout na odkaz "Create Project..." v okně "Recent Projects", nebo můžeme tentýž příkaz najít i v horním menu aplikace v nabídce *File -> New -> Project...*

Po zadání tohoto příkazu se objeví okno, ve kterém je možné si vybrat příslušnou projektovou šablonu. V našem případě tu správnou šablonu najdeme ve skupině Visual C# -> Silverlight. Zde vybereme šablonu s názvem "Silverlight Application", zvolíme název a umístění aplikace a stiskneme tlačítko "OK".

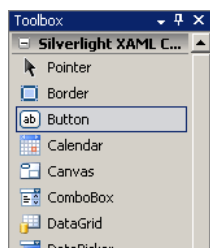


Posledním krokem, který je třeba udělat před samotným vytvořením projektu, je volba ze dvou možností. První možností je, že Visual Studio automaticky vytvoří testovací webovou stránku a přidruží ji k projektu, druhou z nabízených možností je generovat testovací stránku až při samotném spuštění aplikace. Zde zvolíme první možnost a opět stiskneme tlačítko "OK".



Po tomto kroku je již vytvořen projekt podle námi vybrané projektové šablony. Přesněji řečeno je vytvořeno tzv. "solution" obsahující dva projekty. Prvním z nich je samotná Silverlight aplikace (v našem případě s názvem "helloWorld") a druhým je automaticky vygenerovaná webová stránka, která aplikaci hostuje. Solution je struktura, která v sobě zapouzdřuje jednotlivé projekty a stará se o jejich správné propojení. Se stejným principem se setkáme u programování jiných v .NET technologiích, jako jsou ASP.NET nebo programování Windows aplikací.

Nyní se tedy prostředí Visual Studia skládá z následujících částí. Uprostřed okna přibily ke startovní stránce dvě nové záložky - "default.aspx" (to je ona automaticky vytvořená webová stránka) a "page.xaml", což je již důvěrně známý soubor, v němž je definovaná scéna Silverlight aplikace. Tento panel je rozdělen na dvě části - náhled scény a editor zdrojového kódu. Nalevo od

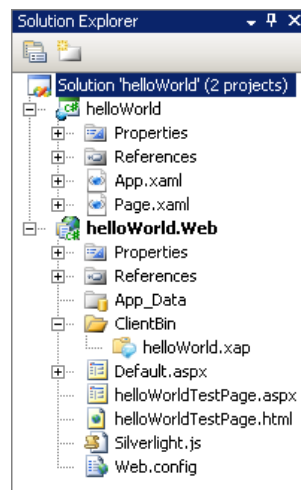


tohoto největšího okna se nachází panel nástrojů, tzv. "Toolbox", ve kterém se nacházejí všechny základní prvky použitelné na scéně. Přesunutím některého prvku z panelu nástrojů do zdrojového kódu XAML souboru vytvoříme na scéně příslušný objekt, kterému pak můžeme definicí parametrů měnit vzhled a chování. Na pravé straně Visual Studia se nacházejí panely "Properties" a "Solution Explorer". První jmenovaný je zde v případě Silverlight 2.0 zcela zbytečně, neboť jak je uvedeno výše, Visual Studio neumožňuje interaktivní změny parametrů, nýbrž pouze ruční zápis kódu. Druhý jmenovaný panel zobrazuje stromovou strukturu celého solution. Jsou zde vidět oba projekty, jak Silverlight aplikace, tak webová stránka a je zde zobrazena i adresářová struktura celé aplikace.

## 5.2 Rozvržení a adresářová struktura:

V této části popíšeme adresářovou strukturu uvnitř solution a celý jeho další obsah. Základem je rozdělení na dva projekty, které solution tvoří.

Prvním z nich, v našem případě s názvem "helloWorld" je projekt Silverlight aplikace, který obsahuje především soubory "Page.xaml" a "App.xaml" a jejich příslušné code-behind soubory. Dále jsou zde složky "Properties" a "References", ve kterých jsou uloženy soubory obsahující definice globálních vlastností celé aplikace a reference na používané .NET knihovny.



Druhým projektem je automaticky generovaná webová ASP.NET stránka, která hostuje Silverlight aplikaci. Tento projekt obsahuje opět složky "Properties" a "References", sloužící ke stejnému účelu jako v případě Silverlight aplikace. Dále se zde nachází dva velmi důležité adresáře - "App\_Data" a "ClientBin". Adresář "App\_Data" slouží jako úložiště datových souborů (např. databáze), které jsou spojeny s aplikací. Veledůležitým adresářem je pak "ClientBin". Tento adresář je totiž kořenovým adresářem Silverlight aplikace na serveru. Jinými slovy pokud zapisujeme do mediálních elementů cestu např. k videoklipu a zapíšeme ji ve tvaru "/videoclip.wmv", pak adresářem, kde bude Silverlight hledat tento videoklip, bude právě "ClientBin". Důvodem toho je fakt, že v tomto adresáři se nachází soubor ".xap", o kterém budu hovořit dále. V projektu webové stránky se dále nachází soubor "Default.aspx" a dva soubory s názvem "helloWorldTestPage". Pokud by nešlo o webovou stránku hostující aplikaci, sloužil by soubor "Default.aspx" jako startovací stránka. Jelikož zde jsou však automaticky generované testovací stránky, je zde soubor celkem zbytečně. V základním nastavení slouží jako startovací stránka soubor "helloWorldTestPage.html". Posledními dvěma soubory nacházejícími se v tomto projektu jsou "Silverlight.js" (kód jazyka

javascript zajišťující správné zobrazení Silverlight aplikace na testovací stránce) a "Web.config", který obsahuje základní konfigurační příkazy ASP webového projektu.

### 5.3 Soubor ".xap":

Asi nejdůležitějším souborem v celém solution je pro silverlight aplikaci soubor s příponou ".xap". Do tohoto souboru se totiž kompilují všechny XAML a code-behind soubory Silverlight aplikace. Pro čtenáře pracující v technologii Adobe Flash by se dalo říci, že soubor .xap je protějškem .swf souboru (kompilovaná Flash aplikace). Přesněji řečeno, soubor .xap je v podstatě archiv obsahující všechno, co potřebuje plugin v prohlížeči, aby správně zobrazil Silverlight aplikaci. Pokud se podíváme dovnitř tohoto archivu uvidíme tam zhruba to, co je vidět na obrázku

napravo. Jedná se vždy o soubor "AppManifest.xaml"



a další soubory ".dll", z nichž jeden má vždy stejný

AppManifest

helloWorld.dll

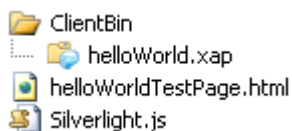
název jako Silverlight aplikace. Soubor "AppManifest.xaml" obsahuje informace o tom, jaké soubory jsou součástí souboru .xap a kde se v něm nacházejí. Soubor "helloWorld.dll" (v našem případě) reprezentuje kompilovaný Silverlight projekt. Mohou tu být i další .dll soubory, a to tehdy, když v aplikaci využíváme některé přídatné .NET knihovny.

### 5.4 Publikování Silverlight Aplikace:

Ve chvíli, kdy máme hotovou nějakou Silverlight aplikaci, jistě si položíme otázku, co je třeba udělat, aby byla tato aplikace přístupná na nějaké webové stránce (jiné než testovací). V této části vysvětlím, jak je třeba postupovat při umístování aplikace na webový server a uvedu, co všechno je třeba na server umístit.

První věc, kterou je třeba zajistit pro správný běh aplikace, je vhodný typ serveru. Stejně jako ASP technologie totiž Silverlight vyžaduje podporu ze strany serveru. Onou podporou je myšleno správné nastavení MIME typů na

serveru, aby bylo možné odkazovat se na soubor .xap. Z českých hostingových webů plně podporuje Silverlight například server ASPone.cz. Neméně důležitá je znalost toho, které soubory jsou klíčové pro správné zobrazení Silverlight aplikace na webové stránce. Následující obrázek ukazuje, co vše má na serveru být (soubory se samozřejmě vztahují k našemu příkladu).



Z obrázku je patrné, že na serveru nesmí chybět soubor .xap patřičně umístěný v adresáři ClientBin.

Ve stejném adresáři by se měly vyskytovat také mediální soubory, na které se aplikace odkazuje.

Dále by zde měla být nějaká webová stránka, na které bude aplikace hostována a pro správné zobrazení (včetně upozornění, nemá-li uživatel nainstalovaný plugin v prohlížeči) je nezbytný soubor "Silverlight.js". Pro bezchybné zobrazení aplikace je velmi doporučováno zachovat v hostující webové stránce její zdrojový kód automaticky generovaný Visual Studiem.