8 5. května (pražského povstání) – Xamarin, UWP Applications

Ve Visual Studiu je možno psát i jiné aplikace, nežli jsme se doposud učili. Výhodou je, že i ty ostatní aplikace (které si ukážeme dnes) se dají vyvíjet v C#.

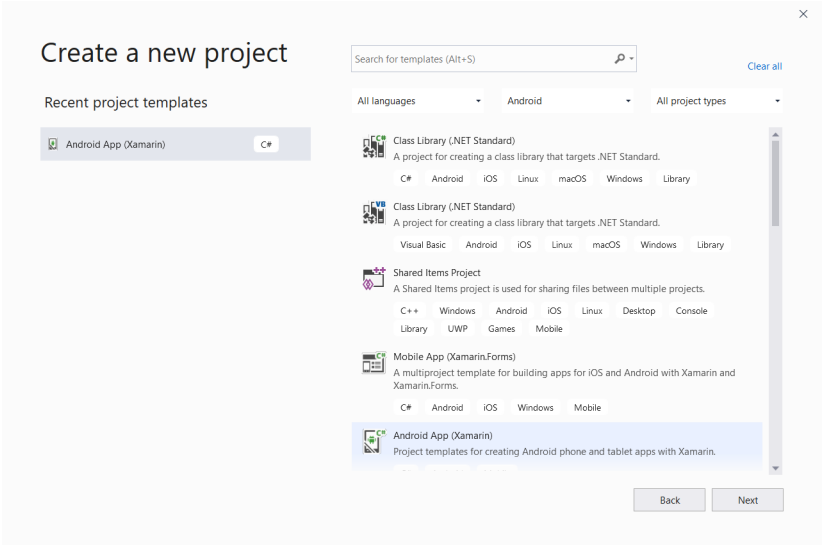
8.1 Xamarin

Slouží k vývoji aplikací pro jiná prostředí (nežli Windows). Typické využití spočívá ve vývoji aplikací pro Android, kterým je dnes vybavena většina mobilních telefonů.

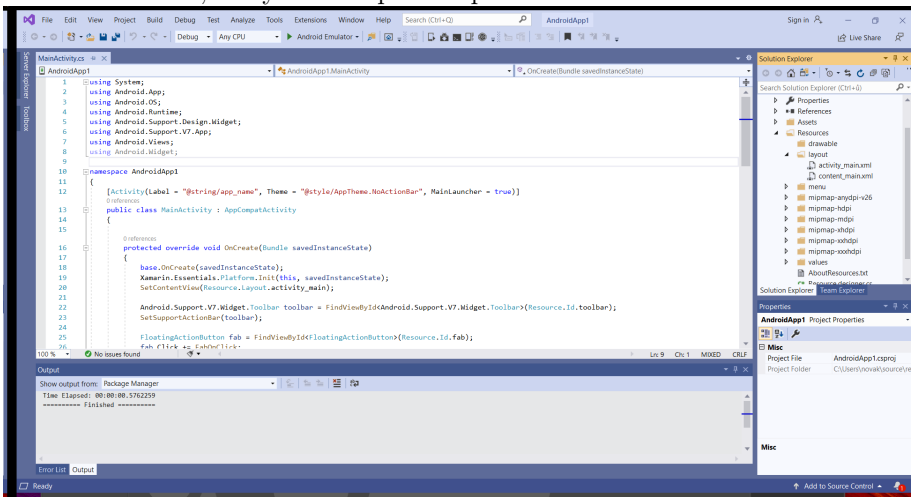
Vývoj aplikací pro Android umí Microsoft Visual Studio až v posledních verzích. Chceme-li vyvíjet tyto aplikace, při instalaci (ve Visual Studio Installeru) je třeba zakliknout *Mobile Development with .NET* (buďto při nové instalaci, nebo v sekci *Installed* vybrat *Modify,* v tabu *Workloads* vybrat *Mobile Development with .NET* a nechat instalovat).

Jelikož prostředí Visual Studia běží v Microsoft Windows, budeme potřebovat emulator přístroje s Androidem. O tyto emulatory se stará `AVD-Manager,` který najdeme v menu. Tam si vybereme, jaký model telefonu chceme emulovat, na jakém hardwaru a s jakou verzí Androidu. Jelikož kromě Visual Studia spustíme ještě emulator mobilního telefonu, je vhodné mít buďto hodně paměti RAM, nebo vybrat co nejslabší emulovaný telefon. Vhodným telefonem je například `Nexus One.` Je velmi vhodné zprovoznit dotyčný telefon s procesorem famílie x86, nikoliv ARM. Studio vám totiž běží obvykle na x86kovém procesoru a jeho emulace se dá virtualizovat (což je výrazně rychlejší). Pokud se nedaří zprovoznění telefonu s x86, může za tím být špatné nastavení virtualizace (v BIOSu či Windows). Telefon s ARMem startuje přes deset minut (což je daleko za pomezím praktické použitelnosti).

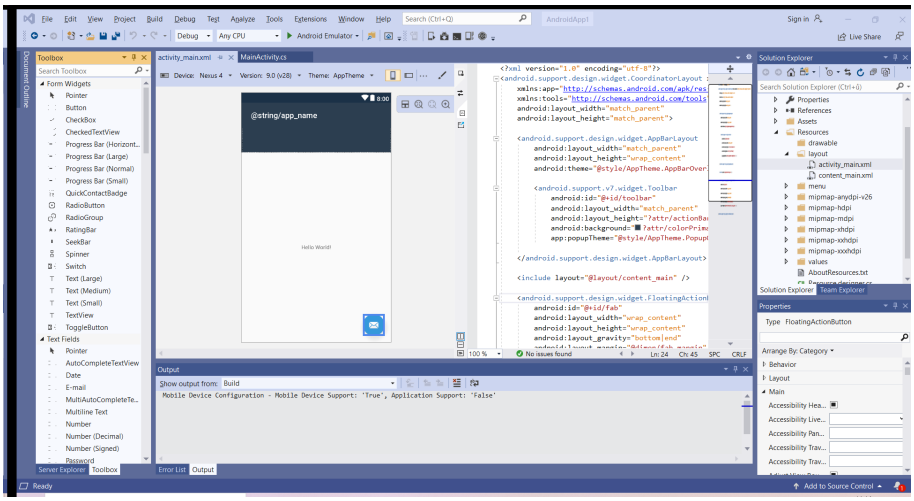
Máme-li vše instalováno (tedy jak podporu mobilních apliací ve Studiu, tak emulator vhodného telefonu), můžeme se pustit do práce. Při zakládání projektu vybereme `Android app (Xamarin)`



Následně se nám objeví další z tisíce tváří Visual Studia. Textový editor je pořád stejný, jen zdrojový text je poněkud kryptičtější. Povšimněte si, že v něm není funkce `Main`. Životní cyklus aplikací pro Android je totiž řízen šesti (úplně jinými) funkcemi, a to: `onCreate`, `onStart`, `onResume`, `onPause`, `onStop` a `onDestroy`. Jednotlivé funkce se vyvolají při příchodu dotyčné události. Těchto šest funkcí tvoří tři páry (první s poslední, druhá s předposlední a zbývají dvě prostřední), kde funkce v páru jsou k sobě duální. Tedy duální k události vytvoření je událost zničení. Duální ke Startu je Stop. Pro nás bude zajímavá především událost `onCreate`. Ta se spustí při startu programu. Do příslušné funkce dáme kód, který chceme provést při inicializaci.



Povšimněme si, že i zde se (podobně jako u formulářových aplikací) masívně využívá událostmi řízené programování. Není tedy divu, že programování začne naklikáním formuláře.



Ovládací prvky se jmenují jinak, nežli tomu bylo u formulářových aplikací. Shoduje se snad jen `Button`, kdežto to, co znáte jako `label`, najdete jako `TextView` a `TextBox` je přejmenovaný na `EditText`. Důvodem je, že ačkoliv programujeme v C#, o objektovém modelu rozhoduje společnost Google. Ta například také nemá formulář, ale layout. S tímto objektovým modelem souvisí i poněkud méně příjemná práce se vstupem a výstupem z formuláře. Zatímco ovladače událostí se zavolají tak, jak jsme zvyklí (a dokonce i syntax ovládacích funkcí vypadá, jak bychom předpokládali), nemáme tu přímo přístup k prvkům na formuláři. K těm se dostaneme pomocí třídy `Resource` a v ní pomocí položky `Id`. Až v této třídě můžeme využít ID, která jsme přidělili v designeru.

Aby to ale nebylo příliš snadné, společnost Google zavedla tato ID poněkud zvláštním způsobem. Pojmenovat si prvek v designeru můžete celkem libovolně. Ale jen do chvíle, kdy k němu chcete pomocí tohoto ID přistupovat. Abyste se k prvku dostali, musí jeho jméno začínat `@+id/` a až za toto lomítko napíšete název, na který jste zvyklí při běžném pojmenovávání prvků. Vyrobíme-li se tedy tlačítko, kterému chceme říkat *knoflík*, pojmenujeme jej v designeru `@+id/knoflik`, budeme-li k němu chtít v programu přistoupit, odkážeme na něj jako na `Resource.Id.knoflik`. Ovšem ID je pouze klíč dotyčného prvku. Nejde o odkaz na objekt. Chceme-li s daným objektem pracovat, musíme na získané ID zavolat funkci `FindViewById`. Tedy abychom se dostali do stavu, na který jsme zvyklí z Windows Forms Application, čili aby prvek, který má ID knoflik, tedy přesněji @+id/knoflik, byl přístupný jako `button1`, musíme provést něco takovéhoto:

```
Button button1=FindViewById(Resource.Id.knoflik);
```

Obsah vstupně-výstupních prvků se ovládá voláním metod a wrapperů (jak jsme zvyklí). Takže máme-li `TextView`, jeho textový obsah ovládáme (čteme/zapisujeme) pomocí wrapperu `Text`. S touto informací si tedy začínáme připadat ve světě aplikací pro Android jako sloni v porcelánu, ale už vlastně umíme pracovat se vstupem a výstupem (pro každý prvek si stačí najít správné jméno, jehož pomocí se pracuje s hodnotami dotyčného prvku).

Nežli se pustíme do programování, musíme ztratit pár slov o rituálech (jichž nás aplikace pro Android neušetří). Jelikož se aplikace pro Android píší jako potomky třídy `Activity`, a ještě přesněji se dnes dědí od nějakých potomků této třídy, například `AppCompatActivity`, ke spuštění aplikace nedochází zavoláním konstruktoru, ale zavoláním metody `onCreate`. Pokud tuto metodu overridujeme, jako první krok té overridované metody musíme zavolat metodu `onCreate` v rodiči. O to se stará první řádek těla funkce `onCreate`, který nám prostředí vygenerovalo. Následně musíme zinicializovat Xamarin. To se děje na druhém řádku. Na třetím řádku stanovíme, jaký layout (tedy terminologií společnosti Microsoft formulář) se má zobrazit. Tyto tři řádky je tedy zapotřebí vždy ponechat. Vlastní tvořivosti se můžete věnovat až následně. Prohlédněte si tedy vygenerovaný zdroják. Další řádky ve funkci `onCreate` definují (prázdné) menu a to divné tlačítko v pravém dolním rohu (přesněji reakci na jeho stisknutí). Nežli si program spustíme, budeme muset nastavit, jak to provést.

Chceme-li program spustit, vidíme známou zelenou šipku známou z magnetofonů pod mnemonikem `play`. Vlevo od ní vidíme políčko, ve kterém musíme vybrat, na čem kód spustíme. A ještě dále vlevo si můžeme vybrat, zda chceme spustit ladicí či distribuční verzi. Nyní to začne být důležité. Distribuční verze se nedá ladit a ladicí verze naopak porůznu při startu čeká na připojení debuggeru (který když se nepřipojí, program se nespustí). Vyberte tedy nastavený emulator, řekněte `play` a kochejte se tím, jak zvolna startuje emulator mobilního telefonu, jak se do něj instaluje vaše aplikace a jak se spouští.

Povšimněte si, že Studio má za určitých okolností dojem, že Váš program běží, i když Vy ten dojem nemáte. Napadá Vás pro to nějaké vysvětlení? Pokud ne, plyne to z toho, co jsme si řekli hned na začátku (s těmi šesti funkcemi). Přerušíte-li běh programu, ještě to zdaleka neznamená, že jste jej ukončili. Program je tedy pouze přerušený, metoda `onDestroy` neproběhla a debugger tedy správně program dále sleduje.

Chcete-li si aplikaci instalovat do telefonu, je obvykle tam třeba povolit instalaci z neznámých zdrojů (což vyvolá velmi vděčnou reakci antivirů). Instalaci můžete provést tak, že do telefonu nakopírujete soubor s koncovkou "apk", který naleznete v projektovém adresáři.

Nyní zbývá sice drobnost, ale podstatná: Jak se v tomhle divném prostředí pracuje s běžnými věcmi? Na některé věci budete muset zapomenout. Například na tak běžnou činnost jako čtení dat ze souboru (v tomto případě na MicroSD-kartě). Samozřejmě to jde nastavit, ale pokud to není nutné, vyhněte se tomu. Telefon totiž často obsahuje citlivá data. Ta nezřídka bývají právě na kartě a asi nechcete, aby jakákoliv aplikace mohla tato data přečíst a bez kontroly kamkoliv poslat. Proto kontrole propadá jak přístup do filesystemu, tak přístup k síti. Chcete-li provádět tyto operace, objeví se před vámi tzv. Manifest. Tam každá aplikace manifestuje, co je zač. A mimo jiné, co bude dělat. Možná si vzpomínáte, že v každém novém telefonu je po jeho zprovoznění potřeba projít nastavení práv jednotlivých aplikací a pozakazovat ta, která vypadají podezřele. Tato práva se právě zjišťují z manifestu. Pokud se aplikace pokusí provést něco, co nemanifestuje, je drasticky ukončena. Pokud má aplikace zákaz provádět něco, co manifestuje, jsou výsledky legrační (funkce nemůže proběhnout, uživatel

je dotazován, zda opravdu nechce aplikaci povolit provést dotyčnou operaci - a když se aplikace neujistí, že operace proběhla, spadne na dereferenci nullového objektu).

Cvičení: Ve vzorové (vygenerované) aplikaci změňte obsah uprostřed stojícího textového políčka. Následně si přiklikejte textové vstupní políčko (`EditText`), tlačítko, tlačítku nastavte ovladač události a při kliknutí (na tlačítko) přečtěte text z textového vstupního políčka a zobrazte do textového (výstupního) políčka.

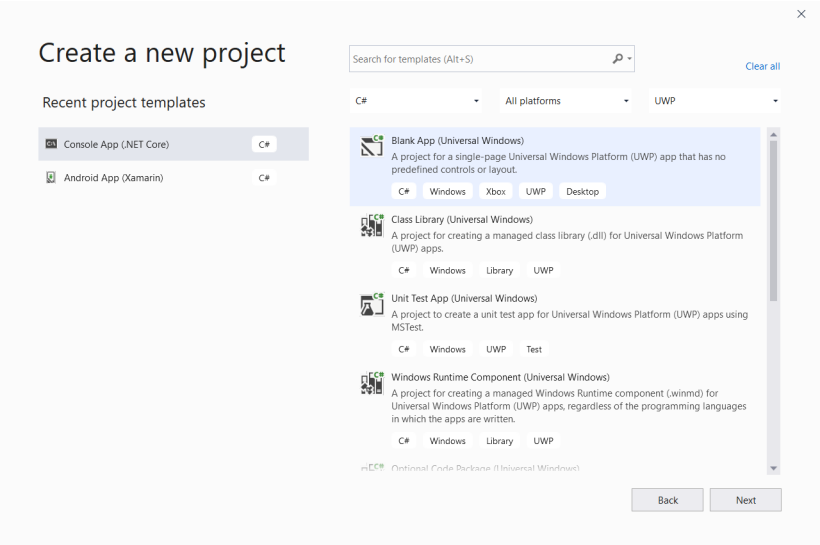
Cvičení: Navrhněte a implementujte vhodný netriviální příklad aplikace pro Android. Já teď, bohužel, nemám žádné použitelné nápady. Vyhněte se aplikacím, které mají průběžně překreslovat display. K těm si něco řekneme příště (kdy budou Thready). Bez nich to nepůjde. Jak to půjde s nimi, se dozvíte za týden.

Stále myslete na to, že i zde máte omezení známá již z Windows Forms Applications, tedy že dokud ovladač události nedoběhne, formulář se nepřekreslí. A běží-li funkce 15 sekund (v popředí a proces tudíž nereaguje), začne se Android starat, zda ho nemá zabít jako spadlý.

8.2 Universal Windows Platform

Jistě si vzpomínáte, jak na poslední přednášce v zimním semestru přednášející pro pobavení obecenstva promítal z robota ovládaného Raspberry Pi. To by to nebyl přednášející, kdyby i letní semestr nezakončil podobně (byť jeho konec přijde až přespříští týden, příště budou nedodělky a přespříště Labyrint světa programovacích jazyků, kam se tato položka nehodí).

Pro některé modely Raspberry Pi (konkrétně 2 a 3, pozor na 3B+) existují Windows 10 IoT Core. Toto je operační systém navržený pro internet věcí. S Windows tak, jak je znáte nemají společné téměř nic. Není tam Notepad, ani Paintbrush a už vůbec ne Solitaire či Miny. Instalovat jej musíte na MicroSD-kartu pomocí `Windows Assessment and Deployment Kit`. Výhodou naopak je, že tento systém můžete udržovat po síti (není tedy třeba klávesnice a display). Jde o operační systém, pro který lze psát programy ve Visual Studiu. Tentokrát je potřeba (v Installeru) zprovoznit podporu Universal Windows Platform. Následně si necháme vytvořit prázdný projekt pro Universal Windows Platform.



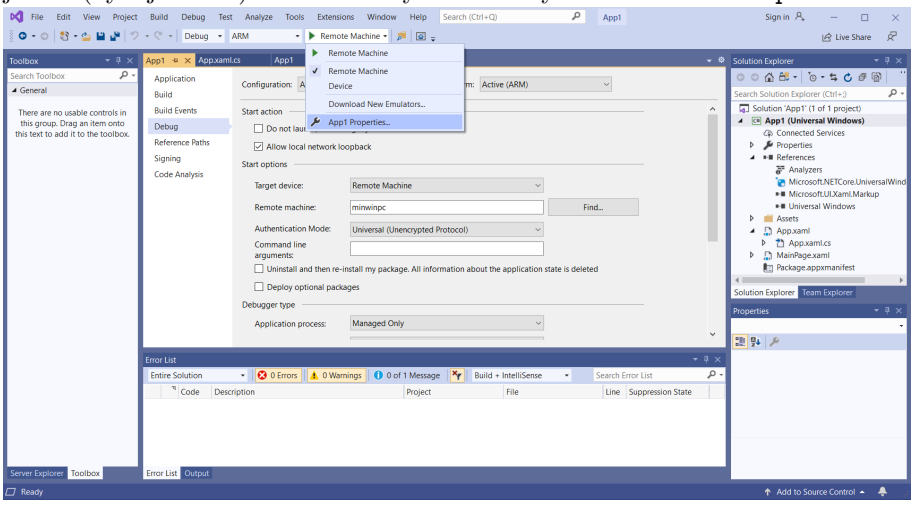
Opět budeme programovat v tom C#, který se celý semestr učíme. Samotné programování tedy bude opět stejné. Co bude jiné, bude opět vstup a výstup. Jelikož však nyní jde o prostředí navržené společností Microsoft, nebude to tak dramatické, jako při vývoji aplikací pro Android.

Podobně jako když jsme vyvíjeli aplikace pro Android, můžeme ve Visual Studiu aplikaci uploadovat do Raspberry a dokonce tam vzdáleně ladit. A v případě, že bychom s tím měli problémy, můžeme zkusit ladit lokálně (proč jinak by se dotyčná platforma jmenovala Universal Windows Application). Ladění nám lokálně půjde do chvíle, kdy si začneme hrát s GPIO (to obvykle lokálně přítomné nemáme).

Nyní se tedy opět pustíme do příkladů. Začneme tím, že i nově vytvořenou aplikaci lze spustit. Toto je vhodné si vyzkoušet, abyste zjistili, co vám případně brání v úspěšném spuštění aplikace. Vytvořte tedy novou aplikaci a spusťte ji na vzdáleném zařízení.

Vytvořit aplikaci nebude těžké. Spustit ji bude obtížnější. Přesněji obtížnější to bude, chcete-li ladit na vzdáleném hardwaru. Ne nadarmo se typ aplikace jmenuje Universal Windows Application, ladit ji tedy můžete i lokálně. Chcete-li však ladit jinde, je třeba toto nastavit. Konkrétně chceme-li ladit na Raspberry Pi, které má jako CPU některý z famílie ARMů, je třeba ve Visual Studiu přenastavit obsah v liště popisující spouštění a ladění (sestávající ze tří okének obvykle obsahujících stringy Debug, x86 a Device. Spouštět sice stále můžete ladicí verzi (tedy slovo Debug můžete ponechat). Místo x86 už ale nastavíte ARM, protože Raspberry Pi nemá CPU intelské famílie ale ARM, jak jsme si již řekli a namísto `Device` nastavíte `Remote Machine`. Nastavení je pak zapotřebí dokončit v em Project *rightarrow* Properties. Tam v tabu Debug musíme nastavit jméno nebo IP adresu dotyčného přístroje do položky Remote Machine. Dále je třeba v nastavení zapnout Developer mode. O ten se nemusíme až tolik starat. Jen si necháme vytvořit Universal Windows Application, Visual Studio

se nás rovnou při každém startu (dokud to tak nenastavíme) bude vyptávat, v jakém (vývojářském) režimu má být. Tam tedy naklikáme `Developer mode`.



Ostatní nastavení (projektu) nehraje až takovou roli. Některé tutoriály radí změnit i Authentication Mode na None, případně do Remote machine za adresu (či jméno) přidat dvojtečku a číslo portu (8116). Někteří zase říkají, že bez spuštění remote debuggeru na dotyčném zařízení (ve webovém interfacu od Raspberry v sekci Debug) zapnout (tlačítkem vpravo dole) debugger. Autorovi textu se jako jediná důležité věc rozhodující o úspěchu či neúspěchu uploadu a následného spuštění jeví připojení k Internetu. S připojením funguje jakékoliv nastavení zmíněných položek, bez připojení pak nefunguje nic. Toto je poněkud zvláštní, protože přístroje provozující Windows 10 IoT Core tak musejí být při každém nahrávání programů připojené k Internetu. To vypadá jako významné bezpečnostní riziko. Po uploadu (přesněji po ukončení ladění) však již lze spojení odpojit.

Máme-li prázdnou aplikaci spuštěnu, vidíme, že aplikace pro Universal Windows Platform vypadají poněkud jinak nežli ostatní aplikace. S tím souvisí také to, co jsme již viděli na Androidu, a to velmi opatrná práce s vybavením. Konkrétně a například proces opět ani nesmí hledat data na filesystemu (potažmo na microSD-kartě). Svůj majetek musí mít aplikace přibalený u sebe. Chceme-li tedy pracovat se souborem, musíme říci, že aplikace bude přistupovat do filesystemu. Nechceme-li toto nastavovat, musíme využít resourcy, tedy potřebné soubory, se kterými budeme pracovat, přibalit do projektu. Toto uděláme hned ve druhém příkladu (kde nám bude aplikace vyhrávat). Příklady však teprve začnou a je třeba postupovat od začátku (hrát si teprve budeme).

Máme-li naklikaný prázdný projekt (který jsme před chvíli úspěšně spustili), zdrojový kód, který budeme modifikovat, najdeme v souboru `MainPage.xaml.cs`. Ten najdeme v Project Manageru pod souborem `MainPage.xaml`. Chceme-li formulář a designera, otevřeme `MainPage.xaml`. V designeru se kliká způsobem, který znáte z Windows Forms Applications. Časem se všimněte, že prvky na

formuláři se také chovají způsobem, na který jsme zvyklí. Tedy textový obsah se skrývá pod wrapperem `Text`, tlačítko je stále instance třídy `Button`,... Taktéž si ale všímejte, že některé prvky jsou jiné!

Příklad 1: V designeru naklikejte na formulář tlačítko, tedy prvek typu `Button` a místo pro zobrazování textu typu `TextBlock` (který tu zastupuje prvek *Label*, který znáte z Windows Forms Applications. Tlačítko pojmenujte `cudlik`, textové políčko pojmenujte natext a nechte tlačítku vygenerovat ovladač události (ten s defaultním jménem `cudlik_Click`). Tento ovladač nastavte takto:

```
private void cudlik_Click(object sender, RoutedEventArgs e)
{
        natext.Text = "Buch!";
}
```

Příklad spusťte a obdivujte, jak stále všechno funguje stejně, jako jste zvyklí z Windows Forms Applications (jen ty prvky se jmenují občas jinak).

Příklad 2: Jako *Asset* přidejte soubor jménem *Brahms.mp3*, příklad laďte na stroji se sluchátky nebo reproduktorem. Povšimněte si, že Raspberry Pi má port na sluchátka. Tím se na vás případně bude linout zvuk. Zvuk přes HDMI ve Windows 10 IoT Core na Raspberry Pi není podporován aspoň zatím). Týž ovladač události jako v minulém příkladu nyní definujte takto:

```
private async void cudlik_Click(object sender, RoutedEventArgs e)
{
    StorageFolder Folder = Windows.ApplicationModel.Package.Current.InstalledLocation;
    Folder = await Folder.GetFolderAsync("Assets");
    StorageFile sf = await Folder.GetFileAsync("Brahms.mp3");
    MediaElement PlayMusic = new MediaElement();
    PlayMusic.SetSource(await sf.OpenAsync(FileAccessMode.Read), sf.ContentType);
    PlayMusic.Play();
}
```

Tentokrát (provádíme-li pokus na Raspberry Pi nebo počítači bez reproduktorů) je třeba připojit sluchátka. Všimněte si, kolik práce dá přehrát uložený zvuk. V příkladu vidíte přístup k vlastním Assetům (prvních 5 řádků). Následuje spuštění vyhrávání (volání metody Play). Taktéž si všimněte klíčových slov `async` a `await`. První klíčové slovo používáme jako modifikátor funkce, která použije to druhé klíčové slovo. To druhé klíčové slovo znamená, že se má volání provést asynchronně, tedy nemá se čekat na jeho výsledek. Ještě přesněji klíčové slovo `async` v definici funkce říká, že se nemá čekat na výsledek našeho volání, jelikož si spustíme asynchronní funkci.

Příklad 3: Formulář s tlačítkem a textovým prvkem se nám osvědčil, tak ho využijeme i nyní. Opět jen modifikujeme ovladač kliknutí na tlačítko:

```
private void cudlik_Click(object sender, RoutedEventArgs e)
{
    MediaElement mediaElement = new MediaElement();
```

```
    var synth = new SpeechSynthesizer();
    //var voices = SpeechSynthesizer.AllVoices;
    //synth.Voice = voices.First(x => x.Gender == VoiceGender.Female);
    Windows.Media.SpeechSynthesis.SpeechSynthesisStream stream =
                await synth.SynthesizeTextToStreamAsync("Good-bye world!");
    mediaElement.SetSource(stream, stream.ContentType);
    mediaElement.Play();
}
```

Vidíme, že zvukový výstup je pomalu jednodušší, nežli přístup k vlastním souborům. Zakomentované řádky je možné odkomentovat, tím se nastavují vlastnosti hlasu loučícího se se světem.