Przedmowa

Branża komputerowa nieustannie się zmienia. Sprzedaż komputerów PC powoli spada, natomiast wzrasta sprzedaż urządzeń nowej generacji – tabletów i smartfonów. Zmianę tę łatwo zrozumieć, bowiem komputery nie są już dłużej wykorzystywane do pracy – komputery wykorzystywane są obecnie do *życia*, które na całe szczęście nie polega jedynie na wykonywaniu pracy.

Ogromny sukces komercyjny urządzeń nowej generacji sugeruje akceptację tej zmiany przez znaczną większość społeczeństwa, jednak dla takich firm jak Microsoft tworzy ona spory problem. Komputery PC przez następne 20 lat nie będą już tak ważne, jak to miało miejsce przez ostatnie dwie dekady. Systemy Windows 8.1 i Windows RT są pierwszą podjętą przez Microsoft próbą rozwiązania tego problemu, poprzez uczynienie systemu operacyjnego Windows „bardziej przyjaznego" tabletom.

Microsoft dokonał tego, wprowadzając nowy interfejs użytkownika o nazwie *Modern UI*. Interfejs ten jest monochroniczny (tj. wykonujemy w nim jedną rzecz na raz), podczas gdy normalny okienkowy system operacyjny jest polichroniczny (kilka rzeczy na raz). Został on również zoptymalizowany pod obsługę dotykiem.

Poza utworzeniem nowego interfejsu użytkownika, Microsoft wprowadził również nowy interfejs programowania aplikacji o nazwie *Windows Runtime* (WinRT) oraz nowy model wykonywania i pakowania aplikacji, nazywany *aplikacjami dla Sklepu Windows*. O konstruowaniu aplikacji dla Sklepu Windows porozmawiamy szczegółowo w rozdziale 2.

W niniejszej książce systemy Windows 8.1 i Windows 8.1.1 RT traktowane są na równi. Innymi słowy, wszystkie wykonywane przez nas czynności będą działać na obu wspomnianych wersjach tego systemu. Wszystko, co zrobimy w tej książce, będzie też miało zastosowanie do aplikacji rozprowadzanych przy użyciu Sklepu Windows.

W większości przypadków cały kod będziemy tworzyć sami, przy czym od czasu do czasu będziemy posługiwać się produktami zewnętrznymi. Wszystkie one, z wyjątkiem komponentu Bing Maps omawianego w rozdziale 11, są produktami otwarto źródłowymi, którymi możemy się swobodnie posługiwać.

Zatem zaczynajmy! Na początek powiemy sobie więcej na temat aplikacji, którą zbudujemy.

Potencjalni odbiorcy

Pisząc tę książkę staraliśmy się stworzyć pewną opowieść. W miarę jej czytania pozwoli Ci ona rozwinąć Twoje umiejętności tworzenia oprogramowania w środowisku .NET do poziomu, w którym będziesz mógł tworzyć aplikacje dla Sklepu Windows. Założyliśmy w tej książce, że większość czytelników pracuje na co dzień przy rozwijaniu aplikacji sieciowych, lecz zostali oni poproszeni o zapoznanie się z procesem tworzenia aplikacji przeznaczonych na tablety z systemem operacyjnym firmy Microsoft.

Niektórzy czytelnicy będą mieć zapewne jakieś doświadczenie w tworzeniu tradycyjnych aplikacji Windows, wykorzystując przy tym technologie Silverlight i/lub Windows Presentation Foundation (WPF). Książka ta nie skupia się na tworzeniu kodu XAML, choć zawiera ona wystarczającą liczbę przykładów, by móc biegle opanować ten język.

Powiemy sobie nieco więcej na temat tworzonej w tej książce aplikacji, byś mógł upewnić się, że książka ta będzie dla Ciebie właściwa. Aplikacja ta zawiera w sobie wszystkie standardowe funkcje, jakie można zazwyczaj znaleźć w aplikacjach biznesowych (ang. line-of-business, LOB). W czasie pisania tej książki tablety Windows nie zdążyły się jeszcze dobrze przyjąć na rynku, dlatego też omawiane w niej elementy będą miały zastosowanie do prawdziwych aplikacji działających na systemach Windows Mobile, Android i iOS, które mieliśmy okazję budować przez ostatnie 10 lat.

Mimo że budowana przez nas aplikacja będzie aplikacją biznesową, wszystko, co zobaczymy i wykonamy, będzie miało również zastosowanie do zwykłych aplikacji konsumenckich.

Aplikacja

Jak już wspomnieliśmy, w książce tej zajmiemy się budową aplikacji biznesowej, nie zaś aplikacji konsumenckiej.

Różnica pomiędzy nimi polega na tym, że w przypadku aplikacji konsumenckiej dostawca oprogramowania nie jest zazwyczaj ściśle związany z klientem końcowym. Klient znajduje aplikację poprzez pośrednie rekomendacje i/lub z poziomu katalogu sklepu z aplikacjami. W aplikacjach biznesowych stosuje się pewne proaktywne zabiegi marketingowe oraz podejmuje czynności tworzące relacje, mając na celu powiązanie klienta z dostawcą przy pomocy jakiejś komercyjnej oferty. Z technicznego punktu widzenia pomiędzy tymi aplikacjami nie ma jednak żadnej większej różnicy.

Szczególnym przykładem w tej książce jest aplikacja dedykowana „serwisowi terenowemu". Jest to klasyczna aplikacja przeznaczona do pracy mobilnej. Do naszych obowiązków należy sprawowanie kontroli nad działaniami pracowników. Wysyłamy ich w teren w celu wykonania jakieś pracy – może to być coś konkretnego („idź tam i napraw to") lub też coś reaktywnego (np. ktoś „patroluje" jakiś obszar i zgłasza ewentualne problemy).

Aplikacja, którą zbudujemy – o nazwie StreetFoo – będzie stanowić połączenie tych dwóch ostatnich przykładów.

Utworzyliśmy prosty serwer hostowany na platformie AppHarbor, który będzie służył jako usługa wewnętrzna dla naszej aplikacji. Po zalogowaniu się w aplikacji użytkownik pobierze z serwera zestaw „zawierających problemy raportów". Każdy raport będzie czymś, co wymaga podjęcia jakichś działań – w naszym przykładzie chodzi o usunięcie graffiti, ale mogłoby to być cokolwiek innego. Cała koncepcja aplikacji opiera się na tym, że jej użytkownik albo naprawi istniejący problem, albo zgłosi w niej jakiś nowy problem. Zaktualizowane lub nowe problemy są następnie przesyłane na serwer.

Tak wygląda podstawowa funkcjonalność naszej aplikacji. Dodatkowo przyjrzymy się kilku innym rzeczom, takim jak wykonywanie zdjęć oraz pozyskiwanie bieżącej lokalizacji, wykorzystując przy tym wszystkie specjalne funkcje oferowane przez Windows 8.1/ Windows 8.1.1 RT, takie jak udostępnianie, widok przyciągania, wyszukiwanie, itd.

Rozdziały

W rozdziałach 1 i 2 zbudujmy sobie odpowiedni grunt, który pozwoli nam przenieść się ze świata .NET do świata WinRT.

Rozdział 1, Przejście z platformy .NET (Część 1)

Rozdział ten wyjaśnia różnice pomiędzy .NET i powiązanymi z nim technologiami (w szczególności WPF) a środowiskiem Windows Runtime (WinRT). Budujemy w nim prosty interfejs użytkownika i implementujemy w nim wzorzec MVVM.

Rozdział 2, Przejście z platformy .NET (Część 2)

W tym rozdziale rozbudowujemy interfejs użytkownika z rozdziału 1, by był on bardziej użyteczny. Poruszamy tu kwestię odwoływania się do serwera w celu utworzenia nowego konta użytkownika, jak również szczegółowo przyglądamy się *asynchroniczności* – prawdopodobnie najważniejszej rzeczy, jakiej nauczymy się w czasie tworzenia naszej aplikacji dla Sklepu Windows.

Pozostałe rozdziały tej książki skupiają się już na konkretnych obszarach dostępnego API.

Rozdział 3, Lokalne dane trwałe

Rozdział ten omawia bazę danych SQLite. Temat ten poruszamy już na tym etapie, dlatego że obecnie bez posiadania jakiegoś trwałego magazynu praktycznie nie jest możliwe zbudowanie jakiejkolwiek przydatnej aplikacji. Choć korzystając z odpowiednich API możemy przechowywać informacje na dysku, to SQLite jest de facto standardową relacyjną bazą danych wykorzystywaną w dzisiejszych rozwiązaniach mobilnych. Z tego względu posłużymy się nią w naszej aplikacji.

Rozdział 4, Pasek aplikacji

W rozdziale tym prezentujemy pierwszą z nowych, dostępnych w systemie Windows 8.1 funkcji: pasek aplikacji. Paski aplikacji są małymi panelami, które wyskakując w górnej i dolnej części ekranu dają nam dostęp do różnych opcji i zakładek (pasek aplikacji w pewnym stopniu zastępuje paski narzędziowe). Przyjrzymy się, w jaki sposób możemy zbudować pasek aplikacji oraz jak możemy utworzyć nasze własne obrazki do wykorzystania na przyciskach.

Rozdział 5, Powiadomienia

Rozdział ten w całości poświęcony jest powiadomieniom. Powiadomienia w aplikacjach dla Sklepu Windows mogą być wykorzystywane do aktualizowania kafelków na ekranie startowym, dodawania do nich znaczków oraz do wyświetlania tzw. *powiadomień wyskakujących* (pojawiają się one w prawej dolnej części ekranu). Powiadomienia możemy tworzyć i wyświetlać lokalnie lub też tworzyć je na serwerze i wysyłać na wszystkie połączone urządzenia, wykorzystując do tego usługi Windows Push Notification Services (WNS). W rozdziale tym przyjrzymy się obu tym sposobom.

Rozdział 6, Praca z plikami

W tym miejscu przyjrzymy się szczegółowo pracy z plikami. Planując tę książkę nie zamierzaliśmy tworzyć tego rozdziału, jako że temat ten zawsze jest wyczerpująco omawiany przez społeczność, gdy tylko pojawia się jakaś nowa platforma. Zamieściliśmy go jednak ze względu na obsługę obrazów. Każdy śledzony w aplikacji raport będzie zawierał dokładnie jeden obraz. Zamiast przechowywać je w bazie SQLite, co byłoby dość niepraktyczne, będziemy zapisywać je na dysku.

Rozdział 7, Udostępnianie

W tym miejscu skupimy się na funkcji udostępniania w Windows 8.1. Udostępnianie zdecydowanie wyróżnia strategię firmy Microsoft dotyczącą tabletów na tle pozostałych platform. Większość platform oddziela od siebie aplikacje, utrudniając im wzajemne współdzielenie danych. Windows 8.1 dysponuje deklaratywnym modelem udostępniania danych, w którym aplikacja wskazuje typy danych, jakie będzie udostępniać. Dane te mogą być później odczytane przez inną aplikację obsługującą ten model udostępniania. W rozdziale tym omówimy kwestie zarówno udostępniania danych, jak również ich konsumpcji przez pozostałe aplikacje.

Rozdział 8, Wyszukiwanie

Rozdział ten dotyczyć będzie funkcji wyszukiwania w systemie Windows 8.1. Funkcja wyszukiwania, choć w różnych wariantach, implementowana jest w zasadzie we wszystkich aplikacjach. W systemie Windows 8.1 uzyskujemy do niej dostęp z poziomu panelu funkcji lub przy wykorzystaniu kontrolki SearchBox. My skupimy

się tu na implementowaniu funkcji wyszukiwania, która pozwoli nam odnajdywać nasze raporty dotyczące problemów.

Rozdział 9, Ustawienia

W tym rozdziale dokonamy podsumowania funkcji specyficznych dla systemu Windows 8.1, omawiając przy tym panel ustawień. Jak wskazuje jego nazwa – panel ustawień jest miejscem, w którym programiści mogą umieszczać ustawienia aplikacji. Jest to również powszechnie stosowane miejsce do zamieszczania łączy prowadzących do informacji na temat wsparcia technicznego oraz związanych z zasadami prywatności. W rozdziale tym wykorzystamy kontrolkę `SettingsFlyout` w celu załadowania i wyrenderowania tekstu sformatowanego metodą Markdown.

Rozdział 10, Lokalizacja

Na tym etapie zajmiemy się pozyskiwaniem bieżącej lokalizacji urządzenia. Większość mobilnych aplikacji firmowych wymaga takiej funkcjonalności, ponieważ często zachodzi potrzeba posiadania pewnych „dowodów" dokonania określonej czynności. Lokalizacja użytkownika może zostać również wykorzystana przy opracowywaniu nowych danych. W rozdziale tym zajmiemy się prostym pozyskiwaniem informacji o lokalizacji urządzenia, jak również posłużymy się kontrolką Bing Maps w celu wyświetlenia w naszej aplikacji mapy.

Rozdział 11, Korzystanie z aparatu

W tym rozdziale nauczymy się korzystać z aparatu urządzenia. W mobilnych aplikacjach biznesowych często konieczne jest gromadzenie pewnych dowodów zdjęciowych dotyczących wykonanej pracy (na przykład, jeśli ktoś ma za zadanie naprawić zlew, powinien on wykonać jego zdjęcie zarówno przed, jak i po jego naprawie). Przyjrzymy się tu, w jaki sposób możemy tworzyć nowe raporty dotyczące problemów, rozpoczynając od wykonywania zdjęć aparatem.

Rozdział 12, Responsywność projektu

Rozdział ten nauczy nas sposobu tworzenia responsywnych projektów, dzięki czemu interfejs użytkownika naszych aplikacji będzie mógł automatycznie dostosowywać się do ich bieżącej szerokości, obsługując nawet najmniejszy, 320-pikselowy rozmiar widoku, nazywany również „trybem przyciągnięcia". Kolejną funkcją wprowadzoną w Windows 8.1, która wyróżnia ten system na tle innych platform, jest możliwość uruchamiania aplikacji obok siebie. Innymi słowy, jedna aplikacja może zajmować cienki pasek po lewej lub po prawej stronie ekranu, podczas gdy kolejna zajmować będzie jego całą pozostałą przestrzeń. Jedynym problemem jest to, że dla takiego trybu działania aplikacji musimy zbudować całkowicie nowy interfejs użytkownika. Tak naprawdę nie jest to wcale takie trudne, gdyż wykorzystywany przez nas wzorzec

MVVM wykonuje dla nas sporą część tej pracy. W rozdziale tym wyposażymy naszą aplikację w zdolność dostosowywania się do różnych rozmiarów wielkości.

Rozdział 13, Zasoby i lokalizacja aplikacji

W tym rozdziale przyjrzymy się bliżej zasobom oraz porozmawiamy o lokalizowaniu naszej aplikacji. Zanim dotrzesz do tego tematu, będziesz już znał kilka sposobów pracy z zasobami, dlatego część tego rozdziału poświęcimy na omówienie rzeczy, którymi nie zajmowaliśmy się zbyt szczegółowo. Następnie przedyskutujemy prawidłowy sposób lokalizowania naszej aplikacji (tj. implementowania w niej obsługi wielu różnych języków).

Rozdział 14, Zadania w tle i cykl życia aplikacji

Rozdział ten omawia *zadania w tle*, czyli specjalny mechanizm wydzielania z aplikacji funkcji, które będą uruchamiane przez Windows w ustalonym harmonogramie. Podobnie jak na innych platformach, Windows stara się ograniczyć działanie naszej aplikacji, w przypadku gdy nie jest ona uruchomiona na pierwszym planie. Porozmawiamy tu o szczegółach implementacji zadań w tle, skupiając się na wykorzystaniu ich do pobierania nowych raportów oraz wysyłania wprowadzonych lokalnie zmian na serwer.

Rozdział 15, Pobieranie lokalne i dystrybucja

W tym rozdziale szczegółowo porozmawiamy o pakowaniu i dystrybuowaniu naszych aplikacji przy użyciu Sklepu Windows. Zobaczymy, w jaki sposób wykorzystujemy nasze licencje dewelopera w celu utworzenia pakietów *pobierania lokalnego* dla testowania wewnętrznego, jak również omówimy sobie poprawne wykorzystywanie tej funkcji w przedsiębiorstwie (pobieranie lokalne jest procesem, w którym dystrybuujemy nasze aplikacje do prywatnych odbiorców, zamiast udostępniać je publicznie w Sklepie Windows). Powiemy tu sobie również o regułach, jakie muszą zostać spełnione, by nasza aplikacja została dopuszczona przez Microsoft do dystrybucji w ramach Sklepu Windows.

Książka zawiera ponadto dwa dodatki:

Dodatek A, Kryptografia i tworzenie skrótów

Dodatek ten omawia pewne wymagania związane z kryptografią i tworzeniem skrótów, o których powinniśmy wiedzieć, przy czym nie nawiązuje on w żaden sposób do głównej treści tej książki.

Dodatek B, Podstawy testowania jednostkowego aplikacji dla Sklepu Windows

W tym dodatku przyglądamy się sposobom testowania naszego kodu przy użyciu oferowanych przez Visual Studio projektów testów jednostkowych. Skorzystamy tu z kontenerów odwrócenia sterowania, które budowaliśmy i wykorzystywaliśmy w przykładach tej książki.

I to wszystko! Po przeczytaniu całego materiału powinieneś być w stanie budować w pełni funkcjonalne aplikacje dla Sklepu Windows.

Wymagania wstępne

Jedyną rzeczą, której będziesz potrzebować do rozpoczęcia pracy, jest środowisko Visual Studio 2013. Wszystko, co będzie Ci potrzebne do budowy aplikacji dla Sklepu Windows, jest już zawarte w tej wersji Visual Studio. To, z której edycji będziesz korzystać, nie ma tu żadnego znaczenia. My co prawda korzystaliśmy z edycji Professional, ale wszystko przetestowaliśmy również w edycji Express.

Jeśli zechcesz umieścić swoje aplikacje w Sklepie Windows, będzie Ci potrzebne konto dewelopera. Żaden z naszych przykładów w tej książce nie wymaga od Ciebie zakupu takiego konta. Wystarczy, że utworzysz sobie bezpłatne konto, co umożliwi Ci pozyskanie licencji dewelopera pozwalającej na lokalne wdrażanie dowolnych zbudowanych przez Ciebie aplikacji.

Kod źródłowy

Kod źródłowy dla tej książki dostępny jest w witrynie GitHub.

Najprostszym sposobem pracy z tym kodem jest pobranie całego repozytorium na lokalny dysk. Każdy rozdział prezentowany jest w osobnym folderze, a każdy z nich zawiera kod aplikacji znajdującej się w takim samym stanie, w jakim znajdowała się ona na końcu danego rozdziału (na przykład, folder *Chapter8* zawiera wszystko od rozdziału 2 aż do rozdziału 8 włącznie; folder *Chapter9* zawiera wszystko to, co znajduje się w folderze *Chapter8*, oraz całą pracę, którą wykonaliśmy w rozdziale 9). W niektórych przypadkach udostępniony do pobrania kod może zawierać więcej zmian, niż tylko te, które zostały dokonane w tej książce.

Jeśli nie jesteś zaznajomiony z działaniem systemu *git* lub witryny GitHub, skorzystaj z zamieszczonego poniżej krótkiego szkolenia.

Korzystanie z systemu *git*

W tej części poznasz podstawy dotyczące instalacji i obsługi systemu *git*, które pozwolą Ci pozyskać kod z repozytorium. Nie mamy tu zamiaru pokazywać sposobów wykorzystywania systemu *git* jako systemu kontroli wersji – informacje na ten temat można znaleźć w serwisie GitHub.

Na początek potrzebny Ci będzie klient *git*. Możesz go pobrać z oficjalnej witryny tego projektu.

Po zakończeniu pobierania przeprowadź instalację uzyskanego pakietu.

Instalator zainstaluje zarówno klienta w wersji dla wiersza poleceń, jak i klienta z interfejsem graficznym. Wiele osób korzysta z wersji graficznej, przy czym my korzystaliśmy z klienta git dla wiersza poleceń. Jeśli chcesz jedynie pozyskać kod dołączony do tej książki, to najłatwiej Ci będzie posłużyć się klientem graficznym.

Aby uruchomić graficznego klienta git w systemie Windows 8.1, przejdź do ekranu Start wciskając klawisz Windows. Znajdując się na ekranie startowym, wpisz **`git`**. Po chwili otrzymasz dwie opcje: Git Bash oraz Git GUI. Otwórz klienta `Git GUI` i wybierz opcję Clone Existing Repository (Sklonuj istniejące repozytorium).

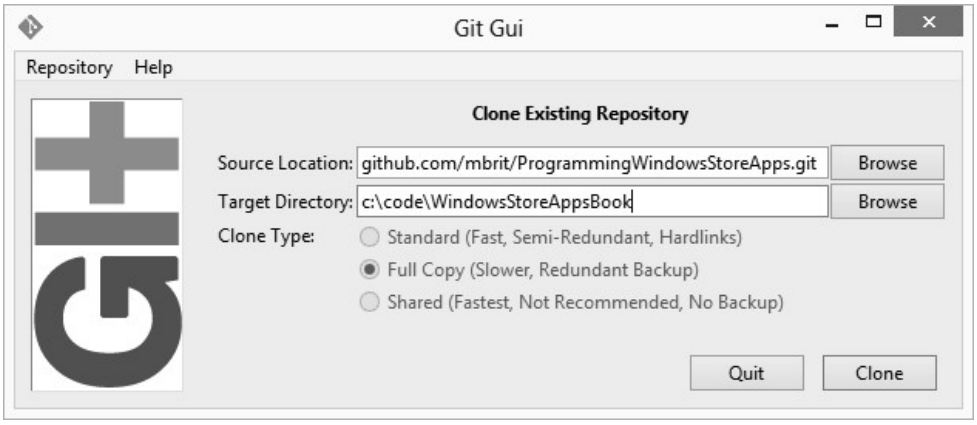
Będziesz musiał skopiować i wkleić ścieżkę do repozytorium z witryny GitHub. Aby tego dokonać, otwórz w przeglądarce poniższy adres: *https://github.com/mbrit/ProgrammingWindowsStoreApps*.

Na stronie tej, w sekcji „Quick setup", znajdziesz właściwy adres URL repozytorium.

Adres ten musisz skopiować do schowka. Powinien on wyglądać mniej więcej tak: *https://github.com/mbrit/ProgrammingWindowsStoreApps.git*. Ilustruje to rysunek P-1.

Rysunek P-1. Obszar strony GitHub prezentujący właściwy adres URL repozytorium

Wróć do klienta i wklej w nim skopiowaną ścieżkę do pola Source Location (Lokalizacja źródłowa). Następnie w polu Target Directory (Katalog docelowy) podaj ścieżkę do dowolnie wybranego przez siebie folderu lokalnego, podobnie jak na rysunku P-2.



Rysunek P-2. Konfiguracja operacji klonowania

Kliknij przycisk Clone (Sklonuj) i poczekaj na pobranie repozytorium na Twój komputer. Po zakończeniu tego procesu będziesz mógł otworzyć w Visual Studio zawarte w poszczególnych folderach pliki rozwiązań.

Kontakt z autorami

Najlepszym sposobem nawiązania kontaktu z Mattem jest jego Twitter (@mbrit).

Alternatywnie możesz też odwiedzić jego witrynę. Iris Classon dostępna jest zarówno na Twitterze (@irisclasson), jak i poprzez jej własną stronę.

Zaczynamy!

I to wszystko. Powinieneś już być gotowy do rozpoczęcia budowy aplikacji dla Sklepu Windows.

Konwencje stosowane w tej książce

W niniejszej książce wykorzystywane są następujące konwencje typograficzne:

Zwykły tekst
: Wskazuje tytuły, opcje i przyciski menu oraz skróty klawiszowe (takie jak Alt i Ctrl).

Kursywa
: Wskazuje nowe terminy, adresy URL, adresy e-mail, nazwy plików i ich rozszerzenia, ścieżki, katalogi i narzędzia systemu Unix.

`Stała szerokość`
: Wskazuje polecenia, opcje, przełączniki, zmienne, atrybuty, klucze, funkcje, typy, klasy, przestrzenie nazw, metody, moduły, właściwości, parametry, wartości, obiekty, zdarzenia, metody obsługi zdarzeń, znaczniki XML, makra, zawartości plików lub otrzymane z poleceń wyjście.

`Stała szerokość i kursywa`
: Komentarz wewnątrz kodu.

`Stała szerokość z pogrubieniem`
: Wskazuje składnię poleceń lub inny tekst, który powinien zostać wpisany przez użytkownika dokładnie tak, jak został przedstawiony.

`Stała szerokość z pogrubieniem i kursywą`
: Wskazuje tekst, który powinien zostać zastąpiony wartościami podanymi przez użytkownika.

Ta ikona określa ogólną uwagę.

Ta ikona określa wskazówkę lub radę.

Ta ikona wskazuje ostrzeżenie lub konieczność zachowania ostrożności.

Korzystanie z przykładowego kodu

Książka ta ma za zadanie pomóc Ci w wykonaniu Twojej pracy. W ogólnym przypadku, jeśli zawiera ona przykładowy kod, możesz wykorzystać go w swoich programach oraz w swojej dokumentacji. Nie musisz pytać nas o zgodę na jego wykorzystanie, chyba że reprodukujesz większą część naszego kodu. Na przykład, napisanie programu, który wykorzystuje kilka fragmentów kodu z tej książki, nie wymaga uzyskania od nas zgody. Sprzedaż lub dystrybucja płyt CD z przykładami z książek O'Reilly wymaga uzyskania takiej zgody. Udzielanie odpowiedzi na pytanie poprzez wskazanie tytułu tej książki i wstawienie kodu danego przykładu nie wymaga uzyskania zgody. Zawieranie większej ilości przykładowego kodu z tej książki w dokumentacji produktu czytelnika wymaga uzyskania od nas zgody.

Doceniamy (ale nie wymagamy) stosowanie atrybucji uwzględniającej tytuł, autorów, wydawcę i numer ISBN tej książki. Na przykład: „*Programowanie aplikacji dla Sklepu Windows w języku C#*, Matt Baxter-Reynolds i Iris Classon (O'Reilly/APN Promise). Copyright 2014 Matthew Baxter-Reynolds, 978-83-7541-152-2".

Jeśli masz wątpliwości, czy wykorzystanie przez Ciebie naszych przykładów nie naruszy powyższych postanowień, zapytaj nas o to pod adresem *permissions@oreilly.com*.

Jak się z nami skontaktować

Wszelkie uwagi i pytania dotyczące tej książki prosimy kierować bezpośrednio do jej wydawcy:

O'Reilly Media, Inc.
1005 Gravenstein Highway North Sebastopol, CA 95472
800-998-9938 (w Stanach Zjednoczonych lub Kanadzie)
707-829-0515 (międzynarodowo lub lokalnie)
707-829-0104 (faks)

Dla książki tej stworzyliśmy specjalną stronę, na której zamieściliśmy erratę, przykłady oraz wszelkie dodatkowe informacje. Strona ta jest dostępna pod adresem:

http://oreil.ly/prog-win-store-apps-csharp

Jakiekolwiek komentarze lub pytania techniczne prosimy kierować na adres:

bookquestions@oreilly.com

Więcej informacji na temat naszych książek, kursów i konferencji oraz wszelkie aktualności można znaleźć na stronie *http://www.oreilly.com*.
Znajdź nas na Facebooku: *http://facebook.com/oreilly*
Obserwuj nas na Twitterze: *http://twitter.com/oreillymedia*
Oglądaj nas na YouTube: *http://www.youtube.com/oreillymedia*

Podziękowania

Dziękujemy naszym recenzentom technicznym, którymi są Oren Novotny, Stefan Turalski, Matt Fitchett oraz Nathan Jepson. Bez nich ta książka nie mogłaby powstać.

Gdyby nie Twitter, książka ta nie byłaby tak dobra i kompletna. Twitter jest prawdopodobnie najlepszym źródłem zasobów do nauki dla osób związanych z branżą komputerową, jakie kiedykolwiek powstało. Niniejsza książka zawiera w sobie sporo cennych porad, z których jednak najważniejszą jest: jeśli jesteś profesjonalnym twórcą oprogramowania i nie korzystasz z Twittera, lepiej zacznij to robić.

Poniżej znajduje się lista naszych przyjaciół na Twitterze, którzy wspierali nas w opracowywaniu tej książki, zaoszczędzili nam wiele godzin pracy, poddali nam mnóstwo nowych pomysłów i wyrazili wiele cennych opinii:

- Alex Papadimoulis (@apapadimoulis)
- Casey Muratori (@cmuratori)
- Chris Field (@mrcfield)
- Chris Hardy (@chrisntr)
- Craig Murphy (@camurphy)
- Daniel Plaisted (@dsplaisted)
- David Kean (@davkean)
- Duncan Smart (@duncansmart)
- Edward Behan (@edwardbehan)
- Filip Skakun (@xyzzer)
- Frank Krueger (@praeclarum)
- Gill Cleeren (@gillcleeren)

- Ginny Caughey (@gcaughey)
- Haris Custo (@hariscusto)
- Hermit Dave (@hermitdave)
- Iris Classon (@irisclasson)
- Jamie Mutton (@jcmm33)
- Joel Hammond-Turner (@rammesses)
- Jose Fajardo (@josefajardo)
- Keith Patton (@kpatton)
- Kendall Miller (@kendallmiller)
- Liam Westley (@westleyl)
- Mark Tepper (@binaerforceone)
- Matt Hidinger (@matthidinger)
- Matthieu GD (@tewmgd)
- Mike Harper (@mikejharper)
- Nic Wise (@fastchiken)
- Peter Provost (@pprovost)
- Ross Dargan (@rossdargan)
- Tim Heuer (@timheuer)
- Tomas McGuinness (@tomasmcguinness)

Na koniec dziękujemy Rachel Roumeliotis, Marii Gulick, Melanie Yarbrough oraz całej reszcie zespołu O'Reilly za ich ciężką pracę i cierpliwość podczas opracowywania tej książki.

Rozdział 1

Przejście z platformy .NET (cz.1)

W tym i kolejnym rozdziale pokażemy, jak wykorzystać znajomość technologii .NET do rozpoczęcia pracy z platformą WinRT i budowania aplikacji dla Sklepu Windows (Windows Store). Pierwsze dwa rozdziały, w odróżnieniu od pozostałych rozdziałów, które koncentrują się na wybranych możliwościach interfejsu API, zawierają omówienie wielu różnych zagadnień. Zrozumienie tych podstawowych funkcji oraz zależności między nimi jest konieczne do skutecznego przejścia na platformę WinRT.

W sprzeczności z dotychczasową polityką firmy Microsoft środowisko WinRT nie jest następnikiem platformy .NET i stanowi przejaw rewolucyjnej zmiany strategii, na jaką zdecydowały się zespoły odpowiedzialne za interfejs API systemu Windows. Technologia ta została opublikowana w czasach szybkiej ewolucji w świecie oprogramowania. Era komputerów PC, która pozwoliła firmie Microsoft osiągnąć przewagę na rynku, powoli dobiega końca.

Dlaczego WinRT?

Powstanie technologii WinRT zbiegło się w czasie z opublikowaniem przez firmę Microsoft dwóch nowych wersji systemu Windows: Windows 8 oraz Windows RT, co stanowi raczej przypadek niż wynik zaplanowanej strategii. Platforma WinRT ma na celu zlikwidowanie ograniczeń związanych z pisaniem macierzystego oprogramowania dla systemu Windows. Do pisania aplikacji macierzystych w systemie Windows służy interfejs Win32, który jest bardzo stary i nie jest zorientowany obiektowo. Oprócz Win32 dostępna jest także technologia COM, czyli zorientowany obiektowo system podrzędny, który umożliwia podłączanie (i odłączanie) komponentów do systemu Windows. Technologia ta może być zupełnie nieznana osobom, które stosunkowo niedawno zainteresowały się pisaniem oprogramowania dla systemu Windows lub korzystają z platformy .NET. Nawet programiści, którzy kiedyś często korzystali z tego rozwiązania, z reguły porzucili je lata temu i do pisania oprogramowania dla systemów operacyjnych Windows wykorzystują obecnie platformę .NET.

.NET przypomina bardziej technologię Java niż Win32 czy COM i stanowi bibliotekę oraz środowisko wykonania, które ma ułatwić rozwijanie oprogramowania dla systemu Windows.

Platforma .NET jest nazywana środowiskiem „kodu zarządzanego", ponieważ środowisko uruchamiania przejmuje dużą część odpowiedzialności za zarzadzanie wykonaniem kodu. Natomiast aplikacje Win32 są „niezarządzane". Platforma .NET została wprowadzona z myślą o budowaniu witryn sieci Web w technologii ASP.NET lub aplikacji macierzystych przy użyciu technologii Windows Forms (na razie zignorujemy usługi systemu Windows i skoncentrujemy się na technologiach związanych z interfejsem użytkownika). Wspomniane technologie rozwijania interfejsów użytkownika ulegały przez lata różnym zmianom, ale główną zaletą platformy .NET pozostała łatwość, z jaką programiści mogą realizować własne cele. Wchodząca w skład pakietu .NET biblioteka Base Class Library (BCL) zapewniła łatwy, zorientowany obiektowo dostęp do funkcji systemu operacyjnego Windows (np. `System.IO.FileStream`), a także klasy usprawniające pracę programistów (np. `System.Collections.Generic.List`). Ten zestaw funkcji jest szczególnie istotny, ponieważ duża część biblioteki BCL odwołuje się do interfejsu Win32 i w ten sposób zapewnia dostęp do funkcji systemu operacyjnego. Poza biblioteką BCL środowisko uruchomieniowe CLR oferuje takie funkcje, jak Garbage Collection, które zdejmują z programisty ciężar zarzadzania pamięcią i innymi istotnymi aspektami.

Fundamentalne różnice

Problem polega na tym, że technologiom WinRT oraz .NET przyświeca zupełnie inna filozofia. Inspiracją dla technologii .NET była Java i zarządzanie pamięcią przy użyciu funkcji Garbage Collection, co skutkuje częściową utratą kontroli nad systemem. Java nie została zaprojektowana z myślą o programistach rozwijających systemy operacyjne (takich jak zespół Windows Division w firmie Microsoft, nazywany także zespołem WinDiv) lub twórcach złożonych produktów, takich Office. Java (a w konsekwencji także technologia .NET) jest zbyt abstrakcyjna, zbyt „magiczna", aby mogła być wykorzystywana do budowania tego typu oprogramowania. Jednak większość z nas nie rozwija „tego typu oprogramowania", lecz zajmuje się tworzeniem niewielkich aplikacji w ramach niedużego budżetu. W takiej sytuacji ograniczenia wynikające z pracy w środowisku wykonania takim jak CLR są mniej odczuwalne.

Programiści .NET, którzy decydują się na przejście do środowiska WinRT, muszą pamiętać o tych fundamentalnych różnicach. Choć musimy zaznaczyć, że mimo iż rozumiemy argumenty przemawiające za odejściem od filozofii platformy .NET i powrotem do technologii COM, nie do końca zgadamy się z tą decyzją. Możliwość budowania aplikacji dla Sklepu Windows przy użyciu technologii .NET pozwoliłaby uniknąć wielu

trudności wynikających z zastosowania technologii COM. Jednak musimy pogodzić się z rzeczywistością.

W tym rozdziale przyjęliśmy założenie, że czytelnik posiada przynajmniej podstawową znajomość technologii .NET oraz najnowszych rozwiązań, takich jak .NET Framework v3.5. Będziemy często stosować typy generyczne. Zaprezentujemy także fragmenty kodu LINQ, jednak będziemy objaśniać ich działanie. Do lektury książki nie jest potrzebna znajomość biblioteki Task Parallel Library (TPL), którą będziemy często wykorzystywać. Wystarczy umiejętność pisania profesjonalnego kodu na platformie .NET.

Wielu programistów .NET posiada pewne doświadczenie w rozwijaniu aplikacji ASP.NET, w związku z tym przyjęliśmy założenie, że większość czytelników preferuje tę metodę rozwijania interfejsu użytkownika. Do budowania interfejsu aplikacji dla Sklepu Windows wykorzystywać będziemy technologię XAML, która zostanie omówiona w dalszej części książki. Do tego samego celu można również użyć technologii Silverlight bądź Windows Presentation Foundation (WPF), ale przyjęliśmy założenie, że czytelnik nie ma doświadczenia w stosowaniu żadnej z tych metod.

Cele

W dwóch kolejnych rozdziałach przedstawimy proces budowania aplikacji dla Sklepu Windows z interfejsem użytkownika oraz pewną funkcją biznesową wymagającą nawiązania połączenia z serwerem. Ponadto udowodnimy, że możliwe jest zbudowanie biblioteki testów jednostkowych (ang. unit test) dla tego typu aplikacji:

- Zbudujemy nowy projekt aplikacji dla Sklepu Windows w środowisku Visual Studio 2012 po to, aby zaprezentować nowe szablony projektów oraz wyjaśnić różnice między plikami wyjściowymi generowanymi dla nowych projektów aplikacji dla Sklepu Windows oraz dla starszych typów projektów .NET.
- Dodamy do projektu nową stronę XAML i jednocześnie przedstawimy zagadnienia związane z budowaniem infrastruktury zgodnej z wzorcem projektowym Model/Widok/Model widoku (Model/View/View-Model – MVVM), który zostanie omówiony w dalszej części rozdziału. Dodatkowo pokażemy, jak wspierać paradygmat odwrócenia sterowania, który również zostanie przedstawiony w dalszej części rozdziału. Kod będzie zawierał wywołania metod ogólnie dostępnego serwera, aczkolwiek zaczniemy od zasymulowania tych wywołań.
- Zbudujemy test jednostkowy dla modelu widoku tylko po to, aby pokazać, że jest to możliwe.

- Po udowodnieniu, iż model może być testowany, zaimplementujemy prawdziwą komunikację z serwerem. Na zakończenie będziemy mogli przetestować całą aplikację i przekonać się, że działa ona prawidłowo.

Nowe szablony projektów

Warto zacząć od uświadomienia sobie, że w aplikacjach Sklepu Windows wykorzystywana jest okrojona, ograniczona wersja platformy .NET nazywana .NETCore. Nie pierwszy raz firma Microsoft zdecydowała się na stworzenie podzbioru technologii .NET – inny przykład stanowi platforma .NET Compact Framework zastosowana w systemie Windows CE lub platforma Client Profile wprowadzona w wersji v3.5 i „zoptymalizowana" z myślą o aplikacjach klienckich. Analogicznie zbiór narzędzi dla systemu Windows Phone 8 stanowi okrojoną wersję bibliotek Silverlight, które z kolei stanowią okrojoną wersję technologii WPF.

Proces „okrajania" bardziej zaawansowanych wersji ma na celu ograniczenie operacji, które mogą być wykonywane przez programistów. Firma Microsoft decyduje się na takie rozwiązanie z kilku przyczyn. Po pierwsze, pozwala ono na dodatkowe zabezpieczenie interfejsu API poprzez usunięcie potencjalnych punktów ataku. Po drugie, prowadzi ono do osiągnięcia „lepszych efektów końcowych". Innymi słowy, firma Microsoft kontroluje interfejs API w taki sposób, aby uniemożliwić programistom budowanie oprogramowania, które wpływałoby negatywnie na wizerunek produktu Windows 8.

Zależność tę można zilustrować na przykładzie czasu pracy baterii, który jest bardzo istotny w przypadku uruchomienia systemu Windows 8 na tablecie. Zasadniczo, im dłuższy czas pracy baterii, tym bardziej zadowoleni są użytkownicy. Jeśli firma Microsoft zaprojektuje interfejs API w taki sposób, aby utrudnić programistom budowanie rozwiązań, które znacznie podnoszą zużycie baterii, zdaniem użytkowników czas pracy baterii będzie dłuższy i w konsekwencji będą oni rekomendowali system Windows 8 jako system operacyjny dla tabletów.

Problem ten zostanie zilustrowany na przykładzie zadań w tle omówionych w rozdziale 14.

Podsumowując, możemy uzyskiwać dostęp tylko do tych funkcji, które zostały dodane do platformy .NETCore. Nawet gdybyśmy mogli pominąć to ograniczenie, nie byłoby to zalecane, ponieważ aplikacja prawdopodobnie nie otrzymałaby certyfikatu potrzebnego do opublikowania jej w Sklepie Windows (do tego zagadnienia powrócimy w rozdziale 15).

Metadane WinRT

Jedną z ogromnych zalet platformy .NET był od samego początku poziom szczegółowości systemu metadanych (używając pojęcia metadane w tym kontekście, mamy po prostu na myśli opis struktury wszystkich typów oraz ich elementów członkowskich). Każdy zestaw .NET mógł zawierać niezwykle dokładny opis. Natomiast biblioteki DLL systemu Windows DLL zawierały bardzo niewiele metadanych, zazwyczaj jedynie tabelę EXPORTS z listą funkcji do wywołania. Biblioteki COM oferowały nieco więcej informacji o sobie przy użyciu języka IDL, jednak poziom szczegółowości był nieporównywalny do metadanych zestawów .NET (ponadto praca z metadanymi COM przysparzała dużo większych trudności niż korzystanie z metadanych .NET).

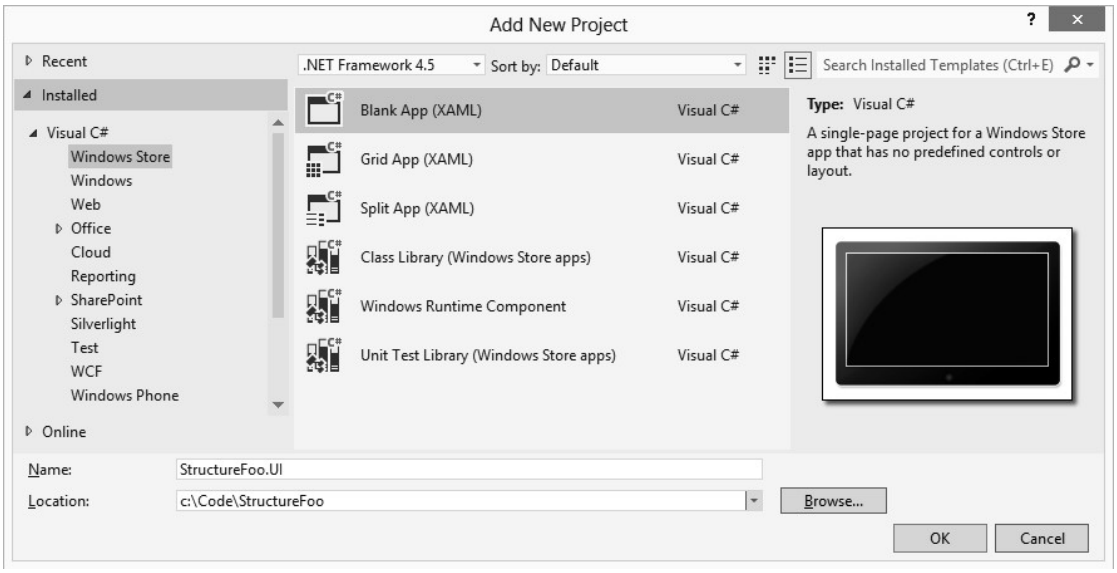
WinRT stanowi w rzeczywistości ulepszoną wersję COM, którą można byłoby nazwać COM++ (choć stosując ten termin, można wywołać wiele zdziwionych spojrzeń). Firma Microsoft zastosowała w środowisku WinRT system metadanych .NET. W związku z tym po skompilowaniu biblioteki DLL WinRT otrzymujemy plik *.winmd*, który zawiera zarówno metadane, jak i binarny kod wykonywalny. Format metadanych umieszczonych w tym pliku *.winmd* jest zgodny z formatem stosowanym na platformie .NET.

Podobnie jak na platformie .NET, metadane zostają dodane automatycznie podczas kompilacji zestawu.

Takie rozwiązanie ułatwia integrację środowisk .NET i WinRT. Zgodność metadanych sprawia, że firma Microsoft może bez większych trudności przekazywać wywołania między tymi dwoma platformami.

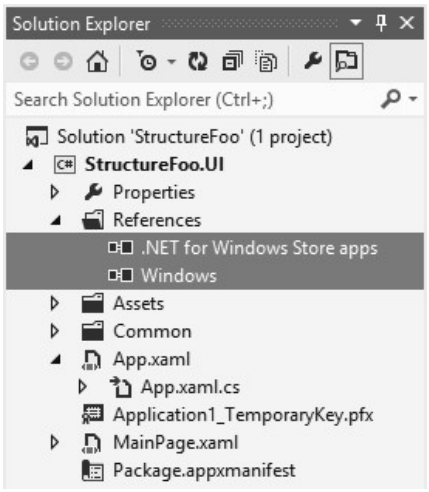
Ponieważ struktura zestawów .NET jest prawdopodobnie znana większości czytelników, przedstawimy jedynie strukturę nowych zestawów (przy użyciu narzędzia Reflector omówionego w dalszej części rozdziału) oraz pokażemy, w jaki sposób aplikacje dla Sklepu Windows odwołują się do centralnych komponentów WinRT.

Możemy zacząć od stworzenia projektu w programie Visual Studio. Rysunek 1-1 ilustruje proces dodawania do istniejącego rozwiązania nowego projektu typu Blank App (Pusta aplikacja) dla Sklepu Windows w języku C#. Jak widać, projekty związane z aplikacjami dla Sklepu Windows są umieszczone na innej gałęzi (Windows Store) niż zwykłe projekty .NET.



Rysunek 1-1. Typy nowych projektów aplikacji dla Sklepu Windows

Jeśli stworzymy nowy projekt (w tym przykładzie wybraliśmy typ Blank App), po zajrzeniu do sekcji References (Odwołania) w oknie Solution Explorer zobaczymy komponenty zaprezentowane na rysunku 1-2.



Rysunek 1-2. Zastępcze odwołania do zestawu .NETCore oraz bibliotek Windows WinRT

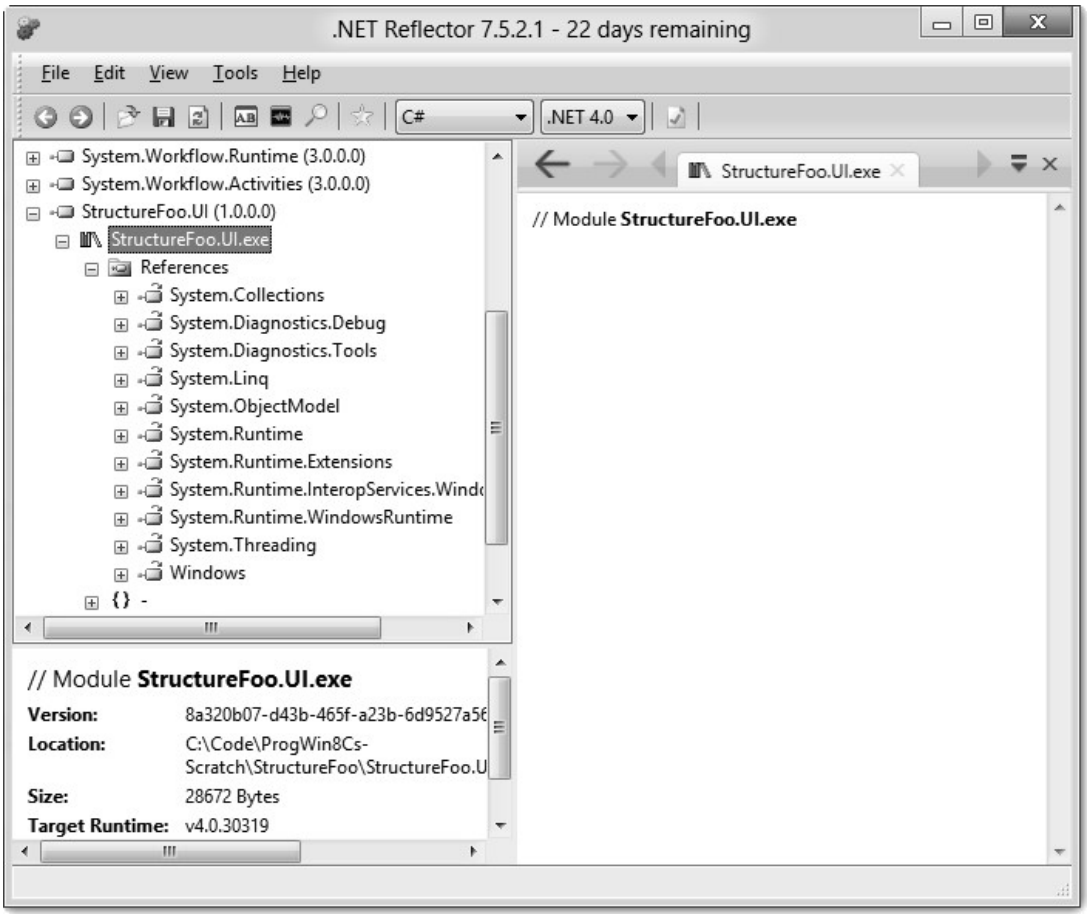
Zawartość sekcji References jest inna niż w typowym projekcie .NET. Przedstawione opcje stanowią w rzeczywistości zastępcze odwołania do odpowiednich bibliotek oraz zestawów wykorzystywanych do kompilacji projektu. A konkretnie są to odwołania do zestawu .NETCore oraz podstawowych bibliotek Windows WinRT.

Ponieważ istnieją restrykcyjne reguły określające, jakie odwołania ma zawierać prawidłowa aplikacja dla Sklepu Windows, nie należy modyfikować sekcji References.

Jak widać w oknie programu Visual Studio, zbiory odwołań do wielu zestawów/bibliotek są reprezentowane przez pojedyncze elementy. W środowisku .NET widzieliśmy poszczególne, jasno określone zestawy, natomiast w tym przypadku z pomocą przychodzi program Reflector.

Reflector to popularne narzędzie wykorzystywane na platformie .NET do sprawdzania struktury typów i elementów członkowskich w zestawie, a także do dekompilowania zestawów. Dodatkowe informacje znaleźć można na stronie *http://www.reflector.net*.

Aby móc wykonać omawianą procedurę na własnym komputerze, trzeba pobrać wersję próbną programu Reflector (oczywiście w przypadku braku pełnej licencji). Jeśli wskażemy w programie Reflector zestaw wyjściowy (w tym przykładzie noszący nazwę `StructureFoo.UI.exe`), zobaczymy elementy podobne do przedstawionych na rysunku 1-3.

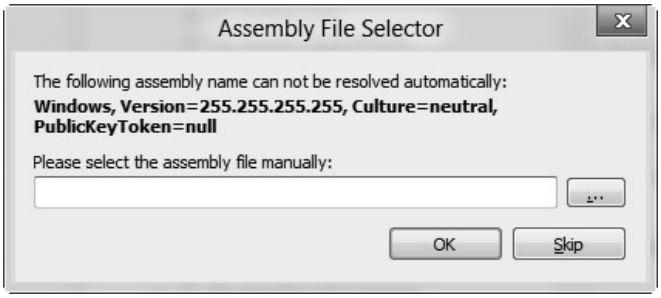


Rysunek 1-3. Rzeczywiste odwołania w zestawie wyjściowym

Przedstawiony zostanie zbiór standardowych zestawów .NET wchodzących w skład platformy .NETCore (`System.Collections` itd.) oraz obiektu `Windows`. Jak pamiętamy z rysunku 1-1, program Visual Studio wyświetlał pojedynczy element „.NET for Windows Store apps", natomiast Reflector prezentuje faktyczne odwołania.

Element `Windows` nie odwołuje się do zestawu, lecz do pliku metadanych WinRT. Plik ten zawiera informacje, które pozwalają środowisku uruchomieniowemu ustanawiać powiązania z komponentami niezarządzanego, macierzystego kodu zawierającymi implementacje. W dalszej części książki będzie można się przekonać, że tego typu wywołania WinRT (bazujące na metodzie zwanej thunking*) są często stosowane, choć proces ten zachodzi w sposób niewidoczny dla programisty.

Po zaznaczeniu elementu `Windows` w programie Reflector wyświetlone zostanie okno podobne do przedstawionego na rysunku 1-4 (wykorzystywana przez nas wersja programu Reflector nie umożliwia jeszcze automatycznego odwoływania się do metadanych). Jak widać, numer wersji to `255.255.255.255`, co sygnalizuje, że odwołujemy się do metadanych WinRT, a nie zestawu .NET.



Rysunek 1-4. Monit o wskazanie pliku metadanych Windows

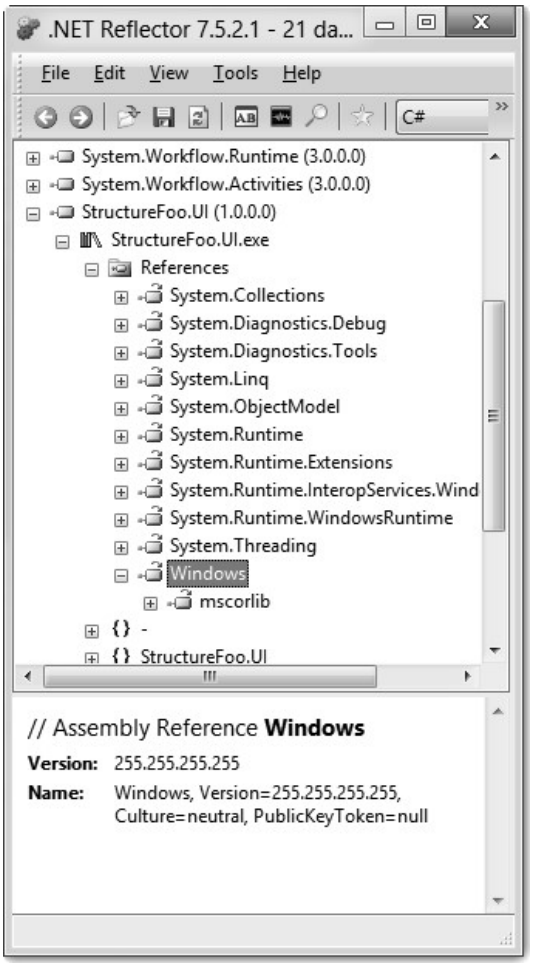
Poszukiwany plik metadanych jest przechowywany w folderze *C:\Program Files (x86)\WindowsKits \8.0\References\CommonConfiguration\Neutral\Windows.winmd*, czyli folderze zestawu narzędzi WinRT.

Po wskazaniu lokalizacji pliku `Windows.winmd` program Reflector wyświetli okno podobne do zaprezentowanego na rysunku 1-5.

Obecność elementu `mscorlib` sygnalizuje, że środowisko WinRT odwołuje się do platformy .NET. Jednak w rzeczywistości wynika ona jedynie z faktu, iż system metadanych zastosowany w środowisku WinRT został zapożyczony z platformy .NET.

Mamy nadzieję, że udało nam się zademonstrować, iż światy .NET oraz WinRT przeplatają się ze sobą. W tym celu stworzyliśmy w miarę standardowy zestaw wykonywalny .NET i odwołaliśmy się do szeregu standardowych zestawów .NET oraz bibliotek WinRT.

* *Thunking* to proces przekazywania wywołań między różnymi systemami. W tym kontekście wywołania są przekazywane ze świata „zarządzanego kodu" .NET do niezarządzanego kodu na platformie WinRT.

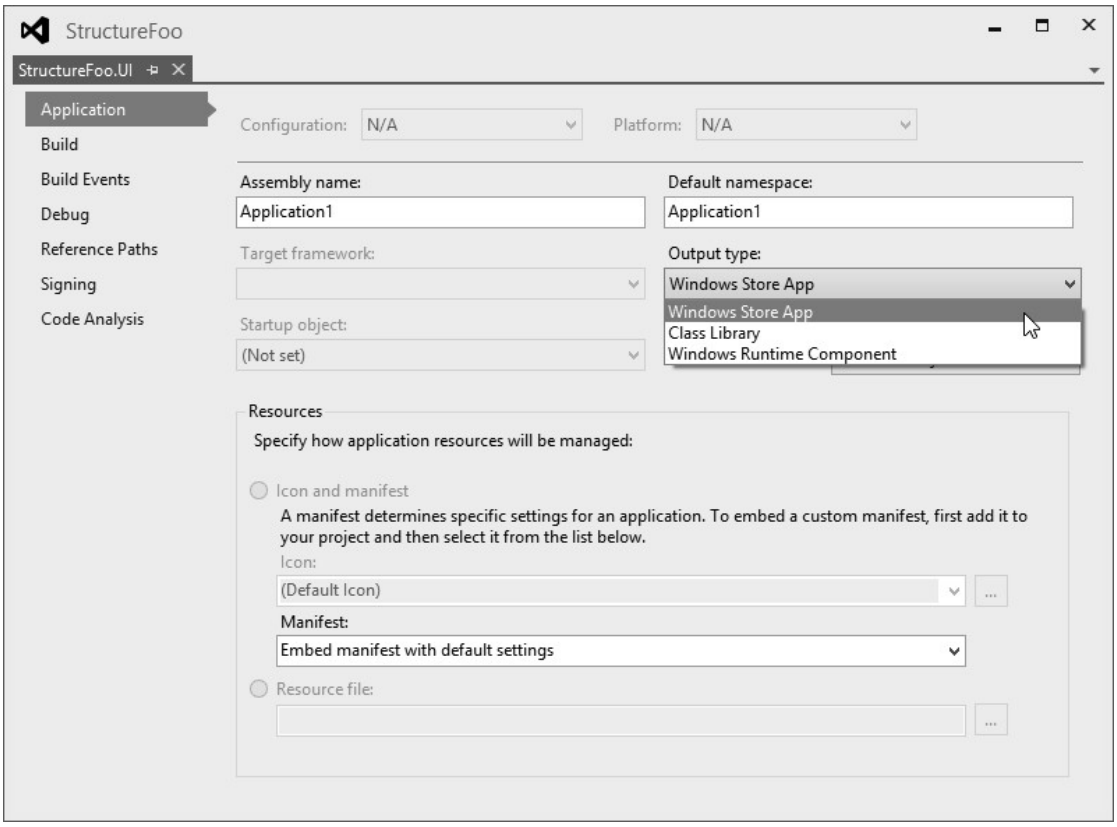


Rysunek 1-5. Pośrednie odwołanie do biblioteki WinRT

A teraz powróćmy do programu Visual Studio i przyjrzyjmy się ustawieniom projektu oraz mechanizmowi dodawania odwołań.

Ustawienia projektu i dodawanie odwołań

Co ciekawe, po otworzeniu okna właściwości projektu okazuje się, że nie możemy zmienić platformy docelowej. W normalnym projekcie .NET, *moglibyśmy* dokonać zmiany i ta opcja nie byłaby nieaktywna (ograniczenie to istnieje w momencie pisania tej książki i mamy nadzieje, że zostanie ono w przyszłości wyeliminowane i urządzenia będą wspierały wiele wersji platformy). Dostępne są różne typy wyjściowe. Jak ilustruje rysunek 1-6, można stworzyć aplikację dla Sklepu Windows (Windows Store App), bibliotekę klas (Class Library) lub komponent środowiska uruchomieniowego systemu Windows (Windows Runtime Component).



Rysunek 1-6. Nieaktywna opcja platformy docelowej oraz dostępne typy projektów

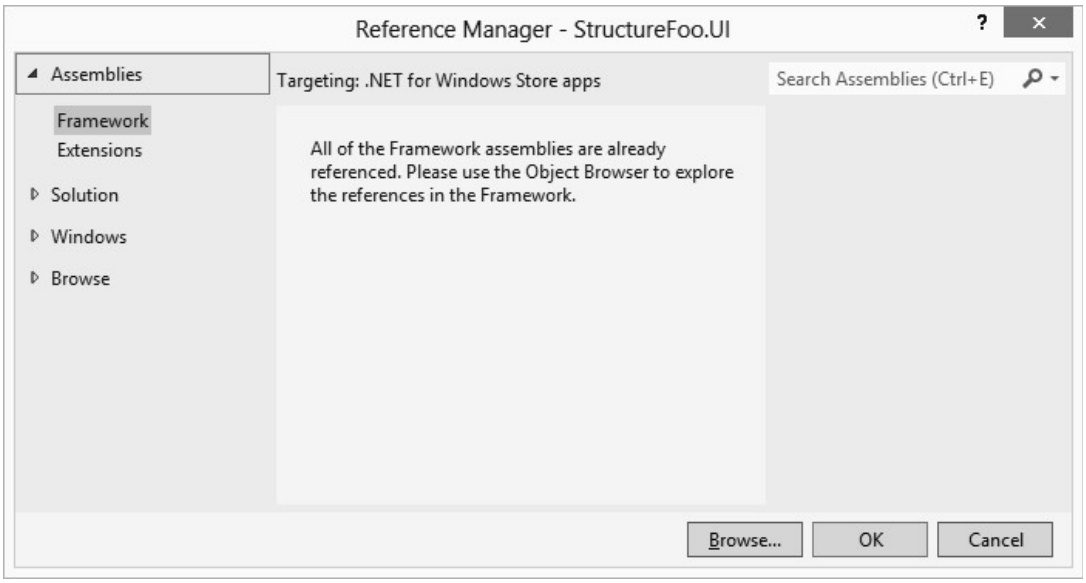
Aplikacja dla Sklepu Windows to zwykły plik wykonywalny, który może być instalowany i uruchamiany. Biblioteka klas to zestaw .NET, który może być wykorzystywany z poziomu kodu zarządzanego. Natomiast plik WinMD jest wykorzystywany w sytuacjach, w których planowane jest zastosowanie zestawu/biblioteki JavaScript lub C++/C#. Drugi przypadek wykracza poza zakres niniejszej książki, choć zostanie wspomniany w rozdziale 14 podczas omawiania zadań w tle. W prezentowanych w tej książce przykładach będziemy zwykle tworzyć jeden plik wykonywalny wraz z biblioteką pomocniczą. A teraz przyjrzyjmy się odwołaniom.

Wersja Visual Studio 2012 zawiera ulepszone okno do dodawania odwołań, które oferuje bardzo przydatną opcję wyszukiwania!

Jeśli klikniemy projekt prawym przyciskiem myszy w oknie Solution Explorer i wybierzemy opcję Add Reference (Dodaj odwołanie), zobaczymy okno przedstawione na rysunku 1-7 i zatytułowane „Targeting: .NET for Windows Store apps". Jak widać, nie możemy dodać żadnych dodatkowych odwołań do zestawów z kategorii Framework. W tego typu

projektach dostępne jest tylko i wyłącznie odwołanie do zestawu .NETCore i nie można dodawać ani usuwać odwołań bądź ich części.

Gdybyśmy wybrali gałąź Windows (czyli biblioteki WinRT), sytuacja wyglądałaby podobnie.



Rysunek 1-7. Po wybraniu wersji .NETCore nie możemy wybrać dodatkowych zestawów, jak w standardowych projektach .NET.

Sekcja Solution zawiera listę projektów wchodzących w skład rozwiązania. Aby dodać odwołanie do zestawu wyjściowego określonego projektu, wystarczy go zaznaczyć. Aczkolwiek stosując tę metodę, należy zachować ostrożność, aby nie dodać przypadkiem odwołania do projektu zwykłej biblioteki klas .NET. Ta pomyłka będzie bowiem skutkowała wyświetleniem dość enigmatycznego komunikatu o błędzie: „Unable to add a reference to project" (Nie można dodać odwołania do projektu <*NazwaProjektu*>).

Opcja Browse (Przeglądaj) pozwala na dodanie dowolnego zestawu z różnym skutkiem. Na przykład, udało nam się dodać odwołanie do zwykłego zestawu .NET 2.0, jednak później wyświetlony został komunikat o braku klas w bibliotece `mscorlib`. Co jest zrozumiałe, ponieważ .NETCore stanowi okrojoną wersję platformy.

Budowanie podstawowego interfejsu użytkownika

Po przeanalizowaniu struktury projektów aplikacji dla Sklepu Windows czas przyjrzeć się metodom projektowania interfejsu użytkownika (ang. user interface – UI).

Warianty UI

Budując interfejs użytkownika w aplikacjach dla Sklepu Windows, należy zacząć od wybrania jednego z trzech *wariantów*, które decydują o zastosowanej technologii wyświetlania oraz języku programowania.

W tej książce zdecydowaliśmy się na zastosowanie wariantu XAML, który za chwilę omówimy bardziej szczegółowo.

Wariant DirectX/C++ wykracza poza zakres niniejszej książki, ponieważ służy głównie do tworzenia gier, a nie aplikacji.

HTML

Wariant HTML5/CSS3/JavaScript jest ciekawszy i pod wieloma względami bardzo użyteczny dla programistów rozwijających aplikacje dla systemów Windows 8 oraz Windows RT. Technologia HTML5 wiąże się z budowaniem niezależnej, wykonywanej lokalnie aplikacji sieci Web, która jest uruchamiana na urządzeniu w przeglądarce IE. Jest ona spakowana jak zwykła aplikacja dla Sklepu Windows (pakowaniem aplikacji zajmiemy się w rozdziale 15). Do rozwijania aplikacji służy język JavaScript. Warto podkreślić, że tego typu aplikacje nie zawierają kodu wykonywanego po stronie serwera (jak np. w ASP.NET) – wszystkie operacje są realizowane w języku JavaScript, choć dostępna jest biblioteka WinJS, która zapewnia dostęp do pełnej biblioteki WinRT (panuje przekonanie, że właśnie ze względu na możliwość odwoływania się do biblioteki WinRT przy pomocy WinJS specjaliści z firmy Microsoft nie zdecydowali się na użycie pełnej platformy.NET do budowania aplikacji dla Sklepu Windows).

Co ciekawe, standardowe aplikacje Poczta (Mail), Kalendarz (Calendar) oraz Kontakty (People) w systemie Windows 8 bazują na technologii HTML5, a nie XAML. Jednak z naszej nieformalnej analizy Sklepu Windows dokonanej kilkakrotnie w drugiej połowie 2012 roku wynika, że język XAML jest najbardziej popularny i bazuje na nim około

70% aplikacji publikowanych w Sklepie Windows. Co więcej, na języku HTML5 bazują zazwyczaj aplikacje, które wymagają dostępu do zawartości sieci Web.

Możliwość budowania aplikacji przy użyciu języka HTML5 jest bardzo kusząca i może sprawiać wrażenie najtrafniejszej opcji. Konsolidacja rynku zachodząca w czasie pisania tej książki (w 2013 roku) utrudnia decydowanie, w jaką technologię warto zainwestować. Wybór technologii niezależnej od platformy, takiej jak HTML5, wydaje się być optymalną strategią, ponieważ końcowy rezultat można bez ogromnych trudności i kosztów przenosić na inne platformy.

Jednak użytkownicy nowych urządzeń mobilnych mają dużo wyższe wymagania dotyczące pracy z aplikacjami i oczekują perfekcyjnej interakcji. A platforma, która stanowi w pewnym sensie kompromisowe rozwiązanie (tzn. HTML oraz sieć Web), nie pozwala sprostać tym wymaganiom (wariant ten można nazwać „prawie macierzystym", ponieważ jest wystarczająco dobry, aby był użyteczny, ale nie tak dobry jak „macierzysty"). Na przykład iPad zawdzięcza swoją popularność wysokiej jakości aplikacjom, które bazują zwykle na domyślnym zestawie narzędzi dostarczanym przez firmę Apple. Choć programiści mogliby użyć bardziej uniwersalnych technologii (np. Apache Cordova, zwanej też PhoneGap), rzadko decydują się na taki krok, ponieważ doceniają ważność doskonałej interakcji z użytkownikiem.

Wykorzystanie na tabletach Windows technologii HTML5 w celu uzyskania uniwersalnego rozwiązania pociąga za sobą dodatkowy problem. Nowoczesne interfejsy mają zwykle orientację poziomą, natomiast większość aplikacji w sieci Web ma układ pionowy. Te trudności podają w wątpliwość uniwersalność wariantu HTML5.

Lepsza interakcja

Zastosowanie w aplikacjach dla Sklepu Windows technologii macierzystych pomaga w usprawnieniu interakcji z użytkownikiem. W związku z tym lepiej jest wybrać wariant XAML, który stanowi macierzystą technologię do budowania nowoczesnych interfejsów użytkownika, zamiast uniwersalnego, „prawie macierzystego" wariantu HTML 5, który stanowi w pewnym sensie kompromisowe rozwiązanie.

Warto wziąć pod uwagę dwa dodatkowe aspekty. Po pierwsze, firma Microsoft odnosi większe sukcesy w rozwijaniu macierzystych technologii UI niż technologii sieci Web. Po drugie, rozwojem aplikacji z nowoczesnym interfejsem użytkownika zajmować się będą na początku przede wszystkim programiści Silverlight i najprawdopodobniej wybiorą oni opcję XAML ze względu na znajomość tego języka. Dzięki temu istnieje większa szansa, że powstanie liczna grupa specjalistów zainteresowanych wariantem

XAML i publikujących różnego typu materiały, które ułatwią stosowanie tej technologii. Co więcej, naszym subiektywnym zdaniem, budowanie aplikacji dla Sklepu Windows w języku HTML5 jest trudniejsze niż stosowanie technologii XAML.

Aby pomóc w zrozumieniu wariantu XAML, przedstawimy pokrótce historię jego powstania.

Podczas pracy nad projektem Longhorn (który doprowadził do powstania systemu Vista) w firmie Microsoft pojawił się pomysł, aby zastąpić komponent odpowiedzialny za budowanie interfejsu użytkownika w systemie Windows. Ta nowa wizja została opatrzona nazwą Windows Presentation Foundation (WPF). Stosowany ówcześnie komponent do budowania interfejsu użytkownika w systemie Windows (GDI) był już przestarzały. Technologia WPF miała stanowić nowy aparat bazujący na modelu deklaratywnym, zamiast na modelu programistycznym (w interfejsie GDI trzeba było pisać kod służący do umieszczania elementów interfejsu użytkownika na ekranie). Model deklaratywny miał przypominać język HTML, który z oczywistych względów był bardzo popularny. W ten sposób powstał język eXtensible Application Markup Language (XAML). Komponent WPF został umieszczony na platformie .NET 3.0 i wykorzystywał bibliotekę DirectX do fizycznego rozmieszczania pikseli na ekranie.

Technologia WPF nie spotkała się z ogromnym zainteresowaniem ze strony pozostałych programistów w firmie Microsoft, głównie ze względu na fakt, iż bazowała ona na platformie .NET. Jak wspomnieliśmy wcześniej, zespoły ds. systemu Windows oraz pakietu Office nie przepadają za kodem zarządzanym, a WPF stanowi tego typu rozwiązanie. Jak można było się spodziewać, wspomniane zespoły nie zdecydowały się na zastosowanie tego rozwiązania.

Po opublikowaniu technologii WPF firma Microsoft postanowiła podjąć rywalizację z pakietem Adobe Flash i w związku z tym opublikowała nowy produkt o nazwie Silverlight. Silverlight miał być uruchamiany wewnątrz przeglądarki sieci Web (w postaci kontrolki ActiveX w przeglądarce IE). W tym czasie platforma .NET służyła przede wszystkim do rozwijania systemów biznesowych, czyli aplikacji sieci Web (ASP.NET) lub klasycznych aplikacji tworzonych przy użyciu formularzy systemu Windows (Windows Forms) bazujących na interfejsie GDI. Ze względu na początkowy brak zainteresowania ze strony zespołów w firmie Microsoft i brak rzeczywistej potrzeby zastąpienia tradycyjnych formularzy systemu Windows, technologia Silverlight pozostała typowym kanałem dostarczania dla technologii WPF.

Silverlight nigdy nie stał się właściwą konkurencją dla pakietu Flash. W czasie jego publikowania pakiet Flash tracił już na popularności, a technologia HTML5 szybko się rozwijała. Niektórzy programiści zaczęli wykorzystywać technologię Silverlight w wewnętrznych aplikacjach biznesowych instalowanych w postaci aplikacji Silverlight działających poza przeglądarką (ang. Out Of Browser – OOB). W efekcie, w czasie kiedy

firma Microsoft skłaniała się ku zaniechaniu technologii Silverlight oraz WPF, Silverlight wzmocnił swoją pozycję na rynku wewnętrznych aplikacji biznesowych.

A teraz przeanalizujmy, co działo się w międzyczasie z systemem Windows Phone. Gdy firma Microsoft porzuciła system Windows Mobile na rzecz systemu Windows Phone, potrzebowała platformy dla interfejsów użytkownika i wybrała do tego celu technologię Silverlight. A dokładniej, wybrała i jak wspomnieliśmy wcześniej okroiła technologię Silverlight. „Silverlight for Windows Phone" to jedyny efektywny zestaw narzędzi do budowania aplikacji na telefony Windows Phone (za wyjątkiem gier).

Podsumowując, macierzysta technologia WPF była praktycznie niewykorzystywana, a technologia Silverlight miała niewielki wpływ na sieć Web, była stosowana w niektórych wewnętrznych aplikacjach biznesowych i stanowiła jedyne narzędzie wspierane w systemie Windows Phone.

Dodatkowo firma Microsoft chciała „zmodernizować" sposób budowania aplikacji Windows i w związku z tym zdecydowała się na wprowadzenie platformy WinRT i uniezależnienie technologii WPF od platformy .NET. W ramach tego procesu firma Microsoft porzuciła nazwę WPF i zastąpiła ją nazwą XAML. Następnie uczyniła kod XAML *kodem niezarządzanym* tzn. macierzystą implementacją zbudowaną przy użyciu komponentów WinRT umieszczonych w bibliotekach WinRT. Zmiana nazwy wynika z tego, że w odróżnieniu od WPF kod XAML w środowisku WinRT nie jest zarządzany.

Jednak interfejsy API pozostały w dużej mierze zgodne. Na przykład, technologia WPF oferuje klasę o nazwie `System.Windows.Controls.Button`, która reprezentuje przycisk na stronie. Klasa ta zawiera właściwość `Content` służącą do ustawiania tekstu oraz zdarzenie `Click` wywoływane, gdy użytkownik kliknie przycisk. Jest to zarządzany kod umieszczony w zestawie `System.Windows.dll`. Natomiast w środowisku WinRT/XAML dostępna jest reprezentująca przycisk klasa `Windows.UI.Xaml.Controls.Button`, która stanowi niezarządzaną implementację tej samej kontrolki i również zawiera elementy członkowskie `Content` oraz `Click`, jednak w tym przypadku stanowi kod niezarządzany zaimplementowany z wykorzystaniem kontrolki WinRT powiązanej przy użyciu metadanych `Windows.winmd` (model ten nie jest doskonały, ale zasadniczo interfejsy API są ze sobą zgodne).

Podstawowe zasady analizowania kodu XAML

Rozpoczynając pracę z językiem XAML, trudno jest zrozumieć, dlaczego firma Microsoft nie zdecydowała się po prostu na użycie języka HTML. Aby zrozumieć potrzebę wprowadzenia języka XAML, trzeba mieć świadomość, co odróżnia go od języka HTML.

HTML to język znaczników definiujących dokument. Początkowy zamysł był taki, że programista posiadający pewien dokument tekstowy będzie mógł opatrzyć go specjalnymi adnotacjami w celu dostosowania jego wyglądu (np. pogrubienia czcionki) lub

działania (np. przechodzenia do innego dokumentu). Z czasem społeczność programistów korzystających z różnych technologii wykorzystała tę podstawową koncepcję do opracowania metody deklarowania interfejsu użytkownika.

XAML to odwrotna koncepcja, zgodnie z którą programista nie opisuje dokumentu, lecz interfejs użytkownika. W związku z tym, że język XAML został zaprojektowany z myślą o systemie Windows, służy w rzeczywistości do opisywania okien oraz rozmieszczonych w nich elementów interfejsu użytkownika. HTML to dokument, który wymaga przeanalizowania i zinterpretowania. Natomiast XAML to graf obiektów zdefiniowany w formacie XML.

Oto przykładowy plik XAML. Dla uproszczenia usunęliśmy z niego niektóre atrybuty, dlatego przedstawionego kodu nie można byłoby użyć w rzeczywistej aplikacji:

```
<Page>
    <StackPanel>
        <Button Content="Foo" />
        <Button Content="Bar" />
    </StackPanel>
</Page>
```

Analizator XAML działa na dość prostych zasadach. Przetwarza dokument XML Document Object Model (DOM), tworząc wystąpienia napotkanych klas. Pierwszy element po elemencie `Page` nosi nazwę `StackPanel`. Analizator XAML znajduje typ o tej nazwie (o ile ma dostęp do listy potencjalnych przestrzeni nazw, która dla uproszczenia została usunięta z tego dokumentu XML), tworzy na jego podstawie wystąpienie i dodaje je do właściwości `Content` analizowanego aktualnie elementu UI – w tym przypadku elementu `Page`. Następny element nosi nazwę `Button` i zostaje dodany do właściwości `Children` elementu `StackPanel`. W ten sposób uzyskujemy element z atrybutami, których wartości zostały skonfigurowane na podstawie właściwości znalezionych w kontrolce (w podobny sposób analizowany jest dokument ASP.NET). Następny element to drugi element `Button` itd.

Analizator HTML tworzy wewnętrzną reprezentację struktury HTML DOM w podobny sposób, jednak ponieważ dokument HTML nie musi być prawidłowym dokumentem XML, analizator XAML ma dużo łatwiejsze zadanie. Ponadto plik XAML nie zawiera wielu dodatkowych informacji umieszczanych w dokumentach HTML, takich jak skrypty, alternatywne widoki, style itp.

Ta książka nie opisuje metod budowania zachwycających wizualnie aplikacji przy użyciu języka XAML, ponieważ to wymagałoby zbyt szczegółowego omówienia procesu projektowania interfejsu użytkownika. Naszym celem jest przedstawienie metod budowania użytecznych interfejsów, w związku z tym skoncentrujemy się na przedstawieniu różnych pomocnych kontrolek oraz metod implementowania typowych komponentów.

Budowanie podstawowej strony

Rozpoczniemy od stworzenia podstawowego interfejsu użytkownika. A konkretnie zbudujemy formularz rejestracyjny, który służy do wpisywania nazwy użytkownika, adresu e-mail i hasła oraz zawiera przycisk. Efekt końcowy został przedstawiony na rysunku 1-8.

Rysunek 1-8. Docelowy układ formularza rejestracyjnego

Paski aplikacji

Warto mieć świadomość, że pozycja przycisku Register (Zarejestruj) na rysunku 1-8 jest prawdopodobnie niezgodna z wytycznymi dotyczącymi budowania interfejsów w systemie Windows 8. Prawidłową lokalizacją byłby pasek aplikacji, czyli mały panel pojawiający się w dolnej części ekranu, gdy użytkownik chce uzyskać dostęp do dodatkowych opcji (wykonując gest przesunięcia z górnej lub dolnej krawędzi ekranu bądź używając prawego przycisku myszy).

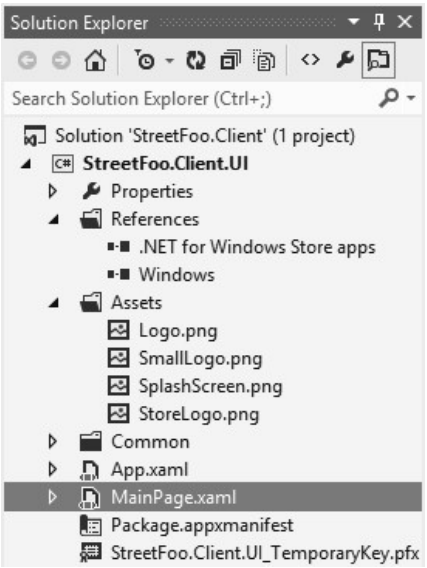
Jednak zgodnie z innymi zaleceniami, gdy akcja reprezentuje podstawowe zastosowanie formularza, można umieścić ją w głównym widoku. W wielu aplikacjach tego typu przycisk zostaje umieszczony na pasku aplikacji, ale pasek ten jest stale wyświetlony, nawet jeśli użytkownik nie podjął jeszcze żadnej akcji.

Sytuację dodatkowo komplikują inne aplikacje – np. standardowa aplikacja Poczty, w której przycisk New (Nowy) jest wyświetlony w prawym górnym rogu strony.

Budowaniem pasków aplikacji zajmiemy się dopiero w rozdziale 4, który zawiera szczegółowe omówienie problemu prawidłowego rozmieszczania i grupowania elementów. Na razie pozostawimy przycisk w głównym widoku.

W niniejszej książce będziemy wykorzystywać powszechnie dostępną usługę, którą umieściliśmy w środowisku AppHarbor. (AppHarbor – https://appharbor.com/ – to dostawca środowisk typu platforma .NET jako usługa). Usługa ta realizuje funkcję biznesową omówioną we Wprowadzeniu, czyli w uproszczeniu umożliwia użytkownikom raportowanie problemów występujących w sąsiedztwie np. zgłaszanie graffiti lub śmieci wymagających usunięcia. Wybraliśmy właśnie ten problem, ponieważ wiąże się on z użyciem wszystkich najpopularniejszych komponentów aplikacji mobilnych, takich jak: obsługa kont użytkowników, robienie i przesyłanie zdjęć, przeglądanie historycznych danych oraz sprawdzanie lokalizacji. W kolejnej części rozdziału zaimplementujemy rejestrację użytkowników. Pobierzemy dane nowego użytkownika i prześlemy je na serwer, umożliwiając użytkownikowi realizowanie takich operacji, jak zgłaszanie problemu. Nasza przykładowa aplikacja będzie nosiła nazwę StreetFoo.

Rozpoczniemy od nowego rozwiązania i projektu w programie Visual Studio. Naszym zdaniem lepiej jest rozpocząć od stworzenia pustego rozwiązania, a następnie dodawać do niego projekty, ponieważ to prowadzi do powstania na dysku bardziej logicznej i naturalnej struktury folderów. Jednak w tym przykładzie stworzymy nowe rozwiązanie Visual C#, wskazując opcję Windows Store→Blank App i nadając projektowi nazwę Street-Foo. Client.UI. Rysunek 1-9 ilustruje efekt końcowy.



Rysunek 1-9. Struktura nowoutworzonego projektu Windows Store→Blank App

Folder Assets przedstawiony na rysunku 1-9 zawiera po prostu obrazy, które są potrzebne do sterowania aplikacją (do tego tematu powrócimy za chwilę). Sekcję References omówiliśmy już we wcześniejszej części rozdziału. Sekcja Properties (Właściwości) oraz zawarty w niej plik *AssemblyInfo.cs* pełnią podobną funkcję, jak w projekcie .NET.

App.xaml łączy w sobie cechy pliku *Global.asax* stosowanego w rozwiązaniach ASP.NET oraz metody `void static main` służącej do uruchamiania aplikacji Windows Forms i plików wykonywalnych aplikacji konsoli. Automatycznie wygenerowany, standardowy kod reaguje na zdarzenie `Launched` i służy do stworzenia pierwszej wyświetlanej strony (zdarzenia związane z cyklem życia aplikacji zostaną omówione w rozdziale 14).

MainPage.xaml to domyślnie tworzona strona, którą wkrótce usuniemy i zastąpimy stroną *RegisterPage.xml*.

Plik *Package.appxmanifest* zawiera metadane opisujące aplikację i decydujące o sposobie wdrażania (instalowania) aplikacji oraz jej funkcji (działa w podobny sposób, jak manifest na platformie Android). W dalszej części książki będziemy stopniowo objaśniali różne aspekty działania tego manifestu, ale na razie można go zignorować. Podobnie jak znajdujący się w dolnej części okna plik *.pfx*, który służy do podpisywania i zostanie bardziej szczegółowo omówiony w rozdziale 15.

Domyślnie standardowy szablon projektu jest bardzo prosty i generuje jedynie aplikację, która zostanie zainstalowana i wyświetli pustą stronę. Jednak dostępny jest pewnego rodzaju „projekt podrzędny", który pozwala uzyskać dostęp do wielu dodatkowych szablonów i funkcji. Te dodatkowe funkcje umożliwiają m.in. dodawanie zbioru domyślnych stylów, definiowanie układu strony z domyślnym nagłówkiem oraz tworzenie stron, które wspierają nawet najmniejszy rozmiar okna dla tego typu aplikacji (do tego tematu powrócimy w rozdziale 12).

Aby uzyskać dostęp do tych dodatkowych funkcji, musimy usunąć plik *MainPage.xaml*, a następnie dodać do projektu nowy element typu Basic Page (Podstawowa strona) i nadać mu nazwę *RegisterPage.xaml*. Program Visual Studio wyświetli monit o dodanie brakujących plików.

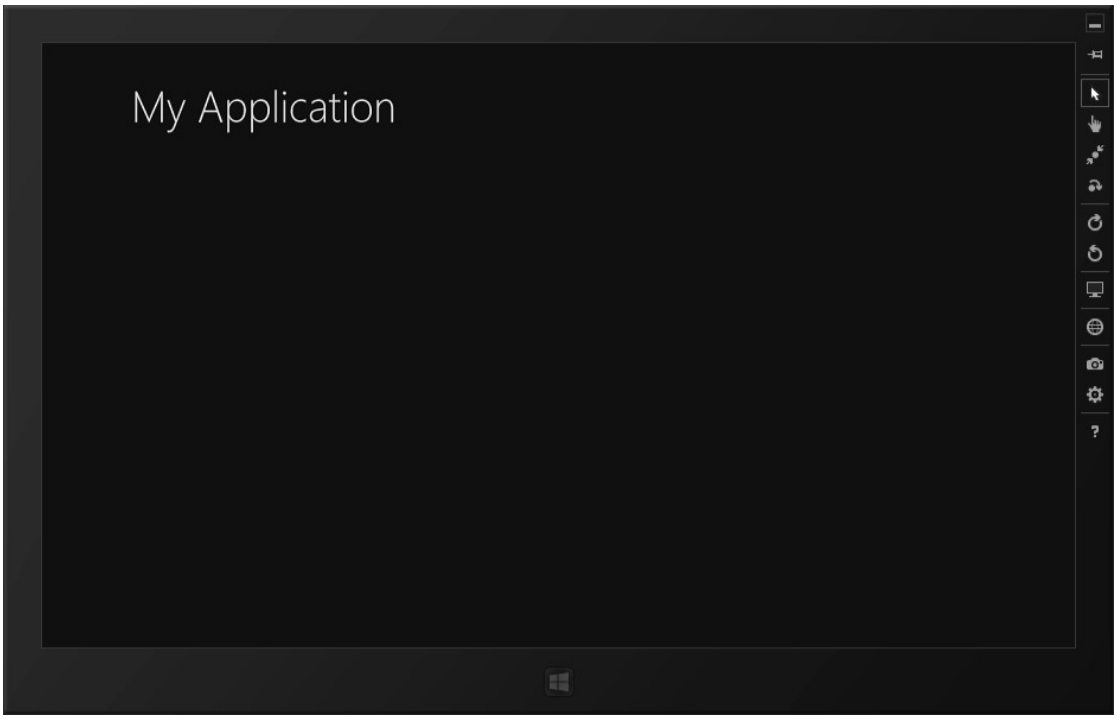
Gdy wykonamy tę operację, szablon projektu aplikacji dla Sklepu Windows doda do projektu nowy folder o nazwie *Common*, który zawierać będzie wiele nowych funkcji wspierających specjalne szablony. W folderze tym znajduje się plik *readme* z ostrzeżeniem, że programiści nie powinni modyfikować zawartości plików umieszczonych w danym folderze, ponieważ grozi to uszkodzeniem szablonu projektu Visual Studio. Nie trzeba traktować tego ostrzeżenia zbyt poważnie. Oczywiście zawsze należy przeprowadzać tego typu operacje z rozwagą, ale ryzyko poczynienia nieodwracalnych szkód jest dość niewielkie.

Jak wspomnieliśmy, kod w pliku *App.xaml* służy do uruchamiania aplikacji. Kod wygenerowany podczas tworzenia projektu zawiera odwołanie do klasy `MainPage`, która zniknęła z projektu w wyniku usunięcia pliku *MainPage.xaml*. Dlatego musimy zastąpić to odwołanie odwołaniem do klasy `RegisterPage`.

Otwieramy plik *App.xaml* i modyfikujemy metodę `OnLaunched` przy użyciu klasy `RegisterPage`:

```
          protected override void OnLaunched(LaunchActivatedEventArgs e)
           {
#if DEBUG
              if (System.Diagnostics.Debugger.IsAttached)
               {
                   this.DebugSettings.EnableFrameRateCounter = true;
              }
#endif
              Frame rootFrame = Window.Current.Content as Frame;

              // Nie powtarzaj inicjowania aplikacji, gdy w oknie znajduje się już
              // zawartość,upewnij się tylko, że okno jest aktywne

              if (rootFrame == null)
               {

                   // Utwórz ramkę, która będzie pełnić funkcję kontekstu nawigacji
                  // i przejdź do pierwszej strony

                   rootFrame = new Frame();
                  if (e.PreviousExecutionState ==
                        ApplicationExecutionState.Terminated)
                   {

                      //TODO: Wczytaj stan z wstrzymanej wcześniej aplikacji

                   }

                  // Umieść ramkę w bieżącym oknie

                   Window.Current.Content = rootFrame;
              }
              if (rootFrame.Content == null)
               {

             // Kiedy stos nawigacji nie jest przywrócony, przejdź do pierwszej strony,
             // konfigurując nową stronę przez przekazanie wymaganych informacji jako
             // parametru nawigacji

                   if (!rootFrame.Navigate(typeof(RegisterPage), e.Arguments))
                   {
                       throw new Exception("Failed to create initial page");
                   }
              }

               // Upewnij się, że bieżące okno jest aktywne

               Window.Current.Activate();
          }
```

Strona wyświetlona po uruchomieniu projektu została zaprezentowana na rysunku 1-10.



Rysunek 1-10. Podgląd nowo utworzonej strony

Kod XAML tej strony jest dość długi, więc nie będziemy go w całości prezentować. Oto fragment zasługujący na szczególną uwagę:

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="140"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

            <!-- przycisk Wstecz i strona tytułowa -->

    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="120"/>
            <ColumnDefinition Width="*"/>
        </Grid.ColumnDefinitions>
        <AppBarButton x:Name="backButton" Icon="Back" Height="95"
                        Margin="10,46,10,0"
                      Command="{Binding NavigationHelper.GoBackCommand,
                      ElementName=pageRoot}"
                      Visibility="{Binding IsEnabled, Converter=
                       {StaticResource BooleanToVisibilityConverter},
                        RelativeSource={RelativeSource Mode=Self}}"
                      AutomationProperties.Name="Back"
                      AutomationProperties.AutomationId="BackButton"
                      AutomationProperties.ItemType="Navigation Button"/>
         <TextBlock x:Name="pageTitle" Text="{StaticResource AppName}"
```

```
                    Style="{StaticResource HeaderTextBlockStyle}"
                  Grid.Column="1"
                IsHitTestVisible="false" TextWrapping="NoWrap"
                VerticalAlignment="Bottom" Margin="0,0,30,40"/>
        </Grid>

     <!-- pominięto dla uproszczenia -->

    </Grid>
```

Na podstawie tego fragmentu można zauważyć różnicę między językiem XAML a HTML. Dawniej do definiowania układu w dokumencie HTML użylibyśmy znacznika TABLE (oczywiście obecnie cel ten realizuje się przy użyciu elementów DIV powiązanych z deklaracjami CSS), jednak plik XAML nie zawiera żadnych pomocniczych elementów TR czy TD. Zamiast tego zawiera zagnieżdżone kontrolki `Grid` (nazywane siatkami). Dodatkowo zastosowane zostały tajemnicze definicje `RowDefinition` określające układ wierszy oraz definicje `ColumnDefinition` określające układ kolumn. Jak łatwo zauważyć, możemy umieścić kontrolkę w komórce siatki, wykorzystując do tego celu atrybut `Grid.Column` tej kontrolki. To rozwiązanie zupełnie nie przypomina metod HTML i CSS.

Zrozumienie języka XAML

Język XAML doskonale sprawdza się w sytuacjach, gdy potrzebujemy interfejsu użytkownika o bardzo precyzyjnym układzie (podobnie jak w bardzo starych technologiach, takich jak m.in. bezpośrednie wywołania Win16/Win32, MFC czy VB), ale jednocześnie chcemy zachować pewną swobodę (np. łatwiejszą obsługę widoków o zmiennym rozmiarze) oraz nadal czerpać korzyści wynikające z użycia deklaratywnego interfejsu użytkownika. Podobna koncepcja została zrealizowana w technologii Windows Forms poprzez zastosowanie standardowych właściwości `Dock` oraz `Anchor` kontrolki. Język XAML stanowi ewolucję tego podejścia, zamiast podążać za nowymi trendami w technologii HTML. Język XAML bazuje na oknach, natomiast język HTML nadal bazuje na dokumentach. Co więcej, język XAML jest dużo bardziej precyzyjny. W definicji pierwszego wiersza określamy dokładną wysokość w pikselach. Nie jest to wartość minimalna czy wartość modyfikowana w zależności od innych elementów, jak w przypadku kodu HTML i CSS. Jeśli ustawimy „140” pikseli, otrzymamy 140 pikseli.

Zanim zajmiemy się tworzeniem formularza, zmieńmy nagłówek. Na razie nagłówek jest zdefiniowany w kontrolce `TextBlock`, a wartość atrybutu `Text` odwołuje się do rozszerzenia `StaticResource`.

Oto fragment kodu XAML wygenerowanego w momencie tworzenia strony *RegisterPage.xaml*:

```
<!-- przycisk Wstecz i strona tytułowa -->

        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="120"/>
                <ColumnDefinition Width="*"/>
            </Grid.ColumnDefinitions>
            <AppBarButton x:Name="backButton" Icon="Back" Height="95"
                          Margin="10,46,10,0"
                        Command="{Binding NavigationHelper.GoBackCommand,
                        ElementName=pageRoot}"
                        Visibility="{Binding IsEnabled, Converter=
                         {StaticResource BooleanToVisibilityConverter},
                          RelativeSource={RelativeSource Mode=Self}}"
                        AutomationProperties.Name="Back"
                        AutomationProperties.AutomationId="BackButton"
                        AutomationProperties.ItemType="Navigation Button"/>
             <TextBlock x:Name="pageTitle" Text="{StaticResource AppName}"
                         Style="{StaticResource HeaderTextBlockStyle}"
                       Grid.Column="1"
                       IsHitTestVisible="false" TextWrapping="NoWrap"
                       VerticalAlignment="Bottom" Margin="0,0,30,40"/>
         </Grid>
```

Gdy wartość atrybutu jest umieszczona w nawiasie klamrowym, analizator XAML wie, że standardowe przetwarzanie musi zostać zatrzymane, aby mogła zostać wykonana jakaś inna operacja. Tego typu konstrukcje są nazywane *rozszerzeniami znaczników*. `StaticResource` to jeden z wielu typów rozszerzeń. W dalszej części rozdziału omówimy innego typu rozszerzenia: `Binding` (tego typu rozszerzenie zostało zastosowane w kontrolce `Button` i służy do pokazywania lub ukrywania domyślnego przycisku Wstecz w zależności od tego, czy bieżąca strona jest pierwszą stroną). Na razie zastąpimy po prostu wartość atrybutu `Text` ciągiem „Register” (Zarejestruj) w następujący sposób:

```
        <!-- przycisk Wstecz i strona tytułowa -->

        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="120"/>
                <ColumnDefinition Width="*"/>
            </Grid.ColumnDefinitions>
            <AppBarButton x:Name="backButton" Icon="Back" Height="95"
                          Margin="10,46,10,0"
                        Command="{Binding NavigationHelper.GoBackCommand,
                        ElementName=pageRoot}"
                        Visibility="{Binding IsEnabled, Converter=
                         {StaticResource BooleanToVisibilityConverter},
```

```
                        RelativeSource={RelativeSource Mode=Self}}"
                      AutomationProperties.Name="Back"
                      AutomationProperties.AutomationId="BackButton"
                      AutomationProperties.ItemType="Navigation Button"/>
            <TextBlock x:Name="pageTitle" Text="Register" Style="{StaticResource
                       HeaderTextBlockStyle}" Grid.Column="1"
                      IsHitTestVisible="false" TextWrapping="NoWrap"
                      VerticalAlignment="Bottom" Margin="0,0,30,40"/>
        </Grid>
```

Dokonana zmiana zostanie od razu odzwierciedlona na wyświetlonym podglądzie.

Następnie zmienimy strukturę strony tak, aby uzyskać pojedynczą kontrolkę `Grid` zajmującą całą stronę. Domyślny układ został zaprojektowany z myślą o widokach przewijanych w poziomie. Nasz formularz ma jedynie zawierać miejsce pod tytułem, w którym można wpisać określone wartości. W związku z tym zdefiniujemy siatkę z wierszem u góry o wysokości 140 pikseli oraz kolumną po lewej stronie o szerokości 120 pikseli. Możemy umieścić przycisk Wstecz w lewej górnej komórce, etykietę w prawym górnym rogu, a formularz w prawej dolnej części (potrzebujemy przycisku Wstecz, ponieważ w przyszłości aplikacja będzie na początku wyświetlała stronę logowania, a przycisk Wstecz będzie służył do przechodzenia do strony rejestracji).

Teraz możemy już użyć kontrolki `StackPanel`, która bardzo pomaga w definiowaniu pionowego (domyślnego) lub poziomego układu kontrolek. Wykorzystamy również kontrolki `TextBlock` (pełniące rolę etykiet), `TextBox`, `PasswordBox` oraz `Button`. Oto zmodyfikowany kod formularza. Rysunek 1-11 ilustruje pożądany efekt końcowy.

```
<local:StreetFooPage
     x:Name="pageRoot"
    x:Class="StreetFoo.Client.UI.RegisterPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:StreetFoo.Client.UI"
    xmlns:common="using:StreetFoo.Client.UI.Common"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
     mc:Ignorable="d">
  <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="140"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="120"/>
            <ColumnDefinition Width="*"/>
        </Grid.ColumnDefinitions>

        <!-- przycisk Wstecz i strona tytułowa -->
```

```
        <AppBarButton x:Name="backButton" Icon="Back" Height="95"
                        Margin="10,46,10,0"
                      Command="{Binding NavigationHelper.GoBackCommand,
                      ElementName=pageRoot}"
                      Visibility="{Binding IsEnabled, Converter={StaticResource
                        BooleanToVisibilityConverter}, RelativeSource=
                       {RelativeSource Mode=Self}}"
                       AutomationProperties.Name="Back"
                      AutomationProperties.AutomationId="BackButton"
                      AutomationProperties.ItemType="Navigation Button"/>
         <TextBlock x:Name="pageTitle" Text="Register" Style="{StaticResource
                     HeaderTextBlockStyle}" Grid.Column="1"
                    IsHitTestVisible="false" TextWrapping="NoWrap"
                    VerticalAlignment="Bottom" Margin="0,0,30,40"/>

<! – Formularz rejestracyjny-->

        <StackPanel Grid.Row="1" Grid.Column="1">
            <TextBlock Text="Username"></TextBlock>
            <TextBox HorizontalAlignment="Left" Width="400" Text="{Binding
               Username, Mode=TwoWay}"/>
            <TextBlock Text="Email"></TextBlock>
            <TextBox HorizontalAlignment="Left" Width="400" Text="{Binding
               Email, Mode=TwoWay}"/>
            <TextBlock Text="Password"></TextBlock>
            <PasswordBox HorizontalAlignment="Left" Width="400" Password="
              {Binding Password, Mode=TwoWay}"/>
            <TextBlock Text="Confirm password"></TextBlock>
            <PasswordBox HorizontalAlignment="Left" Width="400" Password="
              {Binding Confirm, Mode=TwoWay}"/>
            <Button Content="Register"></Button>
        </StackPanel>

     <!-- element VisualStateManager został dla uproszczenia usunięty -->

</Grid>
</local:StreetFooPage>
```

Warto zwrócić uwagę na dwa aspekty związane z zaprezentowanym fragmentem kodu. Do określenia wartości właściwości `Text` oraz `Password` kontrolek edycji wykorzystaliśmy rozszerzenia `Binding`. Szczególnie istotna jest specyfikacja `Mode=TwoWay`, która sprawia, że zmienione dane zostają przekazane z interfejsu użytkownika do modelu widoku (do tego zagadnienia jeszcze powrócimy). Później wytłumaczymy także, w jaki sposób wykorzystać polecenia do wykonywania określonych operacji po kliknięciu przycisku.

Po uruchomieniu projektu wyświetlona zostanie strona rejestracji, jak widać na rysunku 1-11.

Struktura strony jest już gotowa, teraz trzeba powiązać ją z pewnymi operacjami.

Rysunek 1-11. Formularz rejestracji

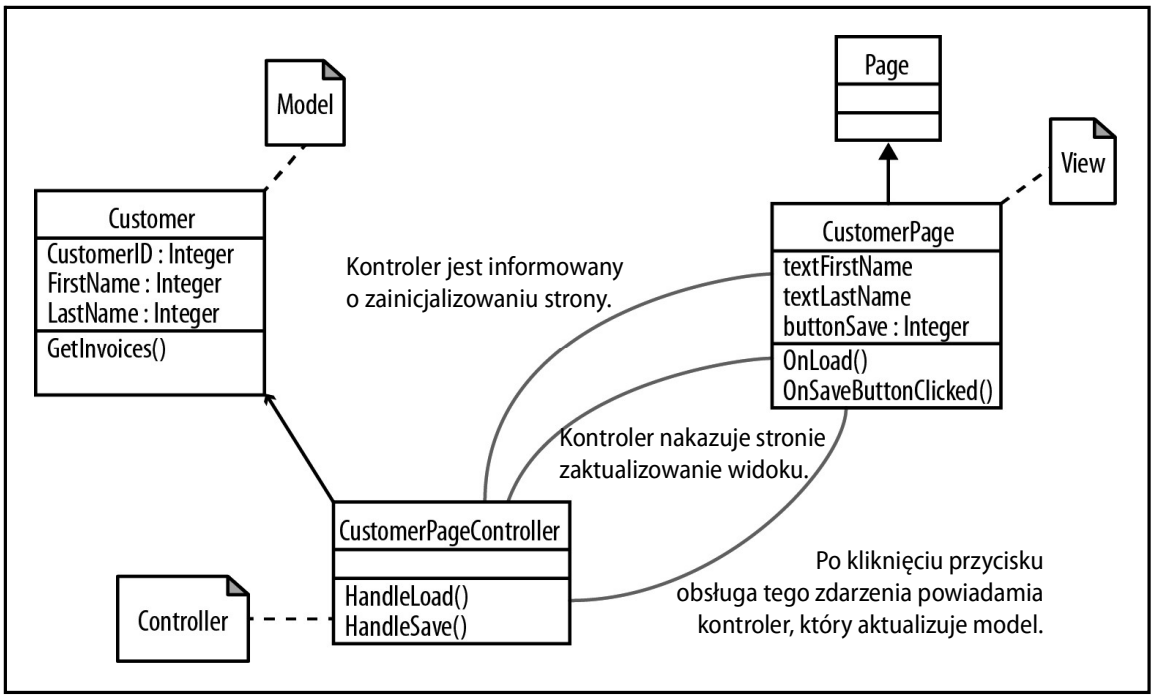
Implementowanie wzorca MVVM

W tej książce będziemy implementować interfejs użytkownika przy użyciu wzorca Model-Widok-Model widoku (Model/View/View-Model – MVVM). Wzorzec MVVM ma na celu oddzielenie danych przetwarzanych przez aplikację od widoku prezentowanego użytkownikowi.

Najprostsza metoda budowania interfejsu użytkownika zakłada brak jakiegokolwiek podziału. Załóżmy, że mamy bazę danych z informacjami o klientach oraz formularz umożliwiający edytowanie danych wybranego klienta. Brak separacji oznacza, że moglibyśmy zbudować jedną wielką klasę, w której kod służący do pobierania danych z bazy danych przeplata się z kodem służącym do wyświetlania tych danych na ekranie, do reagowania na kliknięcie przycisku „Zapisz", do sprawdzania poprawności danych oraz do zapisywania zmian w bazie danych. Tego typu podejście polegające na wymieszaniu fragmentów kodu służących do realizowania różnych operacji pociąga za sobą szereg problemów. Przede wszystkim utrudnia ponowne wykorzystywanie funkcji służących do obsługi bazy danych oraz sprawdzania poprawności, a ponadto nie pomaga programiście w pisaniu bezpośrednich testów jednostkowych. Stworzenie osobnych klas odpowiedzialnych za przesyłanie danych z i do bazy danych oraz za sterowanie interfejsem użytkownika znacznie ułatwia wielokrotne wykorzystywanie kodu i wykonywanie testów jednostkowych.

Jeszcze raz podkreślamy – brak podziału funkcjonalności w kodzie to prawie zawsze fatalny pomysł.

Najbardziej znany wzorzec architektury, w którym funkcje przechowywania danych są oddzielone od interfejsu użytkownika, nosi nazwę Model-Widok-Kontroler (Model--View-Controller, w skrócie MVC). Wzorzec MVC składa się z komponentu trwałych danych (*modelu*), fizycznego *widoku* oraz *kontrolera*, który pośredniczy pomiędzy modelem i widokiem (np. może zainicjować akcję „po kliknięciu tego przycisku należy pobrać dane wejściowe z interfejsu użytkownika, uruchomić procedurę sprawdzania poprawności, a następnie uaktualnić model"). Schemat wzorca MVC ilustruje rysunek 1-12. Przedstawiliśmy uproszczony diagram – zwykle między kontrolerem a klasą strony istnieją interfejsy, które oddzielają implementację strony.



Rysunek 1-12. Schemat wzorca MVC

Wcześniej wspomnieliśmy, że model „wie", jak przesyłać dane z i do bazy danych. W rzeczywistości model stanowi przechowywaną w pamięci reprezentację trwałych danych. W (anachronicznej) architekturze skoncentrowanej na danych model wiedziałby, jak sterować bazą danych. W (nowocześniejszej) architekturze zorientowanej obiektowo model, jak sama nazwa wskazuje, stanowi zestaw klas modelujących domenę charakterystyczną dla danej aplikacji. Na przykład, do modelowania klienta służy klasa o nazwie

`Customer`. Jeśli klient złożył zamówienia, klasa `Customer` będzie zawierała metodę, która zwraca kolekcję obiektów `Order` (Zamówienie).

WPF a Silverlight

Firma Microsoft zaleca, aby w rozwiązaniach WPF, Silverlight oraz XAML stosować architekturę MVVM. Co więcej, społeczność programistów aplikacji dla Sklepu Windows prawie zawsze przestrzega tego zalecenia i my też jesteśmy jego zwolennikami.

Model jest taki sam, jak we wzorcu MVC, czyli stanowi reprezentację przetwarzanych danych. *Widok* także pozostał niezmieniony i reprezentuje wyświetlane elementy interfejsu użytkownika. Nowość stanowi komponent *model widoku*. Łączy on funkcjonalność kontrolera oraz wyspecjalizowanej fasady dla modelu. W naszym prostym przykładzie składającym się z jednej strony służącej do edycji danych klienta nasza wyspecjalizowana fasada stanowić będzie mapowanie jeden do jednego z wystąpieniem klasy `Customer`, jednak zdarzają się bardziej skomplikowane rozwiązania. Czasami trzeba opracować zupełnie nowy zestaw klas reprezentujących dane widoku, grupując i agregując różne dane pobrane z modelu. Jednak dane modelu widoku wcale nie muszą reprezentować trwałych danych aplikacji. Ilustruje to nasz pierwszy przykład, w którym pobieramy dane nowego klienta, którego konto nie istnieje jeszcze w lokalnym zestawie trwałych danych.

Wprowadzenie architektury MVVM w rozwiązaniach WPF/Silverlight służy przede wszystkim do wyeliminowania jednej z największych zmór programowania interfejsów użytkownika, a mianowicie konieczności pobierania danych z modelu i umieszczania ich w kontrolkach na ekranie. Budowanie formularzy wiązało się niegdyś z koniecznością wielokrotnego powtarzania kodu typu `textFirstName.Content = Customer.FirstName` oraz `Customer.FirstName = textFirstName.Content`. Na szczęście w rozwiązaniach WPF/Silverlight, a co ważniejsze w implementacjach XAML/WinRT, platforma może przejąć ten obowiązek, wystarczy zbudować odpowiednie powiązanie, a platforma zajmie się realizowaniem tej żmudnej operacji. Dzięki przypisaniu wartości `{Binding Customer.FirstName}` właściwości `Content` kontrolki `TextBox` w kodzie XAML, nie musimy pisać kodu służącego do przekazywania danych tam i z powrotem z modelu widoku. Ten proces jest nazywany *wiązaniem danych* i był dostępny już w pierwszych wersjach języka Visual Basic. Dzięki niemu, wystarczy określić w elementach UI, skąd mają pochodzić wybrane dane, a platforma zajmie się całą resztą.

A teraz przyjrzymy się drugiej korzyści płynącej z podziału funkcjonalności, a mianowicie możliwości wykonywania testów jednostkowych.