

Odnieś sukces na platformie Windows 8!

# Windows<sup>®</sup> 8

## Tworzenie aplikacji z użyciem **C#** i **XAML**



Jeremy Likness

**Wintellect**  
Know how.

Tytuł oryginału: Building Windows 8 Apps with C# and XAML

Tłumaczenie: Paweł Gonera

ISBN: 978-83-246-7062-8

Authorized translation from the English language edition, entitled:  
BUILDING WINDOWS 8 APPS WITH C# AND XAML; ISBN 0321822161;  
by Jeremy Likness; published by Pearson Education, Inc, publishing as Addison Wesley.  
Copyright © 2013 Pearson Education, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

Polish language edition published by HELION S.A. Copyright © 2013.

The .NET logo is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries and is used under license from Microsoft.  
Microsoft, Windows, Visual Basic, Visual C#, and Visual C++ are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and/or other countries/regions.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION  
ul. Kościuszki 1c, 44-100 GLIWICE  
tel. 32 231 22 19, 32 230 98 63  
e-mail: [helion@helion.pl](mailto:helion@helion.pl)  
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/w8twap>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)



# Spis treści

---

**Słowo wstępne 9**

**Wprowadzenie 11**

**Podziękowania 15**

**O autorze 17**

## **1 Nowe środowisko uruchomieniowe Windows 19**

Spojrzenie wstecz — Win32 oraz .NET 19

Spojrzenie w przód — rozwój NUI 24

Wprowadzenie do aplikacji Windows Store 26

*Projekt Windows 8 28*

*Szybkość i płynność 28*

*Przyciąganie i skalowanie 28*

*Użycie właściwego kontraktu 28*

*Duże kafelki 29*

*Żywy i podłączony 30*

*Zgodność z zasadami projektowania Windows 8 31*

Niezbędne narzędzia dla Windows 8 31

*Blend dla Visual Studio 32*

*HTML5 wraz z JavaScript 32*

*C++ wraz z XAML 34*

*VB/C# oraz XAML 35*

Wewnątrz WinRT 35

WPF, Silverlight oraz niebieski stos 36

Podsumowanie 37

*Cytowane prace 37*

## **2 Zaczynamy 39**

Konfigurowanie środowiska 39

*Windows 8 39*

*Visual Studio 2012 43*

*Blend 44*

## 6 ■ Spis treści

Witaj, Windows 8	44
<i>Tworzenie pierwszej aplikacji Windows 8</i>	44
<i>Szablony</i>	44
Aplikacja ImageHelper	47
<i>Pod maską</i>	56
Podsumowanie	60

## 3 Język XAML (Extensible Application Markup Language) 61

Deklarowanie interfejsu użytkownika	61
<i>Drzewo wizualne</i>	63
<i>Właściwości zależne</i>	65
<i>Właściwości dołączane</i>	67
Wiązanie danych	69
<i>Konwertery wartości</i>	72
Serie ujęć	74
Style i zasoby	76
Układ	78
<i>Element Canvas</i>	78
<i>Element Grid</i>	79
<i>Element StackPanel</i>	81
<i>Elementy VirtualizingPanel oraz VirtualizingStackPanel</i>	81
<i>Element WrapGrid</i>	82
<i>Element VariableSizedWrapGrid</i>	83
<i>Element ContentControl</i>	85
<i>Element ItemsControl</i>	86
<i>Element ScrollViewer</i>	86
<i>Element ViewBox</i>	86
<i>Element GridView</i>	88
<i>Element ListView</i>	91
<i>Element FlipView</i>	91
<i>Element ListBox</i>	92
Wspólne kontrolki	92
Podsumowanie	92

## 4 Aplikacje Windows 8 95

Układy i widoki	95
<i>Symulator</i>	95
<i>Visual State Manager</i>	98
<i>Semantyczne powiększanie</i>	101
Obsługa zdarzeń użytkownika	103
<i>Zdarzenia wskaźników</i>	103
<i>Zdarzenia manipulacji</i>	105
<i>Obsługa myszy</i>	106
<i>Obsługa klawiatury</i>	107
<i>Efekty wizualne</i>	108
<i>Celowanie</i>	110
<i>Menu kontekstowe</i>	111
Pasek aplikacji	112

Ikony i ekrany powitalne	117
Strona informacyjna	117
Czujniki	119
Przyspieszeniometerz	120
Kompas	121
Geolokalizacja	121
Żyroskop	122
Inklinometr	122
Czujnik światła	123
Czujnik orientacji	123
Podsumowanie	124

## 5 Cykl życia aplikacji 127

Zarządzanie czasem życia procesu	129
Aktywacja	130
Wstrzymanie	131
Zakończenie działania	133
Wznowienie	133
Nawigacja	134
API Application Data	137
Żywy i podłączony	140
Własny ekran startowy	140
Podsumowanie	141

## 6 Dane 143

Ustawienia aplikacji	143
Odczyt i zapis danych	144
Potrzeba szybkości i wielowątkowości	148
Użycie <i>async</i> i <i>await</i>	150
Wyrażenia <i>lambda</i>	152
Metody pomocnicze IO	152
Osadzone zasoby	153
Kolekcje	155
Zapytania zintegrowane z językiem (LINQ)	156
Zawartość WWW	157
Udostępniana zawartość	159
Strumienie, bufory i tablice bajtów	160
Kompresja danych	160
Szyfrowanie i podpisywanie danych	162
Usługi sieciowe	164
Obsługa OData	167
Podsumowanie	168

## 7 Kafelki i powiadomienia 169

Proste kafelki	169
Kafelki aktywne	170
Wskaźniki	174
Kafelki podrzędne	176

## 8 ■ Spis treści

Powiadomienia wyskakujące	179
Usługa Windows Notification Service	183
Podsumowanie	189

## 8 Tworzenie paneli w aplikacji 191

Wyszukiwanie	192
Udostępnianie	200
<i>Źródło treści do udostępniania</i>	200
<i>Pobieranie treści przy pracy jako cel udostępniania</i>	205
Ustawienia	209
Podsumowanie	212

## 9 MVVM i testowanie 213

Wzorce projektowania interfejsu użytkownika	214
<i>Model</i>	217
<i>Widok</i>	218
<i>Model widoku</i>	219
Przenośna biblioteka klas	220
Dlaczego testujemy?	223
<i>Testowanie eliminuje założenia</i>	223
<i>Testowanie usuwa błędy u źródła</i>	224
<i>Testowanie pomaga dokumentować kod</i>	224
<i>Testowanie ułatwia rozszerzanie i utrzymywanie aplikacji</i>	225
<i>Testowanie poprawia architekturę i projekt</i>	225
<i>Testowanie pozwala rozwijać się programistom</i>	226
<i>Konkluzja — pisz testy jednostkowe!</i>	226
Testy jednostkowe	226
<i>Platforma testów jednostkowych Windows Store</i>	227
<i>Imitacje i zręby</i>	230
Podsumowanie	232

## 10 Pakiety i instalacja 233

Sklep Windows	233
<i>Wyszukiwanie</i>	233
<i>Zasięg</i>	235
<i>Modele biznesowe</i>	237
<i>Reklamy</i>	240
<i>Przygotowanie aplikacji do sklepu</i>	241
<i>Proces</i>	242
<i>Użycie App Certification Kit</i>	242
<i>Czego możesz się spodziewać?</i>	245
Ładowanie boczne	245
Podsumowanie	247

## Skorowidz 249

# 5

## Cykl życia aplikacji

---

**W** TRADYCYJNYM ŚRODOWISKU WINDOWS użytkownik zarządza czasem życia aplikacji — uruchamia ją, i ona działa do momentu, gdy użytkownik zdecyduje o jej zamknięciu. W modelu tym występuje poważny problem, ponieważ aplikacja wykorzystuje takie zasoby systemu, jak pamięć i CPU, nawet jeżeli nie znajduje się ona na pierwszym planie. Wpływa to na wydajność aplikacji uruchamianych przez użytkownika, a także powoduje szybsze wyczerpanie baterii, jeżeli urządzenie nie jest podłączone do sieci elektrycznej.

Bardzo łatwo jest pokazać to tradycyjne podejście w trybie pulpitu Windows 8. Jeżeli nie posiadasz pliku wideo, skorzystaj z poniższej witryny, udostępniającej wszystkie wideo z sesji konferencji BUILD, na której Microsoft zaprezentował Windows 8: <http://channel9.msdn.com/Events/BUILD/BUILD2011>.

(Być może czytasz to po konferencji z roku 2012; jeżeli tak, to powinieneś być w stanie skorzystać ze zaktualizowanej witryny i pobrać z niej aktualne nagrania).

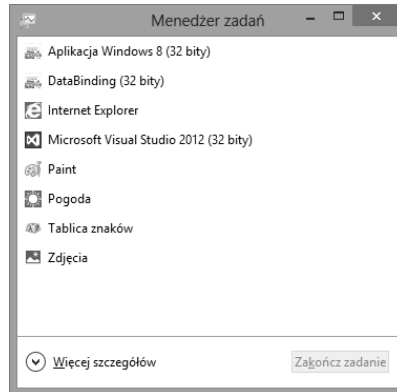
Przejdź do wybranego filmu i pobierz go w średniej lub wysokiej jakości na swój dysk. Może to zająć od kilku minut do godziny, w zależności od rozmiaru pliku. W czasie oczekiwania na pobranie pliku uruchom Menedżer zadań, który pozwala na przegląd aplikacji działających w systemie. Istnieje kilka sposobów na jego uruchomienie.

- W menu *Start* Windows 8 wpisz *menedżer zadań* i wybierz program z wyświetlonych wyników wyszukiwania.
- Kliknij na pulpicie prawym przyciskiem myszy pasek zadań i wybierz opcję *Menedżer zadań*.
- W dowolnym miejscu naciśnij jednocześnie *Ctrl+Shift+Esc*.

Po uruchomieniu Menedżera zadań wyświetli się prosty widok pokazany na rysunku 5.1.

Domyślny widok zawiera listę działających aplikacji. Aby zobaczyć bardziej zaawansowany widok, kliknij łącze *Więcej szczegółów* w lewej dolnej części ekranu. Spowoduje to otwarcie widoku zaawansowanego przedstawionego na rysunku 5.2. Zostaną pokazane nie tylko działające aplikacje z informacjami na temat zużycia zasobów, ale również działające procesy tła, takie jak sterowniki w systemie, które obsługują różne urządzenia.

Upewnij się, że Menedżer zadań działa w widoku zaawansowanym i że jest widoczny na ekranie. Teraz uruchom pobrany plik wideo przez kliknięcie prawym przyciskiem myszy w oknie Eksploratora Windows i wybranie *Odtwarzanie za pomocą programu Windows Media Player*. Możesz zauważyć, że zużycie procesora oraz pamięci znacznie rosną. Wielkość tych wzrostów zależy od konfiguracji systemu. Teraz otwórz Notatnik i rozciągnij go, aby całkowicie zakrył film. Po tej operacji zużycie procesora nie zmieni



Rysunek 5.1. Prosty widok Menedżera zadań

The screenshot shows the 'Menedżer zadań' window in an advanced view. It features a menu bar (Plik, Opcje, Widok) and a tabbed interface with tabs for 'Procesy', 'Wydajność', 'Historia aplikacji', 'Uruchamianie', 'Użytkownicy', 'Szczegóły', and 'Usługi'. The 'Procesy' tab is active, displaying a table of running processes with columns for Name, Status, CPU usage, Memory usage, Disk usage, and Network usage.

Nazwa	Stan	10%	48%	0%	0%
		Procesor...	Pamięć	Dysk	Sieć
<b>Aplikacje (9)</b>					
Aplikacja Windows 8 (32 bity)		0%	15,5 MB	0 MB/s	0 Mb/s
DataBinding (32 bity)		0%	7,5 MB	0 MB/s	0 Mb/s
Internet Explorer		0%	106,6 MB	0 MB/s	0 Mb/s
▶ Menedżer zadań		0%	7,2 MB	0 MB/s	0 Mb/s
▶ Microsoft Visual Studio 2012 (32...		0%	162,6 MB	0 MB/s	0 Mb/s
▶ Paint		0%	11,3 MB	0 MB/s	0 Mb/s
Pogoda		0%	56,7 MB	0 MB/s	0 Mb/s
▶ Tablica znaków		0%	1,6 MB	0 MB/s	0 Mb/s
Zdjęcia		0%	35,1 MB	0 MB/s	0 Mb/s
<b>Procesy w tle (12)</b>					
Adobe® Flash® Player Utility		0%	1,4 MB	0 MB/s	0 Mb/s
Communications Service		0%	5,5 MB	0 MB/s	0 Mb/s
▶ Indeksator programu Microsoft ...		0%	9,6 MB	0 MB/s	0 Mb/s

Rysunek 5.2. Zaawansowany widok Menedżera zadań

się, choć już nie oglądamy filmu. Aplikacja i tak realizuje zadanie przetwarzania wideo, choć przez otwarcie programu Notatnik na pierwszym planie pokazaliśmy, że nie jesteśmy obecnie zainteresowani filmem.

W aplikacjach Windows 8 scenariusz ten uległ zmianie. Użytkownik decyduje, która aplikacja będzie działać na pierwszym planie, i tylko jedna aplikacja może działać w taki sposób (z wyłączeniem widoku przyciągniętego, pozwalającego na działanie dwóch aplikacji, z których jedna zajmuje mały pasek przestrzeni). Następnie system określa, co się powinno stać z pozostałymi aplikacjami tła. Jeżeli nie zbudujesz swojej aplikacji prawidłowo, będzie ona powodowała negatywne efekty uboczne, wywołujące problemy w interakcji z użytkownikiem. Zarządzanie aplikacjami w Windows Runtime jest nazywane zarządzaniem czasem życia procesu (ang. *Process Lifetime Management* — PLM).



## Zarządzanie czasem życia procesu

Aplikacje Windows 8 działają wyłącznie wtedy, gdy są na pierwszym planie. Pozwala to użytkownikowi na skupienie się na głównej aplikacji, z którą zamierza pracować. Aplikacje w tle przechodzą w stan wstrzymania, w którym wątki aplikacji są zamrożone. Aplikacja przestaje korzystać z zasobów systemowych, dzięki czemu zmniejsza się tempo rozładowywania baterii. W większości przypadków aplikacja pozostaje w pamięci. Pozwala to na szybkie przełączanie aplikacji, więc gdy użytkownik cofnie się do aplikacji poprzedniej, będzie ona natychmiast wznawiana, tak jakby cały czas działała.

Istnieją jednak przypadki, gdy wstrzymana aplikacja może być wyłączona przez system. Jest to zależne od zasobów systemu. Jeżeli na przykład aplikacja działająca na pierwszym planie wymaga przydzielenia dużego obszaru pamięci, który spowoduje zajęcie niemal całej dostępnej pamięci, system może wyłączyć przynajmniej jedną aplikację będącą w stanie wstrzymania. System wybiera do wyłączenia tę aplikację, która zajmuje najwięcej pamięci.

Aby zobaczyć ten proces w działaniu, uruchom menu *Start* przez naciśnięcie klawisza *Windows*. Otwórz Menedżer zadań i umieść go na ekranie Windows 8. Upewnij się, że wybrana została opcja *Zawsze na wierzchu*, znajdująca się w menu *Opcje*. Teraz uruchom aplikację, a następnie wróć do menu *Start* i uruchom kolejną. Powtarzaj ten proces kilka razy. Zauważysz, że po kilku sekundach aplikacje będą przechodziły w stan *Wstrzymany*. Jeżeli status nie jest wyświetlany, przejdź do *Widok, Wartości stanu* i zaznacz *Pokaż stan wstrzymany*. Na rysunku 5.3 pokazany jest wynik uruchomienia kilku aplikacji, gdy na pierwszym planie znajduje się aplikacja Pogoda. Zwróć uwagę, że inne aplikacje Windows 8 są w stanie *Wstrzymany*.



**Rysunek 5.3.** Aplikacje Windows 8 w stanie Wstrzymany

Jeżeli w systemie zacznie brakować zasobów, wiele z aplikacji w stanie *Wstrzymany* może być zakończonych i usuniętych z listy Menedżera zadań. Możesz również emulować wstrzymanie z Visual Studio 2012 w trybie debugowania. Pokażę to w dalszej części rozdziału, ale na początek warto poznać pełny cykl PLM. Życie aplikacji zaczyna się od *aktywacji*.

## Aktywacja

Aktywacja jest pierwszym krokiem w cyklu życia aplikacji. W poprzednich wersjach Windows aktywacja zazwyczaj następowała w wyniku uruchomienia aplikacji przez użytkownika — za pośrednictwem menu *Start* albo przez wpisanie jej nazwy w wierszu polecenia. W Windows 8 aplikacje są zawsze aktywowane poprzez kontrakt. Informacje na temat sposobu aktywacji znajdziesz w rozdziale 2, „Zaczynamy”.

Najpowszechniej stosowanym kontraktem jest *Launch*. Jest to kontrakt wywołujący w momencie stuknięcia kafelka w menu *Start*. Kontrakt *Launch* aktywuje aplikację i wywołuje metodę *OnLaunched*. Innym kontraktem, który może aktywować aplikację, jest *Share*, jeżeli aplikacja jest celem udostępniania, podobnie jak *ImageHelper* z rozdziału 2. Aktywacja aplikacji może być wykonana również przez kilka innych kontraktów. Aplikacja *Windows8Application2* bazuje na aplikacji z rozdziału 4, „Aplikacje Windows 8”, ale została rozbudowana o obsługę kontraktu *Search*. Aplikację tę możesz pobrać z serwera wydawnictwa Helion: <ftp://ftp.helion.pl/przyklady/w8twap.zip>.

Dodanie tego kontraktu do aplikacji Windows 8 jest bardzo proste. Zostało to wykonane w przykładowej aplikacji, ale jeżeli zaczynasz jej tworzenie od początku, możesz kliknąć prawym przyciskiem myszy węzeł projektu w oknie *Solution Explorer* i wybrać *Add/New Item* lub zastosować kombinację klawiszy *Ctrl+Shift+A*. Wybierz opcję *Search Contract* i nazwij stronę wyników wyszukiwania, jak jest to pokazane na rysunku 5.4.



**Rysunek 5.4.** Dodawanie kontraktu Search do projektu

Kiedy jest dodawany nowy element, w tle jest wykonywane kilka operacji. Oprócz utworzenia nowej klasy *Page* do pliku *App.xaml.cs* dodawana jest metoda *OnSearchActivated*, co pozwala na obsługę przychodzących żądań wyszukiwania. Generowany jest fragment kodu, który powinien zostać przez nas przeniesiony do procedury obsługi *OnLaunched*, dzięki czemu będziemy mogli obsłużyć zapytania bez uruchamiania aplikacji. Aktualizowana jest również sekcja z pakietami aplikacji, do której dodawana jest deklaracja *Search*.

Gdy cała infrastruktura jest na swoim miejscu, szablony mogą być wykorzystane do udostępnienia funkcji wyszukiwania. W moim przykładzie wyniki wyszukiwania są pokazywane na stronie z grupą filtrów. Pozwala to użytkownikowi na określenie wyszukiwania we wszystkich elementach lub ograniczyć wyszukiwanie do elementów z danej grupy. Wyszukiwanie pozwala na obsługiwane przychodzących zadań niezależnie od tego, czy aplikacja właśnie działa, czy nie.

Gdy aplikacja jest aktywowana przez którykolwiek z kontraktów, wyświetlony zostaje ekran powitalny. Ekran powitalny składa się z tła o wybranym kolorze oraz statycznego obrazu (sposób definiowania tych elementów został opisany w rozdziale 4., a tutaj, w dalszej części rozdziału, opiszę sposób modyfikacji tego ekranu). Aplikacja jest powiadamiana o aktywacji przez wywołanie odpowiedniej metody z pliku *App.xaml.cs* (*OnLaunching* dla kontraktu *Launch*, *OnSearchActivated* dla kontraktu *Search* itd.).

Aplikacja ma około 15 sekund na pokazanie pierwszej strony. Jeżeli w tym czasie strona nie będzie wyświetlona, aplikacja zostanie zakończona, a dla użytkownika końcowego będzie to raportowane jako awaria. Jest to ważne, ponieważ aplikacje wymagające pobrania dużej ilości danych lub złożonej inicjalizacji muszą pokazać pierwszą stronę przed tymi operacjami, a następnie prawidłowo obsłużyć oczekiwania na zakończenie tych zadań. Można to zrealizować przez rozszerzenie ekranu powitalnego. Więcej informacji na temat rozszerzonych ekranów powitalnych znajduje się w dalszej części rozdziału.

Gdy aplikacja zostanie aktywowana, staje się aplikacją pierwszoplanową i będzie działać w sposób normalny do momentu przełączenia się użytkownika do innej aplikacji, w tym pulpitu. Gdy użytkownik dokonuje przełączenia, aplikacja jest o tym fakcie powiadamiana za pomocą zdarzenia *Suspending*. Pozwala to poinformować aplikację o operacji wstrzymania i umożliwia programiście wykonanie kodu niezbędnego do zachowania stanu przed zakończeniem działania aplikacji.

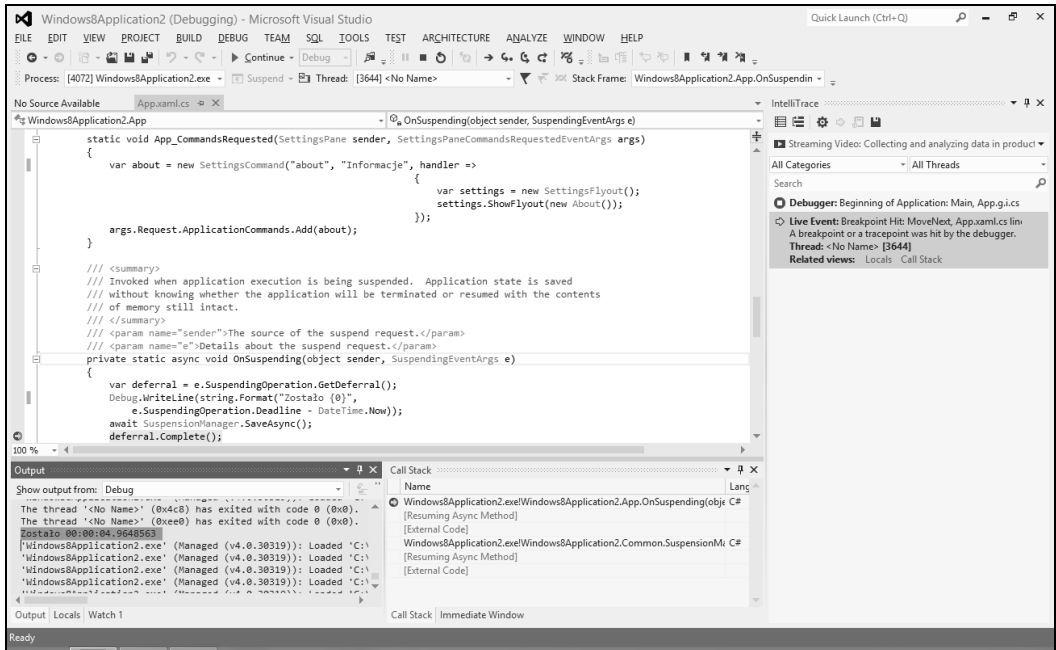
## Wstrzymanie

W czasie normalnego działania aplikacja zostanie wstrzymana od razu po tym, jak przestanie działać na pierwszym planie. Nie jest łatwo wymusić wstrzymanie aplikacji w czasie testowania, więc Visual Studio 2012 pozwala na ręczne wstrzymanie programu w celu wykonania testów tego procesu. Aby zapoznać się ze wstrzymywaniem aplikacji, załaduj projekt *Windows8Application2*.

Otwórz plik *App.xaml.cs* i umieść punkt zatrzymania w wyróżnionym wierszu kodu metody *OnSuspending*:

```
private static async void OnSuspending(object sender,
    SuspendingEventArgs e)
{
    var deferral = e.SuspendingOperation.GetDeferral();
    Debug.WriteLine(string.Format("Zostało {0}",
        e.SuspendingOperation.Deadline - DateTime.Now));
    await SuspensionManager.SaveAsync();
    deferral.Complete();
}
```

Uruchom aplikację w trybie debugowania. Na pasku narzędzi możesz znaleźć przycisk z ikoną, która wygląda jak przycisk pauzy dla aplikacji na drugim planie (nie pomył jej z przyciskiem pauzy, który wymusza operację *Break All*). Dymek podpowiedzi jest pokazany na rysunku 5.5. Użyj tego przycisku do emulowania wstrzymania aplikacji. Po kliknięciu przycisku aplikacja zostanie zatrzymana w wyróżnionym wierszu metody *OnSuspending*. Zwróć uwagę na wartość wypisaną w oknie *Output* (jeżeli nie widzisz go, zastosuj kombinację klawiszy *Ctrl+Alt+O*), a następnie kliknij *Continue*.



Rysunek 5.5. Wstrzymanie aplikacji w trybie debugowania

Wartość `Deadline` zawiera czas, po którym nastąpi wstrzymanie. Jeżeli aplikacja nie wykona niezbędnych zadań w tym okresie, zostanie zakończona. Przy wykonywaniu zadań asynchronicznych musisz zająć wprowadzenia opóźnienia. Gdy nastąpi przekazanie opóźnienia w argumentach żądania, można wykonać operacje asynchroniczne, na przykład zapisanie stanu aplikacji na dysku. W przykładowej aplikacji zapisanie stanu jest realizowane w zdarzeniu, więc nie trzeba wykonywać dodatkowych czynności przy wstrzymaniu aplikacji. Gdy aplikacja jest gotowa do wstrzymania, wywołujemy metodę `Complete`, aby przejść dalej.

## WSKAZÓWKA

Do tej pory nauczyłeś się, w jaki sposób używać debugera Visual Studio 2012 do debugowania aplikacji działających lokalnie lub w symulatorze. Istnieje również możliwość uruchomienia debugowania na zdalnym komputerze. Jest to szczególnie przydatne, jeżeli posiadasz urządzenie do testowania operacji dotykowych, ale wolisz tworzyć aplikacje na laptopie lub stacji roboczej. Aby zdalnie debugować aplikacje, musisz wcześniej uruchomić na zdalnym komputerze aplikację, która konfiguruje środowisko debugowania. Następnie możesz zdefiniować zdalny komputer w opcji *Remote Machine*. Może zająć konieczność uwierzytelnienia się przed rozpoczęciem sesji. Dokładna instrukcja konfiguracji tej opcji jest dostępna pod adresem [http://msdn.microsoft.com/en-us/library/bt727f1t\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/bt727f1t(v=vs.110).aspx).

Na tym etapie aplikacja stale znajduje się w pamięci, ale nie zużywa żadnych innych zasobów. Jeżeli uruchomisz Menedżer zadań, zauważysz, że użycie procesora wynosi 0%. Aplikacja nie wpływa już na

zużycie baterii, bo po prostu czeka. Jeżeli wznowisz działanie (przez przełączenie aplikacji na pierwszy plan lub przez kliknięcie operacji *Resume*, znajdującej się w Visual Studio 2012 w tym samym menu rozwijanym co *Suspend*), aplikacja zostanie przywrócona do działania bez widocznego opóźnienia.

Jądro posiada jeszcze jedną usługę do zatrzymywania aplikacji. Może się zdarzyć, że aplikacja może wykonywać sekcję krytyczną kodu, na przykład zwalnianie blokad plików. Jądro nie zatrzyma aplikacji w czasie wykonywania sekcji krytycznej kodu (zwykle, gdy sterowanie jest przekazywane do komponentu WinRT), a zamiast tego poczeka do momentu zakończenia wykonywania tej sekcji krytycznej, unikając dzięki temu powstania blokad w systemie.

Możesz zauważyć jeszcze jedną opcję obok *Suspend* oraz *Resume*, opisaną jako *Suspend and shutdown*. Przycisk ten jest używany do zakończenia działania aplikacji. Symuluje on sytuację, w której aplikacja jest w stanie zatrzymanym, a brak zasobów wymusza na systemie wyłączenie Twojej aplikacji w celu zwolnienia pamięci.

## Zakończenie działania

Gdy system potrzebuje pamięci, kiedy następuje przełączenie aktywnego użytkownika, system jest zamykany lub nasza aplikacja ulegnie awarii, Windows Runtime kończy działanie aplikacji. Nie istnieje zdarzenie informujące o zakończeniu działania. Dlatego tak ważne jest zapisanie stanu, gdy aplikacja jeszcze działa lub po wywołaniu procedury zdarzenia `OnSuspending`.

Typowym zachowaniem aplikacji jest wyświetlenie pierwszego ekranu, gdy została ona uruchomiona po raz pierwszy lub po jej zamknięciu przez użytkownika. Aplikacja powinna zostać przywrócona do zapisanego stanu wyłącznie wtedy, gdy nastąpił powrót ze stanu zakończenia. Na szczęście poprzedni stan aplikacji zawsze jest przekazywany do zdarzenia `OnLaunched`. Pozwala to na sprawdzenie poprzedniego stanu i podjęcie odpowiedniej akcji w zależności od tego, czy użytkownik przywraca aplikację po jej zakończeniu, czy uruchamia ją normalnie po jej zamknięciu.

## Wznowienie

Aplikacja może zarejestrować zdarzenie `Resuming`. Zdarzenie to jest wywoływane, gdy aplikacja jest nadal w pamięci, a użytkownik przełącza ją na pierwszy plan. Nie ma potrzeby odtwarzania stanu, ponieważ wszystkie wątki aplikacji, stos oraz magazyn pozostają dostępne. Do czego może być przydatne takie zdarzenie?

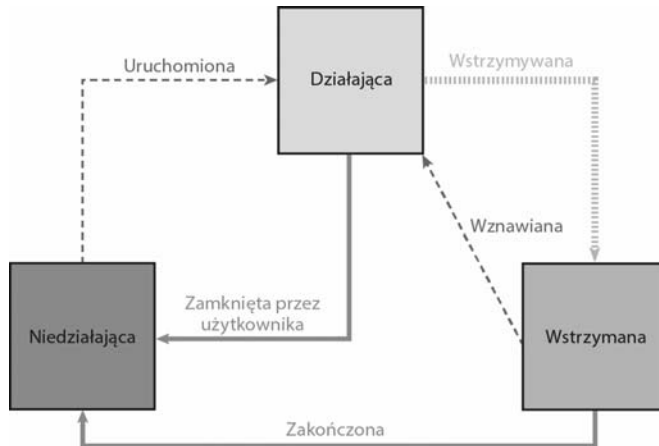
Głównym jego zastosowaniem jest zapewnienie okresowego odświeżania danych w aplikacji. Wyobraźmy sobie aplikację z prognozą pogody, która była wstrzymana przez kilka godzin, na czas pasjonującej rozgrywki w Twojej ulubionej grze. Aplikacja pozostawała zamrożona w pamięci wraz ze wszystkimi wątkami. Następnie wracasz do aplikacji. Co widzisz? Bez zdarzenia `Resuming` będziesz widział stare informacje pogodowe sprzed kilku godzin. Zdarzenie to pozwoli aplikacji pobrać najnowszą prognozę i pokazać bieżący stan pogody, w tym alarmy, jakie mogły się pojawić.

Jeżeli aplikacja jest zakończona, to jest ponownie uruchamiana przy próbie powrotu do niej przez użytkownika. Zdarzenie `Resuming` nie jest wtedy uruchamiane. Zamiast tego aplikacja może sprawdzać poprzedni stan w celu określenia, który wariant kodu startowego należy wykonać. Zajrzyj do pliku `ExtendedSplashScreen.xaml.cs`, do metody `ExtendedSplashScreen_Loaded`, w której znajduje się następujący warunek:

```
if (_activationArgs.PreviousExecutionState ==
    ApplicationExecutionState.Terminated)
```

Jeżeli aplikacja była wyłączona, zostanie odczytany poprzedni stan i użytkownik wróci do miejsca, w którym wcześniej skończył pracę. Sposób zapisu i przywracania stanu jest przedstawiony w dalszej części rozdziału. W przeciwnym razie aplikacja inicjuje się i jest przywoływany ekran *Start*. Innymi stanami, które można sprawdzić, są: `ClosedByUser`, `NotRunning`, `Running` oraz `Suspended`.

Na rysunku 5.6 przedstawiony jest schemat cyklu życia aplikacji.



**Rysunek 5.6.** Schemat cyklu życia aplikacji

Poniższa lista wskazówek pomoże Ci budować aplikację w taki sposób, aby zapewnić użytkownikowi możliwie płynne działanie przez zarządzanie PLM:

- **przyrostowo zapisuj dane użytkownika** — pozwala to uniknąć zapisywania wszystkiego naraz przy zatrzymywaniu aplikacji, co niesie ze sobą niebezpieczeństwo przekroczenia czasu,
- **zapisuj i przywracaj stan aplikacji** — jest to ważne, gdyż należy się upewnić, czy użytkownik będzie miał wrażenie ciągłej pracy po przywróceniu aplikacji,
- **zapisuj i przywracaj wyłącznie metadane sesji (dotyczące tego, w którym punkcie znajduje się użytkownik)** — pozostałe dane powinny być obsługiwane w innym miejscu aplikacji.

Teraz, gdy znasz już cykl życia aplikacji, możesz użyć kodu z szablonów do zarządzania stanem aplikacji. Pomoże Ci to zrozumieć, w jaki sposób działa nawigacja w aplikacjach Windows 8. Gdy wcześniej zatrzymana aplikacja jest uruchamiana z ekranu *Start*, będzie ona wznawiona, a nie uruchomiona od początku.

## Nawigacja

W aplikacjach Windows 8 występują dwa podstawowe typy nawigacji — hierarchiczna i płaska. W aplikacjach hierarchicznych zaczynamy od widoku wysokiego poziomu i zagłębiamy się aż do interesującego nas obszaru. W przykładowej aplikacji operowaliśmy na grupach kolekcji, następnie wchodziliśmy do kolekcji i ostatecznie do poszczególnych elementów. Nawigacja płaska działa podobnie jak w kreatorze, kierując nas zgodnie z kolejnymi krokami procesu.

Po pierwszym uruchomieniu aplikacji pocztowej najprawdopodobniej zaczniemy od scenariusza nawigacji płaskiej. Kreator poprowadzi nas przez proces wprowadzania informacji na temat serwera

pocztowego, danych uwierzytelniających, testowania połączenia oraz dodawania konta do listy. Gdy zaczniemy korzystać z aplikacji pocztowej, widok stanie się hierarchiczny. Na początku otwiera się widok kont, a następnie wchodzimy głębiej, do poszczególnych wiadomości.

W przypadku aplikacji Windows 8 zbudowanych z wykorzystaniem XAML oraz C# domyślnym kontenerem dla aplikacji jest `Frame`. Zjrzyj do pliku `ExtendedSplashScreen.xaml.cs` w aplikacji `Windows8Application2`. Kod w metodzie `ExtendedSplashScreen_Loaded` tworzy obiekt `Frame`, podłącza zdarzenia, a następnie przechodzi do gotowej strony. Ekran powitalny wykonuje pewne operacje przed podłączeniem głównego mechanizmu nawigacyjnego aplikacji (więcej informacji na temat tworzenia własnych ekranów startowych znajduje się w dalszej części rozdziału):

```
var rootFrame = new Frame();
rootFrame.Navigating += rootFrame_Navigating;
// Tu znajduje się dodatkowy kod
rootFrame.Navigate(target, parameter);
```

Obiekt `Frame` jest kontenerem wizualnym obsługującym nawigację. Jednostką nawigacji wewnątrz `Frame` jest `Page`. Obiekt `Page` jest specjalnym rodzajem kontrolki, który ma możliwość współpracy z mechanizmem nawigacji. Można go uznać za kontener dla elementów wizualnych, które chcemy zaprezentować w określonym obszarze aplikacji. Obiekt `Frame` pozwala na nawigowanie przez przekazywanie typu obiektu `Page`, który chcemy wyświetlić, jak również opcjonalnego parametru. W przykładowej aplikacji celem jest domyślnie typ głównego okna zawierającego różne grupy, a jako parametr przekazywana jest kolekcja grup ze źródła danych:

```
if (!rootFrame.Navigate(typeof(GroupedItemsPage), "ItemGroups"))
```

Wywołanie metody `Navigate` z `Frame` powoduje utworzenie obiektu docelowej strony. Gdy docelowa strona zostanie załadowana, będzie wywołana metoda `OnNavigateTo`. Pozwala to skonfigurować interfejs użytkownika. Przykładowa aplikacja korzysta z tej metody do podłączenia danych z bazowego modelu widoku (więcej na temat modelu widoku znajdziesz w rozdziale 9., „MVVM i testowanie”), jak również do ustawienia źródła dla elementów kontrolki.

Obiekt `Page` przechowuje historię wszystkich obiektów `Page`, do których nawigował użytkownik. Pozwala to na realizację funkcji zbliżonej do przeglądarki internetowej, ponieważ możesz cofnąć się do poprzedniego obiektu lub przejść do przodu, jeżeli wcześniej nawigowałeś wstecz. Akcje te są obsługiwane za pomocą metod `GoBack` oraz `GoForward`, a aby sprawdzić, czy akcje te są możliwe, powinieneś odczytać wartości właściwości `CanGoBack` i `CanGoForward` z obiektu `Frame`.

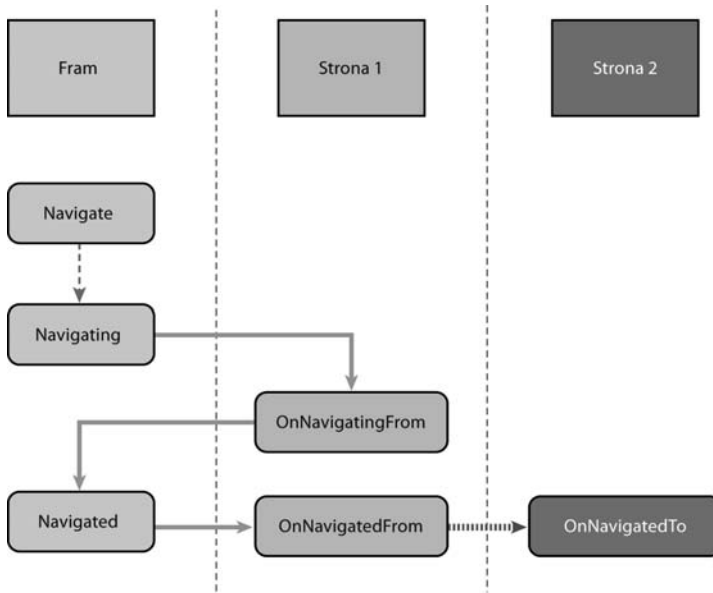
Otwórz plik `GroupDetailPage.xaml`. Kod XAML dla przycisku cofnięcia wygląda następująco:

```
<Button x:Name="backButton" Click="GoBack"
    IsEnabled="{Binding Frame.CanGoBack,
    ElementName=pageRoot}"
    Style="{StaticResource BackButtonStyle}"/>
```

Przycisk jest aktywny wyłącznie wtedy, gdy w historii znajduje się poprzednia strona. Jeżeli jest on aktywny i użytkownik kliknie przycisk, wywołana zostanie metoda `GoBack`. Metoda ta jest zdefiniowana w bazowej klasie `LayoutAwarePage`, która z kolei dziedziczy po `Page`:

```
protected virtual void GoBack(object sender, RoutedEventArgs e)
{
    if (this.Frame != null && this.Frame.CanGoBack)
        this.Frame.GoBack();
}
```

Na rysunku 5.7 zamieszczony jest typowy przebieg wystąpień zdarzeń nawigacji. Na obiekcie *Frame* została wywołana metoda *Navigate*, co powoduje przejście do nowej strony o nazwie *Strona 2*. W obiekcie *Frame* generowane jest zdarzenie *Navigating*. Obiekt *Strona 1* jest informowany o nawigacji za pośrednictwem zdarzenia *OnNavigatingFrom*. Jeżeli istnieje powód anulowania nawigacji (na przykład użytkownik wprowadził dane w formularzu i nie zapisał ich), *Strona 1* może to wykonać przez ustawienie znacznika *Cancel* w zdarzeniu.



**Rysunek 5.7.** Przegląd zdarzeń nawigacji

Gdy *Strona 1* nie decyduje się na anulowanie nawigacji, obiekt *Frame* usuwa z widoku stronę i generuje zdarzenie *Navigated*. Informuje obiekt *Strona 1* o jego usunięciu z widoku za pomocą metody *OnNavigatedFrom*. Następnie umieszcza obiekt *Strona 2* w widocznej ramce i powiadamia go o tym fakcie za pomocą zdarzenia *OnNavigatedTo*. Obiekt *Frame* może utworzyć nowy obiekt *Strona 2* lub ponownie użyć istniejącego. *Strona 1* wchodzi teraz do historii, więc właściwość *CanGoBack* jest włączona i wywołanie *GoBack* w obiekcie *Frame* spowoduje ponowne pokazanie *Strona 1*.

Zarządzanie obiektami stron przez *Frame* odbywa się za pośrednictwem właściwości *CacheSize*. Jej wartość reprezentuje liczbę stron znajdujących się w pamięci podręcznej. Jeżeli strona jest umieszczona w pamięci podręcznej, to nie będzie tworzony nowy obiekt, tylko zostanie ona powtórnie wykorzystana. Aby obiekt *Page* był przechowywany w pamięci podręcznej, należy ustawić w nim właściwość *Navigat ion- CacheMode* na *Enabled* lub *Required*. Gdy wartością tej właściwości jest *Required*, będzie ona zapamiętywana niezależnie od ustawienia *CacheSize* w obiekcie *Frame* i nie będzie uwzględniana przy ustalaniu liczby w *CacheSize*.

Zrozumienie sposobu działania nawigacji ułatwia zarządzanie lokalizacją użytkownika. Wbudowane szablony w Visual Studio 2012 posiadają klasę pomocniczą o nazwie *SuspensionManager*, która wspiera programistę przy zapisywaniu i odtwarzaniu stanu. Po załadowaniu aplikacji główny obiekt *Frame* jest rejestrowany wraz z *SuspensionManager* w pliku *App.xaml.cs*:

```
SuspensionManager.RegisterFrame(rootFrame, "AppFrame");
```



Powoduje to skonfigurowanie obiektu `SessionManager`, który obserwuje nawigację użytkownika w ramce. Następnie zachowuje stos nawigacji razem z parametrami przekazanymi do stron, dzięki czemu po wznowieniu działania aplikacji może odtworzyć zarówno lokalizację użytkownika, jak i historię nawigacji. Stan jest zapisywany w lokalnej pamięci podręcznej za pomocą `API Application Data`.

## API Application Data

`API Application Data` zapewnia mechanizmy manipulowania danymi specyficznymi dla określonej aplikacji. Można korzystać z niego do zapisywania stanu, preferencji, lokalnej pamięci podręcznej oraz innych ustawień. `Windows Runtime` zarządza szczegółami zapisu specyficznymi dla aplikacji i izoluje je w ramach konta użytkownika i aplikacji w systemie. Dane są zachowywane przy aktualizacji aplikacji i usuwane w przypadku jej odinstalowania.

Mamy do dyspozycji trzy typy danych lokalnych dostępnych dla aplikacji. W tabeli 5.1 wymienione są te typy wraz z opisem ich przeznaczenia.

**Tabela 5.1.** Typy przechowywanych danych aplikacji

Magazyn	Opis
Local	Są dostępne dla aplikacji i użytkowników bieżącego urządzenia. Są zachowywane pomiędzy sesjami działania aplikacji oraz uruchomieniami komputera.
Roaming	Są dostępne dla aplikacji i użytkowników wszystkich urządzeń. Magazyn ten jest ograniczony do około 100 kB danych. Wymaga konta Windows.
Temporary	Magazyn ten może być w dowolnym momencie usunięty z systemu i zwykle jest używany do lokalnych „plików roboczych” lub innych danych tymczasowych, które nie muszą być zachowywane pomiędzy kolejnymi uruchomieniami aplikacji.

Każdy magazyn zapewnia dwa rodzaje dostępu. Pierwszy rodzaj, *ustawienia aplikacji*, pozwala na przechowywanie dowolnego typu danych `WinRT`. Można w ten sposób zapisywać takie dane, jak pojedyncze wartości, wartości złożone lub w kontenerach, uporządkowane zbiory danych obu typów. Magazyn ten jest przeznaczony dla małych zbiorów danych. Zgodnie z dokumentacją ustawienia te są w rzeczywistości przechowywane w rejestrze. Dokumentacja dotycząca ustawień aplikacji znajduje się pod adresem <http://msdn.microsoft.com/en-us/library/windows/apps/hh464917.aspx>.

Drugim rodzajem są *pliki aplikacji*. Są to pliki w tradycyjnym sensie, które można odczytywać i zapisywać. Każdy magazyn ma przestrzeń przydzieloną dla aplikacji i użytkownika. Możemy tworzyć katalogi do 32 poziomów głębokości oraz odczytywać i zapisywać tyle plików, ile potrzebujemy. Pliki zapisane w magazynie tymczasowym mogą być usunięte w każdym momencie. Gdy użytkownik jest zalogowany do konta Windows (na przykład Live), pliki zapisywane do magazynu wspólnego są synchronizowane pomiędzy urządzeniami, ale muszą się one mieścić w ograniczeniach konta, które można odczytać za pomocą `ApplicationData.RoamingStorageQuota`.

Aplikacja kontroluje dane lokalne, więc zazwyczaj nie są one zmieniane poza kodem aplikacji. Dane wspólne są inne, ponieważ użytkownik może zmienić je na jednym urządzeniu, a inna instancja aplikacji działa na drugim urządzeniu. Gdy dane są synchronizowane, zmiany te powodują wygenerowanie zdarzenia `DataChanged`. Twoja aplikacja może używać tego zdarzenia do reagowania na zmiany i odświeżenia danych z lokalnego urządzenia. Możesz również jawnie zasygnalizować modyfikacje danych przez wywołanie `SignalDataChanged`.

Przykładowa aplikacja *Windows8Application2* korzysta z ustawień lokalnych do zapisywania ustawień lokalizacji dla aplikacji, co jest domyślnym działaniem klasy *SuspensionManager*. Korzystamy tu z ustawień lokalnych zamiast wspólnych, ponieważ dodany nowy element jest przechowywany wyłącznie lokalnie. Lista elementów może rozrosnąć się do sporych rozmiarów, więc nie ma sensu przechowywać jej w ustawieniach wspólnych. Więcej informacji na temat danych oraz zarządzania dużymi kolekcjami pomiędzy urządzeniami znajdziesz w rozdziale 6., „Dane”.

Ustawienia przechowywania danych aplikacji są niezmiernie proste w użyciu. Jeżeli nie chcesz zapamiętywać historii nawigacji ani parametrów przekazanych do kolejnych stron, możesz bez trudu zapisać bieżącą stronę przez zaktualizowanie właściwości *NavigatedPage* w pliku *App.xaml.cs*:

```
set
{
    _navigatedPage = value;
    ApplicationData.Current.LocalSettings.Values["NavigatedPage"]
        = value.ToString();
}
```

Jest to naprawdę proste — wystarczy wybrać klucz dla właściwości i ustawić wartość. Możliwe jest również zapisanie grup wartości. Jeżeli potrzebujesz operacji atomowej, dzięki której zostanie zapisana cała grupa wartości lub nie zostanie zapisana żadna z nich, użyj właściwości *ApplicationDataCompositeValue*. Powoduje to przygotowanie atomowej grupy wartości do jej zapisania w postaci jednego ustawienia. Bieżąca grupa oraz element mogą być zapisane w następujący sposób, za pomocą kodu z pliku *App.xaml.cs*:

```
private void PersistCurrentSettings()
{
    var value = new ApplicationDataCompositeValue();
    value.Add("Group", _currentGroup == null ? string.Empty :
        _currentGroup.UniqueId);
    value.Add("Item", _item == null ? string.Empty : _item.UniqueId);
    ApplicationData.Current.LocalSettings.Values["Current"] = value;
}
```

Odczyt wartości jest równie prosty. Aby przywrócić bieżącą stronę, aplikacja może po prostu odczytać wartość jako ciąg znaków, a następnie skonwertować ją na aktualny typ dla nawigacji:

```
target = Type.GetType(ApplicationData.Current.LocalSettings
    .Values["NavigatedPage"].ToString(), true);
```

Do mechanizmu nawigacji zostanie przekazana wartość bieżącego elementu lub grupy. Wartości złożone są odczytywane w poniższy sposób:

```
var value = ApplicationData.Current.LocalSettings.Values["Current"]
    as ApplicationDataCompositeValue;
```

Następnie wartość może być odtworzona i pobrana do wewnętrznych kolekcji, co pozwoli wygenerować pełny obiekt:

```
var currentItemId = value["Item"];
CurrentItem = (from g in DataSource.ItemGroups
    from i in g.Items
    where i.UniqueId.Equals(currentItemId)
    select i).FirstOrDefault();
```

Zapytanie jest zapisane z użyciem składni Language Integrated Query, czyli LINQ. Więcej informacji na temat LINQ można znaleźć pod adresem [http://msdn.microsoft.com/en-us/library/bb397926\(vs.110\).aspx](http://msdn.microsoft.com/en-us/library/bb397926(vs.110).aspx).

Możliwe jest również rozdzielenie ustawień na osobne kontenery. Do tego celu należy użyć klasy `ApplicationDataContainer`. Magazyn posiada metodę `CreateContainer`, jak również listę bieżących kontenerów w `Containers`. Po odwołaniu się do danego kontenera możesz odczytać słownik `Values` i określić lub odczytywać ustawienia specyficzne dla tego kontenera.

Magazyn nie jest ograniczony do prostych par klucz-wartość. Można również tworzyć foldery i pliki. Zamiast zapisywać pojedyncze strony i parametry, jak przed chwilą pokazałem, można użyć generycznej klasy pomocniczej `SuspensionManager` do obsługi większości scenariuszy nawigacji. Za pomocą `SuspensionManager` deklarujemy nazwę pliku, w którym będą zapisane wszystkie dane stanu dla aplikacji:

```
private const string sessionStateFilename = "_sessionState.xml";
```

Klasa ta korzysta z `DataContractSerializer` do utworzenia obrazu zapisanego stanu nawigacji. Klasa ta może przejrzeć wszystkie właściwości i wartości w obiekcie i zapisać je w postaci reprezentacji XML. Poniżej przedstawiony jest kod, który realizuje tę operację przez przekształcenie obiektu na postać pamięciowego dokumentu XML:

```
MemoryStream sessionData = new MemoryStream();
DataContractSerializer serializer =
    new DataContractSerializer(typeof(Dictionary<string, object>),
        _knownTypes);
serializer.WriteObject(sessionData, _sessionState);
```

Kod ten jest zapisywany na urządzeniu. API `Application Data` jest używane do utworzenia instancji pliku, a następnie serializowany strumień jest zapisywany na dysku:

```
StorageFile file = await ApplicationData.Current.LocalFolder
    .CreateFileAsync(sessionStateFilename,
        CreationCollisionOption.ReplaceExisting);
using (Stream fileStream = await file.OpenStreamForWriteAsync())
{
    sessionData.Seek(0, SeekOrigin.Begin);
    await sessionData.CopyToAsync(fileStream);
    await fileStream.FlushAsync();
}
```

Aby przy powrocie użytkownika do aplikacji odtworzyć stan, wykonywana jest odwrotna operacja:

```
StorageFile file = await ApplicationData.Current.LocalFolder
    .GetFileAsync(sessionStateFilename);
using (IInputStream inputStream = await file.OpenSequentialReadAsync())
{
    DataContractSerializer serializer = new DataContractSerializer(
        typeof(Dictionary<string, object>), _knownTypes);
    _sessionState = (Dictionary<string, object>)serializer
        .ReadObject(inputStream.AsStreamForRead());
}
```

Skompiluj i uruchom aplikację, a następnie przejdź do grupy i elementu. Zakończ aplikację przez przeciągnięcie palcem poza dolną krawędź ekranu lub naciśnięcie `Alt+F4`. Teraz otwórz *Eksplorator plików* i przejdź do następującego katalogu:

```
C:\users\{nazwa_użytkownika}\AppData\Local\Packages\{nazwa_pakietu}
```

W miejsce `{nazwa_użytkownika}` wstaw swoją nazwę użytkownika, a w miejsce `{nazwa_pakietu}` nazwę pakietu ustawioną w polu `Package name`, na zakładce `Packaging`, w manifeście aplikacji. Na moim komputerze ścieżka ta wygląda następująco:

```
C:\Users\Jeremy\AppData\Local\Packages\34F9460F-0BA2-4952-8627-439D8DC45427_req6rhny9ggkj
```

Otwórz katalog *LocalState*, w którym powinieneś znaleźć plik *\_sessionState.xml*. W pliku tym zapisany jest stan aplikacji.

Przykładowy element XML przechowujący historię nawigacji wygląda następująco:

```
<KeyValueOfstringanyType>
<Key>Navigation</Key>
<Value i:type="a:string" xmlns:a="http://www.w3.org/2001/XMLSchema">1,3,2,33,
↳Windows8Application2.GroupedItemsPage,12,10,ItemGroups,32,Windows8Application2.
↳GroupDetailPage,12,7,Group-2,31,Windows8Application2.ItemDetailPage,12,14,Group-2-Item-2
</Value>
</KeyValueOfstringanyType>
```

Więcej informacji na temat zapisywania i odczytywania danych znajduje się w rozdziale 6., „Dane”.

## Żywy i podłączony

Choć aplikacje są zatrzymywane i potencjalnie wyłączane po przełączeniu, to jednak nie oznacza to, że nie mogą pozostawać podłączone lub sprawiać wrażenia, że normalnie działają. Jednym z efektywnych sposobów na zapewnienie aktualności aplikacji jest użycie kafelków i powiadomień. Więcej informacji na te tematy znajduje się w rozdziale 7., „Kafelki i powiadomienia”.

Istnieją również inne sposoby na zapewnienie ciągłej pracy aplikacji. Poniższa lista zawiera kilka z możliwości:

- **dźwięk** — niektóre aplikacje, takie jak odtwarzacze podcastów i muzyki, mogą zarejestrować operację odtwarzania dźwięku w tle, korzystając z podstawowego zbioru kontrolki dźwięku,
- **pobieranie** — do pobrania dużych ilości danych przy oszczędnym wykorzystaniu baterii można użyć Background Transfer API,
- **ekran blokady** — aplikacje ekranu blokady mogą wyświetlać podstawowe informacje na zablokowanym ekranie (na przykład aplikacja pocztowa może pokazywać liczbę nieprzeczytanych wiadomości),
- **gniazda** — klasa `ControlChannelTrigger` może być użyta do zarejestrowania zdarzeń sieciowych, które powiadamiają aplikację nawet, gdy jest zatrzymana.

Nieprawdziwe jest twierdzenie, że aplikacja Windows 8 zatrzymuje się w momencie jej przełączenia w tło. Kombinacja usług zapewnianych przez platformę pozwala na zachowanie połączeń i aktualizowanie danych bez konieczności działania na pierwszym planie. Użycie odpowiednich API pozwala aplikacji na oszczędzanie zasobów systemowych oraz zwiększanie czasu pracy na bateriach dzięki obsłudze zdarzeń we właściwy sposób.

## Własny ekran startowy

We wcześniejszej części tego rozdziału wspomniałem, że aplikacja musi wyświetlić pierwszą stronę w określonym czasie i że w przeciwnym razie zostanie ona zakończona. Gdy potrzebujesz więcej czasu na inicjalizację aplikacji, możesz wyświetlić własny ekran startowy i z jego poziomu realizować inicjalizację. W tym podrozdziale pokażę, w jaki sposób zbudować własny ekran startowy.

W aplikacji *Windows8Application2* długi czas inicjalizacji jest symulowany przez wykonanie operacji w pętli. W trybie debugowania pętla ta działa długo, ponieważ zapisuje wartości do okna debugera:

```
await Task.Run(() =>
{
    for (var x = 0; x < 2000; x++)
    {
        Debug.WriteLine(x);
    }
});
```

Pętla ta ma symulować operacje często wykonywane w czasie inicjalizacji aplikacji. Operacje takie mogą obejmować podłączenie się do usług sieciowych i pobranie treści lub odczyt elementów zapisanych w pamięci podręcznej. Jeżeli aplikacja wykonuje te operacje zbyt długo, po 15 sekundach zostanie zamknięta przez Windows 8.

Aby rozwiązać ten problem, dane mogą być ładowane przez własny ekran startowy przed wyświetleniem głównej strony aplikacji. W pliku *App.xaml.cs*, w metodzie `OnLaunched`, jako główny ekran ustawiany jest obiekt typu `ExtendedSplashScreen`, do którego przekazywany jest bieżący ekran startowy:

```
var splashScreen = args.SplashScreen;
var eSplash = new ExtendedSplashScreen(splashScreen, false, args);
splashScreen.Dismissed += eSplash.DismissedEventHandler;
Window.Current.Content = eSplash;
Window.Current.Activate();
```

Kontrolka `ExtendedSplashScreen` przechowuje referencję oryginalnego ekranu startowego. Zawiera ona kopię obrazu wyświetlanego na oryginalnym ekranie startowym; w celu ustawienia tego obrazu na ekranie korzysta ze współrzędnych określonych przez system. Obraz pokazuje się dokładnie w tym samym miejscu, dzięki czemu wykonywane jest niezauważalne przejście od obrazu z ekranu startowego do rozszerzonego ekranu startowego.

Po załadowaniu kontrolki wykonywana jest metoda `Initialize` na źródle danych i następuje asynchroniczne oczekiwanie na załadowanie elementów. Gdy wszystkie elementy zostaną załadowane, kontrolka jest zastępowana przez obiekt `Frame` i następuje przejście do pierwszej strony. Cała logika normalnie znajduje się w pliku kodu ukrytego *App.xaml.cs*, ale została przeniesiona do kontrolki `ExtendedSplashScreen`, dzięki czemu może być opóźniona do momentu załadowania kompletu danych.

Kontrolka `ProgressRing` została umieszczona na ekranie startowym poniżej grafiki, dzięki czemu użytkownik widzi, że jest realizowane przetwarzanie. Aby zobaczyć wskaźnik postępu, uruchom aplikację w trybie debugowania. Zobaczysz wtedy obraz ekranu startowego wraz z obracającym się wskaźnikiem postępu. Jest on animowany nawet pomimo zablokowania programu przez asynchroniczną obsługę operacji, więc nie blokuje on wątku UI. Przykład ten ilustruje bardzo prosty mechanizm sprzężenia zwrotnego, ale aplikacje z bardziej złożonymi procesami inicjującymi mogą korzystać z paska postępu i wyświetlać tekstowe informacje o ładowanych komponentach. Bardziej złożone przykłady są przedstawione w kolejnych rozdziałach; tutaj ekran startowy wykonuje jedynie pętlę spowalniającą, a następnie uruchamia główną stronę.

## Podsumowanie

W tym rozdziale przedstawiłem cykl życia aplikacji oraz rolę, jaką spełnia mechanizm `Process Lifetime Management (PLM)`. Aplikacje mogą być zatrzymywane i wyłączane, ale mają mechanizmy pozwalające na zapisanie i odtworzenie stanu, które umożliwiają synchronizację wielu urządzeń. Zapoznałeś się z nawigacją bazującą na ramkach, która jest wykorzystywana w aplikacjach Windows 8, oraz ze sposobami

wykorzystania lokalnego magazynu do zapisania lokalizacji użytkownika w aplikacji. Lokalny magazyn pozwala na zapisanie i odtwarzanie danych specyficznych dla użytkownika i aplikacji.

Nauczyłeś się również, w jaki sposób utworzyć własny ekran startowy, który pozwala aplikacjom na wykonanie inicjalizacji bez niebezpieczeństwa wyłączenia, gdy zajmuje ona ponad 15 sekund. Wspomniałem też o danych aplikacji, demonstrując lokalny system plików. Aplikacje Windows 8 mogą tworzyć dane i korzystać z nich na wiele sposobów, od usług sieciowych do plików lokalnych. W następnym rozdziale, „Dane”, przedstawię dokładniej obsługę danych.



# Skorowidz

---

.NET, 19  
.NET 1.0, 22  
.NET Framework, 22

## A

ABI, Application Binary Interface, 35  
aktualizacja kafelków, 171  
aktywacja aplikacji, 130  
algorytm  
  CMAC, 164  
  MD5, 164  
  SHA, 164  
algorytmy  
  kompresji, 162  
  szyfrowania, 162  
animacje, 75  
API, 9, 19  
API Application Data, 137  
API Win32, 22  
API WinRT, 27, 33  
aplikacja  
  BlogPostTarget, 207  
  ILDASM, 60  
  ImageHelper, 47  
  VirtualBox, 43  
  Wintellog, 146, 185  
aplikacje  
  Windows 8, 26, 28, 30  
  Windows Store, 26  
architektura aplikacji Win32, 22

asercje dla testu, 232  
asynchroniczny przepływ sterowania, 151

## B

BCL, Base Class Libraries, 36  
biblioteka  
  BCL, 36  
  Callisto, 209  
  Json.NET, 203  
  klas, 46  
  MFC, 21  
  testów jednostkowych, 47  
  TPL, 149  
  WPF, 23  
BindableBase, 45  
Blend, 32, 39, 44  
błędy, 146  
BooleanNegationConverter, 45  
BooleanToVisibilityConverter, 45  
bufory, 160

## C

C#, 35  
C++, 34  
cel aktualny, 110  
cel wizualny, 110  
celowanie, 110  
centrowanie obrazu, 48  
ciągła praca aplikacji, 140

CLI, Common Language Infrastructure, 27  
CLR, Common Language Runtime, 22  
CLR, Core Language Runtime, 14, 69  
COM, Common Object Model, 22  
cykl życia aplikacji, 127  
czujnik, 119  
  orientacji, 123  
  światła, 123

## D

dane, 143  
dane lokalne, 137  
debuger, 64  
debugowanie aplikacji, 96, 132, 179  
definicja  
  dla stanu Snapped, 99  
  manifestu, 49  
  siatki, 85  
  słownika, 77  
definiowanie  
  grup, 98  
  opcjonalnych przejść, 98  
  stanów, 98  
deklarowanie  
  interfejsu użytkownika, 61  
  opcji Share Target, 206  
DI, 222

dodawanie  
kontraktu, 130  
usługi sieciowej, 165  
dotyk, 12  
drzewo wizualne, 63  
dwukrotne stuknięcie, 104  
dysk USB, 40  
dźwięk, 140

**E**

efekty wizualne, 108  
ekran  
blokad, 140  
startowy, 141  
startowy Windows 8, 31  
ekrany powitalne, 117  
element  
Canvas, 78  
CollectionViewSource, 89  
ContentControl, 85  
ContextMenu, 111  
FlipView, 91  
Grid, 47, 67, 79  
GridView, 88  
ItemsControl, 86  
ListBox, 92  
ListView, 91  
ScrollViewer, 86  
StackPanel, 81  
Style, 76  
TextBlock, 67  
VariableSizedWrapGrid, 83,  
89  
ViewBox, 86  
VirtualizingPanel, 81  
VirtualizingStackPanel, 81, 90  
VisualStateGroups, 98  
WrapGrid, 82

**F**

filtrowanie  
elementów, 197  
według pliku testów, 228  
filtry, 156, 197  
fokus, 107  
folder Groups, 147

format  
JSON, 147  
PNG, 53  
funkcja  
odbijania, 76  
rotowania, 76

**G**

GDI, Graphics Device Interface,  
27  
geolokalizacja, 121  
gesty, 52  
gesty dotykowe, 65  
glify, 175  
główny element wizualny, 78  
gniazda, 140  
grupa, 98  
grupowanie, 89, 157

**H**

HLSL, High Level Shading  
Language, 34  
HTML5, 12, 33

**I**

IDE, 22  
ikony, 12, 117  
ILDASM, 57  
imitacja interfejsu IDialog, 230  
inklinometr, 122  
instalacja Windows 8  
maszyna wirtualna, 40  
pełna instalacja, 40  
równoległe systemy, 40  
instalowanie certyfikatów, 247  
instrukcja using, 53  
integracja ze Sklepem Windows,  
234  
interakcje dotykowe, 103  
interfejs  
Inspectable, 35  
INotifyPropertyChanged, 71  
IUnknown, 35  
IValueConverter, 72

programowania aplikacji,  
API, 20  
użytkownika, UI, 9, 47, 214

**J**

JavaScript, 32, 34  
jądro, 133  
język WSDL, 165  
JSON, JAVASCRIPT OBJECT  
NOTATION, 147

**K**

kafelek Twitter, 29  
kafelki, 29, 169  
aktywne, 170  
podrzędne, 176  
proste, 169  
kanał  
OData, 167  
RSS, 158  
katalog NotificationExtensions,  
175  
Kinect, 25  
klasa  
App, 200  
ApplicationDataContainer,  
139  
AsymmetricKeyAlgorithm  
↳ Provider, 163  
BadgeGlyphNotification  
↳ Content, 175  
BindableBase, 45  
BlogDataSource, 153–159,  
231  
BlogGroup, 147  
Compressor, 161  
Control, 98  
ControlChannelTrigger, 140  
CryptographicEngine, 162  
DataBindingHost, 71  
DataContractJsonSerializer,  
145  
DataContractSerializer, 144,  
148  
DataPackage, 201



- DataServiceCollection, 167
  - Decompressor, 161
  - DependencyObject, 65–69, 72, 77
  - DialogTest, 230
  - Encoding, 161
  - FileIO, 152
  - Filter, 197
  - FrameworkElement, 77
  - HttpClient, 157, 158
  - ItemDetailPage, 203
  - LayoutAwarePage, 45, 99, 215
  - MacAlgorithmProvider, 164
  - MagicButton, 68
  - Package, 154
  - PathIO, 152
  - SettingsFlyout, 118, 211
  - SimpleItem, 88
  - SplashPage, 171
  - SuspensionManager, 46, 139, 143
  - SymmetricKeyAlgorithm
    - ↳ Provider, 162
  - SyndicatedClient, 159
  - TileUpdateManager, 172
  - Timeline, 74
  - ToastNotificationManager, 181
  - TranslateTransform, 75
  - Visual State Manager, 98
  - WindowsRuntimeBuffer
    - ↳ Extensions, 160
  - klucz
    - asymetryczny, 163
    - symetryczny, 162
  - kod
    - aplikacji ImageHelper, 59
    - MAC, 164
    - niezarządzany, 21
  - kodowanie
    - ASCII, 161
    - Base64, 163
    - UTF-16, 161
    - UTF-8, 161
  - kojarzenie aplikacji ze Sklepem
    - Windows, 243
  - kolekcje, 155
  - kolizje, 147
  - kompas, 121
  - komponent
    - CameraCaptureUI, 56, 59
    - DataTransferManager, 200
    - FileSavePicker, 56
    - Windows Runtime, 47
  - komponenty WinRT, 35, 59
  - kompresja danych, 160
  - konfigurowanie
    - Produktów, 239
    - siatki, 87
    - środowiska, 39
    - testów, 227
  - konto Windows Live, 185
  - kontrakt, 12, 28, 191
    - Launch, 130
    - Search, 195
    - Settings, 209
    - Share, 208
    - Share Target, 205
    - Szukaj, 29
    - Windows.ShareTarget, 57
    - Windows.File, 57
    - Windows.Launch, 57
  - kontrakty
    - systemowe, 57
    - Windows 8, 192
  - kontroler Kinect, 25
  - kontrolka
    - AppBar, 115
    - ContentControl, 85
    - ExtendedSplashScreen, 141
    - Grid, 112
    - GridView, 88, 89
    - ItemDisplay, 90
    - ListBox, 81, 92, 107
    - ListView, 91
    - ProgressRing, 141
    - RichTextBlock, 204
    - ScrollView, 86
    - SemanticZoom, 101, 102
    - Slider, 66, 69
    - SplashPage, 180
    - StackPanel, 87
    - Storyboard, 99
    - UserControl, 112
    - WrapGrid, 81, 83
  - kontrolki wspólne, 92
  - konwerter
    - BooleanNegationConverter, 45
    - BooleanToVisibilityConverter, 45
  - konwertery wartości, 72
- ## L
- LayoutAwarePage, 45
  - Likeness Jeremy, 17
  - LINQ, Language Integrated Query, 89, 156
  - lista kontrolek, 92
- ## Ł
- ładowanie
    - boczne, 245, 247
    - własnego formatu danych, 208
    - zawartości, 207
- ## M
- MAC, Message Authentication Code, 164
  - magazyn
    - Local, 137, 144
    - Roaming, 137
    - Temporary, 137
  - manifest, 48
    - Application UI, 48
    - Capabilities, 49
    - Declarations, 49
    - Packaging, 49
  - manipulacje, 105
  - mapa bitowa, 55
  - maszyna wirtualna, 40, 43
  - mechanizm IoC, 229
  - MEF, Managed Extensibility Framework, 222
  - menedżer
    - pakietów, 203
    - zadań, 128
  - menu kontekstowe, 111

metadane, 27, 59  
 metadane komponentu WinRT, 60  
 metoda  
   Activate, 196  
   AddToSchedule, 182  
   Begin, 77  
   Convert, 72  
   ConvertBack, 72  
   CreateCompletionOption, 144  
   CreateContainer, 139  
   DoFactorialExample, 151  
   ExtendedSplashScreen\_Loaded, 133  
   Focus, 107  
   GoBack, 135  
   Initialize, 141  
   LoadBitmap, 53  
   LoadLiveGroups, 154  
   Navigate, 135  
   OnLaunched, 141  
   OnSuspending, 131  
   RaisePropertyChanged., 71  
   Register, 65  
   Resolve, 229  
   ShowAsync, 111  
   TestInitialize, 229  
 metody  
   asynchroniczne, 149, 151  
   pomocnicze, 71  
   pomocnicze IO, 152  
 MFC, Microsoft Foundation Class, 21  
 model widoku, 219  
 modele biznesowe, 237  
 MS-DOS Executive, 20  
 MSIL, Microsoft Intermediate Language, 22  
 MSTest, 227  
 MVC, model-widok-kontroler, 214  
 MVVM, 213  
   model, 217  
   widok, 218

## N

naciśnięcie wskaźnika, 104  
 narzędzie, 31  
   Blend, 32, 39  
   ILDASM, 59  
   Visual Studio 2012, 31, 39  
 naturalny interfejs użytkownika, NUI, 19, 24  
 nawigacja, 134, 136, 138  
 NGWS, Next Generation Windows Services, 22  
 niebieski stos, 37  
 NTFS, 41  
 NUI, 24

## O

obiekt  
   Binding, 71  
   BlogGroup, 147  
   Frame, 135  
   HttpClient, 188  
   HttpResponseMessage, 158  
   PopupMenu, 111  
   Task, 151  
   UICommand, 111  
   UserControl, 112  
   WritableBitmap, 50  
 obiekty  
   ContentControl, 85  
   Style, 78  
 obrót, 103  
 obsługa  
   asynchroniczności, 12  
   błędów, 146  
   języków programowania, 12  
   karty graficznej, 34  
   klawiatury, 107, 108  
   myszy, 106  
   OData, 167  
   OnLaunched, 130  
   urządzeń wejściowych, 64  
   zdarzenia PointerPressed, 65  
   zdarzeń użytkownika, 103  
 OData, Open Data Protocol, 167

odczyt

  czujnika światła, 123  
 danych, 144  
 danych o produktach, 240  
 informacji o lokalizacji, 121  
 inklinometru, 122  
 orientacji, 124  
 położenia kompasu, 121  
 wartości z  
   przyspieszeniomierza, 120  
 żyroskopu, 122  
 okno  
   dialogowe, 115  
   dialogowe potwierdzenia, 115  
   ILDASM, 58  
   Immediate Window, 173  
   NuGet Package Manager, 210  
   Solution Explorer, 246  
   udostępniania, 206  
   ustawień, 12  
   wysuwane, 177  
 opcja  
   Przytnij, 178  
   Remote Machine, 132  
   Search Contract, 195  
 opcje kafelka, 172  
 operacja  
   FailIfExists, 144  
   GenerateUniqueName, 144  
   OpenIfExists, 144  
   ReplaceExisting, 144  
 operacje asynchroniczne, 150  
 osadzanie zasobów, 153

## P

pakiet App Certification Kit, 242  
 panel  
   informacyjny, 211  
   Launch, 57  
   Share, 50  
   Sklepu Windows, 246  
   Start, 191  
   Udostępnianie, 29, 200, 203  
   Urządzenia, 29  
   Ustawienia, 29, 119  
   wirtualizujący, 81  
   Wyszukiwanie, 194

parametry wiązania, 70

pasek

- aplikacji, 112, 114, 116
- paneli, 178, 194
- przewijania, 87

PCL, 220, 232

pełna instalacja, 41

pierwsze uruchomienie, 145

plik

- \_sessionState.xml, 140
- About.xaml, 118
- App.xaml, 77
- App.xaml.cs, 50, 115, 119, 130
- ApplicationCommands.xaml.cs, 114
- BlogDataSource.cs, 154
- Blogs.js, 154
- ExtendedSplashScreen.xaml.cs, 133, 135
- GroupedItemsPage.xaml, 98–102
- ImageHelper, 58
- ItemDisplay.xaml, 88
- MainPage.xaml, 47
- MainPage.xaml.cs, 50, 54, 107, 162
- Package.appxmanifest, 48, 117
- SplashPage.xaml.cs, 145
- SplashScreen.xaml.cs, 172
- StandardStyles.xaml, 77, 113
- TinylocTests.cs, 227
- ViewboxExample.xaml, 88
- Windows.Media.winmd, 59

pliki

- WINMD, 59
- wynikowe, 58

PLM, Process Lifetime Management, 128, 134, 141

pobieranie, 140

- danych, 54
- treści, 205

POCO, Plain Old CLR Object, 69, 72

podpisywanie danych, 162

polecenie

- Anuluj, 115
- Informacje, 119

OK, 115

- przypięcia grupy, 177
- Uprawnienia, 119, 179

powiadomienia

- wypychane, 187
- wyskakujące, 179, 183

powiadomienie o zmianie wartości, 71

powiększanie semantyczne, 101

powolne operacje, 28

prefiksy

- ms-appdata, 152
- ms-appx, 154

procedura obsługi zdarzenia, 111

proces, 129

proces publikacji, 245

profile mobilne, 12

programowanie sterowane testami, TDD, 226

Project Natal, 25

projekcje, 27, 157

projektowanie Windows 8, 31

protokół

- Atom, 159
- OData, 167
- SOAP, 164

przechowywanie danych, 137

przechwytywanie obrazu, 51

przeciąganie, 103, 109

przeglądanie elementów

- przechowywania, 55

przejścia, 100

przelot, 55

przenośna biblioteka klas, PCL, 220, 232

przestrzeń nazw

- Windows.ApplicationModel.DataTransfer, 200
- Windows.UI.StartScreen, 177
- Windows.UI.Xaml.Controls, 92
- Wintello, 154

przesunięcie wskaźnika, 104

przesuwanie, 103, 109

przetwarzanie danych WWW, 157

przyciąganie, 28

przycisk

- cofnięcia, 135
- Zapisz obraz, 64

przygotowanie aplikacji, 241

przyspieszeniomierz, 120

przytrzymanie, 103

publikacja aplikacji, 233

pulpit, 30

## R

RCW, Runtime Callable Wrapper, 36

RDF, Resource Description Framework, 159

rejestracja strony w panelu, 119

rejestracja zdarzenia, 148

rejestrowanie właściwości, 65

reklamy, 240

REST, 167

rezultaty testów jednostkowych, 228

RichTextColumns, 46

rodzaje API WinRT, 27

rodzaje dostępu

- pliki aplikacji, 137
- ustawienia aplikacji, 137

rozciąganie zawartości, 86

rozdzielczość ekranu, 28

rozmiar kafelka, 174

rozszerzenia Windows 8, 193

rozszerzenie .winmd, 27, 59

równoległe systemy, 41

## S

schemat

- cyklu życia aplikacji, 134
- wiązania danych, 69

semantyczne powiększanie, 101, 236

serializator, 144

serie ujęć, 74

SHA, Secure Hash Algorithm, 164

silnik

- Chakra, 27
- Trident, 27

Silverlight, 23  
 Silverlight 5, 36  
 skalowanie, 28  
 Sklep Windows, 12, 233, 235  
 skrót klawiszowy  
   Alt+Enter, 179  
   Alt+F4, 52  
   Ctrl+Alt+I, 173  
   Ctrl+Alt+O, 64, 131  
   Ctrl+F5, 55  
   Ctrl+R, 227  
   Ctrl+Shift+A, 130  
   Shift+F5, 52  
   Windows+C, 194  
   Windows+Q, 194  
 słownik stylów StandardStyles, 46  
 słowniki zasobów, 77  
 słowo kluczowe  
   async, 53, 150  
   await, 53, 149  
   Binding, 69  
   new, 148  
 SOAP, Simple Object Access  
   Protocol, 164  
 sortowanie, 157  
 sposoby skalowania, 88  
 stan, 98  
   procesu, 129  
   Snapped, 99  
 standard ECMA-335, 27  
 StandardStyles, 46  
 stany puste, 99  
 statyczny zasób, 73  
 stos komponentów aplikacji  
   .NET Framework, 23  
   Win32, 22  
   Windows 8, 27  
 strona informacyjna, 117  
 strumienie, 160  
 strzałka =>, 152  
 stuknięcie, 103  
 styl, 76  
   jawny, 78  
   niejawny, 77  
   SettingsBackButtonStyle, 118  
 SuspensionManager, 46  
 symbol, 114  
 symulator, 95, 97

symulator Sklepu Windows, 239  
 symulowanie opcji  
   licencjonowania, 238  
 szablon, 28  
   Aplikacja podzielona, 46  
   Aplikacja siatki, 45  
   Pusta aplikacja, 45  
 szablony  
   kafelków, 170  
   powiadomień, 181  
   projektów C#, 45  
   wbudowane, 32  
 szczypanie, 103  
 szyfrator strumienia, 162  
 szyfrowanie, 162

## Ś

środowisko  
   Desktop, 36  
   programowania, IDE, 22  
   Windows Store, 36

## T

tablice bajtów, 160  
 TDD, 226  
 technologia  
   .NET Framework, 14  
   CLR, 14  
   XAML, 14  
 testowanie aplikacji, 223–226, 244  
 testy jednostkowe, 226  
 TPL, Task Parallel Library, 149  
 tryb  
   debugowania, 132, 173  
   manipulacji, 106  
 tryby wprowadzania danych, 104  
 tworzenie  
   pliku, 139  
   kafelka, 172  
   łączy, 177  
   metod asynchronicznych, 151  
   paneli, 191  
   pierwszej aplikacji, 44  
   polecenia OK, 115  
   powiadomień, 180  
   statycznego zasobu, 73  
 UI, 47, 61

typy  
   danych, 201  
   kafelków, 169  
   kolekcji, 155  
   nawigacji, 134  
   przechowanych danych, 137

## U

udostępnienie, 200  
   adresu URL, 204  
   artykułu, 209  
   fragmentu tekstu, 205  
   zawartości, 159  
 UI, 47, 62, 214  
 układ, 78, 95  
   Canvas, 79  
   GridView, 90  
   listy, 91  
   VariableSizedWrapGrid, 84  
   WrapGrid, 83  
 uprawnienia aplikacji, 158  
 urządzenia DLNA, 191  
 usługa WNS, 183  
 usługi sieciowe, 164  
 ustawienia, 12, 209  
   aplikacji, 143  
   powiadomień, 180  
   przechowywania danych, 138  
   symbolu, 114  
 uwierzytelnianie dla WNS, 186  
 użycie  
   kanału OData, 167  
   kontraktu Search, 189

## V

VB, Visual Basic, 21, 35  
 VB.NET, 35  
 View Code, 48  
 Visual Studio 2012, 32, 39  
   Premium, 43  
   Professional, 43  
   Test Professional, 43  
   Ultimate, 43  
 VSM, Visual State Manager, 98,  
 100, 214

## W

wartości ManipulationModes, 105  
 wątek, 148  
 wątki aplikacji, 129  
 weryfikacja aplikacji, 243  
 WF, Workflow Foundation, 61  
 wiązanie  
   danych, 69, 166  
   danych i konwerterów, 73  
   dwukierunkowe, 72  
 widok, 95  
   Menedżera zadań, 128  
   pomniejszony, 101  
   przyciągnięty, 98  
   w MVVM, 219  
   ZoomedInView, 102  
 wielowątkowość, 148  
 Win32, 19  
 Windows 7, 26  
 Windows 95, 20  
 Windows RT, 37  
 Windows Runtime, WinRT, 9–12, 26, 35  
 Windows Store, 12, 19, 26  
 własne dane, 203  
 własny ekran startowy, 140  
 właściwości  
   CLR, 66, 67  
   dołączane, 67  
   klasy Timeline, 74  
   zależne, 65  
   zasobu, 154  
 właściwość  
   automatyczna, 113  
   Content, 158  
   Setting, 182  
   tag, 174  
   zależna DataContext, 66  
   ZIndex, 79  
 włączanie powiadomień, 181  
 WNS, Windows Notification Service, 183, 189  
 WPF, 36, 214  
 WSDL, Web Services  
   Description Language, 165  
 wskaźnik numeryczny, 175

wskaźniki, 174  
 wstrzykiwanie zależności, 222  
 wstrzymanie aplikacji, 131  
 wybór symulatora, 96  
 wymiary komórek, 80  
 wyniki wyszukiwania, 198  
 wyrażenia regularne, 158  
 wyrażenie lambda, 152  
 wysyłanie aplikacji, 242  
 wyszukiwanie, 199  
   aplikacji, 235  
   programu Notatnik, 194  
   w Dictionary.com, 195  
   w Internet Explorerze, 194  
 wywołanie  
   asynchroniczne, 150  
   PostToCloud, 188  
 wznowienie aplikacji, 133  
 wzorce projektowania UI, 214  
 wzorzec MVVM, 213, 215

## X

XAML, Extensible Application Markup Language, 13, 61  
 XAML dla układu  
   Grid, 80  
   VariableSizedWrapGrid, 84  
   WrapGrid, 83  
 XML dla kafelka, 173

## Z

zakładka Application UI, 117  
 zakończenie działania aplikacji, 133  
 zalecenia  
   dotyczące kafelków, 174  
   dotyczące paska aplikacji, 115  
   dotyczące wskaźników, 176  
 zapis  
   danych, 144  
   obrazu, 53  
 zapisywanie w pamięci, 137  
 zapytanie LINQ, 89, 138, 156, 199  
 zarządzanie  
   aplikacjami, 28  
   PLM, 134

stanami, 99  
 wywołaniami, 149  
 zasady projektowania, 31  
 zasoby, 77  
 zawartość WWW, 157  
 zdalny komputer, 132  
 zdarzenia  
   kierowane, 63  
   manipulacji, 105  
   nawigacji, 136  
   wskaźników, 103  
 zdarzenie  
   Activated, 182  
   Click, 64  
   CollectionChanged, 156  
   CommandRequested, 119  
   DataRequested, 200  
   Dismissed, 182  
   DoubleTapped, 111  
   Navigating, 136  
   OnNavigatedTo, 102  
   OnNavigatingFrom, 136  
   OnShareTargetActivated, 50  
   PointerEntered, 104  
   PointerExited, 104  
   PointerPressed, 64, 65  
   PropertyChanged, 71, 156  
   Resuming, 133  
   SelectionChanged, 204  
   ViewStateChanged, 99  
 złączenia, 157  
 znacznik msApplication-  
   PackageFamilyName, 234  
 zręby, 232  
 zwolnienie wskaźnika, 104

## Ż

źródło treści, 200  
 źródło udostępniania, 200

## Ź

żądania HTTP, 158  
 życie aplikacji, 129  
 żyroskop, 122



# PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄŻKI**
- 3. ZBIERAJ PROWIZJĘ**

Zmień swoją stronę WWW  
w działający bankomat!

**Dowiedz się więcej i dołącz już dzisiaj!**

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**




# Poznaj kompletny proces przygotowywania i publikowania aplikacji w Sklepie Windows!

Windows 8 na dobre zadomowił się na komputerach użytkowników. Jego nowy interfejs budzi wiele kontrowersji — jedni go wychwalają, drugim ciężko się do niego przyzwyczaić. Pewne jest jednak, że to podejście do interfejsu użytkownika stanie się standardem na kolejne lata. Dlatego już teraz warto poznać zasady tworzenia aplikacji dla nowej platformy.

W tym zadaniu pomoże Ci ten znakomity podręcznik, który jako pierwszy przedstawia kompletny proces tworzenia aplikacji dla Windows 8 — od projektu aż po publikację w Sklepie Windows. W trakcie lektury zobaczysz, jak wykorzystać potencjał najnowszej wersji Visual Studio 2012. Ponadto poznasz wzorzec MVVM, bibliotekę WinRT oraz sposoby wykorzystywania zasobów sieci w Twojej aplikacji. Niezwykle istotny jest rozdział poświęcony testowaniu i debugowaniu aplikacji — tylko bezbłędna aplikacja ma szansę odnieść sukces w Sklepie Windows. Jak sprzedać w nim aplikację? Tego też dowiesz się z tej książki. Warto ją mieć!

Bądź na czasie i:

- przygotuj aplikację dla Windows 8
- poznaj możliwości Visual Studio 2012
- wykorzystaj bibliotekę WinRT
- opublikuj aplikację w Sklepie Windows
- odnieś sukces!

 Addison-Wesley  
Pearson Education

**helion.pl**  
księgarnia  
internetowa

Nr katalogowy: 14213

 Księgarnia internetowa  
<http://helion.pl>

 Zamówienia telefoniczne:  
**0 801 339900**  
 **0 601 339900**

 **Helion**

Sprawdź najnowsze promocje:  
• <http://helion.pl/promocje>  
Książki najchętniej czytane:  
• <http://helion.pl/bestsellery>  
Zamów informacje o nowościach:  
• <http://helion.pl/nowosci>

Helion SA  
ul. Kościuszki 1c, 44-100 Gliwice  
tel.: 32 230 98 63  
e-mail: [helion@helion.pl](mailto:helion@helion.pl)  
<http://helion.pl>

Informatyka w najlepszym wydaniu

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

cena 49,00 zł

ISBN 978-83-246-7062-8



9 788324 670628

PEARSON