

Jacek Matulewski

MVVM i XAML

w Visual Studio 2015

Twórz doskonałe aplikacje zgodne ze wzorcem MVVM z użyciem języka XAML!

- Wzorzec MVVM, czyli jak zespołowo wytwarzać aplikacje, które można łatwo testować
- Budowanie interfejsu w XAML, czyli moc i elegancja
- Aplikacje uniwersalne, czyli jak użyć jednego kodu dla wielu platform

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.

Opieka redakcyjna: Ewelina Burska

Projekt okładki: Studio Gravite/Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: 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/xamlmv>
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
<ftp://ftp.helion.pl/przyklady/xamlmv.zip>

ISBN: 978-83-283-1018-6

Copyright © Helion 2016

Printed in Poland.

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

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

Spis treści

- Część I Wzorzec MVVM. Podstawy XAML 7**
 - Rozdział 1. Szybkie wprowadzenie do XAML 9**
 - Wzorzec widoku autonomicznego 9
 - Tworzenie projektu 10
 - Projektowanie interfejsu 11
 - Kilka uwag na temat kodu XAML opisującego interfejs okna 15
 - Zdarzenia 16
 - Własności 20
 - Zapisywanie i odtwarzanie stanu aplikacji 21
 - Rozdział 2. Wzorzec MVVM 25**
 - Model 25
 - Widok 26
 - Model widoku 27
 - Rozdział 3. Implementacja modelu i model widoku 29**
 - Model 29
 - Warstwa dostępu do danych 30
 - Model widoku 31
 - Alternatywne rozwiązania 33
 - Ratujemy widok 35
 - Zadania 36
 - Rozdział 4. Wiązanie danych (data binding) 37**
 - Instancja modelu widoku i kontekst danych 37
 - Alternatywne rozwiązanie 38
 - Wiązanie pozycji suwaków i koloru prostokąta 39
 - Zmiany w code-behind 40
 - Implementacja interfejsu INotifyPropertyChanged 41
 - Powiadomienia w alternatywnych modelach widoku 44
 - Interfejs INotifyDataErrorInfo 50
 - Klasa ObservedObject 50
 - Rozdział 5. Konwersja danych w wiązaniu 53**
 - Prosta konwersja typów 53
 - Konwersja klas Color i SolidColorBrush 55
 - Multibinding 56

Wiązanie między kontrolkami	57
Konwersje „wbudowane”	60
Zadania	60
Rozdział 6. Polecenia (commands)	61
Interfejs ICommand	61
Przycisk uruchamiający polecenie	62
Sprawdzanie możliwości wykonania polecenia	65
Resetowanie stanu suwaków po naciśnięciu klawisza	66
Klasa RelayCommand	67
Zdarzenia a polecenia	69
Zamykanie okna	71
Zadanie	72
Rozdział 7. Zachowania, własności zależności i własności doczepione	73
Zachowania (behaviors)	73
Własność zależności (dependency property)	75
Własność doczepiona (attached property) i zachowanie doczepione (attached behavior)	79
Zadania	81
Rozdział 8. Testy jednostkowe	83
Testy jednostkowe w Visual Studio 2013	84
Projekt testów jednostkowych	84
Przygotowania do tworzenia testów	85
Pierwszy test jednostkowy	85
Testy jednostkowe w Visual Studio 2015	86
Uruchamianie testów	88
Testy wielokrotne	89
Dostęp do prywatnych pól testowanej klasy	90
Atrapy obiektów (mock objects)	92
Testowanie konwersji	95
Testowanie wyjątków	96
Rozdział 9. Powtórzenie	99
Model	99
Prototyp widoku	100
Model widoku	102
Wiązanie	103
Konwerter	104
Wzorec MVVM	106
Zadania	107
Część II Zaawansowane zagadnienia budowania interfejsu w XAML	109
Rozdział 10. Budowanie złożonych kontroltek	111
Konfiguracja przycisku w podoknie Properties	111
Pędzle	115
Formatowanie tekstu na przycisku	118
StackPanel — liniowe ułożenie elementów	119
Projektowanie własnych kontroltek	121

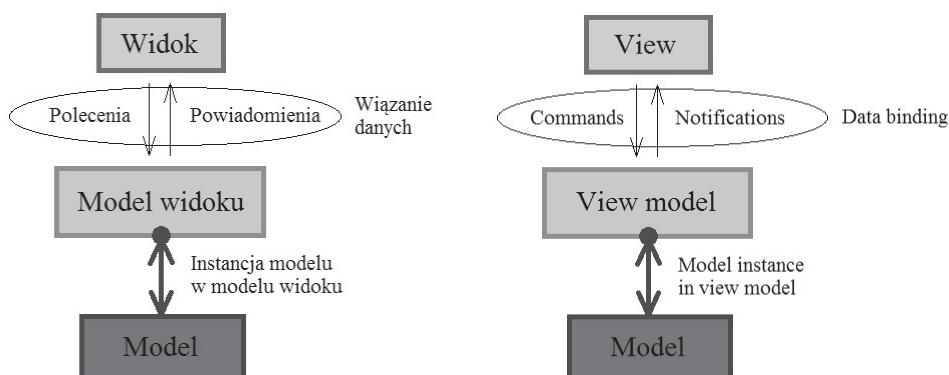
Rozdział 11. Style	123
Siatka i wiele kontrolek	123
Zasoby okna	125
Style	127
Wyzwalacze	129
Zasoby aplikacji	130
Rozdział 12. Transformacje i animacje	133
Transformacje kompozycji i renderowania	133
Uruchamianie transformacji w wyzwalaczu stylu	140
Animacje	142
Animacja w stylu	144
Funkcje w animacji	145
Animacja koloru	147
Rozdział 13. Szablony kontrolek	149
Rozdział 14. Zdarzenia trasowane (routed events)	153
Pojedyncza kontrolka	153
Zagnieżdżanie przycisków	155
Kontrola przepływu zdarzeń trasowanych	156
Przerwanie kolejki	158
Bulgotanie (bubbling) i tunelowanie (tunneling)	158
Dynamiczne tworzenie przycisków zagnieżdżonych	160
Rozdział 15. Kolekcje w MVVM i XAML	163
Model	163
Przechowywanie danych w pliku XML	167
Model widoku zadania	169
Kolekcja w modelu widoku	172
Prezentacja kolekcji w widoku. Szablon danych (data template)	175
Style elementów kontrolki ListBox	177
Konwertery	179
Zapisywanie danych przy zamknięciu okna	182
Modyfikacje kolekcji	184
Sortowanie	190
Zadania	192
Rozdział 16. Okna dialogowe w MVVM	193
Klasa bazowa okna dialogowego	194
Polecenia wykonywane przed wyświetleniem i po wyświetleniu okna dialogowego	196
Okno dialogowe MessageBox	199
Warunkowe wyświetlenie okna dialogowego	203
Okna dialogowe wyboru pliku	205
Łańcuch okien dialogowych	209
Okna dialogowe z dowolną zawartością	210
Zadania	214
Rozdział 17. Grafika kształtów w XAML	215
Model widoku	216
Widok	217
Zmiana kształtu okna	222
Zadania	226

Rozdział 18. Aplikacja WPF w przeglądarce (XBAP)	227
Część III Aplikacje uniwersalne (Universal Apps)	231
Rozdział 19. Kod współdzielony	233
Projekt	234
Kod współdzielony: model i model widoku	235
Konwertery	237
Zadanie	238
Rozdział 20. Warstwa widoku dla Windows 8.1	239
Widok	239
Logo aplikacji	244
Zadanie	246
Rozdział 21. Cykl życia aplikacji i przechowywanie jej stanu	247
Cykl życia aplikacji	247
Przechowywanie stanu	248
Zadanie	252
Rozdział 22. Kafelek	255
Rozdział 23. Tworzenie i testowanie pakietu AppX	259
Rozdział 24. Warstwa widoku dla Windows Phone 8.1	265
Zadania	268
Rozdział 25. Kolekcje w aplikacji mobilnej	271
Dostęp do plików w katalogu lokalnym	271
Współdzielony kod z warstwy widoku	276
Lista zadań w widoku dla Windows Phone 8.1	279
Zdarzenie CanExecuteChanged poleceń	283
Zadanie	285
Rozdział 26. Pasek aplikacji (app bar)	287
Zadania	290
Rozdział 27. Okna dialogowe w aplikacjach Windows Phone	291
Standardowe okna dialogowe	291
Okna dialogowe z dowolną zawartością w Windows Phone	301
Zadania	305
Rozdział 28. Aplikacje uniwersalne w Windows 10	307
Skorowidz	315

Rozdział 2.

Wzorzec MVVM

Opisany w poprzednim rozdziale projekt będziemy teraz krok po kroku modyfikować tak, żeby jego architektura stała się zgodna ze wzorcem MVVM. We wzorcu tym zakłada się obecność trzech warstw: modelu, modelu widoku i widoku (rysunek 2.1). W najprostszym przypadku, takim jak w naszej aplikacji, poszczególne warstwy mogą składać się tylko z jednej klasy, ale zwykle jest ich więcej.



Rysunek 2.1. Warstwy aplikacji we wzorcu MVVM (z lewej polska, a z prawej angielska terminologia)

Model

Funkcja warstwy modelu jest najbardziej intuicyjna — z grubsza odpowiada modelom w innych wzorcach projektowych, chociażby w klasycznej dwuwarstwowej architekturze model-widok lub we wzorcach MVC i MVP. Dobrym pomysłem jest tworzenie tej warstwy w metodologii projektowania domenowego (ang. *domain-driven design*, DDD). W wielkim uproszczeniu oznacza to, że projektujemy zbiór klas składających się na model razem z ekspertem w dziedzinie, której program ma dotyczyć. Często jest to klient lub osoba przez niego wskazana. Wówczas należy uważnie słuchać słownictwa, jakiego ów ekspert używa, bo często stosowane przez niego rzeczowniki są dobrymi

kandydatami na nazwy podstawowych klas modelu. Z kolei czasowniki towarzyszące tym rzeczownikom będą prawdopodobnie nazwami kluczowych metod. Przy czym w DDD nie chodzi oczywiście tylko o wybieranie nazw klas i metod, a przede wszystkim o ich zawartość i wyznaczenie relacji między klasami. Ma ona odzwierciedlać relacje pojawiające się w języku używanym przez eksperta. To oczywiście trywializacja, ale dobrze oddaje ideę DDD.

Modele domenowe powinny być możliwie proste i „lekkie”. Nie powinny korzystać z żadnych konkretnych mechanizmów platformy .NET — najlepiej, gdyby jedyną używaną w nich przestrzenią nazw była przestrzeń `System`¹. W tym podejściu klasy modelu powinny stanowić tylko proste nośniki danych przekazywanych z bazy danych lub innego źródła danych do wyższych warstw aplikacji. Klasy modelu nie mogą, i to jest bardzo ważne, znać żadnych szczegółów dotyczących owych wyższych warstw — powinny być całkowicie autonomiczne. W takim podejściu klasy modelu muszą być bardzo proste, a tym samym łatwe do testowania². Klarowne są też relacje między nimi.

Kluczowy w projektowaniu warstwy modelu jest podział odpowiedzialności — należy jasno ustalić, za co odpowiedzialna jest która klasa. Część odpowiedzialności może, lub nawet powinna, być wydzielona do osobnych modułów w warstwie modelu. Za zapis danych można uczynić odpowiedzialną podwarstwę dostępu do danych (ang. *data access layer*, DAL), która na przykład w postaci klasy statycznej przyjmuje instancje klas domenowych i zapisuje ich stan. Podobnie logika modelu może być wydzielona do osobnego modułu tak zwanej logiki biznesowej (ang. *business logic layer*, BLL), która operuje na instancjach domenowych klas modelu.

Widok

Widok odpowiedzialny jest za kontakt z użytkownikiem. W WPF, a także w aplikacjach Windows Phone i WinRT, widokiem jest kod XAML opisujący graficzny interfejs użytkownika (ang. *graphical user interface*, GUI). Z widokiem związana jest klasa okna, w której w poprzednim rozdziale umieszczaliśmy metody zdarzeniowe. Tworzy ona tak zwany kod zaplecza widoku, czyli *code-behind*. Zgodnie z zaleceniami wzorca MVVM kod ten powinien być ograniczony do minimum, a najlepiej, żeby go w ogóle nie było. W tym sensie wzorzec MVVM całkowicie odwraca wzorzec widoku autonomicznego. Głównym powodem unikania kodu C# w warstwie widoku, a przynajmniej w klasie okna, jest to, że kod ten, jako silnie związany z kontrolkami, jest trudny do przetestowania. Ponadto zanurzenie logiki prezentacyjnej w widoku znacząco utrudnia współpracę między projektantami interfejsu tworzącymi widok a programistami odpowiedzialnymi za niższe warstwy aplikacji. Zmniejsza też elastyczność projektu, utrudniając tym samym jego zmiany.

¹ Klasy tego typu nazywane są POCO, od ang. „*plain-old*” *CRL objects*. To popularne określenie w slangu programistów C#.

² Testowanie klas POCO nie ma jednak sensu, jeżeli zawierają one same własności.

Model widoku

Model widoku jest abstrakcją widoku. Jeżeli możemy sobie wyobrazić kilka wariantów graficznego interfejsu użytkownika naszej aplikacji, dla różnych środowisk i platform, to model widoku w tych wszystkich przypadkach powinien pozostawać taki sam. Myśląc przez analogię: możemy sobie wyobrazić różne stoły, różnej wielkości i o różnych kształtach, z trzema lub czterema nogami. Nie zmienia to jednak definicji stołu jako miejsca, przy którym można usiąść i coś na nim położyć. Podobnie wiele może być projektów widoku. Ale model widoku musi być jak definicja stołu, jego zapisana idea — powinien być jak najprostszy, lecz kompletny. Powinien wobec tego zawierać tylko to, co konieczne do określenia, do czego widoki mają być użyte. Warto podjąć wysiłek, żeby doprowadzić kod modelu widoku do jak najwyższego poziomu abstrakcji. Z powyższych górnolotnych rozważań wynika, że najlepszym sprawdzianem poprawności modelu widoku są zmiany wprowadzane w widoku. Tych w trakcie rozwijania projektu zwykle nie brakuje. Jeżeli model widoku jest dobrze zaprojektowany, takie zmiany widoku powinny się odbyć bez jego modyfikacji. Pamiętajmy jednak, że — jak wiele dobrych praktyk w informatyce — jest to raczej cel, do którego dążymy, niż twarde wymaganie, stawiane osobie projektującej model widoku.

Funkcją modelu widoku jest udostępnienie widokowi instancji klas z warstwy modelu (na rysunku 2.1 odpowiada to ruchowi do góry) oraz zmienianie stanu tych instancji w wyniku działań użytkownika wykrytych w warstwie widoku (ruch w dół). W tym drugim przypadku model widoku odpowiedzialny jest między innymi za weryfikację przekazywanych danych. Model widoku pełni więc rolę pośrednika między warstwami modelu i widoku, a jednocześnie adaptera dla przekazywanych danych. Owo pośredniczenie najczęściej odbywa się w taki sposób, że obiekty modelu są prywatnymi polami modelu widoku. Model widoku udostępnia je lub ich części w swoich własnościach, jest wobec tego świadomy warstwy modelu, nie powinien być natomiast świadomy warstwy widoku — to widok powinien być świadom modelu widoku. Połączenie między modelem widoku a widokiem jest zwykle bardzo „luźne”. Oparte jest nie na odwołaniach w kodzie C#, lecz na wiązaniach danych umieszczonych w kodzie XAML. To luźne wiązanie ułatwia niezależną pracę nad widokiem i modelem widoku i znakomicie ułatwia wprowadzanie zmian w poszczególnych warstwach, z całkowitym ich przebudowywaniem włącznie. Ta druga zaleta jest szczególnie warta docenienia, choć jest ona w większym lub mniejszym stopniu zaletą wszystkich wzorców z wyraźnie rozdzielonymi warstwami (modułami).

W modelu widoku zapisana jest cała logika prezentacyjna określająca procedury kontaktu z użytkownikiem z uwzględnieniem weryfikacji danych. Mimo tego pozostaje łatwa do testowania, nie ma w niej bowiem odwołań do kontrolerek ani założonej bezpośredniej interakcji z użytkownikiem.



Wskazówka

Doskonale zdaję sobie sprawę, że dla osób, które nie miały jeszcze kontaktu ze wzorcem MVVM albo chociażby z MVP lub MVC, większość powyższych zdań o modelu widoku jest trudna do zrozumienia. Zadaniem kolejnych rozdziałów z pierwszej części książki będzie wyjaśnienie tego na konkretnym przykładzie. Po przeczytaniu dalszych rozdziałów warto wrócić do niniejszego i przeczytać go jeszcze raz, w całości lub przynajmniej w części dotyczącej modelu widoku. To powinno pomóc poukładać sobie w głowie wiedzę o MVVM przedstawioną w pierwszej części.

W przypadku aplikacji *KoloryWPF* modelem może być prosta klasa opisująca kolor, zawierająca tylko trzy składowe typu `byte`. Odpowiedzialność za zapis stanu modelu pozostawimy osobnej klasie statycznej należącej do warstwy modelu. Prostota naszej aplikacji spowoduje, że model widoku będzie z początku równie prosty i w istocie bardzo podobny do samego modelu. Z czasem dodamy do niego jednak elementy charakterystyczne dla klas modelu widoku, między innymi polecenia i mechanizm powiadomień. A ponieważ podstawowym celem aplikacji jest możliwość kontrolowania trzech składowych koloru, model widoku musi udostępniać własności reprezentujące te składowe. Oprócz tego wyposażymy go w metodę, którą potem przekształcimy w tak zwane polecenie, umożliwiające zapis stanu aplikacji (czyli *de facto* stanu modelu).

To nie jest oczywiście jedyna architektura, jaką można sobie wyobrazić dla tej aplikacji. Dobrym modelem mogłaby być przecież klasa `Properties.Settings` stworzona przez Visual Studio w momencie określania ustawień aplikacji. Przy takim założeniu naszym jedynym zadaniem pozostaje napisanie modelu widoku, który tę klasę udostępniłby widokowi. Można również rozważyć klasę `System.Windows.Media.Color`, jako klasę modelu, ale nie uważam, żeby korzystanie z klas przeznaczonych do budowania interfejsu było dobrym pomysłem na tworzenie modelu. Dlatego pozostaniemy przy rozwiązaniu „kanonicznym”, lecz pamiętając, że wzorzec MVVM pozwala na pewne wariacje.

Ostrzegałem już, że aplikacja, którą od tego momentu będziemy przebudowywać, jest bardzo prosta. W kontekście uczenia się wzorca MVVM to jest jednak moim zdaniem zaleta. Brak szczegółów związanych z bardziej skomplikowanym projektem pozwoli Czytelnikowi łatwiej dostrzec istotę wzorca.

Skorowidz

A

animacja, 142, 143, 145, 146, 215

ColorAnimation, 147

w stylu, 144

z użyciem ramek kluczowych, 148

aplikacja

AppX, *Patrz:* AppX

cykl życia, 247, 248

dynamika, 16

interfejs, *Patrz:* interfejs

język domyślny, 245

lista, 255

logo, 244, 245

mobilna, *Patrz:* aplikacja na urządzenia

przenośne

na smartfon, 247

na tablet, 247

na urządzenia przenośne, 247, 248, 271

dostęp do pamięci, 271

pasek, 287, 292, 295, 301

plik, *Patrz:* plik

stan, 248

odtwarzanie, 21, 23

przywracanie, 251

resetowanie, 251

wstrzymywanie, 252

zapisywanie, 21, 23, 247, 249, 313

tworzenie, 10, 11

uniwersalna, 231, 234, 239, 279

Windows 10, 307

uruchamianie w przeglądarce, 227, 230

ustawienia lokalne, 249, 250

Windows Phone, 291

wstrzymywanie, 247, 248, 281

wznawianie, 247, 248

zamykanie, 183, 184, 247, 248, 313

zasoby, 130

app bar, *Patrz:* aplikacja pasek

AppX, 259

instalowanie, 263

testowanie, 261, 262

tworzenie, 260

atrybut

DataContext, 38

ExpectedException, 97

Fill, 20

Height, 15

Icon, 288

Label, 288

RelativePanel.AlignLeftWith, 311

RelativePanel.Below, 311

StringFormat, 278, 279

TargetType, 127

TextDecoration, 279

Title, 15

Width, 15

x:Class, 15

x:Name, 15

xmlns, 15

attached property, *Patrz:* własność doczepiona

B

behavior, *Patrz:* zachowanie

biblioteka

Microsoft.Expression.Interaction.dll, 73

Newtonsoft.JSON, 168

System.Windows.Interactivity.dll, 73

BLL, 26

bubbling, *Patrz:* bulgotanie

business logic layer, *Patrz:* BLL

bulgotanie, 158

C

checkbox, *Patrz:* pole opcji
code-behind, 26, 40, 61, 69, 76, 153, 188, 189
czas, 216, 217

D

DAL, 26, 30
dane
 szablon, *Patrz:* szablon danych
 weryfikacja, 27
 wiązanie, 37, 38, 39
data, 186, 188, 189, 216, 217
data access layer, *Patrz:* DAL
data binding, *Patrz:* dane wiązanie
data template, *Patrz:* szablon danych
DDD, 25, 29, 100
domain-driven design, *Patrz:* DDD

E

ekran
 powitalny, 244, 245
 wielkość, 311
element
 AppBarButton, 288
 DoubleAnimation, 143
 MenuFlyout, 288
 Page, 228
 Setter, 127, 129, 312
emulator
 smartfona, 233
 tabletu, 233, 243
 uruchamianie, 243
etykieta, 288

F

formularz, 184, 186
funkcja
 przejęcia, 145
 wyglądania, 146

G

generic type, *Patrz:* typ parametryczny
gradient, 115, 116
graphical user interface, *Patrz:* interfejs
 użytkownika graficzny
GUI, *Patrz:* interfejs użytkownika graficzny

I

interfejs, 166
ikona, 288
interfejs, 11, 172
 ICommand, 61, 240
 IComparable, 190
 IComparer, 190
 IDataErrorInfo, 50
 IEnumerable, 166
 ImultiValueConverter, 57
 IMultiValueConverter, 56, 188
 INotifyCollectionChanged, 174
 INotifyDataErrorInfo, 50
 INotifyPropertyChanged, 41, 42, 44, 50, 169,
 171, 194, 216
 IValueConverter, 53, 237
 użytkownika graficzny, 26, 27
Internet of Things, *Patrz:* urządzenie IoT

K

kafelek, 255
 aktualizowanie, 313
 kolor tła, 255
 logo, 255
 rozmiar, 255
 szablon, 256
 wygląd, 255, 256
klasa
 App, 249, 251, 265, 313
 Application, 249
 ApplicationCommands, 69
 ApplicationData, 249
 Brush, 39
 Brushes, 237
 CommandBar, 288
 CommandDialogBox, 293
 CommandManager, 235
 ContentDialog, 301, 303
 DependencyObject, 75
 EditingCommands, 69
 EventTrigger, 69
 FileIO, 271
 FrameworkElement, 133, 194
 Freezable, 216
 Geometry, 216
 Graphics, 215
 konwersja, 55
 List, 190
 MainWindow, 41
 MediaCommands, 69
 MessageBox, 185

- MessageDialogBox, 298
 - modelu, 100
 - NavigationCommands, 69
 - NotificationDialogBox, 198, 293, 295
 - ObservedObject, 50
 - PrivateObject, 91, 92
 - RelayCommand, 67, 68, 102, 104, 185, 235
 - Shape, 215
 - SolidColorBrush, 20
 - statyczna, 93
 - StorageFile, 271
 - UIElement, 133, 194
 - Windows.Storage.KnownFolders, 271
 - XDocument, 272
 - kod
 - XAML, 26, 111, 123, 239
 - zapleczu widoku, *Patrz:* code-behind
 - kolekcja, 163
 - modyfikowanie, 184
 - w aplikacji mobilnej, 271
 - w modelu widoku, 172
 - zachowań, 74
 - konsola Xbox, 307
 - kontrolka, 111
 - DatePicker, 186, 189
 - definiowanie, 121
 - dziedziczka po Shape, *Patrz:* kształt
 - Ellipse, 215
 - Grid, 38, *Patrz też:* siatka
 - Line, 215
 - ListBox, 177, 184, 289
 - Path, 215
 - projektowana przez użytkownika, 121, 234, 239
 - Rectangle, 215
 - rozmiar, 14
 - Slider, 12
 - styl, *Patrz:* styl
 - szablon, *Patrz:* szablon kontrolki
 - TextBlock, 176
 - wiązanie, 57
 - widoku, 101
 - WPF, 75
 - konwerter, 53, 54, 55, 57, 234, 276
 - AlternationConverter, 60
 - BooleanToVisibilityConverter, 60, 278
 - BoolToBrushConverter, 237
 - BoolToVisibilityConverter, 178
 - BorderGapMaskConverter, 60
 - ColorToSolidColorBrushConverter, 96
 - DataGridLengthConverter, 60
 - definiowanie, 179
 - JournalEntryListConverter, 60
 - JournalEntryUnifiedViewConverter, 60
 - MenuScrollingVisibilityConverter, 60
 - ProgressBarBrushConverter, 60
 - ProgressBarHighlightConverter, 60
 - testowanie, 83, 95
 - wbudowany, 60
 - ZoomPercentageConverter, 60
 - kształt, 215
- L**
- lista, 156, 177, 179
 - sortowanie, *Patrz:* sortowanie
 - ListBox, *Patrz:* lista
- M**
- manifest, 259
 - menedżer stanów wizualnych, 312
 - metoda
 - CanExecute, 61, 65, 102, 103, 188, 240
 - Convert, 53, 56, 181
 - ConvertBack, 53, 56, 181
 - Execute, 61, 63, 102, 103
 - GetProperty, 91
 - OnCanExecuteChanged, 240
 - OnPropertyChanged, 103
 - OnSuspending, 251, 281
 - ScrollToBottom, 189
 - SetField, 91, 92
 - SetProperty, 91
 - ShowAsync, 303
 - Sort, 190
 - XDocument.Save, 271
 - mock object, *Patrz:* obiekt atrapa
 - model, 106
 - pasywny, 193
 - testowanie, 83
 - tworzenie, 99
 - widoku, 27, 29, 44, 46, 53, 103, 106, 169, 188, 193, 216
 - instancja, 37
 - kolekcja, *Patrz:* kolekcja w modelu widoku
 - testowanie, 83
 - tworzenie, 31, 33, 34, 35, 102
 - wiązanie widoku, 103, 104
 - multibinding, 56, 57, 279, 303
- O**
- obiekt
 - atrapa, 92, 94
 - Windows.Storage.ApplicationData.
 - ↪ Current.LocalFolder, 271

Windows.Storage.ApplicationData.
 ↪Current.LocalSettings, 249
 Windows.Storage.ApplicationData.
 ↪Current.RoamingSettings, 249
 wstrzykiwanie, 93
 okno
 dialogowe, 193, 194, 196
 łańcuch, 209
 MessageBox, 199
 w aplikacji Windows Phone, 291
 wyboru pliku, 205
 wyświetlenie warunkowe, 203
 zawartość, 210, 301
 pasek tytułu, 223
 przesuwanie, 223
 przezroczystość, 222
 operator
 ., 40
 ?, 40
 dostępu, 40

P

pasek aplikacji, *Patrz:* aplikacja pasek
 pędzel, 115, 215
 LinearGradientBrush, 115, 116, 125
 RadialGradientBrush, 116
 SolidColorBrush, 237
 plik
 App.config, 22
 App.xaml.cs, 249, 251, 313
 domyślny aplikacji, 11
 JSON, 168
 XML, 167, 168
 pojemnik RelativeLayout, 311
 pole opcji, 69, 156
 polecenie, 61
 CommandAfter, 202
 CommandBefore, 198, 202
 Create IntelliTests, 88
 Show, 198
 uruchamianie, 62, 66
 projekt
 aplikacji uniwersalnej, *Patrz:* aplikacja uniwersalna
 domyślny, 265
 współdzielony, 234, 235, 237, 265, 276
 projektowanie domenowe, *Patrz:* DDD
 przestrzeń nazw
 domyślna, 15
 local, 13, 15
 mc, 16
 Microsoft.VisualStudio.TestTools.UnitTesting, 90
 s, 186

System, 100
 System.Windows, 186
 System.Windows.Data, 53, 237
 System.Windows.Input, 186
 System.Windows.Media, 237
 Windows.UI.Xaml.Data, 237
 x, 15
 przycisk, 111, 311
 aktywny, 118
 definiowanie, 121
 tekst, 118
 wygląd, 111, 119
 zagnieżdżanie, 155, 160

R

reguła DRY, 123
 routed event, *Patrz:* zdarzenie trasowane

S

siatka, 38, 123
 smartfona emulator, *Patrz:* emulator smartfona
 pędzel, 215
 sortowanie, 190
 splash screen, *Patrz:* ekran powitalny
 stos StackPanel, 134
 styl, 127, 151
 lokalizacja, 127, 130, 131
 Surface Hub, 307
 suwak, 16, 53
 szablon
 danych, 175
 kontrolki, 149, 150, 151

Ś

ścieżka, 225
 środowisko
 Blend, 148
 projektowe Expression Blend, 16, 118

T

tabletu emulator, *Patrz:* emulator tabletu
 test
 dostęp do pól testowanej klasy, 90
 IntelliTest, 88
 jednostkowy, 83, 95, 97
 tworzenie, 85, 87, 88
 uruchamianie, 88
 Visual Studio 2013, 84, 86, 87
 konwertera, 95

- wielokrotny, 89
- wyjątku, 96
- testowanie funkcjonalne, 9
- tile, *Patrz:* kafelek
- transformacja
 - animowana, *Patrz:* animacja
 - CompositeTransform, 140
 - kompozycji, 134, 135, 137, 215
 - MatrixTransform, 140
 - obrotu, 220
 - renderowania, 135, 137
 - złożona, 140
- tunelowanie, 159
- tunneling, *Patrz:* tunelowanie
- typ
 - ApplicationDataContainer, 249
 - byte, 53
 - Comparison, 190
 - DateTime, 216
 - DependencyProperty, 79
 - double, 53, 89
 - int, 89
 - konwersja, 53, 55
 - MessageDialogBoxButton, 298
 - parametryczny, 93
 - SuspendingDeferral, 252

U

- Universal Windows Platform, *Patrz:* UWP
- urządzenie IoT, 307
- user control, *Patrz:* kontrolka projektowana przez użytkownika
- UWP, 307, 309

V

- Visual Studio, 255
- Visual Studio 2010, 84
- Visual Studio 2013, 173, 231, 233
- Visual Studio 2015, 86, 87, 103, 173, 231, 233, 307, 309

W

- warstwa, 193
 - dostępu do danych, *Patrz:* DAL
 - logiki biznesowej, *Patrz:* BLL
 - modelu, 25, 26, 29
 - widoku, *Patrz:* widok
 - dla Windows Phone 8.1, 265
- wiązanie danych, *Patrz:* dane wiązanie
- widok, 26, 106, 186, 234

- model, *Patrz:* model widoku
- tworzenie, 100
- warstwa, *Patrz:* warstwa widoku
- Windows 10, 307
- Windows 8.1, 233, 234, 242, 255
- Windows Forms, 9
- Windows Phone, 265, 291
- Windows Phone 8.1, 234, 265
- Windows Phone Runtime, 231
- Windows Phone Store, 259
- Windows Presentation Foundation, *Patrz:* WPF
- Windows Runtime, 231, 239, 282
- Windows Store, 259
- WinRT, *Patrz:* Windows Runtime
- wirtualizacja Hyper-V, 233
- własność, 20
 - Angle, 143
 - Background, 115
 - Caption, 194
 - Center, 111
 - ColumnDefinitions, 123
 - Content, 111, 210
 - Current, 249
 - DataContext, 194
 - DateTime.Now, 188, 216
 - DialogBypassButton, 204
 - DialogResult, 212
 - doczepiona, 79, 311
 - Fill, 39, 215
 - Foreground, 112, 115
 - Height, 111
 - IsDialogBypassed, 203
 - IsEnabled, 178
 - LayoutTransform, 133, 134
 - Opacity, 222
 - OriginalSource, 159
 - Page.BottomAppBar, 288
 - RenderTransform, 133, 215
 - RowDefinitions, 123
 - SelectedIndex, 188, 289
 - Source, 159
 - Stroke, 215
 - StrokeThickness, 215
 - Visibility, 178
 - Width, 111
 - WindowContent, 210
- WPF, 10
- wyjątek
 - ArgumentOutOfRangeException, 100
 - NotSupportedException, 96
 - NullReferenceException, 40
 - testowanie, 96
- wyzwalacz, 129

wzorzec

MVC, 25

MVP, 25

MVVM, 25, 83, 99, 103, 106, 188, 193, 231,
233, 234

Z

zachowanie, 73

definiowanie, 73

kolekcja, 74

zdarzenie, 16

bulgotanie, *Patrz:* bulgotanie

CanExecuteChanged, 61, 235, 283

Click, 189

CollectionChanged, 174

kontrolki, 9

Paint, 215

PreviewKeyDown, 159

PreviewMouseDown, 159

przekształcanie w polecenie, 69

RoutedEventArgs, 159

trasowane, 153

kontrola przepływu, 156

przerwanie sekwencji, 158

Window.Closed, 61

zegar, 216, 217, 227

analogowy, 218

tarcza, 222

znacznik, *Patrz:* element

znak

., 40

?, 40

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>

MVVM i XAML w Visual Studio 2015

Aplikację można budować na wiele sposobów, z użyciem różnych narzędzi. Zawsze jednak trzeba pamiętać o tym, do czego ma ona służyć, kto będzie jej używał, na jakim sprzęcie i jak długi ma być jej cykl życiowy. Jeżeli projekt jest duży lub jego czas życia został zaplanowany na lata, warto od razu zadbać o to, aby jego architektura ułatwiała współpracę wielu osób przy jego tworzeniu, późniejszą rozbudowę, testowanie najbardziej istotnych modułów i możliwość używania aplikacji w wersjach przeznaczonych dla różnych platform sprzętowych oraz systemów operacyjnych. Dobrym wyborem jest trójwarstwowy MVVM — wzorzec przeznaczony dla aplikacji WPF oraz tzw. aplikacji uniwersalnych, w których interfejs jest przygotowywany w języku XAML.

W środowisku Visual Studio 2015 możesz łatwo zbudować aplikację opartą na wzorcu MVVM i wyposażać ją we wspaniały interfejs dzięki pomocy XAML. Z tej książki dowiesz się, jak mądrze zaprojektować strukturę Twojej aplikacji, co powinno znaleźć się w poszczególnych warstwach, jak łączyć interfejs z modelem, jak zdefiniować polecenia, własności i zachowania. Zobaczysz, jak testować kod. Poznasz także aplikacje uniwersalne dla Windows 8.1 oraz Windows 10. Krótka mówiąc, zdobędziesz solidną wiedzę o konstruowaniu znakomitych, łatwych w utrzymaniu aplikacji!

- Wprowadzenie do XAML, wzorzec MVVM
- Projektowanie i implementacja modelu oraz model widoku
- Wiązania i konwersja danych w wiązaniu
- Cykl życia aplikacji i przechowywanie jej stanu
- Polecenia, zachowania, własności zależności i własności doczepione
- Testy jednostkowe
- Budowanie złożonych kontrolerek i szablony kontrolerek
- Style, transformacje i animacje
- Zdarzenia trasowane i kolekcje w MVVM
- Okna dialogowe w MVVM w aplikacjach WPF i Windows Phone
- Kafelki, grafika kształtów w XAML i kod współdzielony
- Aplikacja WPF w przeglądarce
- Tworzenie i testowanie pakietu instalacyjnego AppX

MVVM i XAML — Twoje przepustki do świata nowoczesnych aplikacji!

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

Helion

37364 numer katalogowy

księgarnia internetowa



<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

Sprawdź najnowsze promocje:

① <http://helion.pl/promocje>

Książki najchętniej czytane:

② <http://helion.pl/bestsellery>

Zamów informacje o nowościach:

③ <http://helion.pl/novosci>

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

ISBN 978-83-283-1018-6



9 788328 310186

Informatyka w najlepszym wydaniu

cena: 59,00 zł