

Windows 8[®]

Programowanie aplikacji
z wykorzystaniem C# i XAML

Wydanie VI

Professional



Charles Petzold

Tytuł oryginału: Programming Windows: Writing Windows 8 Apps With C# and XAML

Tłumaczenie: Paweł Gonera

ISBN: 978-83-246-7383-4

© 2013 Grupa Wydawnicza Helion S.A.

Authorized Polish translation of the English edition Programming Windows®, Sixth Edition ISBN 9780735671768

© 2013 Charles Petzold

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

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 the Publisher.

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.

Wydawnictwo HELION dołożyło wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie bierze jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Wydawnictwo HELION nie ponosi 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

WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

<ftp://ftp.helion.pl/przyklady/win8pa.zip>

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/win8pa>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzje.

Printed in Poland.

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

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

Spis treści

Wprowadzenie 9

CZĘŚĆ I PODSTAWY

Rozdział 1. Znaczniki i kod	21
Pierwszy projekt	21
Graficzne pozdrowienia	27
Manipulowanie tekstem	30
Multimedia również	38
Alternatywne programy realizowane za pomocą kodu	39
Wstawianie grafik za pomocą kodu	43
To nawet nie strona	45
Rozdział 2. Składnia XAML	49
Pędzel gradientu w kodzie	49
Składnia element-właściwość	52
Właściwości ContentProperty	55
Właściwość ContentProperty elementu TextBlock	59
Współdzielenie pędzli (oraz innych zasobów)	61
Zasoby są współdzielone	65
Grafika wektorowa	65
Rozciąganie tekstu za pomocą elementu Viewbox	75
Style	78
Trochę o wiązaniu danych	83
Rozdział 3. Podstawowa obsługa zdarzeń	87
Zdarzenie Tapped	87
Obsługa zdarzeń routowanych	90
Przesłanie ustawienia Handled	96
Wejście, wyrównanie i tła	98
Zmiany rozmiaru i orientacji	101
Wiązanie danych dla elementów Run?	106
Odmierzanie czasu i animacja	108

Rozdział 4. Prezentacja z wykorzystaniem elementów Panel	117
Element Border	118
Elementy Rectangle i Ellipse	121
Element StackPanel	123
Stosy poziome	125
Program WhatSize z wiązaniem danych (i elementem Converter)	128
Wykorzystanie elementu ScrollViewer	132
Osobliwość czy normalność?	138
Tworzenie e-booka	140
Bardziej wyszukane elementy StackPanel	143
Pochodne klasy UserControl	145
Tworzenie bibliotek Windows Runtime	147
Alternatywa z zawijaniem	150
Element Canvas i właściwości załączone	153
Właściwość ZIndex	157
Osobliwości elementu Canvas	158
Rozdział 5. Interakcje z kontrolkami	161
Specyfika kontroltek	161
Suwak do zakresów	163
Grid	167
Orientacja i współczynniki proporcji	173
Slider i konwerter sformatowanego tekstu	176
Wskazówki ekranowe i konwersja	177
Szkicowanie za pomocą suwaków	179
Różne odmiany przycisków	181
Definiowanie właściwości zależności	189
Znaczniki RadioButton	198
Wprowadzanie danych z klawiatury a elementy TextBox	205
Dotyk i kontrolka Thumb	208
Rozdział 6. WinRT i MVVM	215
Model MVVM w skrócie	215
Powiadomienia wiązań danych	216
Warstwa ViewModel w programie ColorScroll	218
Skróty składniowe	223
Właściwość DataContext	226
Wiązania i TextBox	228
Przyciski a wzorzec MVVM	233
Klasa DelegateCommand	235
Rozdział 7. Wywołania asynchroniczne	243
Wątki i interfejs użytkownika	243
Korzystanie z klasy MessageDialog	244
Wywołania zwrotne jako funkcje lambda	250
Operator await	251
Anulowanie operacji asynchronicznej	253

Sposoby wykonywania plikowych operacji wejścia-wyjścia	255
Lokalny magazyn aplikacji	256
Okna dialogowe do wskazywania plików	256
Dostęp masowy	257
Okna do wskazywania plików i operacje wejścia-wyjścia	257
Obsługa wyjątków	262
Konsolidacja wywołań asynchronicznych	263
Uproszczona obsługa plikowych operacji wejścia-wyjścia	265
Problemy związane z cyklem życia aplikacji	267
Własne metody asynchroniczne	272
Rozdział 8. Paski aplikacji i wyskakujące okna	283
Implementacja menu kontekstowych	283
Wyskakujące okna dialogowe	287
Paski aplikacji	290
Styl przycisków paska aplikacji	293
Wewnątrz czcionki Segoe UI Symbol	298
Kontrolki pól wyboru i przełączników na paskach aplikacji	304
Pasek aplikacji Notatnika	308
Wprowadzenie do programu XAML Cruncher	314
Ustawienia aplikacji a warstwa ViewModel	329
Element Page aplikacji XAML Cruncher	332
Parsowanie znaczników XAML	337
Plikowe operacje wejścia-wyjścia w XAML	339
Okno dialogowe ustawień	343
Więcej niż Windows Runtime	348
Rozdział 9. Animacje	349
Przestrzeń nazw Windows.UI.Xaml.Media.Animation	349
Podstawy animacji	350
Zmienne animacje	354
Inne animacje wartości typu double	359
Animowanie właściwości załączanych	366
Funkcje łagodzące	369
Animacje All-XAML	378
Animowanie własnych klas	382
Animacje kluczowych klatek	386
Animacje właściwości klasy Object	390
Predefiniowane animacje i przejścia	392
Rozdział 10. Transformacje	397
Przegląd informacji o transformacjach	397
Obroty (ręczne i animowane)	400
Wizualna komunikacja z użytkownikiem	405
Translacje	407
Grupy transformacji	411
Skalowanie	416

Tworzenie zegara analogowego	418
Pochylenie	424
Tworzenie efektów startowych	427
Podstawy matematyczne transformacji	428
Transformacja Composite	436
Transformacje z wykorzystaniem klasy Geometry	438
Transformacje z wykorzystaniem klasy Brush	440
Gdzie się podział mój element?	444
Transformacje rzutów — projekcje	447
Obliczanie macierzy Matrix3D	454
Rozdział 11. Trzy szablony	465
Dane w przyciskach	466
Podejmowanie decyzji	475
Kontrolki kolekcji i rzeczywiste zastosowanie szablonu DataTemplate	479
Kolekcje i interfejsy	489
Dotyk i zaznaczanie	491
Panele i wirtualizacja	496
Własne panele	500
Tworzenie wykresu słupkowego z wykorzystaniem szablonu	513
Kontrolka FlipView	515
Szablon ControlTemplate	518
Menedżer stanu wizualizacji	528
Korzystanie z pliku generic.xaml	535
Części szablonu	536
Własne kontrolki	544
Szablony i kontenery elementów	550
Rozdział 12. Strony i nawigacja	555
Problemy z rozdzielczością ekranu	555
Problemy ze skalowaniem	560
Widok przyciągnięty	564
Zmiana orientacji	569
Prosta nawigacja pomiędzy stronami	572
Stos powrotu	577
Zdarzenia nawigacji i przywracanie stron	579
Zapisanie i odtworzenie stanu aplikacji	583
Akceleratory nawigacji i przyciski myszy	587
Przekazywanie i zwracanie danych	590
Standardowe szablony Visual Studio	596
Modele widoku i kolekcje	602
Grupowanie elementów	620

Rozdział 13. Dotyk itd.	629
Przegląd zdarzeń Pointer	630
Pierwsze podejście do malowania palcami	633
Przechwytywanie wskaźnika	636
Edycja z użyciem menu wyskakującego	644
Odczyt siły nacisku	647
Wygładzanie przewężeń	651
Jak zapisać swoje rysunki?	660
Realistyczne i surrealistyczne rysowanie palcami	660
Pianino dotykowe	663
Manipulacje, palce i elementy	668
Obsługa inercji	676
Kontrolka XYSlider	680
Centrowane skalowanie i obroty	686
Obroty jednym palcem	690
Rozdział 14. Mapy bitowe	697
Bity pikseli	698
Przezroczystość i wstępnie pomnożona alfa	704
Pędzel z okrągłym gradientem	709
Ładowanie i zapisywanie plików obrazów	716
Zmiana głębi kolorów	727
Zapisywanie rysunku namalowanego palcami	734
Wybór koloru HSL	759
Rysowanie odwrotne	770
Dostęp do biblioteki obrazów	775
Użycie kamery	784
Rozdział 15. Operacje natywne	791
Wprowadzenie do P/Invoke	792
Mała pomoc	797
Informacja o strefie czasowej	798
Komponent Windows Runtime dla DirectX	819
DirectWrite a czcionki	821
Konfiguracje i platformy	832
Interpretacja metryki czcionki	835
Rysowanie w obiekcie SurfaceImageSource	841
Rozdział 16. Tekst sformatowany	855
Czcionki prywatne	856
Użycie elementu Glyphs	860
Pliki czcionek w magazynie lokalnym	862
Rozszerzenia typograficzne	866
Elementy RichTextBlock oraz Paragraph	867
Zaznaczanie w RichTextBlock	871

Obsługa nadmiarowego tekstu w RichTextBlock	871
Problemy ze stronicowaniem	878
Edycja tekstu sformatowanego w RichEditBox	885
Własne mechanizmy wprowadzania tekstu	895
Rozdział 17. Współdzielenie i drukowanie	901
Ustawienia i okna wyskakujące	902
Współdzielenie danych poprzez schowek	906
Panel Udostępnianie	911
Proste drukowanie	912
Marginesy drukowalne i niedrukowalne	918
Proces stronicowania	922
Własne właściwości drukowania	929
Drukowanie miesięcznego planu pracy	935
Drukowanie zakresu stron	944
Gdzie można wykonywać czasochłonne zadania?	955
Drukowanie grafiki z FingerPaint	956
Rozdział 18. Sensory i GPS	959
Orientacja	959
Przyspieszenie, siła, grawitacja i wektory	964
Podążaj za kulką	974
Dwie północe	979
Inklinometr = przyspieszeniometer + kompas	981
OrientationSensor = przyspieszeniometer + kompas	986
Azymut i wysokość	991
Mapy oraz kafelki map Bing	1004
Rozdział 19. Pióro (nazywane również rysikiem)	1019
Kolekcje InkManager	1020
Atrybuty atramentu	1023
Usuwanie i inne rozszerzenia	1029
Zaznaczanie pociągnięć	1035
Żółty notatnik	1043
Skorowidz	1061
O autorze	1088

Sensory i GPS

W ostatnim czasie komputery zostały wyposażone w nowe „zmysły”. To nie jest szkic nowego filmu! Wiele komputerów — a w szczególności tabletów i innych urządzeń mobilnych — posiada sensory, które pozwalają maszynie zorientować się w przestrzeni trójwymiarowej, w położeniu na powierzchni Ziemi, sprawdzić ilość światła, a nawet szybkość, z jaką użytkownik obraca w rękach komputer.

Te elementy sprzętowe, nazywane ogólnie *sensorami*, i interfejsy programowe pozwalające na korzystanie z nich znajdują się w większości w przestrzeni nazw Windows. ↪Devices.Sensors, natomiast klasy pomagające programowi w określeniu położenia geograficznego znajdują się w przestrzeni nazw Windows.Devices.Geolocation. Sprzęt realizujący to drugie zadanie jest często nazywany nieformalnie skrótem GPS (ponieważ tego typu system satelitarny nosi nazwę Global Positioning System), ale często komputer może określić położenie geograficzne z użyciem połączenia sieciowego.

W tym rozdziale skupimy się na danych dostępnych poprzez klasy SimpleOrientationSensor, Accelerometer, Compass, InclInometer, OrientationSensor oraz Geolocator, ale musimy pominąć rzadziej używane klasy LightSensor i Gyrometer, które pozwalają na pomiar ilości światła oraz szybkości kątowej komputera.

Aby w pełni wykorzystać informacje z tego rozdziału, powinieneś wziąć do ręki komputer z uruchomionymi programami i poruszyć nim, a nawet trzymać go nad głową. Jeżeli Twój komputer, na którym tworzysz aplikacje Windows 8, jest przywiązany do biurka tak jak mój, możesz postarać się o tablet, taki jak Microsoft Surface, i zainstalować na nim programy w sposób pokazany przez Tima Heuera na jego blogu, dostępnym pod adresem <http://timheuer.com/blog/archive/2012/10/26/remote-debugging-windows-store-apps-on-surface-arm-devices.aspx>.

Część przykładowych programów z tego rozdziału powstało na podstawie artykułów, jakie napisałem na temat sensorów w Windows Phone 7.5, opublikowanych od czerwca do grudnia 2012 r. w „MSDN Magazine”.

Orientacja

Najprostszą klasą sensorów, jaką się zajmiemy, jest SimpleOrientationSensor, która pozwala w przybliżony sposób zorientować się, jakie jest położenie komputera w przestrzeni trójwymiarowej, ale bez większych szczegółów. Aby utworzyć obiekt Simple ↪OrientationSensor, korzystamy z metody statycznej:

```
SimpleOrientationSensor simpleOrientationSensor = SimpleOrientationSensor.GetDefault();
```

Operację tę wykonujemy w aplikacji tylko raz, więc kod ten powinien pojawić się jako definicja pola, co pozwoli na dostęp do tego obiektu dla całej klasy. Jeżeli metoda `SimpleOrientationSensor.GetDefault` zwróci `null`, komputer nie posiada urządzenia pozwalającego na odczyt orientacji.

W dowolnym momencie z obiektu `SimpleOrientationSensor` możemy odczytać wartość oznaczającą bieżącą orientację:

```
SimpleOrientation simpleOrientation = simpleOrientationSensor.GetCurrentOrientation();
```

`SimpleOrientation` to typ wyliczeniowy z sześcioma składowymi:

- `NotRotated`,
- `Rotated90DegreesCounterclockwise`,
- `Rotated180DegreesCounterclockwise`,
- `Rotated270DegreesCounterclockwise`,
- `Faceup`,
- `Facedown`.

Ograniczenie ilości danych do tych sześciu wartości tłumaczy, dlaczego w nazwie klasy `SimpleOrientationSensor` znalazła się fraza *simple* — prosta.

Możemy również być powiadomieni o zmianie orientacji za pomocą zdarzenia. Ustawienie metody obsługi zdarzenia `OrientationChanged` jest realizowane w następujący sposób:

```
simpleOrientationSensor.OrientationChanged += OnSimpleOrientationChanged;
```

Zdarzenie to jest generowane w momencie zmiany orientacji, co nie dzieje się, gdy komputer jest względnie nieruchomy. Jeżeli potrzebujesz wartości początkowej, to oprócz ustawienia metody obsługi zdarzenia wywołaj metodę `GetCurrentOrientation`.

Metoda obsługi zdarzenia działa we własnym wątku, więc interakcja z wątkiem interfejsu użytkownika musi zachodzić z użyciem obiektu `CoreDispatcher`:

```
async void OnSimpleOrientationChanged(SimpleOrientationSensor sender,
    SimpleOrientationSensorOrientationChangedEventArgs args)
{
    await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
    {
        ...
    });
}
```

Argument zdarzenia mający typ o bardzo długiej nazwie posiada właściwość `Orientation` typu wyliczeniowego `SimpleOrientation` oraz właściwość `Timestamp` typu `Date` ↪ `TimeOffset`.

Możesz tu zadać pytania: „Czy nie korzystałem już z tych danych orientacji? Czy nie są one dostępne w przestrzeni nazw `Windows.Graphics.Display`? Czy nie korzystałem z klasy `DisplayProperties` i jej właściwości statycznych `NativeOrientation` oraz `CurrentOrientation`, jak również zdarzenia `OrientationChanged` do odczytania danych o orientacji?”. Na pewno pamiętasz, że te dwie właściwości statyczne zwracają wartości typu wyliczeniowego `DisplayOrientations`:

- Landscape,
- Portrait,
- LandscapeFlipped,
- PortraitFlipped.

Klasy `SimpleOrientationSensor` i `DisplayProperties` są oczywiście ze sobą związane, ale trzeba wiedzieć, w jaki sposób. Klasa `SimpleOrientationSensor` pozwala sprawdzić, jak komputer jest zorientowany w przestrzeni. Właściwość `DisplayProperties.CurrentOrientation` pokazuje, jak Windows dostosował się do orientacji komputera przez zmianę orientacji okna programu. Inaczej mówiąc, `SimpleOrientationSensor` raportuje orientację sprzętu, natomiast `DisplayProperties.CurrentOrientation` raportuje orientację oprogramowania, które zareagowało na zmianę w położeniu sprzętu.

Projekt *OrientationAndOrientation* ma za zadanie pokazać różnice pomiędzy tymi orientacjami. W pliku XAML zdefiniowanych jest kilka elementów `TextBlock` dla etykiet oraz do wyświetlania informacji:

Listing 18.1. Projekt: OrientationAndOrientation | Plik: MainPage.xaml (fragment)

```
<Page... FontSize="18">
  <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
    <Grid HorizontalAlignment="Center"
          VerticalAlignment="Center">
      <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
      </Grid.RowDefinitions>

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

      <TextBlock Text="SimpleOrientationSensor:&#x00A0;"
                Grid.Row="0"
                Grid.Column="0" />

      <TextBlock Name="orientationSensorTextBlock"
                Grid.Row="0"
                Grid.Column="1"
                TextAlignment="Right" />

      <TextBlock Text="DisplayProperties.CurrentOrientation:&#x00A0;"
                Grid.Row="1"
                Grid.Column="0" />

      <TextBlock Name="displayOrientationTextBlock"
                Grid.Row="1"
                Grid.Column="1"
                TextAlignment="Right" />
    </Grid>
  </Grid>
</Page>
```

W pliku kodu ukrytego zdefiniowane są dwie metody mające za zadanie ustawić dwa elementy TextBlock w drugiej kolumnie elementu Grid. Te dwie metody są wywoływane zarówno w konstruktorze, aby ustawić wartości początkowe, jak i z dwóch metod obsługi zdarzeń.

Listing 18.2. Projekt: OrientationAndOrientation | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
    SimpleOrientationSensor simpleOrientationSensor =
        ↪ SimpleOrientationSensor.GetDefault();

    public MainPage()
    {
        this.InitializeComponent();

        // Inicjalizacja SimpleOrientationSensor
        if (simpleOrientationSensor != null)
        {
            SetOrientationSensorText(simpleOrientationSensor.GetCurrentOrientation());
            simpleOrientationSensor.OrientationChanged += OnSimpleOrientationChanged;
        }

        // Inicjalizacja DisplayProperties
        SetDisplayOrientationText(DisplayProperties.CurrentOrientation);
        DisplayProperties.OrientationChanged += OnDisplayPropertiesOrientationChanged;
    }

    // Metoda obsługująca SimpleOrientationSensor
    async void OnSimpleOrientationChanged(SimpleOrientationSensor sender,
                                           SimpleOrientationSensorOrientation
                                           ↪ ChangedEventArgs args)
    {
        await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
        {
            SetOrientationSensorText(args.Orientation);
        });
    }

    void SetOrientationSensorText(SimpleOrientation simpleOrientation)
    {
        orientationSensorTextBlock.Text = simpleOrientation.ToString();
    }

    // Metoda obsługująca DisplayProperties
    void OnDisplayPropertiesOrientationChanged(object sender)
    {
        SetDisplayOrientationText(DisplayProperties.CurrentOrientation);
    }

    void SetDisplayOrientationText(DisplayOrientations displayOrientation)
    {
        displayOrientationTextBlock.Text = displayOrientation.ToString();
    }
}
```

Zwróć uwagę, że obiekt `SimpleOrientationSensor` jest tworzony jako pole, ale konstruktor sprawdza przed dostępem do niego, czy zawiera wartość różną od `null`.

Jeżeli uruchomisz ten program na tablecie mającym standardową orientację poziomą — czyli właściwość `DisplayProperties.NativeOrientation` zawiera wartość `DisplayOrientations.Landscape` — i jeżeli nie zrobisz nic, aby uniemożliwić Windows zmianę orientacji (na przykład umieszczając tablet w stacji dokującej), to w czasie obracania tabletu w kierunku zgodnym z ruchem wskazówek zegara zazwyczaj można zauważyć następującą korelację pomiędzy dwoma wskaźnikami orientacji:

SimpleOrientationSensor	DisplayProperties.CurrentOrientation
<code>NotRotated</code>	<code>Landscape</code>
<code>Rotated270DegreesCounterClockwise</code>	<code>Portrait</code>
<code>Rotated180DegreesCounterClockwise</code>	<code>LandscapeFlipped</code>
<code>Rotated90DegreesCounterClockwise</code>	<code>PortraitFlipped</code>

Obiekt `SimpleOrientationSensor` raportuje również wartości `Faceup` i `Facedown`, które nie mają odpowiedników w typie `DisplayOrientations`.

Choć powyższa tabela zwykle zgadza się dla tabletów z natywną orientacją poziomą, to jednak urządzenia mające standardowo orientację pionową wykazują następującą korelację:

SimpleOrientationSensor	DisplayProperties.CurrentOrientation
<code>NotRotated</code>	<code>Portrait</code>
<code>Rotated270DegreesCounterClockwise</code>	<code>LandscapeFlipped</code>
<code>Rotated180DegreesCounterClockwise</code>	<code>PortraitFlipped</code>
<code>Rotated90DegreesCounterClockwise</code>	<code>Landscape</code>

Co więcej, aplikacja może zażądać od Windows, aby nie wykonywał zmian kompensujących orientację komputera — czy to za pomocą ustawienia w pliku `Package.appxmanifest`, czy to programowo, przez ustawienie właściwości `DisplayProperties.AutoRotationPreferences`. W takim przypadku wartość `DisplayProperties.CurrentOrientation` nie zmienia się w czasie działania aplikacji. Niektóre tablety mają przełącznik sprzętowy pozwalający zablokować obracanie ekranu przez Windows. W takim przypadku możesz nawet zobaczyć coś takiego:

SimpleOrientationSensor	DisplayProperties.CurrentOrientation
<code>NotRotated</code>	<code>PortraitFlipped</code>
<code>Rotated270DegreesCounterClockwise</code>	<code>PortraitFlipped</code>
<code>Rotated180DegreesCounterClockwise</code>	<code>PortraitFlipped</code>
<code>Rotated90DegreesCounterClockwise</code>	<code>PortraitFlipped</code>

Jeżeli chcesz, możesz nawet samodzielnie wykonać kompensację orientacji. Możesz poinstruować Windows, aby nie były wykonywane zmiany orientacji, a następnie użyć `SimpleOrientationSensor` do sprawdzenia, jak jest zorientowany komputer. Jednak pamiętaj, że ustawienie z pliku `Package.appxmanifest` oraz `DisplayProperties.AutoRotationPreferences` jedynie informuje Windows o Twoich preferencjach, więc gdy Windows postąpi niezgodnie z tymi wskazówkami, mogą być konieczne dalsze korekty.

Chyba najbezpieczniejszym podejściem do wyłączenia automatycznego obrotu jest ustawienie właściwości `DisplayProperties.AutoRotationPreferences` na `DisplayProperties.NativeOrientation`, z czego będziemy korzystać w dalszej części rozdziału.

Przyspieszenie, siła, grawitacja i wektory

Klasa `SimpleOrientationSensor` bez wątplenia wykorzystuje urządzenie sprzętowe nazywane *przyspieszeniemierzem* albo *akcelerometrem*. Przyspieszeniomierz jest urządzeniem mierzącym przyspieszenie, na początek może się wydawać, że znajomość przyspieszenia nie jest zbyt przydatna. Jednak wiemy z fizyki — a dokładniej z drugiego prawa dynamiki Newtona — że:

$$F = ma$$

Siła jest równa iloczynowi masy i przyspieszenia, a jedną z sił, od której trudno uciec, jest siła grawitacji. Przez większość czasu przyspieszeniomierz komputera mierzy grawitację i odpowiada na proste pytanie, gdzie jest dół.

Bardziej bezpośredni dostęp do przyspieszeniomierza mamy poprzez klasę `Accelerometer`. Aby utworzyć obiekt klasy `Accelerometer`, należy użyć metody statycznej o tej samej nazwie co w `SimpleOrientationSensor`:

```
Accelerometer accelerometer = Accelerometer.GetDefault();
```

Jeżeli metoda `Accelerometer.GetDefault` zwróci `null`, komputer nie posiada przyspieszeniomierza albo Windows 8 nic nie wie na jego temat. Jeżeli Twoja aplikacja nie może działać bez przyspieszeniomierza, powinieneś poinformować użytkownika o jego braku.

W dowolnym momencie możesz odczytać bieżącą wartość przyspieszenia:

```
AccelerometerReading accelerometerReading = accelerometer.GetCurrentReading();
```

Podobna metoda w klasie `SimpleOrientationSensor` nosi nazwę `GetCurrentOrientation`.

Dobrym pomysłem jest sprawdzanie, czy wartością zwracaną przez `GetCurrentReading` jest `null`. W klasie `AccelerometerReading` są zdefiniowane cztery właściwości:

- `AccelerationX` typu `double`,
- `AccelerationY` typu `double`,
- `AccelerationZ` typu `double`,
- `Timestamp` typu `DateTimeOffset`.

Trzy wartości `double` tworzą wektor trójwymiarowy wskazujący od urządzenia w kierunku ziemi. Więcej na jego temat wkrótce.

Do obiektu `Accelerometer` można również dołączyć metodę obsługi zdarzenia:

```
accelerometer.ReadingChanged += OnAccelerometerReadingChanged;
```

Podobne zdarzenie w klasie `SimpleOrientationSensor` nosi nazwę `OrientationChanged`. Podobnie jak w przypadku `OrientationChanged`, metoda obsługująca `ReadingChanged` działa w osobnym wątku, więc powinna być obsługiwana w następujący sposób:

```
async void OnAccelerometerReadingChanged(Accelerometer sender,
AccelerometerReadingChangedEventArgs args)
{
    await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
    {
        ...
    });
}
```

Klasa `AccelerometerReadingChangedEventArgs` ma zdefiniowaną właściwość o nazwie `Reading` typu `AccelerometerReading`, takiego samego jak obiekt zwracany przez `GetCurrentReading`.

Jak często będzie wywoływana metoda obsługi zdarzenia `ReadingChanged`? Jeżeli komputer jest nieruchomy, być może wcale! Z tego powodu, jeżeli potrzebujesz początkowych odczytów przyspieszenia, powinieneś na początek wywołać metodę `GetCurrentReading`.

Jeżeli komputer jest przenoszony lub zmienia orientację w przestrzeni, metoda `ReadingChanged` jest wywoływana przy zmianie wartości (przy określonych kryteriach), ale pomiędzy kolejnymi wywołaniami następuje przerwa o długości nie mniejszej niż liczba milisekund odczytana z właściwości `ReportInterval` obiektu `Accelerometer`. Z właściwości tej odczytałem wartość 112, czyli metoda `ReadingChanged` jest wywoływana nie częściej niż dziewięć razy na sekundę.

Właściwości `ReportInterval` można przypisać inną wartość, ale nie niższą od wartości z właściwości `MinimumReportInterval`, która wynosi 16 milisekund, czyli około 60 razy na sekundę. Ustawienie `ReportInterval` na `MinimumReportInterval` pozwala uzyskać maksymalną ilość danych; ustawienie `ReportInterval` na zero powoduje powrót do ustawień domyślnych.

Pozostałe klasy czujników w przestrzeni nazw `Windows.Devices.Sensors` mają ten sam interfejs programowy co `Accelerometer`. Mają one następujące składniki:

- statyczną metodę `GetDefault`,
- metodę instancyjną `GetCurrentReading`,
- właściwość `ReportInterval`,
- właściwość tylko do odczytu `MinimumReportInterval`,
- zdarzenie `ReadingChanged`.

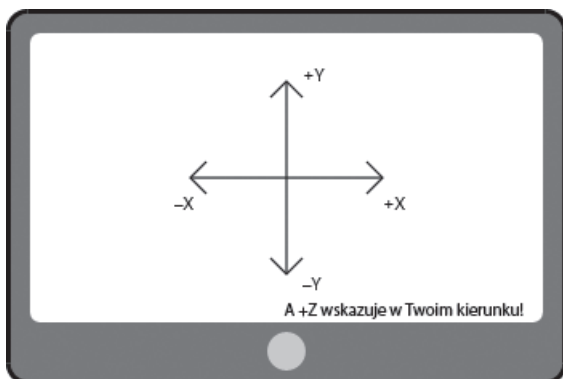
Jedynie klasa `SimpleOrientationSensor` różni się od pozostałych.

Jeżeli komputer jest nieruchomy, właściwości `AccelerationX`, `AccelerationY` oraz `AccelerationZ` klasy `AccelerometerReading` definiują wektor wskazujący w kierunku środka Ziemi. Wektory są zwykle zapisywane pogrubieniem jako współrzędne (x, y, z) , dla odróżnienia od punktów (x, y, z) . Punkt jest lokalizacją w przestrzeni, natomiast wektor jest kierunkiem i modułem. Oczywiście zachodzi związek pomiędzy wektorem i punktem. Kierunek wektora (x, y, z) jest kierunkiem z punktu $(0, 0, 0)$ do punktu (x, y, z) , a modułem wektora jest długość tego odcinka. Jednak sam wektor nie jest odcinkiem i nie posiada lokalizacji.

Moduł wektora może być wyliczony za pomocą trójwymiarowej wersji twierdzenia Pitagorasa:

$$\text{Moduł} = \sqrt{x^2 + y^2 + z^2}$$

Każdy wektor trójwymiarowy musi być usytuowany w określonym układzie współrzędnych trójwymiarowych i wektor odczytany z AccelerometerReading nie jest wyjątkiem. W przypadku tabletu z natywną orientacją poziomą układ współrzędnych jest następujący:



Zwróć uwagę, że wartości Y zwiększają się w górę, czyli odwrotnie niż w przypadku konwencji stosowanych w grafice dwuwymiarowej. Dodatnia oś Z „wystaje” z ekranu. Konwencja taka jest często nazywana układem współrzędnych prawej ręki. Jeżeli palec wskazujący prawej ręki wskazuje w kierunku dodatnich X , duży palec w kierunku dodatnich Y , to kciuk wskazuje w kierunku dodatnich Z .

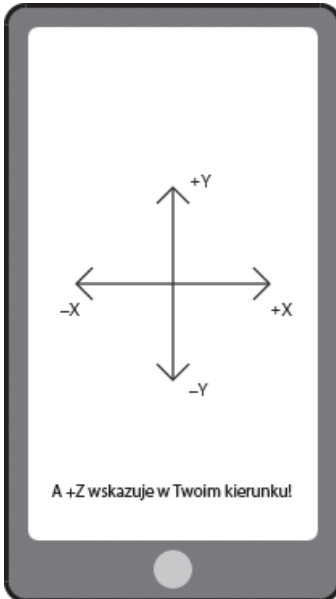
Jeżeli obrócisz palce prawej ręki w kierunku potrzebnym do obrotu dodatniej osi X w kierunku dodatniej osi Y , kciuk będzie wskazywał w kierunku dodatniej osi Z . Działa to dla dowolnych par osi, w kolejności X, Y, Z . Obróć palce prawej dłoni w taki sposób, aby dodatnia oś Y przesunęła się w kierunku dodatniej osi Z , a kciuk będzie wskazywał w kierunku dodatniej osi X . Albo obróć palce prawej dłoni w taki sposób, aby dodatnia oś Z przesunęła się w kierunku dodatniej osi X , a kciuk będzie wskazywał w kierunku dodatniej osi Y .

Zasada prawej dłoni może być używana do określenia kierunku obrotu wokół osi. W przypadku obrotów wokół osi X (dla przykładu) wskaź kciukiem prawej dłoni w kierunku dodatniej osi X , a palce będą obracać się w kierunku dodatnich kątów obrotu wokół osi.

W przypadku urządzeń z natywną orientacją pionową układ współrzędnych jest taki sam z perspektywy użytkownika (patrz rysunek na następnej stronie).

Choć nie byłem w stanie tego potwierdzić, układ współrzędnych dla zwykłych laptopów bazuje na powierzchni klawiatury, a nie ekranu. Oś X jest dłuższy bok klawiatury, oś Y krótszy bok klawiatury, a oś Z wskazuje ponad klawiaturę.

Ten układ współrzędnych jest stały dla urządzenia i wektor obiektu Accelerometer wskazuje w kierunku środka Ziemi, mając wartości określane względem tego układu współrzędnych. Gdy tablet jest na przykład trzymany prosto w swojej natywnej orientacji,



wektor przyspieszenia wskazuje w kierunku $-Y$. Moduł wektora wynosi mniej więcej 1, więc wektor ten można zapisać jako $(0, -1, 0)$. Gdy urządzenie leży na płaskiej powierzchni stołu, ekranem w górę, to wektor ma wartość bliską $(0, 0, -1)$.

Moduł równy 1 wskazuje, że wektor ten jest mierzony w jednostkach g , czyli przyspieszeniem powodowanym przez siłę grawitacji na powierzchni Ziemi, tj. 9,81 metra na sekundę do kwadratu. Jeżeli zabierzesz tablet na Księżyc, moduł wektora spadnie do 0,17. Jeżeli tablet będzie spadał swobodnie (o ile się odważysz rzucić go) to moduł wektora przyspieszenia zmniejszy się do zera, aż do momentu uderzenia w podłoże.

Przedstawię teraz program *AccelerometerAndSimpleOrientation*, który wyświetla wartości z obiektów *Accelerometer* oraz *SimpleOrientationSensor*. Plik XAML zawiera zestaw elementów *TextBlock* na etykiety i wartości ustawiane za pomocą kodu ukrytego.

Listing 18.3. Projekt: *AccelerometerAndSimpleOrientation* | Plik: *MainPage.xaml* (fragment)

```
<Page ... >

    <Page.Resources>
        <Style TargetType="TextBlock">
            <Setter Property="FontSize" Value="24" />
            <Setter Property="Margin" Value="24 12 24 12" />
        </Style>
    </Page.Resources>

    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">

        <Grid HorizontalAlignment="Center"
            VerticalAlignment="Center">
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
            </Grid.RowDefinitions>
        </Grid>
    </Grid>
</Page>
```

```

        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>

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

    <TextBlock Grid.Row="0" Grid.Column="0" Text="Przyspieszeniomierz X:" />
    <TextBlock Grid.Row="1" Grid.Column="0" Text="Przyspieszeniomierz Y:" />
    <TextBlock Grid.Row="2" Grid.Column="0" Text="Przyspieszeniomierz Z:" />
    <TextBlock Grid.Row="3" Grid.Column="0" Text="Moduł:"
        Margin="24 24" />
    <TextBlock Grid.Row="4" Grid.Column="0" Text="Orientacja:" />

    <TextBlock Grid.Row="0" Grid.Column="1" Name="accelerometerX"
        TextAlignment="Right" />
    <TextBlock Grid.Row="1" Grid.Column="1" Name="accelerometerY"
        TextAlignment="Right"/>
    <TextBlock Grid.Row="2" Grid.Column="1" Name="accelerometerZ"
        TextAlignment="Right"/>
    <TextBlock Grid.Row="3" Grid.Column="1" Name="magnitude"
        TextAlignment="Right"
        VerticalAlignment="Center" />
    <TextBlock Grid.Row="4" Grid.Column="1" Name="simpleOrientation"
        TextAlignment="Right" />
</Grid>
</Grid>
</Page>

```

Plik kodu ukrytego posiada nieco więcej kodu sprawdzającego niż poprzednio. Jeżeli nie można utworzyć obiektu `Accelerometer` lub `SimpleOrientationSensor`, informacja o tym jest wyświetlana użytkownikowi. Dodatkowo dobrym pomysłem jest, aby obiekt `Accelerometer` nie działał w czasie, gdy program z niego nie korzysta, ponieważ powoduje to szybsze zużywanie baterii. Aby pokazać, że aplikacja szanuje baterię, program ten podłącza metodę obsługi w `OnNavigateTo` i odłącza ją w `OnNavigateFrom`. Poza tym jego struktura jest zbliżona do poprzedniego programu.

Listing 18.4. Projekt: AccelerometerAndSimpleOrientation | Plik: MainPage.xaml.cs (fragment)

```

public sealed partial class MainPage : Page
{
    Accelerometer accelerometer = Accelerometer.Default();
    SimpleOrientationSensor simpleOrientationSensor =
        SimpleOrientationSensor.Default();

    public MainPage()
    {
        this.InitializeComponent();
        this.Loaded += OnMainPageLoaded;
    }

    async void OnMainPageLoaded(object sender, RoutedEventArgs args)
    {
        if (accelerometer == null)

```

```

        await new AlertDialog("Nie można uruchomić
        ↳ przyspieszoniomierza").ShowAsync();

    if (simpleOrientationSensor == null)
        await new AlertDialog("Nie można uruchomić
        ↳ SimpleOrientationSensor").ShowAsync();
    }

    // Podłączenie metod obsługi zdarzeń
    protected override void OnNavigatedTo(NavigationEventArgs args)
    {
        if (accelerometer != null)
        {
            SetAccelerometerText(accelerometer.GetCurrentReading());
            accelerometer.ReadingChanged += OnAccelerometerReadingChanged;
        }

        if (simpleOrientationSensor != null)
        {
            SetSimpleOrientationText(simpleOrientationSensor.GetCurrentOrientation());
            simpleOrientationSensor.OrientationChanged += OnSimpleOrientationChanged;
        }
        base.OnNavigatedTo(args);
    }

    // Odłączenie metod obsługi zdarzeń
    protected override void OnNavigatedFrom(NavigationEventArgs args)
    {
        if (accelerometer != null)
            accelerometer.ReadingChanged -= OnAccelerometerReadingChanged;

        if (simpleOrientationSensor != null)
            simpleOrientationSensor.OrientationChanged -= OnSimpleOrientationChanged;

        base.OnNavigatedFrom(args);
    }

    // Metoda obsługi przyspieszoniomierza
    async void OnAccelerometerReadingChanged(Accelerometer sender,
        AccelerometerReadingChangedEventArgs
        ↳ args)
    {
        await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
        {
            SetAccelerometerText(args.Reading);
        });
    }

    void SetAccelerometerText(AccelerometerReading accelerometerReading)
    {
        if (accelerometerReading == null)
            return;

        accelerometerX.Text = accelerometerReading.AccelerationX.ToString("F2");
        accelerometerY.Text = accelerometerReading.AccelerationY.ToString("F2");
        accelerometerZ.Text = accelerometerReading.AccelerationZ.ToString("F2");
        magnitude.Text =

```

```

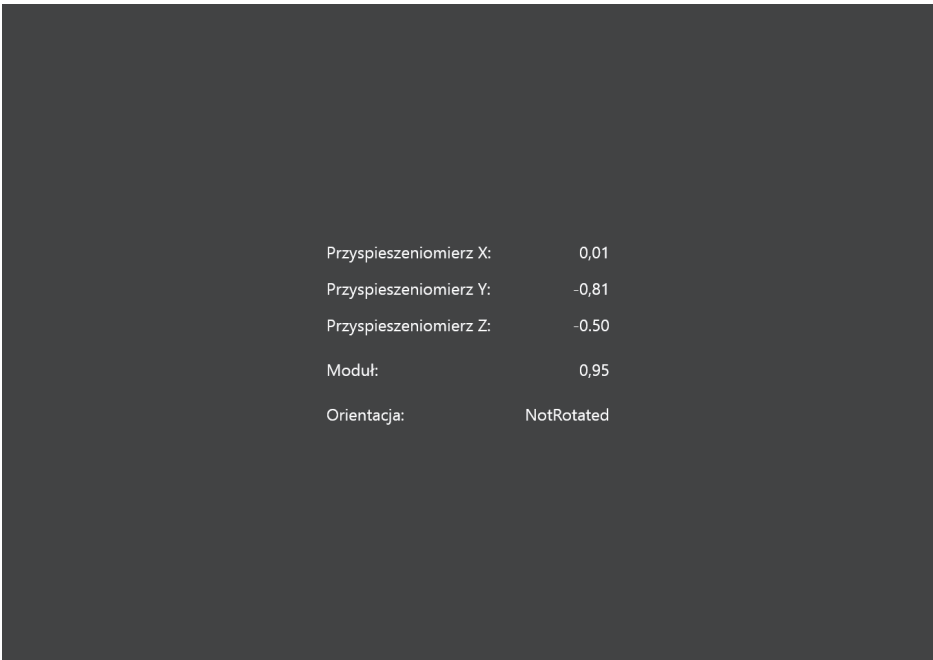
        Math.Sqrt(Math.Pow(accelerometerReading.AccelerationX, 2) +
            Math.Pow(accelerometerReading.AccelerationY, 2) +
            Math.Pow(accelerometerReading.AccelerationZ,
                ↪2)).ToString("F2");
    }

    // Metoda obsługi SimpleOrientationSensor
    async void OnSimpleOrientationChanged(SimpleOrientationSensor sender,
        SimpleOrientationSensorOrientation
            ↪ChangedEventArgs args)
    {
        await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
            {
                SetSimpleOrientationText(args.Orientation);
            });
    }

    void SetSimpleOrientationText(SimpleOrientation simpleOrientation)
    {
        this.simpleOrientation.Text = simpleOrientation.ToString();
    }
}

```

Poniżej pokazany jest widok ekranu tabletu używanego przeze mnie przy pisaniu tej książki, gdy został on umieszczony w stacji dokującej.



Nie przejmuj się, jeżeli zobaczysz wartość modułu odbiegającą od 1. Oznacza to tylko, że przyspieszeniometer nie jest tak precyzyjny, jak byśmy chcieli.

Zarówno składowa X , jak i Y jest ujemna, co wskazuje, że tablet został odchylony w tył. Jak wcześniej wspomniałem, jeżeli tablet jest ustawiony pionowo w górę, teoretycznie wektor ma wartość $(0, -1, 0)$, a jeżeli leży na biurku ekranem w górę, teoretycznie wektor ma wartość $(0, 0, -1)$. Pomiedzy tymi dwoma pozycjami tablet jest obracany wokół osi X . Przekazując wartości Y i Z do metody `Math.Atan2`, otrzymamy kąt obrotu.

Jeżeli uruchomisz ten program na urządzeniu przenośnym, możesz poruszać nim w różnych orientacjach, aby zobaczyć efekty. Generalnie powinieneś zauważyć analogię pomiędzy wartościami z `SimpleOrientationSensor` a z `Accelerometer`:

SimpleOrientationSensor	Wektor Accelerometer
NotRotated	$\sim (0, -1, 0)$
Rotated90DegreesCounterClockwise	$\sim (-1, 0, 0)$
Rotated180DegreesCounterClockwise	$\sim (0, 1, 0)$
Rotated270DegreesCounterClockwise	$\sim (1, 0, 0)$
Faceup	$\sim (0, 0, -1)$
Facedown	$\sim (0, 0, 1)$

Użyty tu symbol przybliżenia (\sim) pokazuje, że powinniśmy traktować te wartości bardzo liberalnie. Wektor `Accelerometer` wykazuje pewne fluktuacje, zanim osiągnie wartości powodujące zmianę w `SimpleOrientationSensor`.

Ten prosty program `AccelerometerAndSimpleOrientation` nie zakłada żadnej preferowanej orientacji, więc gdy będziemy poruszać tabletem, Windows będzie automatycznie zmieniał orientację, zakładając, że nie chcesz czytać liczb do góry nogami. Powinieneś zauważyć korelację pomiędzy wartościami `SimpleOrientationSensor` a orientacją ekranu, ponieważ Windows zmienia sposób wyświetlania ekranu, bazując na tych wartościach! Jeżeli zablokujesz możliwość zmiany orientacji ekranu przez Windows (w dowolny sposób), dane wyświetlane przez program się nie zmieniają.

W rzeczywistości możesz uznać, że ciągła zmiana orientacji jest irytująca. W czasie takiej zmiany aktualizacja ekranu jest na chwilę przerywana, a zawartość ekranu dopasowuje się do zmiany. Jeżeli chwilę o tym pomyślisz, prawdopodobnie dojdiesz do wniosku, że program korzystający z obiektu `Accelerometer` do zmiany zawartości ekranu powinien mieć zablokowaną automatyczną zmianę orientacji.

Z tego powodu wszystkie programy w pozostałej części rozdziału zawierają w konstruktorze programu wiersz powodujący ustawienie preferowanej orientacji:

```
DisplayProperties.AutoRotationPreferences = DisplayProperties.NativeOrientation;
```

Po uruchomieniu programu `AccelerometerAndSimpleOrientation` na urządzeniu przenośnym możesz zauważyć, że w przypadku wykonania szybkiego ruchu zmienia się kierunek i moduł wektora przyspieszenia, który przestaje wskazywać 1g. Jeżeli na przykład ruszysz urządzeniem w lewo, wektor przyspieszenia będzie wskazywał w prawo, ale wyłącznie w czasie, gdy urządzenie przyspiesza. Jeżeli uda Ci się przesuwać je ze stałą szybkością, wektor przyspieszenia wróci do stałej wartości modułu i będzie wskazywał w kierunku środka Ziemi. Nagłe zatrzymanie urządzenia spowoduje, że wektor przyspieszenia wskaże również zmianę w szybkości.

Klasa `Accelerometer` definiuje również zdarzenie o nazwie `Shaken`, które nie niesie ze sobą żadnych informacji. Zdarzenie to jest przydatne, gdy program musi „rzucić” parą kostek, zaproponować inną restaurację, usunąć rysunek lub wycofać przypadkowe usunięcie.

Jednym z częstych zastosowań klasy `Accelerometer` jest oznaczanie poziomu. W pliku XAML tworzone są cztery elementy `Ellipse`. Trzy z nich są narysowane koncentrycznie, a czwarta reprezentuje bąbełek poziomiczy.

Listing 18.5. Projekt: BubbleLevel | Plik: MainPage.xaml (fragment)

```
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
  <Grid Name="centeredGrid"
        HorizontalAlignment="Center"
        VerticalAlignment="Center">
    <Ellipse Name="outerCircle"
            Stroke="{StaticResource ApplicationForegroundThemeBrush}" />

    <Ellipse Name="halfCircle"
            Stroke="{StaticResource ApplicationForegroundThemeBrush}" />

    <Ellipse Width="24"
            Height="24"
            Stroke="{StaticResource ApplicationForegroundThemeBrush}" />

    <Ellipse Fill="Red"
            Width="24"
            Height="24"
            HorizontalAlignment="Center"
            VerticalAlignment="Center">
      <Ellipse.RenderTransform>
        <TranslateTransform x:Name="bubbleTranslate" />
      </Ellipse.RenderTransform>
    </Ellipse>
  </Grid>
</Grid>
```

W pliku kodu ukrytego właściwość `DisplayProperties.AutoRotationPreferences` przypisywana jest wartość `DisplayProperties.NativeOrientation`. Nie ma powodu, aby w tym programie Windows automatycznie zmieniał orientację ekranu. Program posiada również metodę obsługi zdarzenia `SizeChanged`, w którym ustawiana jest wielkość elementów `outerCircle` oraz `halfCircle`.

Listing 18.6. Projekt: BubbleLevel | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
    Accelerometer accelerometer = Accelerometer.GetDefault();

    public MainPage()
    {
        this.InitializeComponent();
        DisplayProperties.AutoRotationPreferences =
            ↪ DisplayProperties.NativeOrientation;
        Loaded += OnMainPageLoaded;
        SizeChanged += OnMainPageSizeChanged;
    }
}
```

```

async void OnMainPageLoaded(object sender, RoutedEventArgs args)
{
    if (accelerometer != null)
    {
        accelerometer.ReportInterval = accelerometer.MinimumReportInterval;
        SetBubble(accelerometer.GetCurrentReading());
        accelerometer.ReadingChanged += OnAccelerometerReadingChanged;
    }
    else
    {
        await new MessageDialog("Przyspieszeniomierz nie jest
        ↳dostępny").ShowAsync();
    }
}

void OnMainPageSizeChanged(object sender, SizeChangedEventArgs args)
{
    double size = Math.Min(args.NewSize.Width, args.NewSize.Height);
    outerCircle.Width = size;
    outerCircle.Height = size;
    halfCircle.Width = size / 2;
    halfCircle.Height = size / 2;
}

async void OnAccelerometerReadingChanged(Accelerometer sender,
                                          AccelerometerReadingChangedEventArgs
                                          ↳args)
{
    await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
    {
        SetBubble(args.Reading);
    });
}

void SetBubble(AccelerometerReading accelerometerReading)
{
    if (accelerometerReading == null)
        return;

    double x = accelerometerReading.AccelerationX;
    double y = accelerometerReading.AccelerationY;

    bubbleTranslate.X = -x * centeredGrid.ActualWidth / 2;
    bubbleTranslate.Y = y * centeredGrid.ActualHeight / 2;
}
}

```

Metoda `SetBubble` jest bardzo prosta. Jej zadaniem jest pobranie składników X i Y z wektora przyspieszenia i ustawienie na ich podstawie współrzędnych X i Y środka bąbla przez skalowanie ich do promienia zewnętrznego okręgu. Zwróć uwagę na przypadek, gdy tablet leży na biurku ekranem w dół lub w górę. Składnik Z przyspieszenia wynosi 1 lub -1 , a składniki X i Y są równe 0, co oznacza, że bąbel znajduje się na środku ekranu. Jest to prawidłowe.

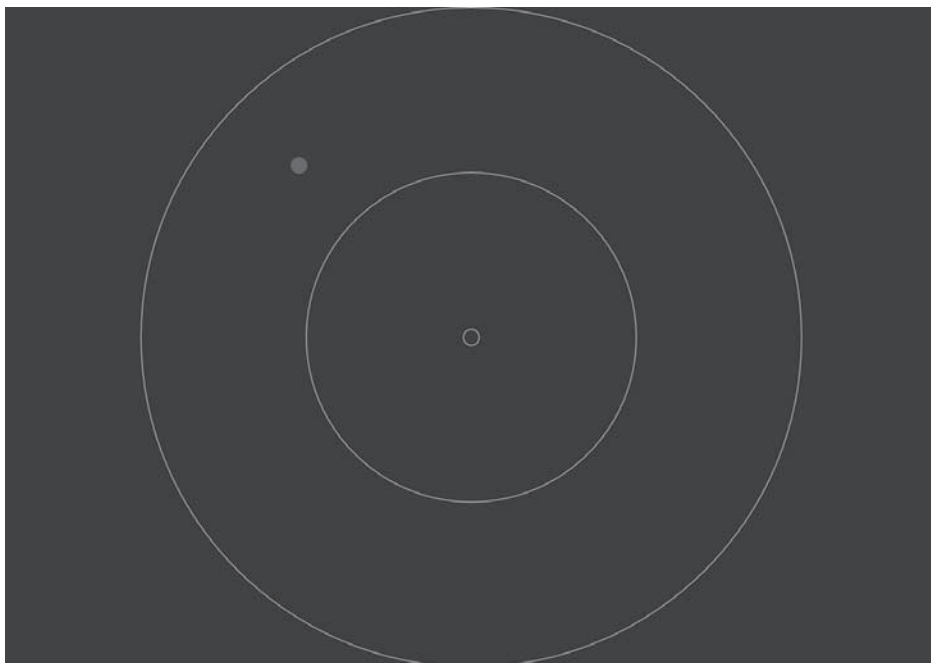
Teraz przestaw tablet tak, aby ekran był prostopadły do powierzchni Ziemi. Składowa Z staje się zerowa. Oznacza to, że moduł przyspieszenia zależy wyłącznie od składników X i Y . Inaczej mówiąc:

$$x^2 + y^2 = 1^2$$

Jest to równanie okręgu w dwóch wymiarach, więc bąbel znajduje się gdzieś na zewnętrznym okręgu. Miejsce, gdzie dokładnie się znajduje, zależy od obrotu tabletu względem osi Z.

Wektor przyspieszenia wskazuje w dół, w kierunku środka Ziemi, a bąbel przesuwa się w górę, więc musimy zamienić znaki składników X i Y , aby skonwertować je na współrzędne dwuwymiarowego ekranu. Pamiętaj jednak, że oś Y wektora przyspieszenia jest i tak odwrócona w stosunku do współrzędnych ekranowych, więc konieczna jest zmiana znaku wyłącznie składnika X , co można zauważyć w ostatnich dwóch wierszach kodu.

Program ten uruchomiony na tablecie Microsoft Surface wygląda następująco:



Oczywiście na rysunku nie widać, jak bardzo drży rysowany bąbel. Wartości z klasy `Accelerometer` są surowe i w rzeczywistej aplikacji na pewno będziesz chciał je nieco wygładzić. Zrobimy to w następnych dwóch programach.

Podążaj za kulką

Klasa `Accelerator` jest często stosowana w grach na urządzenia przenośne. Jeżeli na przykład masz grę symulującą jazdę samochodem, to użytkownik może nim kierować przez przechylenie komputera na lewo i prawo.

Poniższe dwa programy symulują kulkę toczącą się po ekranie. Jeżeli będziesz trzymał tablet równoległe do Ziemi i będziesz balansował rzeczywistą kulką położoną na powierzchni ekranu, możesz ją przetaczać przez pochylanie ekranu w dwóch płaszczyznach.

Im większe pochylenie, tym większe przyspieszenie kulki. Wirtualna kulka w dwóch następnych programach przesuwa się w podobny sposób. Podobnie jak w programie symulującym poziomicę, programy te ignorują składową Z z wektora Accelerometer i do zarządzania przyspieszeniem na dwuwymiarowej powierzchni ekranu korzystają wyłącznie ze składowych X oraz Y .

W programie *TiltAndRoll*, gdy kulka uderza o krawędź, traci całe przyspieszenie w kierunku prostopadłym do krawędzi i kontynuuje toczenie się wzdłuż niej, o ile posiada jeszcze przyspieszenie skierowane w tym kierunku. Kulka jest zdefiniowana w pliku XAML. Element `EllipseGeometry` pozwala na pozycjonowanie kulki w dowolnym miejscu poprzez ustawienie właściwości `Center`.

Listing 18.7. Projekt: *TiltAndRoll* | Plik: *MainPage.xaml* (fragment)

```
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
  <Path Fill="Red">
    <Path.Data>
      <EllipseGeometry x:Name="ball" />
    </Path.Data>
  </Path>
</Grid>
```

Plik kodu ukrytego zaczyna się od zdefiniowania stałej `GRAVITY` określonej w pikselach na sekundę do kwadratu. Teoretycznie tocząca się bez tarcia kulka podlega sile grawitacji, ale przyspieszenie toczącej się kulki wynosi 2/3 przyspieszenia grawitacyjnego (patrz A.P. French, *Newtonian Mechanics*, W.W. Norton 1971, s. 652 – 653). Oznacza to, że można obliczyć wartość `GRAVITY` przez pomnożenie 10 metrów na sekundę do kwadratu przez 0,025 m na cal, przez 95 pikseli na cal i przez 2/3, co daje wartość około 25 000, ale na potrzeby programu znacznie ją zmniejszyłem.

Wartość wektora dwuwymiarowego jest bardzo przydatna w obliczeniach przyspieszenia w dwóch wymiarach, prędkości i położenia, więc dołączyłem do programu strukturę `Vector2` z rozdziału 13., „Dotyk itd.”.

Ponieważ kulka musi się toczyć niezależnie od wywołań zdarzenia `ReadingChanged` z obiektu `Accelerometer`, program nie instaluje metody obsługi, ale zamiast tego korzysta z `CompositionTarget.Rendering` do odczytania bieżącej wartości i zastosowania jej do ruchu kulki. Zwróć uwagę, że składniki X i Y odczytu dokonywanego w obiekcie `Accelerometer` są używane do utworzenia wartości `Vector2`, które są następnie uśredniane z poprzednimi wartościami, które z kolei były uśrednione z poprzednimi itd. Jest to niezwykle prosta metoda wygładzania.

Listing 18.8. Projekt: *TiltAndRoll* | Plik: *MainPage.xaml.cs* (fragment)

```
public sealed partial class MainPage : Page
{
    const double GRAVITY = 5000;    // Piksele na sekundę do kwadratu
    const double BALL_RADIUS = 32;

    Accelerometer accelerometer = Accelerometer.GetDefault();
    TimeSpan timeSpan;
    Vector2 acceleration;
    Vector2 ballPosition;
    Vector2 ballVelocity;
```

```

public MainPage()
{
    this.InitializeComponent();
    DisplayProperties.AutoRotationPreferences =
        ↳DisplayProperties.NativeOrientation;

    ball.RadiusX = BALL_RADIUS;
    ball.RadiusY = BALL_RADIUS;

    Loaded += OnMainPageLoaded;
}

async void OnMainPageLoaded(object sender, RoutedEventArgs args)
{
    if (accelerometer == null)
    {
        await new MessageDialog("Przyspieszeniomierz nie jest
            ↳dostępny").ShowAsync();
    }
    else
    {
        CompositionTarget.Rendering += OnCompositionTargetRendering;
    }
}

void OnCompositionTargetRendering(object sender, object args)
{
    AccelerometerReading reading = accelerometer.GetCurrentReading();

    if (reading == null)
        return;

    // Odczyt czasu od ostatniego zdarzenia
    TimeSpan timeSpan = (args as RenderingEventArgs).RenderingTime;
    double elapsedSeconds = (timeSpan - this.timeSpan).TotalSeconds;
    this.timeSpan = timeSpan;

    // Konwersja danych przyspieszeniomierza na współrzędne wyświetlacza
    double x = reading.AccelerationX;
    double y = -reading.AccelerationY;

    // Pobranie bieżącego przyspieszenia X-Y i wygładzenie go
    acceleration = 0.5 * (acceleration + new Vector2(x, y));

    // Obliczenie nowej szybkości i pozycji
    ballVelocity += GRAVITY * acceleration * elapsedSeconds;
    ballPosition += ballVelocity * elapsedSeconds;

    // Sprawdzenie kolizji z brzegiem
    if (ballPosition.X - BALL_RADIUS < 0)
    {
        ballPosition = new Vector2(BALL_RADIUS, ballPosition.Y);
        ballVelocity = new Vector2(0, ballVelocity.Y);
    }
    if (ballPosition.X + BALL_RADIUS > this.ActualWidth)
    {
        ballPosition = new Vector2(this.ActualWidth - BALL_RADIUS, ballPosition.Y);
    }
}

```

```

        ballVelocity = new Vector2(0, ballVelocity.Y);
    }
    if (ballPosition.Y - BALL_RADIUS < 0)
    {
        ballPosition = new Vector2(ballPosition.X, BALL_RADIUS);
        ballVelocity = new Vector2(ballVelocity.X, 0);
    }
    if (ballPosition.Y + BALL_RADIUS > this.ActualHeight)
    {
        ballPosition = new Vector2(ballPosition.X, this.ActualHeight -
            ↪BALL_RADIUS);
        ballVelocity = new Vector2(ballVelocity.X, 0);
    }
    ball.Center = new Point(ballPosition.X, ballPosition.Y);
}
}
}

```

Dwa najważniejsze obliczenia to:

```

ballVelocity += GRAVITY * acceleration * elapsedSeconds;
ballPosition += ballVelocity * elapsedSeconds;

```

Pamiętaj, że `acceleration`, `ballVelocity` oraz `ballPosition` są wartościami `Vector2`, więc mają one składowe X i Y . Prędkość jest zwiększana o wartość `acceleration` pomnożoną o czas, a położenie jest zwiększane o prędkość pomnożoną przez czas. Pozostało tylko sprawdzać, czy nowa pozycja znajduje się poza granicami strony. Jeżeli tak, jest przesuwana na stronę i jeden z komponentów prędkości jest ustawiany na zero.

Fizyka ruchu kulki jest dosyć realistyczna. Wraz ze zwiększaniem i zmniejszaniem pochylenia zwiększa się i zmniejsza wartość przyspieszenia. Co więcej, ponieważ program korzysta ze wzorów na prędkość i położenie, dosyć łatwe jest dodanie funkcji odbijania. Prosty sposób polega na tym, że gdy kulka trafia w krawędź, można nadać przeciwny znak składnikowi prędkości, zamiast go zerować. Jednak oznacza to, że kulka po odbiciu ma taką samą wartość prędkości, co jest mało realistyczne. Sensowniejsze jest dodanie współczynnika tłumienia, który nazwałem `BOUNCE`. Program *TiltAndBounce* jest niemal identyczny z programem *TiltAndRoll*, poza stałą `BOUNCE` i inną logiką ruchu kulki w metodzie obsługi `CompositionTarget.Rendering`.

Listing 18.9. Projekt: *TiltAndBounce* | Plik: *MainPage.xaml.cs* (fragment)

```

public sealed partial class MainPage : Page
{
    ...
    const double BOUNCE = -2.0 / 3; // Ułamek szybkości
    ...
    void OnCompositionTargetRendering(object sender, object args)
    {
        AccelerometerReading reading = accelerometer.GetCurrentReading();

        if (reading == null)
            return;

        // Pobranie czasu
        TimeSpan timeSpan = (args as RenderingEventArgs).RenderingTime;
        double elapsedSeconds = (timeSpan - this.timeSpan).TotalSeconds;
        this.timeSpan = timeSpan;
    }
}

```

```

// Konwersja danych przyspieszeniomierza na współrzędne wyświetlacza
double x = reading.AccelerationX;
double y = -reading.AccelerationY;

// Pobranie bieżącego przyspieszenia X-Y i wygładzenie go
acceleration = 0.5 * (acceleration + new Vector2(x, y));

// Obliczenie nowej szybkości i pozycji
ballVelocity += GRAVITY * acceleration * elapsedSeconds;
ballPosition += ballVelocity * elapsedSeconds;

// Sprawdzenie kolizji z brzegiem
bool needAnotherLoop = true;

while (needAnotherLoop)
{
    needAnotherLoop = false;

    if (ballPosition.X - BALL_RADIUS < 0)
    {
        ballPosition = new Vector2(-ballPosition.X + 2 * BALL_RADIUS,
            ↪ballPosition.Y);
        ballVelocity = new Vector2(BOUNCE * ballVelocity.X, ballVelocity.Y);
        needAnotherLoop = true;
    }
    else if (ballPosition.X + BALL_RADIUS > this.ActualWidth)
    {
        ballPosition = new Vector2(-ballPosition.X + 2 * (this.ActualWidth -
            ↪BALL_RADIUS),
            ballPosition.Y);
        ballVelocity = new Vector2(BOUNCE * ballVelocity.X, ballVelocity.Y);
        needAnotherLoop = true;
    }
    else if (ballPosition.Y - BALL_RADIUS < 0)
    {
        ballPosition = new Vector2(ballPosition.X, -ballPosition.Y + 2 *
            ↪BALL_RADIUS);
        ballVelocity = new Vector2(ballVelocity.X, BOUNCE * ballVelocity.Y);
        needAnotherLoop = true;
    }
    else if (ballPosition.Y + BALL_RADIUS > this.ActualHeight)
    {
        ballPosition = new Vector2(ballPosition.X,
            -ballPosition.Y + 2 * (this.ActualHeight -
            ↪BALL_RADIUS));
        ballVelocity = new Vector2(ballVelocity.X, BOUNCE * ballVelocity.Y);
        needAnotherLoop = true;
    }
}
ball.Center = new Point(ballPosition.X, ballPosition.Y);
}
}

```

W programie *TiltAndRoll* możliwe jest, że kulka wykroczy poza dwie sąsiednie krawędzie w tym samym zdarzeniu, ale przypadki te są obsługiwane za pomocą serii instrukcji `if`. W tym programie odbijanie kulki od jednego brzegu może spowodować wyjście poza drugi brzeg, przez co niezbędna jest pętla do testowania pozycji, obsługująca wszystkie odbicia.

Choć klasa `Accelerator` pozwala sprawdzić kierunek w dół, nie ujawnia ona pełnej orientacji urządzenia w przestrzeni 3D. Aby zrozumieć, co mam na myśli, uruchom program `AccelerometerAndSimpleOrientation` na urządzeniu przenośnym. Wstań i ustaw urządzenie w jakiś dziwny sposób. Dzięki klasie `Accelerometer` wiemy, gdzie jest dół. Teraz obróć całe ciało wokół osi. Tablet obróci się o 360 stopni w przestrzeni, ale `Accelerometer` będzie raportował w zasadzie taką samą wartość, ponieważ kierunek w dół pozostał niezmienny.

Co się zmienia, gdy obracamy tabletem wokół wektora przyspieszenia? Odpowiedź jest prosta: zmienia się położenie względem północy. Dlatego właśnie ważny jest sensor kompasu, `Compass`, który udostępnia nam brakujący element pozwalający na określenie orientacji tabletu. Po połączeniu odczytów z obiektów `Compass` i `Accelerometer` możemy określić kompletne położenie tabletu w przestrzeni trójwymiarowej. Możemy też pozwolić, aby zrobił to za nas system Windows.

Klasa `Compass` jest zbudowana podobnie do `Accelerometer`, a klasa `CompassReading` zawiera dwie właściwości: `HeadingMagneticNorth` oraz `HeadingTrueNorth`. Obie właściwości zawierają wartości w stopniach i wskazują kąt odchylenia położenia komputera względem północy. Jeżeli położysz tablet na powierzchni równoległej do ziemi i górę ekranu skierujesz w stronę północy (przez „górkę” rozumiem kierunek dodatniej osi *Y* pokazanej na rysunku we wcześniejszej części rozdziału), kąty te powinny być bliskie zeru. Gdy będziesz obracał tabletem w kierunku wschodnim, kąty będą się zwiększać.

Oczywiście, kąty te nie powinny być takie same, poza niektórymi lokalizacjami na świecie. Tablet zawiera magnetometr, który wykrywa północ magnetyczną (która jest zależna od pola magnetycznego Ziemi), ale biegun magnetyczny nie pokrywa się z biegunem geograficznym. Co interesujące, właściwość `HeadingMagneticNorth` ma typ `double`, natomiast `HeadingTrueNorth` jest typu `double` dopuszczającego wartości `null`, co sugeruje, że wartość ta może nie być dostępna.

Wypróbujmy to. W pliku XAML projektu `SimpleCompass` zdefiniowane są dwie strzałki graficzne znajdujące się w środku ekranu i wskazujące w górę.

Listing 18.10. Projekt: `SimpleCompass` | Plik: `MainPage.xaml` (fragment)

```
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
  <Canvas HorizontalAlignment="Center"
    VerticalAlignment="Center">
    <Path Fill="Magenta"
      Data="M -10 0 L 10 0, 10 -300, 0 -350, -10 -300 Z">
      <Path.RenderTransform>
        <RotateTransform x:Name="magNorthRotate" />
      </Path.RenderTransform>
    </Path>

    <Path Name="trueNorthPath"
      Fill="Blue"
      Data="M -10 0 L 10 0, 10 -300, 0 -350, -10 -300 Z">
      <Path.RenderTransform>
        <RotateTransform x:Name="trueNorthRotate" />
      </Path.RenderTransform>
    </Path>
  </Canvas>
</Grid>
```

Kolor magenta będzie odpowiadał północy magnetycznej, natomiast niebieski północy geograficznej.

Jeżeli wartość `HeadingTrueNorth` wynosi `null`, drugi z obiektów `Path` jest ukrywany.

Listing 18.11. Projekt: SimpleCompass | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
    Compass compass = Compass.GetDefault();

    public MainPage()
    {
        this.InitializeComponent();
        DisplayProperties.AutoRotationPreferences =
            ↪DisplayProperties.NativeOrientation;
        Loaded += OnMainPageLoaded;
    }

    async void OnMainPageLoaded(object sender, RoutedEventArgs args)
    {
        if (compass != null)
        {
            ShowCompassValues(compass.GetCurrentReading());
            compass.ReportInterval = compass.MinimumReportInterval;
            compass.ReadingChanged += OnCompassReadingChanged;
        }
        else
        {
            await new MessageDialog("Kompas jest niedostępny").ShowAsync();
        }
    }

    async void OnCompassReadingChanged(Compass sender, CompassReadingChangedEventArgs
    ↪args)
    {
        await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
        {
            ShowCompassValues(args.Reading);
        });
    }

    void ShowCompassValues(CompassReading compassReading)
    {
        if (compassReading == null)
            return;

        magNorthRotate.Angle = -compassReading.HeadingMagneticNorth;

        if (compassReading.HeadingTrueNorth.HasValue)
        {
            trueNorthPath.Visibility = Visibility.Visible;
            trueNorthRotate.Angle = -compassReading.HeadingTrueNorth.Value;
        }
        else
        {
            trueNorthPath.Visibility = Visibility.Collapsed;
        }
    }
}
```

```
}  
}  
}
```

Zwróć uwagę, że kąty obrotu obu strzałek są ustawiane względem ujemnych wartości właściwości `HeadingMagneticNorth` i `HeadingTrueNorth`. Wartości te wskazują na obrót komputera względem północy, więc strzałki powinny obracać się odwrotnie i wskazywać na kierunek północny względem komputera.

Na obu komputerach, z których korzystałem podczas pisania tej książki — w tym Microsoft Surface — wyniki były rozczarowujące. Na obu właściwość `HeadingTrueNorth` zawsze miała wartość `null`. Na Microsoft Surface wartość północy magnetycznej była błędna. Na tablecie Samsung wartości zmieniały się od 0 do 180 stopni! Na tablecie mojego redaktora technicznego właściwość `HeadingMagneticNorth` zawsze wynosiła 0.

Teoretycznie północ geograficzna może być wyliczona na podstawie północy magnetycznej i położenia komputera, ale włączenie w pliku `Package.appxmanifest` funkcji `Location` nie pomogło.

Możemy tylko mieć nadzieję, że w kolejnych komputerach kompas będzie działał na tyle dobrze, że pozwoli w połączeniu z danymi przyspieszeniomierza określić pełne dane o orientacji.

Inklinometr = przyspieszeniomierz + kompas

`Sensor Inclinometer` jest jedną z dwóch klas, która wewnętrznie łączy dane z przyspieszeniomierza i kompasu, wewnętrznie je wygładzając. Klasa ta udostępnia dane w postaci kątów *odchylenia*, *pochylenia* i *obrotu*, które są terminami używanymi w mechanice lotu.

Kąty odchylenia, pochylenia i obrotu są często nazywane kątami Eulera, na cześć osiemnastowiecznego matematyka Leonarda Eulera, który przedstawił matematyczny model obrotów w przestrzeni trójwymiarowej. Jeżeli lecisz samolotem, odchylenie określa kierunek, w którym jest skierowany nos samolotu. Gdy samolot skręca w lewo lub w prawo, zmienia się wartość odchylenia. Pochylenie określa kąt nosa samolotu w stosunku do poziomu. Gdy samolot się wznosi lub opada, zmienia się pochylenie. Obrót jest osiąganym przez przechylenie w lewo lub prawo.

Aby zrozumieć, jak te wartości odnoszą się do komputera, możemy zwizualizować „latanie” na tablecie, jak na magicznym dywanie. Załóżmy, że siedzisz na ekranie twarzą w kierunku góry (oczywiście w natywnej orientacji tabletu) i startujesz. We wcześniejszym pokazanym układzie współrzędnych odchylenie jest obrotem wokół osi *Z*, pochylenie jest obrotem wokół osi *X*, a obrót zachodzi wokół osi *Y*.

Program `YawPitchRoll` pozwala przedstawić te kąty. Plik XAML zawiera kilka elementów `Rectangle` używanych dla linii, niektóre elementy `Ellipse` są wyświetlane jako kulki i nieco tekstu.

Listing 18.12. Projekt: `YawPitchRoll` | Plik: `MainPage.xaml` (fragment)

```
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">  
  <!-- Pochylenie -->  
  <Rectangle Fill="Blue"  
    Width="3"  
    HorizontalAlignment="Center"
```

```

        VerticalAlignment="Stretch" />

<Path Name="pitchPath"
      Stroke="Blue">
  <Path.Data>
    <EllipseGeometry x:Name="pitchEllipse" RadiusX="20" RadiusY="20" />
  </Path.Data>
</Path>

<!-- Obrót -->
<Rectangle Fill="Red"
           Height="3"
           HorizontalAlignment="Stretch"
           VerticalAlignment="Center" />

<Path Name="rollPath"
      Stroke="Red"
      Fill="Red">
  <Path.Data>
    <EllipseGeometry x:Name="rollEllipse" RadiusX="20" RadiusY="20" />
  </Path.Data>
</Path>

<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="*" />
    <RowDefinition Height="*" />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="*" />
    <ColumnDefinition Width="*" />
  </Grid.ColumnDefinitions>

  <!-- Pochylenie -->
  <TextBlock Text="POCHYLENIE"
            Grid.Row="0"
            Grid.Column="0"
            Foreground="Blue"
            HorizontalAlignment="Right"
            Margin="0 0 24 0" />

  <TextBlock Name="pitchValue"
            Grid.Row="0"
            Grid.Column="1"
            Foreground="Blue"
            HorizontalAlignment="Left"
            Margin="24 0 0 0" />

  <!-- Obrót -->
  <TextBlock Text="OBRÓT"
            Grid.Row="1"
            Grid.Column="0"
            Foreground="Red"
            HorizontalAlignment="Left"
            VerticalAlignment="Top"
            Margin="0 108 0 0">
    <TextBlock.RenderTransform>

```



```

        <RotateTransform Angle="-90" />
    </TextBlock.RenderTransform>
</TextBlock>

<TextBlock Name="rollValue"
    Grid.Row="0"
    Grid.Column="0"
    Foreground="Red"
    HorizontalAlignment="Left"
    VerticalAlignment="Bottom">
    <TextBlock.RenderTransform>
        <RotateTransform Angle="-90" />
    </TextBlock.RenderTransform>
</TextBlock>

<!-- Odchylenie -->
<Grid Grid.Row="0"
    Grid.Column="1"
    HorizontalAlignment="Stretch"
    VerticalAlignment="Bottom"
    RenderTransformOrigin="0 1">
    <StackPanel Orientation="Horizontal"
        HorizontalAlignment="Center">
        <TextBlock Text="ODCHYLENIE = " Foreground="Green" />
        <TextBlock Name="yawValue" Foreground="Green" />
    </StackPanel>

    <Rectangle Fill="Green"
        Height="3"
        HorizontalAlignment="Stretch"
        VerticalAlignment="Bottom" />

    <Grid.RenderTransform>
        <TransformGroup>
            <RotateTransform Angle="-90" />
            <RotateTransform x:Name="yawRotate" />
        </TransformGroup>
    </Grid.RenderTransform>
</Grid>
</Grid>
</Grid>

```

Jak można zauważyć w pliku kodu ukrytego, klasa `Inclinometer` jest tworzona podobnie jak `Accelerometer` i `Compass`.

Listing 18.13. Projekt: YawPitchRoll | Plik: MainPage.xaml.cs (fragment)

```

public sealed partial class MainPage : Page
{
    Inclinometer inclinometer = Inclinometer.GetDefault();

    public MainPage()
    {
        this.InitializeComponent();
        DisplayProperties.AutoRotationPreferences =
            ↪ DisplayProperties.NativeOrientation;
        Loaded += OnMainPageLoaded;
    }
}

```

```

}

async void OnMainPageLoaded(object sender, RoutedEventArgs args)
{
    if (inclinometer == null)
    {
        await new MessageDialog("Nie można znaleźć inklinometru").ShowAsync();
    }
    else
    {
        ShowYawPitchRoll(inclinometer.GetCurrentReading());
        inclinometer.ReportInterval = inclinometer.MinimumReportInterval;
        inclinometer.ReadingChanged += OnInclinometerReadingChanged;
    }
}

async void OnInclinometerReadingChanged(Inclinometer sender,
                                         InclinometerReadingChangedEventArgs args)
{
    await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
    {
        ShowYawPitchRoll(args.Reading);
    });
}

void ShowYawPitchRoll(InclinometerReading inclinometerReading)
{
    if (inclinometerReading == null)
        return;

    double yaw = inclinometerReading.YawDegrees;
    double pitch = inclinometerReading.PitchDegrees;
    double roll = inclinometerReading.RollDegrees;

    yawValue.Text = yaw.ToString("F0") + "°";
    pitchValue.Text = pitch.ToString("F0") + "°";
    rollValue.Text = roll.ToString("F0") + "°";

    yawRotate.Angle = yaw;

    if (pitch <= 90 && pitch >= -90)
    {
        pitchPath.Fill = pitchPath.Stroke;
        pitchEllipse.Center = new Point(this.ActualWidth / 2,
                                       this.ActualHeight * (pitch + 90) / 180);
    }
    else
    {
        pitchPath.Fill = null;

        if (pitch > 90)
            pitchEllipse.Center = new Point(this.ActualWidth / 2,
                                             this.ActualHeight * (270 - pitch) /
                                             ↪180);
        else // Pochylenie < -90
            pitchEllipse.Center = new Point(this.ActualWidth / 2,
                                             this.ActualHeight * (-90 - pitch) /
                                             ↪180);
    }
}

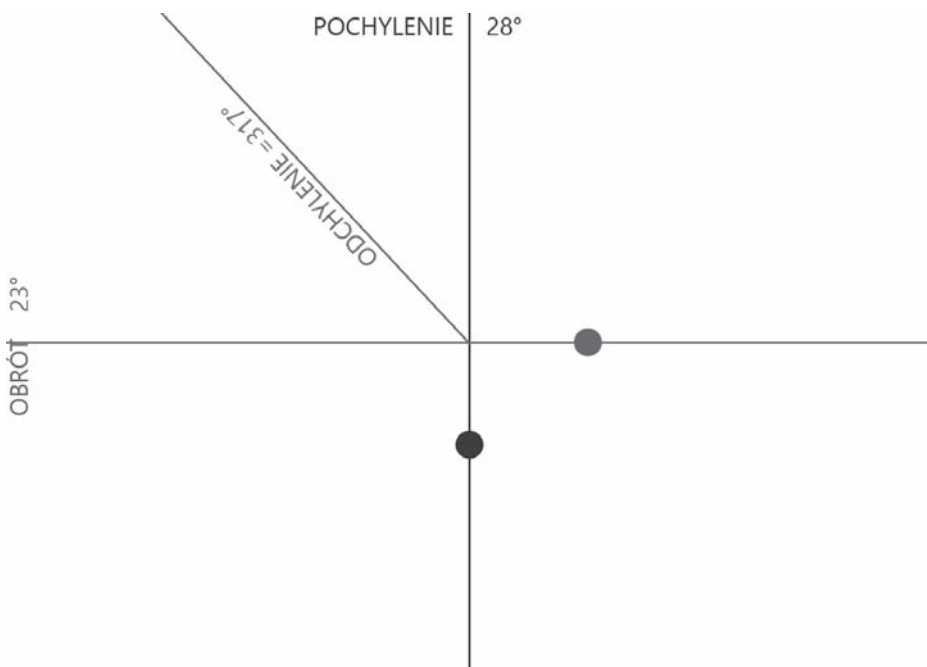
```

```

    }
    rollEllipse.Center = new Point(this.ActualWidth * (roll + 90) / 180,
                                   this.ActualHeight / 2);
    }
}

```

Nie istnieje sekretne źródło danych kompasu dla inklinometru. Właściwość `YawDegrees` ma takie same ograniczenia (lub błędy) jak `Compass`, poza tym że się wzajemnie kompensują — suma wartości `YawDegrees` i odczytu z `Compass` jest zawsze równa 360. Gdy tablet leży na płaskiej powierzchni ekranem do góry, linia odchylenia wskazuje na północ (mniej więcej), a kulki pochylenia i odchylenia znajdują się na środku ekranu. Gdy będziesz pochylał tablet w górę lub w dół, `PitchDegrees` zmienia się od 90 stopni, gdy tablet stoi pionowo w górę, do -90 , gdy tablet stoi pionowo w dół. `RollDegrees` zmienia się od 90 do -90 stopni, gdy tablet pochyla się w lewo lub prawo. Poniższy ekran użyłem, podnosząc lewą i górną krawędź tabletu.



Gdy ekran jest skierowany w dół, `YawDegrees` wskazuje na południe, a `PitchDegrees` przyjmuje wartości od 90 do 180 stopni oraz od -90 do -180 stopni. Program symbolizuje te wartości za pomocą pustego, czerwonego okręgu.

Jeżeli pracujesz nad programem, gdzie obiekt lata po ekranie, te kąty Eulera mogą być wystarczające. Jednak możesz potrzebować czegoś bardziej zorientowanego matematycznie. Do tego służy następująca omawiana klasa.

OrientationSensor = przyspieszeniomierz + kompas

Obroty w przestrzeni trójwymiarowej można reprezentować na kilka rodzajów, które można pomiędzy sobą konwertować. Klasa `OrientationSensor` jest bardzo podobna do `Inclinometer` w tym sensie, że łączy dane z przyspieszeniomierza i kompasu, aby zapewnić pełną orientację w przestrzeni 3D. `OrientationSensor` zapewnia tę orientację dzięki dwóm klasom:

- `SensorQuaternion`,
- `SensorRotationMatrix`.

Kwaterniony są bardzo interesujące z punktu widzenia matematyki. Podobnie jak liczby urojone pozwalają na reprezentowanie obrotów w przestrzeni dwuwymiarowej, kwaterniony reprezentują obroty w przestrzeni trójwymiarowej. Szczególnie programiści gier lubią reprezentować obroty jako kwaterniony, ponieważ mogą być one płynnie interpolowane (kwaterniony przedstawiałem w rozdziale 8. mojej książki *3D Programming for Windows*, Microsoft Press 2007).

Macierz obrotu jest zwykłą macierzą transformacji bez ostatniego wiersza i kolumny. Zwykła trójwymiarowa macierz transformacji ma 4 wiersze i 4 kolumny. W klasie `SensorRotationMatrix` są zdefiniowane 3 wiersze i 3 kolumny. Taka macierz nie jest w stanie reprezentować przesunięcia czy perspektywy, a zgodnie z konwencją nie zapewnia skalowania i pochylania. Może być jednak łatwo używana do obracania obiektów w przestrzeni 3D.

Na tablecie Samsung, którego używałem w czasie pisania tej książki, macierz w `SensorRotationMatrix` zawierała same zera, więc żaden z programów wykorzystujących tę macierz nie działał. Lepsze wyniki uzyskałem w przypadku Microsoft Surface.

Gdy korzystasz z macierzy obrotu, zmiana perspektywy może być pomocna. Wyjaśniałem już, jak wartości z klas `Accelerometer` oraz `Compass` odnoszą się do układu współrzędnych 3D, który prezentowałem we wcześniejszej części rozdziału. Przy korzystaniu z macierzy obrotu z klasy `OrientationSensor` pomocne jest zwizualizowanie dwóch układów współrzędnych, jednego dla urządzenia, a drugiego dla Ziemi:

- w trójwymiarowym układzie współrzędnych dodatnia oś Y wskazuje na górę ekranu, dodatnia oś X wskazuje w prawo, a dodatnia oś Z wychodzi z ekranu, tak jak pokazałem wcześniej,
- w ziemskim układzie współrzędnych dodatnia oś Y wskazuje na północ, dodatnia oś X wskazuje na wschód, a dodatnia oś Z wychodzi z gruntu.

Te dwa układy współrzędnych są takie same, gdy komputer leży na poziomej powierzchni z ekranem wskazującym w górę i gdy górna krawędź ekranu wskazuje na północ. `SensorRotationMatrix` staje się (teoretycznie) macierzą jednostkową — jedynki znajdują się na przekątnej, a w pozostałych komórkach znajdują się zera. W przeciwnym razie macierz opisuje, w jaki sposób Ziemia jest obrócona w stosunku do komputera, co jest przeciwne do obrotu opisywanego przez kąty Eulera.

Różnica ta jest przedstawiona w programie *AxisAngleRotation*, który korzysta z kolejnej metody reprezentowania obrotu w przestrzeni trójwymiarowej — jako obrotu wokół wektora 3D. Plik XAML jest niezbyt interesującym zbiorem elementów `TextBlock`, z których część to etykiety, a część oczekuje na tekst.

Listing 18.14. Projekt: AxisAngleRotation | Plik: MainPage.xaml (fragment)

```
<Page ... >

<Page.Resources>
  <Style x:Key="DefaultTextBlockStyle" TargetType="TextBlock">
    <Setter Property="FontFamily" Value="Lucida Sans Unicode" />
    <Setter Property="FontSize" Value="36" />
    <Setter Property="Margin" Value="0 0 48 0" />
  </Style>

  <Style x:Key="rightText" TargetType="TextBlock"
    BasedOn="{StaticResource DefaultTextBlockStyle}">
    <Setter Property="TextAlignment" Value="Right" />
    <Setter Property="Margin" Value="48 0 0 0" />
  </Style>
</Page.Resources>

<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
  <StackPanel HorizontalAlignment="Center"
    VerticalAlignment="Center">

    <!-- Element Grid pokazujący odchylenie, obrót i pochylenie -->
    <Grid HorizontalAlignment="Center">
      <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
      </Grid.RowDefinitions>

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

      <Grid.Resources>
        <Style TargetType="TextBlock"
          BasedOn="{StaticResource DefaultTextBlockStyle}" />
      </Grid.Resources>

      <TextBlock Text="Odchylenie: " Grid.Row="0" Grid.Column="0" />
      <TextBlock Name="pitchText" Grid.Row="0" Grid.Column="1"
        Style="{StaticResource rightText}" />

      <TextBlock Text="Obrót: " Grid.Row="1" Grid.Column="0" />
      <TextBlock Name="rollText" Grid.Row="1" Grid.Column="1"
        Style="{StaticResource rightText}" />

      <TextBlock Text="Pochylenie: " Grid.Row="2" Grid.Column="0" />
      <TextBlock Name="yawText" Grid.Row="2" Grid.Column="1"
        Style="{StaticResource rightText}" />
    </Grid>

    <!-- Element Grid na RotationMatrix -->
    <Grid HorizontalAlignment="Center"
      Margin="0 48">
      <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
      </Grid.RowDefinitions>
    </Grid>
  </StackPanel>
</Grid>
```

```

        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>

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

    <Grid.Resources>
        <Style TargetType="TextBlock"
            BasedOn="{StaticResource rightText}" />
    </Grid.Resources>

    <TextBlock Name="m11Text" Grid.Row="0" Grid.Column="0" />
    <TextBlock Name="m12Text" Grid.Row="0" Grid.Column="1" />
    <TextBlock Name="m13Text" Grid.Row="0" Grid.Column="2" />

    <TextBlock Name="m21Text" Grid.Row="1" Grid.Column="0" />
    <TextBlock Name="m22Text" Grid.Row="1" Grid.Column="1" />
    <TextBlock Name="m23Text" Grid.Row="1" Grid.Column="2" />

    <TextBlock Name="m31Text" Grid.Row="2" Grid.Column="0" />
    <TextBlock Name="m32Text" Grid.Row="2" Grid.Column="1" />
    <TextBlock Name="m33Text" Grid.Row="2" Grid.Column="2" />
</Grid>

<!-- Wyświetlanie osi i kąta obrotu -->
<Grid HorizontalAlignment="Center">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>

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

    <Grid.Resources>
        <Style TargetType="TextBlock" BasedOn="{StaticResource
            ↳DefaultTextBlockStyle}" />
    </Grid.Resources>

    <TextBlock Text="Kąt:" Grid.Row="0" Grid.Column="0" />
    <TextBlock Name="angleText" Grid.Row="0" Grid.Column="1"
        ↳TextAlignment="Center"/>
    <TextBlock Text="Osie:" Grid.Row="1" Grid.Column="0" />
    <TextBlock Name="axisText" Grid.Row="1" Grid.Column="1"
        ↳TextAlignment="Center" />
    </Grid>
</StackPanel>
</Grid>
</Page>

```

W pliku kodu ukrytego tworzony jest obiekt `Inclinometer` w celu pobrania kątów odchylenia, nachylenia i obrotu oraz `OrientationSensor` w celu uzyskania (i wyświetlenia) macierzy obrotu i jej konwersji na osie (kąty) obrotu.

Listing 18.15. Projekt: AxisAngleRotation | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
    Inclinometer inclinometer = Inclinometer.GetDefault();
    OrientationSensor orientationSensor = OrientationSensor.GetDefault();

    public MainPage()
    {
        this.InitializeComponent();
        DisplayProperties.AutoRotationPreferences =
            ↪DisplayProperties.NativeOrientation;
        Loaded += OnMainPageLoaded;
    }

    async void OnMainPageLoaded(object sender, RoutedEventArgs args)
    {
        if (inclinometer == null)
        {
            await new MessageDialog("Inklinometr nie jest dostępny").ShowAsync();
        }
        else
        {
            // Uruchomienie zdarzeń inklinometru
            ShowYawPitchRoll(inclinometer.GetCurrentReading());
            inclinometer.ReadingChanged += OnInclinometerReadingChanged;
        }

        if (orientationSensor == null)
        {
            await new MessageDialog("OrientationSensor nie jest dostępny").ShowAsync();
        }
        else
        {
            // Uruchomienie zdarzeń OrientationSensor
            ShowOrientation(orientationSensor.GetCurrentReading());
            orientationSensor.ReadingChanged += OrientationSensorChanged;
        }
    }

    async void OnInclinometerReadingChanged(Inclinometer sender,
                                             InclinometerReadingChangedEventArgs args)
    {
        await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
        {
            ShowYawPitchRoll(args.Reading);
        });
    }

    void ShowYawPitchRoll(InclinometerReading inclinometerReading)
    {
        if (inclinometerReading == null)
            return;
    }
}
```

```

yawText.Text = inclinometerReading.YawDegrees.ToString("F0") + "°";
pitchText.Text = inclinometerReading.PitchDegrees.ToString("F0") + "°";
rollText.Text = inclinometerReading.RollDegrees.ToString("F0") + "°";
}

async void OrientationSensorChanged(OrientationSensor sender,
                                    OrientationSensorReadingChangedEventArgs args)
{
    await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
    {
        ShowOrientation(args.Reading);
    });
}

void ShowOrientation(OrientationSensorReading orientationReading)
{
    if (orientationReading == null)
        return;

    SensorRotationMatrix matrix = orientationReading.RotationMatrix;

    if (matrix == null)
        return;

    m11Text.Text = matrix.M11.ToString("F3");
    m12Text.Text = matrix.M12.ToString("F3");
    m13Text.Text = matrix.M13.ToString("F3");

    m21Text.Text = matrix.M21.ToString("F3");
    m22Text.Text = matrix.M22.ToString("F3");
    m23Text.Text = matrix.M23.ToString("F3");

    m31Text.Text = matrix.M31.ToString("F3");
    m32Text.Text = matrix.M32.ToString("F3");
    m33Text.Text = matrix.M33.ToString("F3");

    // Konwersja macierzy obrotu na osie i kąty
    double angle = Math.Acos((matrix.M11 + matrix.M22 + matrix.M33 - 1) / 2);
    angleText.Text = (180 * angle / Math.PI).ToString("F0");

    if (angle != 0)
    {
        double twoSine = 2 * Math.Sin(angle);
        double x = (matrix.M23 - matrix.M32) / twoSine;
        double y = (matrix.M31 - matrix.M13) / twoSine;
        double z = (matrix.M12 - matrix.M21) / twoSine;

        axisText.Text = String.Format("{0:F2} {1:F2} {2:F2}", x, y, z);
    }
}
}

```

Poniżej przedstawiony jest ekran Microsoft Surface pokazujący trzy kąty Eulera na górze, macierz obrotu pośrodku oraz uzyskane osie (kąty) obrotu na dole.

Tworząc ten zrzut ekranu, trzymałem tablet skierowany w kierunku północnym, więc kąt odchylenia jest niemal równy zero. Tablet był pochylony lekko w lewą stronę, przez

Odchylenie: 46°

Obrót: -5°

Pochylenie: 2°

0,997 -0,017 -0,074

-0,042 0,696 -0,717

0,063 0,718 0,693

Kąt: 46

Osie: (-1,00 0,10 0,02)

co kąt obrotu jest ujemny. Góra ekranu była podniesiona do 46 stopni. Ten sam kąt jest wyświetlany na dole, ponieważ jest pobrany z macierzy obrotu. Spójrz na osie — jest to niemal wektor $(-1, 0, 0)$, który odpowiada ujemnej osi X. Korzystając z reguły prawej dłoni, wskaż kciukiem w kierunku ujemnej osi X. Obrót palców wskazuje, że obrót następuje w kierunku dodatnich kątów (i tak jest), więc potwierdza to, co napisałem — macierz obrotu opisuje obrót Ziemi względem komputera.

Oznacza to, że jeżeli chcesz użyć macierzy obrotu reprezentującej obrót komputera względem Ziemi, musisz odwrócić tę macierz. Klasa `SensorRotationMatrix` nie ma możliwości wykonania tej operacji, ale struktura `Matrix3D` może to zrobić (jak pamiętasz, klasa `Matrix3D` jest zdefiniowana w przestrzeni nazw `Windows.UI.Xaml.Media.Media3D` i używana w połączeniu z `Matrix3DProjection`). Wystarczy utworzyć obiekt `Matrix3D` na podstawie `SensorRotationMatrix` i odwrócić go.

Zamierzam użyć tej techniki do utworzenia kolejnej reprezentacji orientacji w przestrzeni trójwymiarowej.

Azymut i wysokość

Można przyjąć, że żyjemy w sferze niebieskiej. Jeżeli chcesz opisać lokalizację obiektu w przestrzeni 3D względem nas, gdy odległość nie ma znaczenia, punkty względne do tej sfery niebieskiej są bardzo wygodne. Ta sfera niebieska szczególnie dobrze nadaje się dla programów, które pozwalają używać komputera do oglądania świata, rzeczywistości wirtualnej lub rzeczywistości rozszerzonej. W takich programach trzymamy tablet tak, jakbyśmy chcieli wykonać zdjęcie za pomocą jego tylnej części, ale to, co widzimy na

ekranie, jest generowane (w całości lub w części) przez program bazujący na orientacji ekranu. Przesuwając tablet po łuku, można obserwować na ekranie odwzorowanie wycinka otaczającego nas świata.

Sfera niebieska ma analogię w ziemskiej rzeczywistości. Gdy musimy określić lokalizację na Ziemi, wykonujemy to z użyciem długości i szerokości geograficznej, które są kątami z wierzchołkiem w środku Ziemi. Dzielimy kulę ziemską na połowy wzdłuż równika. Linie szerokości geograficznej są równoległe do równika i określają dodatni kąt w stosunku do równika (z maksymalną wartością 90 stopni na biegunie północnym) w kierunku północnym oraz ujemny kąt w stosunku do równika w kierunku południowym (aż do -90 stopni na biegunie południowym). Kąty długości bazują na okręgach przechodzących przez oba bieguny i są mierzone w stosunku do południka zerowego, który przechodzi przez Greenwich w Wielkiej Brytanii.

Punkty na sferze niebieskiej możemy opisać mniej więcej w ten sam sposób, ale jesteśmy w środku tej sfery, więc terminologia jest inna.

Wskaż wyprostowanym ramieniem w dowolnym kierunku. W jaki sposób możemy zidentyfikować tę lokalizację? Na początek przesun ramię w górę lub w dół, aby było w położeniu poziomym — czyli równoległym do powierzchni Ziemi. Kąt, o który przesunąłeś ramię w czasie tego ruchu, jest nazywany *wysokością*.

Dodatnie wartości wysokości znajdują się ponad horyzontem, a ujemne poniżej horyzontu. Pionowo w górę znajduje się *zenit*, dla którego wysokość wynosi 90 stopni. Pionowo w dół znajduje się *nadir*, dla którego wysokość wynosi -90 stopni.

Nadal wskazujesz wyprostowanym ramieniem w kierunku horyzontu, prawda? Teraz przesun ramię tak, aby wskazywało północ. Kąt, o który przesunąłeś ramię w czasie tego ruchu, jest nazywany *azymutem*.

Połączenie wysokości i azymutu tworzy *współrzędne horyzontalne*, nazwane w ten sposób, ponieważ horyzont dzieli sferę niebieską na pół — podobnie jak równik we współrzędnych geograficznych.

Współrzędne horyzontalne nie niosą żadnych danych na temat odległości. Gdy jest zaćmienie Słońca, Słońce i Księżyc mają te same współrzędne horyzontalne. Współrzędna horyzontalna nie jest lokalizacją w przestrzeni 3D — jest kierunkiem w przestrzeni 3D od obserwatora. W tym sensie współrzędne horyzontalne są podobne do wektora 3D, ale wektor jest wyrażany we współrzędnych kwadratowych, a współrzędne horyzontalne są sferyczne.

Aby nieco ułatwić zadanie określania współrzędnych horyzontalnych, na początek zdefiniujemy strukturę `Vector3` opisującą wektor trójwymiarowy.

Listing 18.16. Projekt: EarthlyDelights | Plik: Vector3.cs

```
using System;
using Windows.Foundation;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Media.Media3D;

namespace Petzold.Windows8.VectorDrawing
{
    public struct Vector3
    {
        // Konstruktor
        public Vector3(double x, double y, double z)
            : this()
    }
}
```

```

{
    X = x;
    Y = y;
    Z = z;
}

// Właściwości
public double X { private set; get; }
public double Y { private set; get; }
public double Z { private set; get; }

public double LengthSquared
{
    get { return X * X + Y * Y + Z * Z; }
}

public double Length
{
    get { return Math.Sqrt(LengthSquared); }
}

public Vector3 Normalized
{
    get
    {
        double length = this.Length;

        if (length != 0)
        {
            return new Vector3(this.X / length,
                               this.Y / length,
                               this.Z / length);
        }
        return new Vector3();
    }
}

// Właściwości statyczne
public static Vector3 UnitX
{
    get { return new Vector3(1, 0, 0); }
}

public static Vector3 UnitY
{
    get { return new Vector3(0, 1, 0); }
}

public static Vector3 UnitZ
{
    get { return new Vector3(0, 0, 1); }
}

// Metody statyczne
public static Vector3 Cross(Vector3 v1, Vector3 v2)
{
    return new Vector3(v1.Y * v2.Z - v1.Z * v2.Y,

```

```

        v1.Z * v2.X - v1.X * v2.Z,
        v1.X * v2.Y - v1.Y * v2.X);
    }

    public static double Dot(Vector3 v1, Vector3 v2)
    {
        return v1.X * v2.X + v1.Y * v2.Y + v1.Z * v2.Z;
    }

    public static double AngleBetween(Vector3 v1, Vector3 v2)
    {
        return 180 / Math.PI * Math.Acos(Vector3.Dot(v1, v2) /
            v1.Length * v2.Length);
    }

    public static Vector3 Transform(Vector3 v, Matrix3D m)
    {
        double x = m.M11 * v.X + m.M21 * v.Y + m.M31 * v.Z + m.OffsetX;
        double y = m.M12 * v.X + m.M22 * v.Y + m.M32 * v.Z + m.OffsetY;
        double z = m.M13 * v.X + m.M23 * v.Y + m.M33 * v.Z + m.OffsetZ;
        double w = m.M14 * v.X + m.M24 * v.Y + m.M34 * v.Z + m.M44;
        return new Vector3(x / w, y / w, z / w);
    }

    // Operatory
    public static Vector3 operator +(Vector3 v1, Vector3 v2)
    {
        return new Vector3(v1.X + v2.X, v1.Y + v2.Y, v1.Z + v2.Z);
    }

    public static Vector3 operator -(Vector3 v1, Vector3 v2)
    {
        return new Vector3(v1.X - v2.X, v1.Y - v2.Y, v1.Z - v2.Z);
    }

    public static Vector3 operator *(Vector3 v, double d)
    {
        return new Vector3(d * v.X, d * v.Y, d * v.Z);
    }

    public static Vector3 operator *(double d, Vector3 v)
    {
        return new Vector3(d * v.X, d * v.Y, d * v.Z);
    }

    public static Vector3 operator /(Vector3 v, double d)
    {
        return new Vector3(v.X / d, v.Y / d, v.Z / d);
    }

    public static Vector3 operator -(Vector3 v)
    {
        return new Vector3(-v.X, -v.Y, -v.Z);
    }

    // Metody przesłaniające
    public override string ToString()
    {

```

```

        return String.Format("{0} {1} {2}", X, Y, Z);
    }
}

```

W strukturze tej występuje dużo ułatwień, w tym tradycyjny iloraz skalarny oraz iloczyn wektorowy, jak również metoda `Transform`, która mnoży wartość `Vector3` przez wartość `Matrix3D`. W praktyce ta wartość `Matrix3D` będzie prawdopodobnie reprezentowała obrót, więc mnożenie w efekcie obróci wektor w przestrzeni trójwymiarowej.

Gdy trzymamy tablet pionowo i patrzymy na ekran, patrzymy w kierunku względnym do układu współrzędnych komputera, a dokładniej — w kierunku wektora wychodzącego z tyłu ekranu, który jest ujemną osią Z lub $(0, 0, -1)$. Musimy skonwertować tę wartość na współrzędne horyzontalne.

Utwórzmy wartość `Matrix3D` o nazwie `matrix`, bazując na obiekcie `SensorRotation` → `Matrix` dostarczonym przez `OrientationSensor`. Wartość ta może być odwrócona, aby reprezentowała transformację z układu współrzędnych na komputerze na ziemski układ współrzędnych:

```
matrix.Invert();
```

W celu przekształcenia wektora $(0, 0, -1)$ (który jest odwrotnością statycznej właściwości `UnitZ` udostępnianej przez strukturę `Vector3`) na współrzędne ziemskie użyjemy wartości `matrix`:

```
Vector3 vector = Vector3.Transform(-Vector3.UnitZ, matrix);
```

Wektor ten zawiera wartość we współrzędnych prostokątnych, więc musimy je skonwertować na współrzędne horyzontalne. Przypomnij sobie, że w ziemskim układzie współrzędnych oś Z wskazuje w górę. Jeżeli tablet jest trzymany pionowo, oś wychodząca z tyłu urządzenia przekształcona na współrzędne ziemskie ma składnik Z równy zero. Oznacza to, że azymut można wyliczyć za pomocą dobrze znanej konwersji z dwuwymiarowych współrzędnych kartezjańskich na współrzędne kątowe, a jednocześnie z radianów na stopnie:

```
double azimuth = 180 * Math.Atan2(vector.X, vector.Y) / Math.PI;
```

Wzór ten jest poprawny niezależnie od składnika Z przekształcanego wektora. Ponieważ wysokość przyjmuje wartości od minus do plus 90 stopni, może być wyliczona za pomocą funkcji arcus sinus:

```
double altitude = 180 * Math.Asin(vector.Z) / Math.PI;
```

Jednak o czymś zapomnieliśmy. Skonwertowaliśmy trójwymiarową macierz obrotu na współrzędne, które mają tylko dwa składniki, ponieważ ograniczyliśmy wnętrze sfery do jej powierzchni. Co się stanie, gdy wskażemy tabletem na punkt w sferze niebieskiej, a następnie obrócimy tablet wokół tej osi? Jest to ta sama wysokość i azymut, ale widok na ekranie komputera powinien się zmieniać w czasie obrotu. Brakujący element jest czasami nazywany *nachyleniem*. To nieco trudniejsze do wyliczenia, ale jest to pokazane w strukturze `HorizontalCoordinate`.

Listing 18.17. Projekt: EarthlyDelights | Plik: HorizontalCoordinate.cs

```
using System;
using Windows.UI.Xaml.Media.Media3D;
```

```

namespace Petzold.Windows8.VectorDrawing
{
    public struct HorizontalCoordinate
    {
        public HorizontalCoordinate(double azimuth, double altitude, double tilt)
            : this()
        {
            this.Azimuth = azimuth;
            this.Altitude = altitude;
            this.Tilt = tilt;
        }

        public HorizontalCoordinate(double azimuth, double altitude)
            : this(azimuth, altitude, 0)
        {
        }

        // Na wschód od północy
        public double Azimuth { private set; get; }

        public double Altitude { private set; get; }

        public double Tilt { private set; get; }

        public static HorizontalCoordinate FromVector(Vector3 vector)
        {
            double altitude = 180 * Math.Asin(vector.Z) / Math.PI;
            double azimuth = 180 * Math.Atan2(vector.X, vector.Y) / Math.PI;

            return new HorizontalCoordinate(azimuth, altitude);
        }

        public static HorizontalCoordinate FromMotionMatrix(Matrix3D matrix)
        {
            // Odwrócenie macierzy
            matrix.Invert();

            // Transformacja (0, 0, -1) - wektor wychodzący z obiektu
            Vector3 zAxisTransformed = Vector3.Transform(-Vector3.UnitZ, matrix);

            // Odczyt współrzędnych horyzontalnych
            HorizontalCoordinate horzCoord = FromVector(zAxisTransformed);

            // Określenie teoretycznych HorizontalCoordinate dla przekształconego wektora +Y,
            // jeżeli urządzenie jest trzymane pionowo
            double yUprightAltitude = 0;
            double yUprightAzimuth = 0;

            if (horzCoord.Altitude > 0)
            {
                yUprightAltitude = 90 - horzCoord.Altitude;
                yUprightAzimuth = 180 + horzCoord.Azimuth;
            }
            else
            {
                yUprightAltitude = 90 + horzCoord.Altitude;
                yUprightAzimuth = horzCoord.Azimuth;
            }
        }
    }
}

```

```

    }
    Vector3 yUprightVector =
        new HorizontalCoordinate(yUprightAzimuth, yUprightAltitude).ToVector();

    // Określenie rzeczywistego przekształconego wektora +Y
    Vector3 yAxisTransformed = Vector3.Transform(Vector3.Unity, matrix);

    // Pobranie kąta pomiędzy wektorem +Y a rzeczywistym, przekształconym wektorem +Y
    double dotProduct = Vector3.Dot(yUprightVector, yAxisTransformed);
    Vector3 crossProduct = Vector3.Cross(yUprightVector, yAxisTransformed);
    crossProduct = crossProduct.Normalized;

    // Czasami dotProduct jest nieco większy od 1, co
    // generuje wyjątek w obliczeniach angleBetween, więc...
    dotProduct = Math.Min(dotProduct, 1);
    double angleBetween = 180 * Vector3.Dot(zAxisTransformed, crossProduct)
        * Math.Acos(dotProduct) / Math.PI;
    horzCoord.Tilt = angleBetween;

    return horzCoord;
}

public Vector3 ToVector()
{
    double x = Math.Cos(Math.PI * this.Altitude / 180) *
        Math.Sin(Math.PI * this.Azimuth / 180);
    double y = Math.Cos(Math.PI * this.Altitude / 180) *
        Math.Cos(Math.PI * this.Azimuth / 180);
    double z = Math.Sin(Math.PI * this.Altitude / 180);

    return new Vector3((float)x, (float)y, (float)z);
}

public override string ToString()
{
    return String.Format("Azy: {0} Wys: {1} Poch: {2}",
        this.Azimuth, this.Altitude, this.Tilt);
}
}
}

```

Po wykonaniu tej konwersji jesteś na dobrej drodze do napisania programu astronomicznego, który będzie wyświetlać określony fragment nocnego nieba w zależności od orientacji ekranu, podobnie jak zrobiłem to dla Windows Phone 7.5 we wrześniowym numerze „MSDN Magazine” z 2012 roku. Zróbmy teraz coś mniej ambitnego.

Co możemy zrobić, aby obejrzeć mapę bitową, która jest znacznie większa od ekranu komputera, a nie chcemy jej zmniejszać? Tradycyjnym rozwiązaniem jest użycie pasków przewijania. Nieco nowocześniejsze rozwiązanie pozwala na przesuwanie obrazu palcami.

Jednak inne podejście polega na umieszczeniu mapy bitowej na powierzchni sfery niebieskiej. Obraz ten można oglądać przez trzymanie tabletu przed sobą i zmianę orientacji ekranu. Oczywiście nie chcemy rozciągać mapy bitowej, aby pasowała do powierzchni sfery. Zamiast tego użyjemy azymutu do przewijania w poziomie i wysokości do przewijania w pionie.

Program *EarthlyDelights* pozwala na oglądanie dużej (7793 na 4409 piksele) mapy bitowej z 500-letnim obrazem Hieronima Boscha, *Ogród ziemskich rozkoszy*. Program pobiera ten obraz z Wikipedii. Poniżej przedstawiony jest fragment wyświetlany za pomocą programu uruchomionego na Microsoft Surface:



Program nie posiada interfejsu dotykowego do skanowania lub zmiany rozmiaru obrazu. Wszystko bazuje na zmianie orientacji ekranu. Jeżeli jednak stukniesz ekran, program wykona skalowanie, aby pokazać cały obraz z zaznaczonym prostokątem pokazującym obszar wyświetlany w zwykłym trybie.



Funkcja ta nieco komplikuje program, ale uznałem ją za niezbędną.

Najważniejszą częścią pliku XAML jest oczywiście element Image. Zwróć uwagę, że właściwość Stretch elementu Image ma wartość None oraz zawiera obiekt BitmapImage bez ustawionego adresu URI źródła (na razie). Element Grid zawierający Image znajduje się w obiekcie Canvas, więc nie będzie przycięty, gdy będzie większy od ekranu (a na pewno będzie).

Listing 18.18. Projekt: EarthlyDelights | Plik: MainPage.xaml (fragment)

```
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
  <!-- Dwa elementy wyświetlane wyłącznie w czasie pobierania pliku -->
  <ProgressBar Name="progressBar"
    VerticalAlignment="Center"
    Margin="96 0" />

  <TextBlock Name="statusText"
    Text="Ładowanie zdjęcia..."
    HorizontalAlignment="Center"
    VerticalAlignment="Center" />
  <Canvas>
    <Grid>
      <Image Stretch="None">
        <Image.Source>
          <BitmapImage x:Name="bitmapImage"
            DownloadProgress="OnBitmapImageDownloadProgress"
            ImageFailed="OnBitmapImageFailed"
            ImageOpened="OnBitmapImageOpened" />
        </Image.Source>
      </Image>

      <Border Name="outlineBorder"
        BorderBrush="White"
        HorizontalAlignment="Left"
        VerticalAlignment="Top">

        <Rectangle Name="outlineRectangle"
          Stroke="Black" />

        <Border.RenderTransform>
          <CompositeTransform x:Name="borderTransform" />
        </Border.RenderTransform>
      </Border>

      <Grid.RenderTransform>
        <CompositeTransform x:Name="imageTransform" />
      </Grid.RenderTransform>
    </Grid>
  </Canvas>

  <TextBlock Name="titleText"
    Margin="2" />
</Grid>
```

Element Border z zagnieżdżonym Rectangle jest używany w widoku przeskalowanym do pokazania części obrazu, która normalnie zajmuje cały ekran, ale możesz widzieć ten prostokąt również w widoku normalnym. Zewnętrzny element CompositeTransform

odnosi się zarówno do Image, jak i Border. W widoku normalnym transformacja ta nie wykonuje żadnych operacji. Wewnętrzny element CompositeTransform ustawia Border w tym samym obszarze, który jest widoczny w trybie normalnym.

Metoda obsługi zdarzenia Loaded sprawdza, czy dostępny jest obiekt OrientationSensor, i jeżeli jest, zaczyna pobieranie przez ustawienie właściwości UriSource w obiekcie BitmapImage. Po prawidłowym pobraniu obrazu odczytywane są jego wymiary, które razem z wymiarami strony są zapisywane w polach obiektu.

Listing 18.19. Projekt: EarthlyDelights | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
    ...
    OrientationSensor orientationSensor = OrientationSensor.GetDefault();
    double pageWidth, pageHeight, maxDimension;
    int imageWidth, imageHeight;
    string title = "The Garden of Earthly Delights. Autor: Hieronymus Bosch";
    double zoomInScale;
    double rotation;
    bool isZoomView;

    public MainPage()
    {
        this.InitializeComponent();
        DisplayProperties.AutoRotationPreferences =
            ↳DisplayProperties.NativeOrientation;
        Loaded += OnMainPageLoaded;
        SizeChanged += OnMainPageSizeChanged;
    }
    ...
    async void OnMainPageLoaded(object sender, RoutedEventArgs args)
    {
        if (orientationSensor == null)
        {
            await new MessageDialog("OrientationSensor nie jest dostępny",
                "Earthly Delights").ShowAsync();

            progressBar.Visibility = Visibility.Collapsed;
            statusText.Visibility = Visibility.Collapsed;
        }
        else
        {
            bitmapImage.UriSource =
                new Uri("http://upload.wikimedia.org/wikipedia/commons/6/62/
                    ↳The_Garden_of_Earthly_Delights_by_Bosch_High_Resolution_2.jpg");
        }
    }

    void OnMainPageSizeChanged(object sender, SizeChangedEventArgs args)
    {
        // Zapisanie rozmiarów strony
        pageWidth = this.ActualWidth;
        pageHeight = this.ActualHeight;
        maxDimension = Math.Max(pageWidth, pageHeight);

        // Inicjalizacja wartości
    }
}
```

```

        outlineBorder.Width = pageWidth;
        outlineBorder.Height = pageHeight;
        borderTransform.CenterX = pageWidth / 2;
        borderTransform.CenterY = pageHeight / 2;
    }

    void OnBitmapImageDownloadProgress(object sender, DownloadProgressEventArgs args)
    {
        progressBar.Value = args.Progress;
    }

    async void OnBitmapImageFailed(object sender, ExceptionRoutedEventArgs args)
    {
        progressBar.Visibility = Visibility.Collapsed;
        statusText.Visibility = Visibility.Collapsed;

        await new MessageDialog("Nie można pobrać obrazu: " + args.ErrorMessage,
                                "Earthly Delights").ShowAsync();
    }

    void OnBitmapImageOpened(object sender, RoutedEventArgs args)
    {
        progressBar.Visibility = Visibility.Collapsed;
        statusText.Visibility = Visibility.Collapsed;

        // Zapisanie wymiarów obrazu
        imageWidth = bitmapImage.PixelWidth;
        imageHeight = bitmapImage.PixelHeight;
        titleText.Text = String.Format("{0} ({{1}}\x00D7{{2}})", title, imageWidth,
imageHeight);

        // Inicjalizacja przekształcenia obrazu
        zoomInScale = Math.Min(pageWidth / imageWidth, pageHeight / imageHeight);

        // Uruchomienie OrientationSensor
        if (orientationSensor != null)
        {
            ProcessNewOrientationReading(orientationSensor.GetCurrentReading());
            orientationSensor.ReportInterval = orientationSensor.MinimumReportInterval;
            orientationSensor.ReadingChanged += OnOrientationSensorReadingChanged;
        }
    }

    async void OnOrientationSensorReadingChanged(OrientationSensor sender,
                                                OrientationSensorReadingChanged
                                                ↳EventArgs args)
    {
        await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
        {
            ProcessNewOrientationReading(args.Reading);
        });
    }
    ...
}

```

Metoda `ProcessNewOrientationReading` tworzy obiekt `Matrix3D` na podstawie `Sensor ↳RotationMatrix` i korzysta z niego do uzyskania wartości `HorizontalCoordinate`.

Listing 18.20. Projekt: EarthlyDelights | Plik: MainPage.xaml.cs (fragment)

```
void ProcessNewOrientationReading(OrientationSensorReading orientationReading)
{
    if (orientationReading == null)
        return;

    // Pobranie macierzy obrotu i konwersja na współrzędne horyzontalne
    SensorRotationMatrix m = orientationReading.RotationMatrix;

    if (m == null)
        return;

    Matrix3D matrix3d = new Matrix3D(m.M11, m.M12, m.M13, 0,
                                     m.M21, m.M22, m.M23, 0,
                                     m.M31, m.M32, m.M33, 0,
                                     0, 0, 0, 1);

    if (!matrix3d.HasInverse)
        return;

    HorizontalCoordinate horzCoord = HorizontalCoordinate.FromMotionMatrix(matrix3d);

    // Ustawienie środka przekształcenia w elemencie Image
    imageTransform.CenterX = (imageWidth + maxDimension) *
                             (180 + horzCoord.Azimuth) / 360 - maxDimension / 2;
    imageTransform.CenterY = (imageHeight + maxDimension) *
                             (90 - horzCoord.Altitude) / 180 - maxDimension / 2;

    // Ustawienie przesunięcia dla elementu Border
    borderTransform.TranslateX = imageTransform.CenterX - pageWidth / 2;
    borderTransform.TranslateY = imageTransform.CenterY - pageHeight / 2;

    // Pobranie obrotu z Tilt
    rotation = -horzCoord.Tilt;
    UpdateImageTransforms();
}
```

Metoda ta jest odpowiedzialna za ustawienie części transformacji; pozostałe są ustawiane w metodzie `UpdateImageTransforms` (której wywołanie następuje na końcu tej metody). Dla azymutu równego 0 (co następuje, gdy tablet jest skierowany na północ) oraz wysokości równej 0 (czyli gdy tablet jest ustawiony pionowo) właściwości `CenterX` i `CenterY` są ustawione na środek obrazu. W pozostałych przypadkach są ustawione na wartości obejmujące całą szerokość i wysokość, razem z marginesem, dzięki któremu będzie można wyświetlić obszar, w którym nie jest widoczny żaden fragment obrazu (w przeciwnym razie program musiałby pokazywać jednocześnie prawą krawędź obrazu na lewej stronie ekranu oraz lewą krawędź na prawej stronie ekranu).

Chciałem, aby operacja skalowania była animowana, więc dodałem do `MainPage` właściwość zależną sterującą animacją po stuknięciu ekranu.

Listing 18.21. Projekt: EarthlyDelights | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
    // Właściwość zależna do animacji skalowania
    static readonly DependencyProperty interpolationFactorProperty =
        DependencyProperty.Register("InterpolationFactor",
```

```

        typeof(double),
        typeof(MainPage),
        new PropertyMetadata(0.0,
OnInterpolationFactorChanged));
...
    // Właściwość współczynnika interpolacji
    public static DependencyProperty InterpolationFactorProperty
    {
        get { return interpolationFactorProperty; }
    }

    public double InterpolationFactor
    {
        set { SetValue(InterpolationFactorProperty, value); }
        get { return (double)GetValue(InterpolationFactorProperty); }
    }
...
    protected override void OnTapped(TappedRoutedEventArgs e)
    {
        // Animacja właściwości InterpolationFactor
        DoubleAnimation doubleAnimation = new DoubleAnimation
        {
            EnableDependentAnimation = true,
            To = isZoomView ? 0 : 1,
            Duration = new Duration(TimeSpan.FromSeconds(1))
        };
        Storyboard.SetTarget(doubleAnimation, this);
        Storyboard.SetTargetProperty(doubleAnimation, "InterpolationFactor");
        Storyboard storyboard = new Storyboard();
        storyboard.Children.Add(doubleAnimation);
        storyboard.Begin();
        isZoomView ^= true;
        base.OnTapped(e);
    }

    static void OnInterpolationFactorChanged(DependencyObject obj,
        DependencyPropertyChangedEventArgs
        ↪ args)
    {
        (obj as MainPage).UpdateImageTransforms();
    }
...
}

```

Metoda `OnInterpolationFactorChanged` wywołuje również `UpdateImageTransforms`, która realizuje większość trudnych zadań.

Listing 18.22. Projekt: EarthlyDelights | Plik: MainPage.xaml.cs (fragment)

```

void UpdateImageTransforms()
{
    // Jeżeli powiększony, ustaw skalowanie
    double interpolatedScale = 1 + InterpolationFactor * (zoomInScale - 1);
    imageTransform.ScaleX =
    imageTransform.ScaleY = interpolatedScale;

    // Przenieś środek przekształcenia na środek ekranu

```

```

imageTransform.TranslateX = pageWidth / 2 - imageTransform.CenterX;
imageTransform.TranslateY = pageHeight / 2 - imageTransform.CenterY;

// Jeżeli pomniejszony, korekta dla skalowania
imageTransform.TranslateX -= InterpolationFactor *
    (pageWidth / 2 - zoomInScale * imageTransform.CenterX);
imageTransform.TranslateY -= InterpolationFactor *
    (pageHeight / 2 - zoomInScale * imageTransform.CenterY);

// Jeżeli pomniejszony, wyśrodkuj obraz na ekranie
imageTransform.TranslateX += InterpolationFactor *
    (pageWidth - zoomInScale * imageWidth) / 2;
imageTransform.TranslateY += InterpolationFactor *
    (pageHeight - zoomInScale * imageHeight) / 2;

// Ustaw grubość ramki
outlineBorder.BorderThickness = new Thickness(2 / interpolatedScale);
outlineRectangle.StrokeThickness = 2 / interpolatedScale;

// Ustaw obrót i ramkę
imageTransform.Rotation = (1 - InterpolationFactor) * rotation;
borderTransform.Rotation = -rotation;
}

```

Metoda ta jest wywoływana, gdy pojawi się nowa wartość `OrientationSensor` lub gdy zmieni się wartość właściwości `InterpolationFactor` w czasie operacji skalowania. Jeżeli jesteś zainteresowany szczegółami działania tej metody, możesz ją uprościć przez wyeliminowanie całego kodu interpolacji. Ustaw `InterpolationFactor` na 0, a następnie na 1 — zobaczysz, że jest to dosyć proste.

Mapy oraz kafelki map Bing

Klasa `Geolocator` nie jest uznawana za sensor i jest zdefiniowana w całości w innej przestrzeni nazw — `Windows.Devices.Geolocation`. Jednak jest nieco podobna, ponieważ uruchamiamy ją, a następnie informuje nas ona o zmianie lokalizacji geograficznej.

Musimy jawnie wskazać w sekcji `Capabilities` pliku `Package.appxmanifest`, że nasza aplikacja wymaga uprawnień `Location`. Przy pierwszym uruchomieniu aplikacji Windows 8 poprosi użytkownika o potwierdzenie.

Zazwyczaj korzystamy z lokalizacji zwracanej przez `Geolocator` w połączeniu z mapami. Kontrolka map Bing nie jest wbudowana w Windows 8, ale może być pobrana w postaci pakietu umożliwiającego dodanie jej do aplikacji. Konieczny jest również klucz uprawnień, który można uzyskać na witrynie www.bingmapsportal.com.

Jednak w ostatnim programie z tego rozdziału użyję nieco innego podejścia. Zamierzam pokazać mapę, która obraca się zgodnie z orientacją tabletu. Obrót taki pozwoli zorientować mapę względem rzeczywistej północy (lub innego odczytu zwracanego przez tablet). W tym celu nie będę korzystał z kontrolki Bing Maps. Zamiast tego użyję usługi SOAP Bing Maps, która pozwala na pobranie pojedynczych kafelków — połączę je w pełną mapę. Klucz uprawnień jest tu również wymagany.

Po uruchomieniu programu `RotatingMap` być może będziesz chciał przesunąć i skalować mapę palcami. To nie działa. Program nie posiada interfejsu dotykowego! Aby uprościć

przykład, sprawiłem, że program środkuje mapę w bieżącej lokalizacji i zmienia jej orientację po zmianie lokalizacji. Program posiada na pasku aplikacji przyciski do powiększania i zmniejszania skali oraz do przełączania się pomiędzy widokiem mapy a widokiem zdjęcia satelitarnego, ale to wszystko.

Zawartość pliku XAML zamieszczona jest na poniższym listingu. Wszystkie kafelki składające się na mapę są umieszczane w obiekcie Canvas o nazwie imageCanvas. Zwróć uwagę na element RotateTransform pozwalający obracać Canvas wokół jego środka.

Listing 18.23. Projekt: RotatingMap | Plik: MainPage.xaml (fragment)

```
<Page ... >
```

```
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
  <Canvas Name="imageCanvas"
    HorizontalAlignment="Center"
    VerticalAlignment="Center">
    <Canvas.RenderTransform>
      <RotateTransform x:Name="imageCanvasRotate" />
    </Canvas.RenderTransform>
  </Canvas>

  <!-- Okrąg do pokazania lokalizacji -->
  <Ellipse Name="locationDisplay"
    Width="24"
    Height="24"
    Stroke="Red"
    StrokeThickness="6"
    HorizontalAlignment="Center"
    VerticalAlignment="Center"
    Visibility="Collapsed" />

  <!-- Strzałka wskazująca północ -->
  <Border HorizontalAlignment="Left"
    VerticalAlignment="Top"
    Margin="12"
    Background="Black"
    Width="36"
    Height="36"
    CornerRadius="18">
    <Path Stroke="White"
      StrokeThickness="3"
      Data="M 18 4 L 18 24 M 12 12 L 18 4 24 12">
      <Path.RenderTransform>
        <RotateTransform x:Name="northArrowRotate"
          CenterX="18"
          CenterY="18" />
      </Path.RenderTransform>
    </Path>
  </Border>

  <!-- Wyświetlenie "powered by bing" -->
  <Border Background="Black"
    HorizontalAlignment="Center"
    VerticalAlignment="Bottom"
    Margin="12"
    CornerRadius="12"
```

```

        Padding="3">
        <StackPanel Name="poweredByDisplay"
            Orientation="Horizontal"
            Visibility="Collapsed">
            <TextBlock Text=" powered by "
                Foreground="White"
                VerticalAlignment="Center" />
            <Image Stretch="None">
                <Image.Source>
                    <BitmapImage x:Name="poweredByBitmap" />
                </Image.Source>
            </Image>
        </StackPanel>
    </Border>
</Grid>

<Page.BottomAppBar>
    <AppBar Name="bottomAppBar"
        IsEnabled="False">
        <StackPanel Orientation="Horizontal"
            HorizontalAlignment="Right">

            <!-- Usunięcie odwołania do BackgroundCheckedGlyph w
                AppBarButtonStyle w celu użycia go jako CheckBox -->

            <CheckBox Name="streetViewAppBarButton"
                Style="{StaticResource StreetAppBarButtonStyle}"
                AutomationProperties.Name="Street View"
                Checked="OnStreetViewAppBarButtonChecked"
                Unchecked="OnStreetViewAppBarButtonChecked" />

            <Button Name="zoomInAppBarButton"
                Style="{StaticResource ZoomInAppBarButtonStyle}"
                Click="OnZoomInAppBarButtonClick" />

            <Button Name="zoomOutAppBarButton"
                Style="{StaticResource ZoomOutAppBarButtonStyle}"
                Click="OnZoomOutAppBarButtonClick" />
        </StackPanel>
    </AppBar>
</Page.BottomAppBar>
</Page>

```

Możliwe jest „ręczne” wywoływanie usług SOAP Bing Maps przez przesyłanie i analizowanie plików XML, ale znacznie bezpieczniejszym podejściem jest korzystanie z usługi sieciowej poprzez klasy pośredniczące generowane przez Visual Studio. Klasy pośredniczące powodują, że usługa sieciowa jest reprezentowana przez zestaw struktur, typów wyliczeniowych i asynchronicznych wywołań metod. Aby dodać te klasy do programu *RotatingMap*, kliknąłem prawym przyciskiem myszy nazwę projektu w oknie *Solution Explorer* i wybrałem z menu opcję *Add Service Reference*. Gdy zostało wyświetlone okno dialogowe z adresem, wkleiłem do niego URL Imagery Service (który możesz znaleźć na stronie <http://msdn.microsoft.com/en-us/library/cc966738.aspx> zawierającej adresy trzech innych usług sieciowych połączonych z Bing Maps). Nadałem jej nazwę *ImageryService*, co spowodowało, że Visual Studio wygenerował kod w przestrzeni nazw *RotatingMap.ImageryService*.

Usługa zawiera dwa typy żądań — `GetMapUriAsync` oraz `GetImageryMetadataAsync`. Pierwszy typ pozwala na uzyskanie statycznej mapy dla określonej lokalizacji, ale ja skorzystałem z drugiego, który pozwala na pozyskanie informacji potrzebnych do pobrania kafelków mapy, które razem tworzą kompletną mapę.

Zacznijmy zapoznanie się z kodem *RotatingMap* od konstruktora klasy `MainPage`. Jak widać, zapisuje on w ustawieniach aplikacji tylko dwie wartości — styl mapy (wartość typu wyliczeniowego `MapStyle`, który jest generowany w czasie dodawania usługi sieciowej do projektu zawierającego wartości określające mapę lub zdjęcia satelitarne) oraz całkowity poziom powiększenia.

Listing 18.24. Projekt: RotatingMap | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
    ...
    // Zapisanie jako ustawienia aplikacji
    MapStyle mapStyle = MapStyle.Aerial;
    int zoomLevel = 12;
    public MainPage()
    {
        this.InitializeComponent();
        DisplayProperties.AutoRotationPreferences =
            ↪DisplayProperties.NativeOrientation;
        Loaded += OnMainPageLoaded;
        SizeChanged += OnMainPageSizeChanged;

        // Pobranie ustawień aplikacji (i ich późniejsze zapisanie)
        IPropertySet propertySet = ApplicationData.Current.LocalSettings.Values;

        if (propertySet.ContainsKey("ZoomLevel"))
            zoomLevel = (int)propertySet["ZoomLevel"];

        if (propertySet.ContainsKey("MapStyle"))
            mapStyle = (MapStyle)(int)propertySet["MapStyle"];

        Application.Current.Suspending += (sender, args) =>
        {
            propertySet["ZoomLevel"] = zoomLevel;
            propertySet["MapStyle"] = (int)mapStyle;
        };
    }
    ...
}
```

Usługa sieciowa jest wykorzystywana wyłącznie w metodzie obsługi zdarzenia `Loaded`. Konieczne jest wykonanie dwóch wywołań — jednego do pobrania metadanych mapy dla widoku mapy oraz drugiego dla widoku satelitarnego. Dane te są zapisywane w dwóch obiektach lokalnej klasy o nazwie `ViewParams`. Najważniejszą częścią metadanych jest szablon URI do pobierania poszczególnych kafelków mapy. Klasa `ViewParams` posiada również pola na maksymalny i minimalny poziom powiększenia, ale wiem, że zmienia się on w zakresie od 1 do 21 i w kolejnych częściach kodu zakładam maksymalny poziom równy 21.

Listing 18.25. Projekt: RotatingMap | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
    ...
    // Parametry przechowywania dla dwóch widoków
    class ViewParams
    {
        public string UriTemplate;
        public int MinimumLevel;
        public int MaximumLevel;
    }
    ViewParams aerialParams;
    ViewParams roadParams;

    Geolocator geolocator = new Geolocator();
    Inclinometer inclinometer = Inclinometer.GetDefault();
    ...
    async void OnMainPageLoaded(object sender, RoutedEventArgs args)
    {
        // Inicjowanie usługi obrazów Bing Maps
        ImageryServiceClient imageryServiceClient =
            new ImageryServiceClient(
                ImageryServiceClient.EndpointConfiguration.
                ↪BasicHttpBinding_IImageryService);

        // Wykonanie dwóch zadań dla mapy i zdjęć
        ImageryMetadataRequest request = new ImageryMetadataRequest
        {
            Credentials = new Credentials
            {
                ApplicationId = "AkNpobMGtsXUh9o8T9j1doUmjhtcoexUHHTClyBILqnrags-
                ↪ibtcHruZQPfFo61cn8"
            },
            Style = MapStyle.Road
        };
        Task<ImageryMetadataResponse> roadStyleTask =
            imageryServiceClient.GetImageryMetadataAsync(request);

        request = new ImageryMetadataRequest
        {
            Credentials = new Credentials
            {
                ApplicationId = "Tu wstaw klucz uprawnień Bing"
            },
            Style = MapStyle.Aerial
        };
        Task<ImageryMetadataResponse> aerialStyleTask =
            imageryServiceClient.GetImageryMetadataAsync(request);

        // Oczekiwanie na zakończenie obu zadań
        Task.WaitAll(roadStyleTask, aerialStyleTask);

        // Sprawdzenie poprawności
        if (!roadStyleTask.IsCanceled && !roadStyleTask.IsFaulted &&
            !aerialStyleTask.IsCanceled && !aerialStyleTask.IsCanceled)
        {
```

```

// Pobranie mapy bitowej "powered by"
poweredByBitmap.UriSource = roadStyleTask.Result.BrandLogoUri;
poweredByDisplay.Visibility = Visibility.Visible;

// Pobranie URI oraz minimalnych i maksymalnych poziomów powiększenia
roadParams = CreateViewParams(roadStyleTask.Result.Results[0]);
aerialParams = CreateViewParams(aerialStyleTask.Result.Results[0]);

// Pobranie bieżącej lokalizacji
Geoposition geoPosition = await geolocator.GetGeopositionAsync();
GetLongitudeAndLatitude(geoPosition.Coordinate);
RefreshDisplay();

// Pobranie zaktualizowanej lokalizacji
geolocator.PositionChanged += OnGeolocatorPositionChanged;

// Aktywowanie paska aplikacji
bottomAppBar.IsEnabled = true;
streetViewAppBarButton.IsChecked = mapStyle == MapStyle.Road;

// Pobranie bieżącego obrotu
if (inclinometer != null)
{
    SetRotation(inclinometer.GetCurrentReading());
    inclinometer.ReadingChanged += OnInclinometerReadingChanged;
}
}
}

ViewParams CreateViewParams(ImageryMetadataResult result)
{
    string uri = result.ImageUri;
    uri = uri.Replace("{subdomain}", result.ImageUriSubdomains[0]);
    uri = uri.Replace("&token={token}", "");
    uri = uri.Replace("{culture}", "en-us");

    return new ViewParams
    {
        UriTemplate = uri,
        MinimumLevel = result.ZoomRange.From,
        MaximumLevel = result.ZoomRange.To
    };
}
...
}

```

Do odczytania metadanych dla dwóch widoków potrzebne są dwa wywołania asynchroniczne, ale ponieważ nie są one od siebie zależne, mogą być wykonywane w tym samym czasie. Jest to świetne zastosowanie dla metody `Task.WaitAll`, która czeka na zakończenie wykonywania wielu obiektów `Task`.

Gdy oba wywołania usługi sieciowej zakończą się prawidłowo, tworzone są obiekty `Geolocator` oraz `Inclinometer`. Obiekt `Inclinometer` jest używany wyłącznie do odczytania wartości odchylenia używanej przy obracaniu mapy oraz do obracania strzałki wskazującej północ.

Listing 18.26. Projekt: RotatingMap | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
    ...
    async void OnInclinometerReadingChanged(Inclinometer sender,
                                             InclinometerReadingChangedEventArgs args)
    {
        await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
        {
            SetRotation(args.Reading);
        });
    }

    void SetRotation(InclinometerReading inclinometerReading)
    {
        if (inclinometerReading == null)
            return;

        imageCanvasRotate.Angle = inclinometerReading.YawDegrees;
        northArrowRotate.Angle = inclinometerReading.YawDegrees;
    }
    ...
}
```

Na tym kończy się metoda obsługi zdarzenia Loaded i program posiada teraz dwa szablonowy adresu URI, za pomocą których może pobrać poszczególne kafelki mapy. Kafelki tworzące podstawową mapę Bing są kwadratowymi mapami bitowymi o boku 256 pikseli. Każdy kafelek jest skojarzony z określoną długością i szerokością geograficzną, a także poziomem powiększenia oraz zawiera obraz fragmentu świata spłaszczony za pomocą odwzorowania walcowego równokątnego.

Na poziomie 1. cała Ziemia — a właściwie część Ziemi o szerokościach geograficznych pomiędzy dodatnim i ujemnym stopniem 85,05 — jest odwzorowana za pomocą czterech kafelków:



Wkrótce przedstawię sens liczb w kafelkach. Kafelki mają bok o 256 pikselach, więc na równiku każdy piksel obejmuje około 78 kilometrów.

Na poziomie 2. Ziemia jest odwzorowana przez 16 kafelków.



Kafelki mają bok o 256 pikselach, więc na równiku każdy piksel obejmuje około 38 kilometrów.

Każdy kafelek z poziomu 1. obejmuje taki sam obszar jak cztery kafelki z poziomu 2. i trend jest kontynuowany — poziom 3. zawiera 64 kafelki, poziom 4. zawiera 256 kafelków, i tak aż do poziomu 21., który (zgodnie z zasadą) odwzorowuje całą Ziemię za pomocą ponad 4 trylionów kafelków — 2 miliony poziomo i 2 miliony pionowo, przy rozdzielczości na równiku równej 7,5 cm na piksel.

W jaki sposób można spójnie zorganizować tak wiele kafelków? Pamiętaj, że użyte są tu trzy wymiary — długość, szerokość i poziom powiększenia — a dla maksymalnej efektywności przesyłania tych kafelków poprzez usługę sieciową kafelki obejmujące ten sam obszar powinny być przechowywane na serwerach obok siebie.

Oczywiście został użyty bardzo sprytny sposób numerowania, nazywany *quadkey*. Każdy kafelek ma unikatową wartość *quadkey*. Szablony URI odczytane z usługi Bing Maps zawierają znacznik {quadkey}, który powinien być zastąpiony odwołaniem do odpowiedniego kafelka. Przedstawione wcześniej rysunki mają pokazane wartości *quadkey* w lewym górnym narożniku każdego kafelka. Wiodące zera są istotne! Liczba cyfr w numerze jest równa poziomowi powiększenia. Kafelki na poziomie 21. są identyfikowane za pomocą 21 znakowych wartości *quadkey*.

W kodzie tym używane są wyłącznie cyfry 0, 1, 2 i 3, czyli są to liczby w systemie czwórkowym. Binarnie zapisane liczby 0, 1, 2 i 3 to 00, 01, 10 i 11. Pierwszy bit jest współrzędną pionową, a drugi współrzędną poziomą. Dzięki temu bity odpowiadają przeplataną długości i szerokości.

Jak już wiesz, każdy kafelek na poziomie 1. odpowiada czterem kafelkom z poziomu 2., więc można uważać, że kafelki mają relacje „nadrzędny – podrzędny”. Wartość *quadkey* kafelka podrzędnego zaczyna się tymi samymi cyframi co wartość jego kafelka nadrzędnego, ale dodana jest jeszcze jedna cyfra określająca lokalizację w tym elemencie nadrzędnym.

Można łatwo pobrać wartość quadkey kafelka nadrzędnego z podrzędnego, odcinając po prostu ostatnią cyfrę.

Aby użyć usługi sieciowe Bing Maps, konieczne jest określenie wartości quadkey dla dowolnej długości i szerokości geograficznej. Kod pokazanej w następnym listingu metody `GetLongitudeAndLatitude` ilustruje pierwszy krok, czyli konwersję długości i szerokości geograficznej z `Geolocator` na wartości `double` z zakresu od 0 do 1, a następnie na wartości całkowite:

Listing 18.27. Projekt: RotatingMap | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
    const int BITRES = 29;
    ...
    int integerLongitude = -1;
    int integerLatitude = -1;
    ...
    async void OnGeolocatorPositionChanged(Geolocator sender, PositionChangedEventArgs
    ↪args)
    {
        await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
        {
            GetLongitudeAndLatitude(args.Position.Coordinate);
            RefreshDisplay();
        });
    }

    void GetLongitudeAndLatitude(Geocoordinate geoCoordinate)
    {
        LocationDisplay.Visibility = Visibility.Visible;

        // Obliczenie całkowitej długości i szerokości geograficznej
        double relativeLongitude = (180 + geoCoordinate.Longitude) / 360;
        integerLongitude = (int)(relativeLongitude * (1 << BITRES));

        double sinTerm = Math.Sin(Math.PI * geoCoordinate.Latitude / 180);
        double relativeLatitude = 0.5 - Math.Log((1 + sinTerm) / (1 - sinTerm)) / (4 *
        ↪Math.PI);
        integerLatitude = (int)(relativeLatitude * (1 << BITRES));
    }
    ...
}
```

Wartość `BITRES` wynosi 29, co obejmuje 21 bitów wartości quadkey na poziomie 21. oraz 8 bitów na wielkość kafelka w pikselach, co oznacza, że te wartości całkowite identyfikują długość i szerokość geograficzną z dokładnością do najbliższego piksela kafelka na najwyższym poziomie powiększenia. Wyliczenie wartości `integerLongitude` jest bardzo proste, ale `integerLatitude` znacznie bardziej skomplikowane, ponieważ przekształcenie mapy typu Mercator powoduje kompresję szerokości geograficznych wraz ze wzrostem odległości od równika.

Na przykład środek Central Parku w Nowym Jorku ma długość geograficzną równą $-73,965368$ oraz szerokość $40,783271$. Względne wartości `double` (zaokrąglone do kilku miejsc po przecinku) wynoszą $0,29454$ i $0,37572$. 29-bitowe wartości całkowite (pokazane w systemie binarnym i pogrupowane po cztery cyfry dla zwiększenia czytelności) wynoszą:

```
0 1001 0110 1100 1110 0000 1000 0000
0 1100 0000 0101 1110 1011 0000 0000
```

Załóżmy, że potrzebujesz kafelka dla tej długości i szerokości geograficznej i powiększenia poziomu 12. Musimy użyć górnych 12 bitów tej liczby określającej długość i szerokość geograficzną (uważaj — wynikowe cyfry są grupowane nieco inaczej).

```
0100 1011 01100110 0000 0010
```

Są to dwie liczby binarne, ale w celu utworzenia wartości quadkey muszą być one być połączone, aby utworzyły liczbę o podstawie 4. Nie można zrobić tego w kodzie bez przeglądania bitów w pętli, ale dla celów demonstracji możesz po prostu pomnożyć przez dwa wszystkie bity szerokości i dodać te dwie wartości tak, jakby były wartościami o podstawie 4:

```
  0100 1011 0110
+ 0220 0000 0020
-----
 0320 1011 0130
```

Jest to wartość, którą należy podstawić w miejsce symbolu {quadkey} w szablonie URI pobranym z usługi sieciowej. Wynikowy URI definiuje kwadratową mapę bitową o boku 256 pikseli.

Poniżej przedstawiony jest kod z *RotatingMap*, w którym na podstawie długości i szerokości geograficznej tworzona jest wartość quadkey. Dla zwiększenia czytelności kod został rozdzielony, aby pokazać na początku wyznaczanie wartości całkowitej, a następnie ciągu znaków:

Listing 18.28. Projekt: RotatingMap | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
    ...
    StringBuilder strBuilder = new StringBuilder()
    ...
    string ToQuadKey(int longitude, int latitude, int level)
    {
        long quadkey = 0;
        int mask = 1 << (level - 1);

        for (int i = 0; i < level; i++)
        {
            quadkey <<= 2;

            if ((longitude & mask) != 0)
                quadkey |= 1;

            if ((latitude & mask) != 0)
                quadkey |= 2;

            mask >>= 1;
        }

        strBuilder.Clear();

        for (int i = 0; i < level; i++)
```

```

        {
            strBuilder.Insert(0, (quadkey & 3).ToString());
            quadkey >>= 2;
        }

        return strBuilder.ToString();
    }
}
...
}

```

Wartość `quadkey` określa kafelek zawierający oczekiwaną długość i szerokość geograficzną i nasza aktualna lokalizacja znajduje się gdzieś wewnątrz tego kafelka. Lokalizacja piksela w kafelku może być definiowana przez następne 8 cyfr w długości i szerokości geograficznej wymaganej do wyliczenia `quadkey`.

Mamy już prawie wszystkie elementy układanki. Ponieważ cała strona musi być wypełniona kwadratowymi kafelkami o boku 256 pikseli, ta tablica kafelków musi dać się obracać; bieżąca pozycja użytkownika znajduje się na środku ekranu, zatem metoda obsługi zdarzenia `SizeChanged` wylicza liczbę potrzebnych kafelków, więc jednocześnie liczbę potrzebnych elementów `Image`. Pole o nazwie `sqrNumTiles` oznacza „pierwiastek z liczby kafelków”. Dla ekranu 1366 na 768 pikseli wynosi on 9. Całkowita liczba kafelków (i elementów `Image`) jest kwadratem tej liczby, czyli 81.

Listing 18.29. Projekt: RotatingMap | Plik: MainPage.xaml.cs (fragment)

```

public sealed partial class MainPage : Page
{
    ...
    int sqrNumTiles;           // Zawsze liczba nieparzysta
    ...
    void OnMainPageSizeChanged(object sender, SizeChangedEventArgs args)
    {
        // Usuwanie istniejących elementów Image
        imageCanvas.Children.Clear();

        // Określenie liczby potrzebnych elementów
        double diagonal = Math.Sqrt(Math.Pow(args.NewSize.Width, 2) +
                                     Math.Pow(args.NewSize.Height, 2));

        sqrNumTiles = 1 + 2 * (int)Math.Ceiling((diagonal / 2) / 256);

        // Tworzenie elementów Image dla tablicy sqrNumTiles * sqrNumTiles
        for (int i = 0; i < sqrNumTiles * sqrNumTiles; i++)
        {
            Image image = new Image
            {
                Source = new BitmapImage(),
                Stretch = Stretch.None
            };
            imageCanvas.Children.Add(image);
        }
        RefreshDisplay();
    }
    ...
}

```


Metoda RefreshDisplay realizuje główną część zadania, przegląda w pętli elementy Image i określa ich wartości quadkey (więc i jednocześnie URI).

Listing 18.30. Projekt: RotatingMap | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page{
...
    void RefreshDisplay()
    {
        if (roadParams == null || aerialParams == null)
            return;

        if (integerLongitude == -1 || integerLatitude == -1)
            return;

        // Pobranie współrzędnych i przesunięcia pikseli na podstawie bieżącego powiększenia
        int croppedLongitude = integerLongitude >> BITRES - zoomLevel;
        int croppedLatitude = integerLatitude >> BITRES - zoomLevel;
        int xPixelOffset = (integerLongitude >> BITRES - zoomLevel - 8) % 256;
        int yPixelOffset = (integerLatitude >> BITRES - zoomLevel - 8) % 256;

        // Przygotowanie pętli
        string uriTemplate = (mapStyle == MapStyle.Road ? roadParams :
        ↪aerialParams).UriTemplate;
        int index = 0;
        int maxValue = (1 << zoomLevel) - 1;

        // Pętla po tablicy elementów Image
        for (int row = -sqrtNumTiles / 2; row <= sqrtNumTiles / 2; row++)
            for (int col = -sqrtNumTiles / 2; col <= sqrtNumTiles / 2; col++)
            {
                // Tworzenie Image i BitmapImage
                Image image = imageCanvas.Children[index] as Image;
                BitmapImage bitmap = image.Source as BitmapImage;
                index++;

                // Sprawdzenie, czy wykroczyliśmy poza granice
                if (croppedLongitude + col < 0 ||
                    croppedLongitude + col > maxValue ||
                    croppedLatitude + row < 0 ||
                    croppedLatitude + row > maxValue)
                {
                    bitmap.UriSource = null;
                }
                else
                {
                    // Obliczenie quadkey i ustawienie URI mapy bitowej
                    int longitude = croppedLongitude + col;
                    int latitude = croppedLatitude + row;
                    string strQuadkey = ToQuadKey(longitude, latitude, zoomLevel);
                    string uri = uriTemplate.Replace("{quadkey}", strQuadkey);
                    bitmap.UriSource = new Uri(uri);
                }

                // Pozycjonowanie elementu Image
                Canvas.SetLeft(image, col * 256 - xPixelOffset);
                Canvas.SetTop(image, row * 256 - yPixelOffset);
            }
    }
}
```

```

    }
}
...
}

```

Pozostało tylko obsłużyć przyciski paska aplikacji. Przyciski powiększenia i pomniejszenia są włączane i wyłączane na podstawie minimalnych i maksymalnych poziomów powiększenia bieżącego widoku, choć w innych częściach programu przyjąłem maksymalną wartość powiększenia równą 21.

Listing 18.31. Projekt: RotatingMap | Plik: MainPage.xaml.cs (fragment)

```

public sealed partial class MainPage : Page{
...
    void OnStreetViewAppBarButtonChecked(object sender, RoutedEventArgs args)
    {
        ToggleButton btn = sender as ToggleButton;
        ViewParams viewParams = null;

        if (btn.IsChecked.Value)
        {
            mapStyle = MapStyle.Road;
            viewParams = roadParams;
        }
        else
        {
            mapStyle = MapStyle.Aerial;
            viewParams = aerialParams;
        }

        zoomLevel = Math.Max(viewParams.MinimumLevel,
            Math.Min(viewParams.MaximumLevel, zoomLevel));

        RefreshDisplay();
        RefreshButtons();
    }

    void OnZoomInAppBarButtonClick(object sender, RoutedEventArgs args)
    {
        zoomLevel += 1;
        RefreshDisplay();
        RefreshButtons();
    }

    void OnZoomOutAppBarButtonClick(object sender, RoutedEventArgs args)
    {
        zoomLevel -= 1;
        RefreshDisplay();
        RefreshButtons();
    }

    void RefreshButtons()
    {
        ViewParams viewParams = streetViewAppBarButton.IsChecked.Value ? roadParams :
            ↪aerialParams;
        zoomInAppBarButton.IsEnabled = zoomLevel < viewParams.MaximumLevel;
    }
}

```

```
        zoomOutAppBarButton.IsEnabled = zoomLevel > viewParams.MinimumLevel;
    }
}
```

Nie jesteśmy przyzwyczajeni do oglądania znanych regionów na obróconej mapie, więc w tym widoku wyspa Manhattan wygląda nieco dziwnie:



Jeżeli jesteś jednak w nieznanym miejscu, trzymając w ręku tablet i próbując określić miejsce, gdzie się znajdujesz, to orientacja mapy względem rzeczywistych warunków może być bardzo pomocna. Może kiedyś etykiety nazw miast i ulic również będą się dały obracać.

Skorowidz

A

- adres URI, 24, 857
- akapit, 892
- akceleratory nawigacji, 587
- akcesor
 - get, 43
 - set, 43
- aktualizacja
 - mapy bitowej, 731, 734
 - PointerInfo, 744
- alfa, 704
- alokacja pamięci, 385
- animacja, 108
 - DoubleAnimation, 428
 - DoubleAnimationUsingKeyFrames, 428
 - liniowa, 369
 - obiektu Ellipse, 702
 - Storyboard, 353
 - właściwości InterpolationFactor, 1003
- animacje, 349
 - All-XAML, 378
 - klatek kluczowych, 386, 390
 - predefiniowane, 392
 - stanów wizualnych, 536
 - typu Object, 390
 - wartości typu double, 359
 - własnych klas, 382
 - właściwości klasy Object, 390
 - właściwości załączanych, 366
 - zmienne, 354
- animowany efekt tęczy, 114
- antyaliasing, 770
- anulowanie
 - metody, 274
 - nawigacji, 579
 - operacji asynchronicznej, 253, 255
- API Runtime Windows, 11
- API WinRT, 11

- aplikacje
 - biurkowe, desktop applications, 9
 - dotykowe, 663
 - Windows Store, 9
- atrament, 1023
- atrybut, *Patrz:* właściwość
- automatyczne parsowanie, 343
- azymut, 991

B

- badanie trafień, 444
- bajt Alfa, 708
- barwa-nasylenie-jasność, 760
- biblioteka
 - Direct3D, 400
 - DirectX, 819
 - DirectXWrapper, 821
 - kernel32.dll, 793, 797
 - Obrazy, 775
 - Petzold.ProgrammingWindows6.Chapter11, 819
 - Petzold.Windows8.VectorDrawing, 770
 - Petzold.Windows8.Controls, 150
 - ReflectionHelper, 134
 - Shapes, 66, 364
 - SharpDX, 820
 - Windows Runtime Component, 820
 - WinJS, 11
- biblioteki DLL, 791
- bitmapa, 27
- bity pikseli, 698
- blok foreach, 281
- bloki try – catch, 255, 262

C

- cechy operacji asynchronicznych, 253
- celownik, 680

chrom, 518
cień, 839
COM, Component Object Model, 11, 791
cykl życia aplikacji, 267
czas trwania animacji, 352, 355
czas UTC, 798, 814
czcionka, 31, 821, 825, 835, 855
 Segoe UI Symbol, 298, 304, 779
 TrueType, 855
czcionki prywatne, 856
części szablonu, 536
czytnik, 880

D

dane wskaźnika, 629
debuger, 26, 268, 270
debugowanie aplikacji, 13
definiowanie struktur, 797
deklaracja przestrzeni nazw, 24
deserializacja XML, 606
DirectX, 66, 819
długi wskaźnik, 792
dodawanie wpisu do słownika, 1051
dokument XPS, 862
dokumentacja
 interfejsu API, 23
 stanów wizualnych, 536
 SYSTEM_INFO, 793
dołączanie metod obsługi, 926
domyślne atrybuty rysowania, 1031, 1036,
 1047
dostęp
 do animacji, 391
 do API, 791
 do biblioteki obrazów, 775
 do biblioteki Obrazy, 786
 do funkcji Win32, 791
 do map bitowych, 29
 do systemu plików, 257
 masowy, 257
dostosowanie lokalizacji, 288
dotyk, 208, 491, 629
drukowanie, 901, 912
 grafiki, 956
 planu pracy, 935
 zakresu stron, 944, 951
drzewo katalogów, 779
dyrektywa using, 22

dziedziczenie
 ustawień, 81
 właściwości, 38
 zdarzeń, 93

E

e-book, 140
edycja
 tekstu, 885
 XAML, 380
edytor, 315
ekran, 555
ekran startowy, 597, 901
element
 BeginInitStoryboard, 379
 Binding, 83, 85, 132, 609
 Border, 118, 187, 491, 521, 922, 999
 BorderBrush, 148
 Button, 184, 614
 Canvas, 153, 158, 322, 378, 929
 CaptureElement, 786
 Color, 179
 ComboBox, 817
 ControlTemplate, 526
 DataTemplate, 470, 610, 613
 DoubleAnimation, 350
 Ellipse, 121, 124, 156, 360, 361, 468
 EllipseGeometry, 386, 975
 EventTrigger, 379
 FlipView, 880
 Glyph, 929
 Glyphs, 860, 862
 Grid, 41, 158, 168–176, 315, 324, 334
 wewnętrzny, 174
 zewnętrzny, 174
 GridView, 614
 Image, 27, 43, 124, 158, 701, 705, 706, 718,
 719, 870, 958
 InkStrokeRenderingSegment, 1025
 InlineUIContainer, 870
 ItemTemplate, 778
 Line, 324, 649
 LinearGradientBrush, 54, 80
 ListBox, 491, 836
 ListView, 613
 Page, 31, 332
 Path, 72, 363, 1045
 Polygon, 414
 Polyline, 181, 202, 370

- ProgressBar, 541
- ProgressBarIndicator, 542
- RadioButton, 591
- Rectangle, 121, 146, 173, 529, 537
- RectangleGeometry, 439
- RelativeSource, 85
- ResourceDictionary, 544
- RichEditBox, 885
- RichTextBlock, 600, 855, 867, 871, 881
- RichTextBlockOverflow, 872, 875, 883
- RotateTransform, 1005
- Run, 106, 139
- ScrollViewer, 132, 138, 281, 297, 507
- Slider, 173, 176–180, 221, 880
- SolidColorBrush, 535
- StackPanel, 117, 123–128, 138, 143, 147, 281, 346, 875, 926
- StaticResource, 85
- Storyboard, 350
- Style, 78, 80, 519
- TargetType, 544
- TemplateBinding, 522
- TextBlock, 31, 41, 59, 84, 99–102, 107, 159, 187, 412, 857, 867, 870, 923, 1054
- TextBox, 205, 228, 841
- TextWrapping, 262
- Thumb, 537
- UniformGrid, 503, 509
- VariableSizedWrapGrid, 151
- Viewbox, 75, 420
- element-korzeń, 198
- element-właściwość, 52, 57, 203
- element-właściwość Triggers, 379, 380
- elementy zegara, 805
- elipsa, 122
- etykieta gridLinesGrid, 322
- etykieta innerGrid, 322

F

- folder
 - Common, 600
 - Fonts, 858
 - Themes, 545
- format
 - EPUB, 866
 - RTF, 885
 - WMA, 38
 - XML, 34
 - XPS, 866

- formatowanie tekstu, 59
- formaty
 - dokumentów, 866
 - plików graficznych, 29, 717
- framework
 - .NET, 255
 - Prism, 235
 - WPF, 402, 502
- funkcja
 - arcus sinus, 995
 - DWriteCreateFactory, 823
 - Ease, 370
 - ElasticEase, 407
 - EnumDynamicTimeZoneInformation, 799, 803
 - ExponentialEase, 370
 - GetNativeSystemInfo, 793
 - GetTimeZoneInformationForYear, 800
 - IntelliSense, 25
 - Location, 685
 - Math.Asin, 655
 - SineEase, 377
 - SystemTimeToTzSpecificLocalTime, 800
- funkcje
 - lambda, 250
 - łągodzące, 369–372
 - specjalne, 627
 - Win32, 792
 - związane z piórem, 1019
- futura, future, 245

G

- GDI, Graphics Device Interface, 101
- generowanie
 - obrazu, 744
 - stron, 942
- głębia
 - kolorów, 727, 734
 - tekstu, 409
- GPS, 959
- gradient, 49, 442
- grafika, 43
- grafika wektorowa, 65
- grawitacja, 964
- grubość linii, 759
- grupowanie elementów, 597, 620

H

HTML, HyperText Markup Language, 11

I

ID zdarzenia, 658

identyfikator

GUID, 262

zasobu, 65

ignorowanie błędów, 755

IL, Intermediate Language, 833

iloczyn trzech transformacji, 463

implementacja

linii przerywanej, 529

menu kontekstowych, 283

WriteFontCollection, 824

implementacje interfejsu ICommand, 238

inercja, 676

inicjowanie

manipulacji, 690

rozszerzeń nazw plików, 719

rysowania, 1037

urządzenia MediaCapture, 787

usuwania, 1037

właściwości, 41

zaznaczania, 1037

IntelliSense, 25, 33, 43

interakcja z użytkownikiem, 161

interfejs

API, 23

AsyncOperation, 245

DirectWrite, 821

DirectWrite IDWriteFontCollection, 823

IAsyncInfo, 254

IAsyncOperation, 248

ICommand, 234, 236, 238

ID2D1BitmapRenderTarget, 846

IDelegateCommand, 236

IDictionary<TKey, TValue>, 490

IDWriteFontFamily, 825

IEnumerable, 490

IFormatProvider, 130

IFormattable, 130

IList<T>, 490

IMap<K, V>, 490

INotifyCollectionChanged, 490, 604

INotifyPropertyChanged, 217, 223, 329,
471, 490, 602

IRandomAccessStream, 697, 708

ISurfaceImageSourceNative, 821

ITextCharacterFormat, 891

ITextDocument, 890

ITextProvider, 898

IUICommand, 245

IValueConverter, 130, 177

IValueProvider, 898

IVector<T>, 490

kamery, 784

użytkownika, 10, 243

Windows Runtime, 11, 13

WinRT, 11

J

język

IL, 833

JavaScript, 11

pośredni, 833

znaczników, 11

K

kafelki, 1011

kafelki map Bing, 1004

kalendarz, 935

kamera, 784

kartezjański układ współrzędnych, 67

catalog

Fonts, 856

WinMetadata, 11

kąt obrotu, 851, 988

kąty Eulera, 986

klasa

Accelerator, 974, 979

Accelerometer, 972, 974

AccelerometerReading, 965

AccelerometerReadingChangedEventArgs,
965

App, 33, 584, 590, 625

AppBar, 163, 290

Application, 33, 61, 268

ApplicationData, 256

ApplicationDataContainer, 586

ApplicationView, 565, 569

AppSettings, 330–345

AutomationProperties, 294

BezierSegment, 72

BindableBase, 223, 224

Binding, 85, 129, 223

BitmapDecoder, 717, 723
 BitmapEncoder, 718, 725
 BitmapFrame, 722
 BitmapImage, 44
 BitmapPrintDocument, 956
 BitmapSource, 44
 Block, 59, 210, 868
 Border, 119
 Brush, 49, 440
 Buffer, 708
 Button, 163, 183, 189, 466, 524, 528
 ButtonBase, 181, 184
 Calendar, 938
 CameraCaptureUI, 784
 Canvas, 117, 153, 157
 CheckBox, 183
 Clipboard, 907
 Clock, 475
 CollectionViewSource, 620, 623
 ColorAnimationUsingKeyFrames, 388
 ColorItem, 147, 148
 Colors, 40, 133
 Compass, 979
 CompositeTransform, 436
 ContentControl, 163, 184, 466, 550
 ContentPresenter, 525, 550
 ContentPropertyAttribute, 56
 Control, 30, 161–163, 855
 ControlTemplate, 183, 294, 349, 465
 CoreDispatcher, 250, 957
 CoreWindow, 895
 CultureInfo, 130
 CustomPageRange, 948
 DataReader, 260, 265
 DataTemplate, 465, 469
 DataWriter, 261, 265
 DelegateCommand, 235–237
 DependencyObject, 32, 42, 135, 179, 247, 350
 DependencyProperty, 32, 190
 DependencyPropertyChangedEventArgs, 193
 Dial, 692
 DirectWrite, 336, 841
 DirectXWrapper, 821
 DispatcherTimer, 108, 109
 DisplayProperties, 32, 106, 557, 960, 961
 DockPanel, 180
 DoubleAnimation, 351, 354, 359, 366
 DoubleToStringHexByteConverter, 300
 EasingFunctionBase, 356, 369, 373
 Ellipse, 122
 EllipseGeometry, 380
 ElPasoHighSchool.StudentBody, 607
 FileOpenPicker, 256, 258, 718
 FileSavePicker, 256, 258, 718
 FolderPicker, 256
 FontFamily, 855
 FontWeights, 825
 FormattedStringConverter, 176
 Frame, 573, 576
 FrameworkElement, 38, 61, 78, 81, 104, 120, 125, 161, 183, 188, 525
 FrameworkElementAutomationPeer, 205
 FrameworkTemplate, 465
 Geolocator, 684, 1004
 Geometry, 71, 438
 GradientButton, 191–197
 GradientStop, 113
 Grid, 167
 HSL, 762
 HttpClient, 266
 IAsyncOperation, 246
 Image, 28, 70
 ImageBrush, 70
 ImageSource, 44
 Inclinator, 983
 InertiaTranslationBehavior, 677
 InkDrawingAttributes, 1023
 InkFileManager, 1044, 1060
 InkManager, 1029, 1035, 1044
klasa Inline, 59
klasa InlineUIContainer, 60
klasa InMemoryRandomAccessStream, 708
 ItemsControl, 163, 480, 497, 550
 ItemsPanelTemplate, 465
 Key, 663
 LayoutAwarePage, 596
 LinearGradientBrush, 50, 61
 List, 665
 LoggerControl, 636
 MainPage, 22, 812
 ManipulableContentControl, 776
 ManipulationDeltaRoutedEventArgs, 676
 ManipulationInertiaStartingRoutedEventArgs
 ↳ Args, 677
 ManipulationStartingRoutedEventArgs, 690
 Matrix3D, 447

klasa

Matrix3DHelper, 447
Matrix3DProjection, 400
MatrixTransform, 434
MediaElement, 38
MessageDialog, 243–246, 253, 256, 263, 284
NamedColor, 485, 489
ObjectAnimationUsingKeyFrames, 390
ObservableCollection<T>, 491
OrientationSensor, 986
Package.appxmanifest, 685
Page, 555, 573
Page., 117
Panel, 25, 117, 123, 138, 500–502
Paragraph, 59, 868
PassData, 592
Path, 70, 75
PathFigure, 71
PathGeometry, 71
PathIO, 266
PathSegment, 364
PieSlice, 382, 385
PlaneProjection, 447, 450
Point, 68
PointAnimationUsingKeyFrames, 388
Pointer, 632
PointerPointProperties, 1020
PointerRoutedEventArgs, 631
PointKeyFrame, 388
Polyline, 67, 70
Popup, 287
PrintDocument, 917, 955
PrintManager, 930
PrintTaskOptionChangedEventArgs, 932
PrintTaskOptionDetails, 951
PrintTaskOptions, 919
ProgressBar, 163
Projection, 399
PropertyChangedEventArgs, 217
RadialGradientBrushSimulator, 710
RadioButton, 183, 204
RandomAccessStreamReference, 266
RangeBase, 163
RangeBaseValueChangedEventArgs, 165
Rectangle, 121
ResourceDictionary, 65
RgbViewModel, 220, 222, 228
RichTextBlock, 142, 871, 878
RichTextBlockOverflow, 871
RichTextColumns, 875
RotateTransform, 411
RoutedEventArgs, 165
RudimentaryTextBox, 896
Run, 59
SaveStatePage, 586
ScaleTransform, 417, 429
ScrollBar, 164
ScrollViewer, 163, 499
SecondPage, 574, 581
Selector, 480, 610
SelectorItem, 550
SensorQuaternion, 986
SensorRotationMatrix, 986, 991
Setter, 79
SettingsDialog, 343, 816
SettingsPane, 904
Shape, 66, 121
SimpleOrientationSensor, 960–965
SizeChangedEventArgs, 104
SkewTransform, 425
Slider, 162, 164, 167
Span, 59
Spiral, 67
SplitContainer, 315, 321, 329
StackPanel, 108, 123
StorageFile, 258
StorageFolder, 256
Storyboard, 351
String, 723, 824
StructLayoutAttribute, 793
Student, 602
StudentBody, 603, 605
StudentBodyPresenter, 605, 620, 623
StudentGroups, 622, 623
Style, 78
SurfacelImageSource, 821, 842
SurfacelImageSourceRenderer, 841, 842, 852
SuspensionManager, 596
TabbableTextBox, 327, 329
TappedRoutedEventArgs, 92, 96
Task, 264, 272
TextBlock, 25, 60, 855
ThicknessSettingDialog, 757
Timeline, 357, 368, 392
TimeZoneInfo, 798, 800
TimeZoneManager, 801, 803
ToggleButton, 183
TouchInfo, 663
Transform, 366, 424
TransformGroup, 411, 435

- Transition, 394
- TranslateTransform, 407, 410
- TwelveHourClock, 477
- TypeInfo, 134
- Typography, 866
- UIElement, 32, 87, 161, 629, 895
- UIElementCollection, 55
- UISettings, 65
- UniformGrid, 503, 508
- UserControl, 45, 56, 93, 145, 163
- Viewbox, 75, 76
- ViewModel, 227, 489
- ViewParams, 1007
- VisualStateManager, 528
- WebViewBrush, 50
- WriteableBitmap, 697, 706
- WriteableBitmapEx, 698
- WriteFactory, 823
- XmlSerializer, 605
- YellowPadPage, 1048, 1053
- klasy FileIO, 266
- klatka kluczowa, 386, 387
- klawiatura
 - dotykowa, 205, 896
 - ekranowa, 900
 - fizyczna, 896
 - pianina, 663
- klawisz Backspace, 896
- klikanie elementu, 611
- klucz elementu w słowniku, 62
- kod
 - znaczników, 11, 12
 - źródłowy, 13, 833
- kodowanie
 - ASCII, 299
 - Unicode, 34, 298, 894
 - UTF-16, 299
 - UTF-32, 299
 - UTF-8, 299, 894
- kolejność atrybutów, 25
- kolekcja, 602
 - DisplayedOptions, 932, 951
 - DisplayInformation, 803
 - FileTypeFilter, 257
 - InkManager, 1020
 - InkStrokeRenderingSegment, 1025
 - Inlines, 60, 138
 - ItemsSource, 1053
 - List<T>, 490
 - map, 490
 - MergedDictionaries, 64
 - obiektów PointerPoint, 632
 - obiektów typu T, 490
 - ObservableCollection, 603
 - RowDefinitions, 168, 175
 - unikatowych kluczy, 490
 - vector, 490
- kolor, 712, 760
 - HSL, 759
 - standardowy, 535
 - tła, 25, 100, 111, 818
- koło, 736
- kompas, 981
- kompilacja, 833
- kompilat, assembly, 24
- komunikat o błędzie, 337
- konfiguracja
 - Debug, 833
 - Release, 833
- konsolidowanie obliczeń, 273
- konstruktor, 651
 - MainPage, 812
 - WriteableBitmap, 700
- kontenery elementów, 550
- kontrolka, 161
 - Button, 161, 233, 291, 466, 519, 536
 - CheckBox, 304, 672
 - ComboBox, 163, 1056
 - Dial, 692, 694
 - FlipView, 515, 600, 878, 1049
 - HslColorSelector, 765
 - idCustomRangeEdit, 953
 - ItemsControl, 480, 482, 484
 - JiggleButton, 407
 - ListBox, 163, 479, 493, 496–552
 - ListBoxItem, 551
 - LoggerControl, 642
 - ManipulableContentControl, 775
 - MonthYearSelect, 940
 - NewToggle, 545, 548
 - Popup, 347
 - ProgressBar, 541, 544
 - RadioButton, 184, 189, 198–202, 296, 306, 892
 - RangeBase, 163
 - RichEditBox, 855, 886
 - RichTextBox, 205
 - RudimentaryTextBox, 900

kontrolka

- RulerContainer, 322
- ScrollBar, 164
- ScrollView, 550
- Slider, 161–167, 173, 179, 221, 291, 346, 537, 678, 884
- SplitContainer, 348
- StackPanel, 944, 947
- TabbableTextBox, 336, 337
- TextBlock, 185, 187, 286, 480, 495, 520
- TextBox, 205, 228, 284, 291, 327
- Thumb, 208, 460, 537
- ToggleButton, 262
- ToggleSwitch, 183, 346
- TreeView, 777
- UserControl, 290, 405, 902
- XYSlider, 680, 686, 762
- YellowPadPage, 1049

kontrolki

- pól wyboru, 304
- własne, 544
- zegarów, 812

kontrolowanie rozmiaru inercji, 675

konwersja

- danych przyspieszeniomierza, 976
- macierzy obrotu, 990
- na czas lokalny, 804
- na Win32 SYSTEMTIME, 804
- obiekту IBuffer, 266
- obrazu, 727
- RGB na HSL, 769
- stylu czcionki, 827
- SYSTEMTIME na DateTime, 805
- wagi czcionki, 826
- współrzędnych, 995

konwerter, 178

konwerter sformatowanego tekstu, 176

konwertowanie linii, 1027

kończenie programu, 26

kroje czcionek, 855

krzywa

- Béziara, 73, 363, 1021, 1027, 1034
- typu Cubic Bézier, 364

książka Programowanie Windows, 14

kształt

- połączenia linii, 199
- zakończeń linii, 199

kulka, 974

kwadrat jednostkowy, 461

L

liczba

- drukowanych stron, 916, 955
- elementów RichTextBlockOverflow, 875
- wierszy i kolumn, 168

linia

- bazowa, 861
- skanowania, 736, 743
- z zaokrąglonymi końcami, 741

lista

- ComboBox, 888
- czcionek, 832
- drukarek, 915
- elementów TextBlock, 927
- kolorów, 510
- ListBox, 498, 550
- Solution Configurations, 832
- Solution Platforms, 833, 835
- stylów, 294

logiczne DPI, 556

lokalizacja, 288

lokalizacja pliku, 857

lokalny magazyn, 864

lokalny magazyn aplikacji, 256

losowa zmiana koloru, 99

Ł

ładowanie

- atramentu, 1047
- dokumentu, 889
- ustawień, 1047, 1055

łuk, 736

M

macierz

- jednostkowa, 986
- Matrix3D, 454
- Matrix3DProjection, 456
- obrotu, 986, 990

manipulowanie tekstem, 30

mapy, 1004

mapy bitowe, 561, 660, 697

marginesy

- drukowalne i niedrukowalne, 918
- strony, 931

maska przezroczystości, 705

maskowanie pikseli źródłowych, 733

menedżer stanu wizualizacji, 528
 menu kontekstowe, 283, 644
 metadane, 1007
 metoda

- Accelerometer.GetDefault, 964
- AccumulateDelta, 689
- AddHandler, 98
- AddPages, 928
- Application.LoadComponent, 42
- Arrange, 501
- ArrangeOverride, 500, 507
- AsAsyncAction, 265
- Begin, 352
- BitmapDecoder.CreateAsync, 721
- BitmapDecoder.GetDecoderInformation
 - ↳ Enumerator, 720
- BitmapEncoder.CreateAsync, 725
- BitmapEncoder.GetEncoderInformation
 - ↳ Enumerator, 723
- CalculateImageScaleAndOffset, 771
- CalculateMagicNumberAsync, 273, 274, 276
- Cancel, 275
- CanExecute, 235
- CanExecuteDeleteCharacter, 239
- Canvas.SetLeft, 156
- Canvas.SetTop, 156
- CaptureFileAsync, 785
- CapturePhotoToStorageFileAsync, 787
- CapturePhotoToStreamAsync, 787
- CapturePointer, 642
- Clear, 845
- Click, 784
- CommandsRequested, 904
- Complete, 676
- CompositionTarget.Rendering, 853
- Convert, 130
- CreatePrintTask, 920
- CreateTextOption, 951
- DependencyProperty.Register, 190
- DependencyProperty.RegisterAttached, 190
- DeviceInformation.FindAllAsync, 786
- DisplayAndPrinterPrep, 923
- Ease, 369
- Enum.TryParse, 199
- Execute, 235
- ExecuteAddCharacter, 239
- FileIO.ReadLinesAsync, 265
- FileIO.ReadTextAsync, 265, 271
- FormatText, 188
- get, 192
- GetAllIX, 737
- GetBitmapStream, 908
- GetContent, 907
- GetCurrentOrientation, 960
- GetCurrentPoint, 632, 647
- GetCurrentReading, 965
- GetDefault, 965
- GetForCurrentView, 912
- GetIntermediatePoints, 632
- GetKeyState, 590
- GetLeft, 157
- GetLongitudeAndLatitude, 1012
- GetMetrics, 830
- GetNavigationState, 584
- GetPageDescription, 919
- GetPatternCore, 898
- GetPixelDataAsync, 722
- GetPositionFromIndex, 328
- GetPreviewPage, 928
- GetRenderingSegments, 1025
- GetResults, 246, 254
- GetSystemFontCollection, 822
- GetTemplateChild, 536
- GetTop, 157
- GetValue, 43, 134
- GetWordFrequenciesAsync, 276
- GetWordFrequenciesAsync, 277, 282
- GoBack, 589, 594
- InitializeComponent, 23, 42, 64, 104, 197, 241, 545
- InvalidateArrange, 504
- InvalidatePreview, 934
- LoadAsync, 260
- LoadBitmapAsync, 779
- Loaded, 702, 729, 779, 787, 849, 864
- LoadFile, 263
- LoadFileAsync, 264
- LoadFileFromOpenPicker, 341
- LoadFromStream, 890
- ManipulationStarted, 782
- Math.Atan2, 737
- Math.Ceiling, 928
- Measure, 327, 501, 506, 878
- MeasureOverride, 500–506, 512, 875
- Navigate, 579
- NextBytes, 89
- OnApplyTemplate, 547
- OnColorChanged, 193
- OnCreateAutomationPeer, 205, 898

metoda

OnDragThumbDelta, 321
OnInterpolationFactorChanged, 1003
OnKeyDown, 644
OnLaunched, 45, 572, 585, 588
OnLoaded, 336
OnManipulationDelta, 673, 687
OnMenuDelete, 646
OnNavigatedFrom, 23, 583, 913
OnNavigatedTo, 23, 41, 89, 102, 574, 581, 593, 913
OnNavigatingFrom, 23, 576
OnPointerCaptureLost, 644
OnPointerEntered, 665
OnPointerMoved, 745, 771, 1038
OnPointerPressed, 634, 745, 850, 1032, 1037
OnPointerReleased, 660, 1039
OnPointerReleased, 658
OnPrintTaskSourceRequested, 949
OnPropertyChanged, 220, 476
OnSaveAsAppBarButtonClick, 723
OnTapped, 95, 154
OnTextBlockTapped, 96
OnThumbDragStarted, 321
OrderByDescending, 277
Paginate, 917, 926
PathIO.WriteTextAsync, 271
PickSingleFileAsync, 260, 720
PointerCaptures, 642
PointerMoved, 774
ProcessNewOrientationReading, 1001
ProcessPointerDown, 1022
ProcessPointerUp, 1022
ProcessPointerUpdate, 1022, 1033
ReadAsync, 267
ReadBufferAsync, 266
ReadingChanged, 965
ReadLineAsync, 278
RedrawRuler, 326
RefreshBitmap, 712, 715
RefreshDisplay, 1015
Register, 190
Remove, 583
Render, 697
RenderBeziers, 1041
RenderOnBitmap, 741, 774
RenderStroke, 1040
Rotate, 655
RunAsync, 248, 272
SaveAsync, 1046
SaveToStream, 890, 894
Seek, 701
SelectionChanged, 781, 884
set, 192
SetBinding, 161
SetBitmap, 908
SetBubble, 973
SetContent, 907
SetDefaultXamlFile, 337
SetLeft, 157
SetNavigationState, 584
SetPixelData, 725
SetPreviewPage, 917
SetPreviewPageCount, 955
SetProperty, 224
SetSource, 697, 706, 708
SetValue, 42, 156, 192
Show, 904
ShowAsync, 245, 254, 285, 645
ShowForSelectionAsync, 286
SimpleOrientationSensor.GetDefault, 960
SizeChanged, 596, 881–884
StoreAsync, 261
Suspending, 890
Task.Run, 272
Task.WaitAll, 1009
Task.Yield, 281
ThrowIfCancellationRequested, 274
ToString, 131, 467
TransformBounds, 446
TransformToVisual, 313, 444, 446
TryParse, 233
TryY, 739
UIElementColor, 65
UpdateBitmap, 729–733
UpdateImageTransforms, 1002
ValueChanged, 884
VisualStateManager.GoToState, 528, 532, 596
WriteAsync, 700
XamlReader, 314
XamlReader.Load, 74, 337

metody

asynchroniczne, 244, 272
dostępowe, 192
inicjujące właściwości, 192
obsługi zdarzeń, 762, 969
przesłaniające, 899, 994
przesłaniane, 653
SetTop, 157

- statyczne, 652, 993
- wirtualne, 93, 669, 895
- metryka czcionki, 835, 862
- Microsoft Expression Blend, 24
- miniatura, 780
- mnożenie macierzy, 430
- model, 215
 - HSL, 760
 - MVVM, 215
 - RGB, 760
- modele widoku, 602, 625
- moduł
 - przyspieszenia, 973
 - wektora, 966, 967
- modyfikator
 - ref, 822
 - sealed, 150
- modyfikowanie warstwy ViewModel, 228
- MVVM, Model-View-ViewModel, 83, 215

N

- nachylenie, 995
- nadir, 992
- NaN, not a number, 104, 655, 677
- narzędzie, *Patrz*: program
- natywna orientacja pionowa, 966
- nawigacja, 555, 579
- nawigacja pomiędzy stronami, 572
- nazwy
 - kolorów, 33
 - plików bibliotek DLL, 797
 - rodzin czcionek, 856
- normalizacja
 - czasu animacji, 370
 - długości, 714
- notatnik, 308, 1043
- numerowanie quadkey, 1011

O

- obiekt
 - Accelerometer, 964–971
 - AppBar, 292
 - ApplicationDataContainer, 256, 262
 - AppSettings, 759
 - Assembly, 136
 - Binding, 85
 - BitmapCodeclInformation, 723
 - BitmapEncoder, 725

- BitmapImage, 999
- BitmapRenderTarget, 844
- Block, 213
- Buffer, 708
- Button, 211, 234
- Calendar, 938
- Canvas, 999
- contentGrid, 1030
- ContentPresenter, 525, 527
- ControlTemplate, 520
- CoreDispatcher, 960
- CustomPageRange, 949
- DataContext, 817
- DataPackage, 907
- DataPackageView, 907
- DataReader, 266
- DataTemplate, 475, 513
- DataTransferManager, 911
- DataWriter, 266
- DataWriteStoreOperation, 261
- DependencyProperty, 190, 191
- DependencyPropertyChangedEventArgs, 193, 319
- Dictionary, 276, 800
- DiscreteObjectKeyFrame, 391
- DispatcherTimer, 272
- DoubleAnimation, 369, 415
- DoubleAnimationUsingKeyFrames, 453
- FileOpenPicker, 719
- fileStream, 706, 708
- FlipView, 1058
- FolderItem, 778
- GeneralTransform, 444
- Geolocator, 1009
- Geometry, 71
- GradientStop, 113, 469
- Grid, 25, 778
- IAsyncOperation, 245, 248, 251
- IBuffer, 266, 267, 700
- ID2D1BitmapRenderTarget, 845
- IDWriteFactory, 822
- ImageBrush, 703
- Inclinometer, 989, 1009
- InkDrawingAttributes, 1024, 1059
- InkFileManager, 1053
- InkManager, 1020–1027, 1032, 1042
- InkStroke, 1020, 1025, 1028
- InkStrokeRenderingSegment, 1025
- IRandomAccessStream, 782
- ITextCharacterFormat, 891

obiekt

- ITextParagraphFormat, 892
- IUICommand, 251
- KeySpline, 390
- Line, 651
- LinearGradientBrush, 54, 114, 192, 389
- LoggerControl, 639
- MainPage, 334, 577
- ManipulationManager, 689
- Matrix3D, 991, 1001
- MatrixTransform, 776
- MessageDialog, 250
- NamedColor, 489
- Paragraph, 881
- Path, 363, 654
- PathFigure, 72
- PathGeometry, 655
- PathSegment, 72
- pixelStream, 708
- Point, 694
- PointerInfo, 649
- PointerPoint, 632
- PointerPointProperties, 1022
- PointerRoutedEventArgs, 632
- Polyline, 66, 634–648, 663, 1037
- Popup, 287, 307, 903
- PopupMenu, 307
- PrintCustomItemOptionDetails, 951
- PrintDocument, 914, 918
- PrintTask, 915, 930
- PrintTaskOptionDetails, 932
- RadialGradientBrushSimulator, 715
- Random, 89
- RandomAccessStreamReference, 908, 911
- Rectangle, 513
- RectangleGeometry, 439
- RenderTransform, 515
- ResourceDictionary, 61
- RgbViewModel, 222
- RichTextBlockOverflow, 871
- RotateTransform, 400, 475
- Run, 106
- ScrollViewer, 512
- SettingsPane, 904
- SimpleOrientationSensor, 959, 963, 968
- SolidColorBrush, 49, 52, 65, 112, 222, 1025
- SplitContainer, 327
- StorageFile, 256, 259, 720, 724, 748, 785, 894
- StorageFolder, 270, 779

- Storyboard, 351
- Stream, 266, 701
- Student, 609
- Style, 80, 295, 550
- SurfacelImageSource, 820, 849
- SystemInfo, 794
- TextBlock, 26, 47
- Thumb, 685
- TimeZoneClock, 815
- TimeZonelInfo, 798
- UIElementCollection, 55
- UniformGrid, 510, 512
- VirtualizingStackPanel, 282
- WrapOptionsDialog, 311, 312
- WritableBitmap, 660
- WriteableBitmap, 698, 708, 729, 734
- WriteFontCollection, 822

obiekty

- animacji, 351
- DirectX, 845
- graficzne, 770

obliczanie

- kąta obrotu, 852
- macierzy Matrix3D, 454
- rozmiaru tablicy, 700
- wartości animacji, 370
- wysokości elementu, 139
- wysokości wiersza, 169

obraz

- Hieronima Boscha, 998
- WriteableBitmap, 698

obrót, 400, 447, 571, 675, 727, 850, 981, 987

obrót jednym palcem, 690

obserwator automatyzacji, 898

obsługa

- animacji, 110
- anulowania metody, 274
- danych wskaźnika, 629
- dotyku, 183
- inercji, 676
- klawisza Backspace, 896, 898
- klawiszy, 207
- kontrolki ComboBox, 892
- menu wyskakującego, 645
- przyspieszeniomierza, 969
- schowka, 906
- wejścia, 162
- właściwości Color, 769
- wyjątków, 262

- zdarzenia, 87
 - CharacterReceived, 895
 - Click, 213, 279, 358
 - CompositionTarget, 851
 - Loaded, 270, 295, 378
 - ManipulationDelta, 674, 678
 - PrintManager, 915
 - PrintTaskRequested, 914
 - Tapped, 88, 92
 - Tick, 109
 - ValueChanged, 179, 375
- zdarzeń pasków, 768
- zdarzeń routowanych, 90
- zmian właściwości, 319, 323
- odchylenie, 983, 987
- odczyt
 - metadanych, 1009
 - pikseli, 721
 - siły nacisku, 654
 - środka zegara, 816
- odległość
 - dwóch punktów, 714
 - punktu od linii, 714
- odłączanie PathGeometry, 75
- odmierzenie czasu, 108
- ograniczenia bibliotek, 150
 - Windows Runtime Component, 820
- okna
 - dialogowe, 256, 287
 - wyskakujące, 902
- okno
 - Add Reference, 831
 - Additional Dependencies, 820
 - Configuration Manager, 834
 - DataContext, 759
 - dialogowe ustawień, 343
 - FileOpenPicker, 257, 263
 - Informacje, 902
 - MessageDialog, 245, 255, 341
 - Popup, 289, 308, 313, 816, 902
- określanie współrzędnej maksymalnej, 773
- opcja
 - Any CPU, 834, 835
 - Build Action, 857
 - Create App Packages, 835
 - Orientation, 931
 - Win32, 834
 - x86, 834
- opcje
 - dla projektów C#, 834
 - dla projektów C++, 834

- operacje asynchroniczne
 - anulowanie, 253
 - błędy, 253
 - natywne, 791
 - postęp, 253
 - wejścia-wyjścia, 255, 265, 314, 339
- operator, 653, 994
 - __uuidof, 823
 - await, 251, 260
 - OR, 670
- opis strony, 935
- opóźnienie, 677
- orientacja, 173, 959
 - mapy, 1017
 - pozioma, 957
- osobliwość, 138
- oś obrotu, 988
- otaczanie obrazu tekstem, 870

P

- P/Invoke, 792, 833
- pakiet SDK, 9
- panel, 496
 - StackPanel, 343, 513
 - Udostępnianie, 906, 911
 - UniformGrid, 513
 - Urządzenia, 912
 - VariableSizedWrapGrid, 151
 - wrapPanel, 152
- panele własne, 500
- parser XAML, 41, 337
- pasek
 - aplikacji Notatnika, 308
 - AppBar, 292
 - narzędzi, 832
- paski aplikacji, 290, 293, 304
- PDF, Portable Document Format, 862
- pędzel
 - ButtonPressedBackgroundThemeBrush, 536
 - ButtonPressedForegroundThemeBrush, 536
 - gradientu, gradient brush, 49
 - LinearGradientBrush, 70, 79, 189, 361, 441
 - tła, 62
 - z okrągłym gradientem, 709
- pianino, 663
- piksel, 698
- piksele niezależne, 560

pióro, 630, 1019
 pióro elektromagnetyczne, 1020
 plik

- App.xaml, 33, 64, 945
- App.xaml.cs, 33, 584, 586
- AppSettings.cs, 745
- ArcSegment.cs, 738
- BitmapPrintDocument.cs, 956–958
- Block.xaml.cs, 210
- Class1.cs, 148
- CustomPrintRange.cs, 948
- DistributedUniformGrid.cs, 811
- generic.xaml, 535, 544, 551
- GradientButton.cs, 193
- HslColorSelector.xaml, 765
- IndexToPageNumberConverter.cs, 1054
- Kooten.ttf, 858
- LoggerControl.xaml.cs, 636
- MainPage.g.i.cs, 42
- MainPage.Pointer.cs, 743
- MainPage.Share.xaml, 907–911
- MainPage.xaml, 21, 23, 149
- MainPage.xaml.cs, 21, 42, 75
- ManipulableContentControl.xaml.cs, 777
- ManipulationManager.cs, 775
- Miramo.ttf, 859
- MonthYearSelect.xaml.cs, 938
- NamedColor.cs, 484
- Octave.xaml, 665
- Package.appmanifest, 572
- Package.appx, 963
- Package.appxmanifest, 257, 685, 784, 902
- Peric.ttf, 859
- RadialGradientBrushSimulator.cs, 712
- RgbViewModel.cs, 218, 224
- RoundCappedLine.cs, 739
- RoundCappedPath.cs, 740
- RudimentaryTextBox.xaml.cs, 897
- SettingsDialog.xaml, 343, 817
- SettingsDialog.xaml.cs, 819
- StandardStyles.xaml, 64, 78, 83, 293, 597–599, 886
- StudentPage.xaml, 618
- students.xml, 602
- SurfaceImageSourceRenderer.cpp, 843–847
- SurfaceImageSourceRenderer.h, 842
- themeresources.xaml, 535
- ThicknessSettingDialog.xaml, 757
- TimeZoneClock.xaml, 806
- TimeZoneClockViewModel.cs, 808
- TimeZoneManager.cs, 801–804
- TimeZoneManager.Display.cs, 800
- UniformGrid.cs, 504
- windows.ui.aml.media.dxinterop.h, 821
- winnt.h, 796
- WriteFactory.h, 822
- WriteFactory.cpp, 822
- WriteFont.cpp, 830
- WriteFont.h, 829
- WriteFontCollection.cpp, 824
- WriteFontCollection.h, 822, 823
- WriteFontFamily.cpp, 826
- WriteFontFamily.h, 825
- WriteFontMetrics.h, 831
- WriteLocalizedStrings.cpp, 828
- WriteLocalizedStrings.h, 827
- wrl.h, 821
- XYSlider.cs, 762
- YellowPadPage.xaml.cs, 1049, 1050

 pliki

- .cs, 22
- .psl, 835
- .rfl, 885, 890, 893
- .ttf, 855
- .txt, 893
- .winmd, 11
- C#, 24
- czcionek, 862
- czcionek prywatnych, 857
- FixedPage, 863
- map bitowych, 717
- XAML, 24, 625
- XPS, 863

 płynne zwięzanie linii, 659
 płynny układ strony, 866
 pobieranie

- bieżącego obrotu, 1009
- bieżącego przyspieszenia, 978
- bieżącej lokalizacji, 1009
- czasu, 814, 977
- danych, 643, 1051
- danych kodera, 731, 753
- DXGIDevice, 844
- kafelków mapy, 1007
- lokalizacji obiektu, 756
- miniatury, 782
- obrotu z Tilt, 1002
- pędzla, 232
- pikseli, 783
- pikseli źródłowych, 730

- pobieranie
 - pliku XML, 605
 - ramki, 730, 783
 - referencji strumienia, 911
 - wartości koloru, 249
 - wybranego StorageFile, 753
- pochylenie, skew, 424, 982, 987
- pociągnięcia, 1035
- podgląd
 - wideo, 786, 787
 - wydruku, 914, 917
- podkategorie kontrolki, 163
- pola wyboru, 304
- pole Bias, 799
- pole
 - ContentType, 724
 - DaylightDate, 799
 - dwOemId, 793
 - hasPen, 1021
 - LastPosition, 850
 - Padding, 925
 - Solution Platforms, 833
 - sqrtNumTiles, 1014
 - StandardDate, 799
 - StandardName, 799
 - ThisPosition, 853
 - wProcessorArchitecture, 796
- polecenie Stop Debugging, 26
- położenie okna Popup, 816
- powiadomienia wiązań danych, 216
- powiązania dwukierunkowe, 231, 522
- powiązanie
 - RelativeSource, 522
 - TemplateBinding, 521–527
- pozycja elementu, 156
- północ geograficzna, 981
- prefiks
 - local, 24, 195
 - ms-appdata, 865
 - ms-appx://, 858
 - this, 43
 - x, 24, 86
 - On, 93
- prezentacja, 117
- prezentowanie długiego tekstu, 871
- prędkość, 677
- procedura obsługi zdarzenia, 88
- program
 - Microsoft Expression Blend, 12, 146
 - Microsoft Visual Studio Express 2012, 9
 - Notatnik, 885
 - WHATSIZE, 101
 - WordPad, 885
 - WPF, 863
 - XAML Cruncher, 12, 314, 332, 380
 - XAML Designer, 12
- programowanie asynchroniczne, 244
- projekcja, 447
- projekt
 - AccelerometerAndSimpleOrientation, 967, 968
 - AllColorsItemsControl, 509
 - AlphabetBlocks, 209–211
 - AnalogClock, 422, 424
 - AnimateDashOffset, 363
 - AnimatedPieSlice, 383–385
 - AnimateStrokeThickness, 361
 - AnimationEaseGrapher, 371–376
 - AppBarPad, 308–312
 - ApplicationStateSave, 584–589
 - AttachedPropertyAnimation, 366, 367
 - AutoImageSelection, 562
 - AxisAngleRotation, 987–990
 - BareBonesSlider, 537
 - BaselineTiltedShadow, 840, 841
 - BerniceBobsHerHair, 876, 877
 - BetterBorderedText, 119
 - BetterCharacterInput, 896–898
 - BubbleLevel, 972
 - ButtonVarieties, 182
 - CenteredTransforms, 686
 - CheshireCat, 364
 - CircleAnimation, 377
 - CircularGradient, 702, 703
 - ClockButton, 472, 474
 - ClockRack, 800–819
 - ColorItems, 483, 484
 - ColorItemsSource, 486, 488
 - ColorItemsSourceWithBinding, 489
 - ColorList, 143, 144, 146, 149
 - ColorScrollWithDataContext, 224, 227
 - ColorScrollWithValueConverter, 177
 - ColorScrollWithViewModel, 218, 220, 222
 - ColorTextBoxes, 228, 229
 - ColorTextBoxesWithEvents, 231
 - ColorWrap, 152
 - CommonMatrixTransforms, 434
 - ConditionalClockButton, 476–478
 - CustomButtonTemplate, 532, 535
 - CustomGradient, 699

projekt

- CustomizableClassHierarchy, 931, 935
- CustomListBoxItemStyle, 551
- DataPassingAndReturning, 591–596
- DependencyObjectClassHierarchy, 135
- DependencyProperties, 193, 195
- DependencyPropertiesWithBindings, 197, 198
- DepthText, 409
- DialSketch, 692, 693, 694
- DigitalClock, 108
- DirectXWrapper, 821–831, 842–847
- DisplayHighSchoolStudents, 612–615, 618
- EarthlyDelights, 992, 995, 999–1004
- EasyCameraCapture, 784
- EllipseBlobAnimation, 360, 361
- ElPasoHighSchool, 602, 605
- EnumerateFonts, 831
- ExpandingText, 110
- FastNotFluid, 390
- FlickAndBounce, 678
- FlipViewColors, 516
- ForeverColorAnimation, 379
- GettingCharacterInput, 895
- GradientBrushCode, 50
- GradientBrushPointAnimation, 389
- GroupBySex, 620–623
- HarderCameraCapture, 786, 787
- HelloAudio, 38
- HelloCode, 39, 40
- HelloImage, 27
- HelloImageCode, 44
- HelloLocalImage, 29
- HelloLocalImageCode, 45
- HelloPrinter, 913–918
- HelloVectorGraphics, 71
- HelloVideo, 39
- HorizontalListBox, 498
- HowToAsync1, 248, 250, 251
- HowToCancelAsync, 254
- ImageBrushedSpiral, 70
- ImageFileIO, 718–725
- ImageRotate, 411
- ImplicitStyle, 81
- InkAndErase, 1030–1034
- InkEraseSelect, 1035–1042
- InternationalHelloWorld, 37
- JiggleButtonDemo, 406, 407
- KeypadWithViewModel, 235, 237, 239
- LineCapsAndJoins, 199, 201
- LineCapsAndJoinsWithCustomClass, 203, 204
- ListBoxWithItemTemplate, 494
- ListBoxWithUniformGrid, 511
- LookAtAppBarButtonStyles, 295–297
- LookAtFontMetrics, 836, 837
- MadTeaParty, 868
- ManipulationManagerDemo, 687, 689
- ManipulationTracker, 670, 672
- ManualBrushAnimation, 111
- ManualColorAnimation, 112
- NaiveBorderedText, 118
- NativeUp, 570, 571
- NewToggleDemo, 548
- NonAffineStretch, 458–461
- OppositelyScaledText, 416
- OrientableColorScroll, 174, 175
- OrientationAndOrientation, 961, 962
- OverlappedStackedText, 36
- PathMarkupSyntaxCode, 74
- Petzold.ProgrammingWindows11.Chapter11, 547
- Petzold.ProgrammingWindows6.Chapter11, 486, 503–510, 546
- Petzold.Windows8.VectorDrawing, 737–740
- PetzoldWindows8Controls, 148
- PhotoScatter, 775–783
- PointerLog, 636, 639, 640
- Posterizer, 727, 729, 732, 733
- PreconfiguredAnimations, 392, 394
- PrimitivePad, 257–261
- PrintableClassHierarchy, 923, 926, 927
- PrintableTomKitten, 945–954
- PrintMonthlyPlanner, 937–942
- PrintPrintableArea, 919–921
- PrivateFonts, 858, 860, 863, 865
- QuickNotes, 271
- RadialGradientBrushDemo, 710, 712, 715, 716
- RainbowAnimation, 388
- RainbowEight, 113
- RainbowEightTransform, 440
- ReflectedAlphaImage, 705, 706
- ReflectedFadeOutImage, 418
- ReversePaint, 771
- RgbBarChart, 513
- RichTextEditor, 886, 889–893
- RotateAroundCenter, 404, 405
- RotateTheText, 400
- RotatingMap, 1005–1016

RotationCenterDemo, 413
 RoutedEvents0–7, 90–98
 ScalableInternationalHelloWorld, 105
 SegoeSymbols, 299–302
 SharedBrush, 61, 62
 SharedBrushWithBinding, 83
 SharedBrushWithStyle, 78, 79
 SharedStyleWithDataTemplate, 471
 SilasMarner, 880–884
 SilentPiano, 664–668
 SimpleAnimation, 352
 SimpleAnimationCode, 358, 359
 SimpleColorScroll, 170–173
 SimpleCompass, 979, 980
 SimpleContextDialog, 287
 SimpleContextMenu, 284
 SimpleEllipse, 121
 SimpleHorizontalStack, 125
 SimpleInking, 1021–1023, 1026
 SimpleKeyFrameAnimation, 386
 SimpleKeypad, 185, 188
 SimpleListBox, 492
 SimplePageNavigation, 573, 574
 SimpleProjection, 399
 SimpleRotate, 398
 SimpleVerticalStack, 123
 SingleFingerRotate, 690, 691
 SkewPlusSkew, 425
 SkewSlideInText, 427
 SliderBindings, 166
 SliderEvents, 164
 SliderSketch, 180
 SpeedometerProgressBar, 542
 SpinPaint, 848–852
 Spiral, 66, 67
 SpringLoadedSlider, 539–541
 SquaringTheCircle, 380–382
 StackPanelWithScrolling, 133
 StretchedSpiral, 69
 StrippedDownHello, 46
 SuspendResumeLog, 269
 SystemInfoPInvoke, 794, 795
 TapAndShowPoint, 154
 TapTextBlock, 89
 TapToFlip, 451–453
 TextBoxInputScopes, 206
 TextEffects, 408
 TextFormatting, 60
 TextFormattingAppBar, 305, 306
 TextOnCanvas, 153
 TextStretch, 76
 TheTaleOfTomKitten, 140, 141
 ThreeDeeSpinningText, 448, 449
 TiltAndBounce, 977
 TiltAndRoll, 975
 TiltedShadow, 437
 TypographyDemo, 866
 UnconventionalAppBar, 291
 VectorGraphicsStretch, 76
 VisitedPageSave, 581–583
 WhatRes, 556
 WhatSize, 102
 WhatSizeWithBindingConverter, 130
 WhatSizeWithBindings, 128
 WhatSnap, 565
 WheresMyElement, 444, 446
 Whirligig, 660
 WordFreq, 277–280
 WrappedText, 35
 XamlCruncher, 316–324, 327, 330–347
 XYSliderDemo, 680–684
 YawPitchRoll, 981, 983
 YellowPad, 1044–1060
 YoungGoodmanBrown, 872–875
 prosta, 736
 przechowywanie ustawień aplikacji, 256
 przechwytywanie
 wskaźnika, 636, 658, 744
 zdjęć, 787
 przejścia, 392
 przekazywanie danych, 590
 przekształcanie obrazu, 1001
 przekształcanie
 afiniczne, 454
 nieafiniczne, 456
 przyrostowe, 691
 przełączanie
 widoczności elementów, 616
 właściwości Selected, 1035
 przełącznik. 304
 EasingMode, 376
 ToggleButton, 308
 przestrzenie nazw
 System.*, 23
 Windows.*, 23
 przestrzeń kolorów
 ARGB, 34
 RGB, 34
 scRGB, 34
 sRGB, 34

przestrzeń nazw

- System.Collections.Generic, 490
 - System.Collections.Specialized, 490
 - System.ComponentModel, 217, 490
 - System.IO, 255
 - System.Runtime.InteropServices, 793
 - System.Runtime.InteropServices.Windows
 - ↳ Runtime, 266, 700
 - System.Threading.Tasks, 244, 264
 - Windows.Devices.Geolocation, 959, 1004
 - Windows.Devices.Sensors, 965
 - Windows.Foundation, 67
 - Windows.Foundation.Collections, 490
 - Windows.Graphics.Display, 33, 569
 - Windows.Graphics.Imaging, 716
 - Windows.Graphics.Printing, 912
 - Windows.Graphics.Printing.OptionDetails, 912
 - Windows.Media.Capture, 784
 - Windows.Storage, 256, 265
 - Windows.Storage.BulkAccess, 257
 - Windows.Storage.Pickers, 256
 - Windows.Storage.Streams, 708
 - Windows.System.Threading, 272
 - Windows.UI.ApplicationSettings, 904
 - Windows.UI.Core, 205
 - Windows.UI.Input.Inking, 1019
 - Windows.UI.Popups, 284
 - Windows.UI.Text, 885
 - Windows.UI.Text.FontWeight, 825
 - Windows.UI.ViewManagement, 565
 - Windows.UI.Xaml, 11
 - Windows.UI.Xaml.Controls, 25, 50, 162
 - Windows.UI.Xaml.Controls.Primitives, 164
 - Windows.UI.Xaml.Documents, 59, 866
 - Windows.UI.Xaml.Input, 88
 - Windows.UI.Xaml.Markup, 74
 - Windows.UI.Xaml.Media, 44, 50, 67
 - Windows.UI.Xaml.Media.Animation, 349
 - Windows.UI.Xaml.Media.Imaging, 843
 - Windows.UI.Xaml.Media.Media3D, 991
 - Windows.UI.Xaml.Printing, 912
 - Windows.UI.Xaml.Shapes, 66, 75
- przesunięcie, 675, 1002
- przesuwanie
- elipsy, 390
 - spirali, 68
 - na skos, 676
 - obiektu, 400
 - okręgu, 386

- w pionie, 675
 - w poziomie, 675
- przewężenia, 651
- przezroczystość, 34, 704, 705
- przycinanie, 848
- przycisk, 181, 233, 293
- CheckBox, 183
 - Drukuj, 917
 - HyperlinkButton, 183
 - Kopiuj, 906
 - Monochromatyczny, 733
 - Naprzód, 578
 - Otwórz plik, 893
 - Pochylenie, 891
 - Podkreślenie, 891
 - Pogrubienie, 891
 - RadioButton, 184
 - RepeatButton, 183
 - ToggleButton, 183, 640
 - Wklej, 906, 907
 - Wstecz, 578, 905
 - Wyczyść, 850
 - Zapisz, 748
 - Zapisz jako, 723, 893
- przyciski myszy, 587
- przyspieszenie, 677, 964, 973
- przyspieszeniomierz, 969, 976
- przywracanie stron, 579
- punkty
- dotyku, 764
 - styczne, 657, 741

R

- renderowanie
- grafiki wektorowej, 121
 - grubych linii, 199
- rodzaje przycisków, 181
- rodzina czcionek, 825, 827, 855
- routowana obsługa wejścia, 91
- rozciąganie tekstu, 75
- rozdzielczość ekranu, 555, 559, 721
- rozmiar
- czcionki, 289, 934
 - mapy bitowej, 849
 - strony, 816
 - tablicy, 700
 - wiersza i kolumny, 168
- rozszewianie, dithering, 170
- rozszerzenia typograficzne, 866

równanie
 kwadratowe, 712
 prostej, 736
RTF, Rich Text Format, 885
ruch kulki, 974, 977
ryśik, 1019
ryśik cyfrowy, 1020
rysowanie
 krzywych Béziera, 1031
 linii, 735, 772, 841, 1021, 1033, 1038
 linii poziomych, 1049
 linii przerywanej, 364
 mapy bitowej, 847
 na ekranie, 1021
 na SurfaceImageSource, 821, 841
 odwrotne, 770
 palcami, 633, 660, 734
 pociągnięć, 1034, 1045, 1052
 spirali Archimedesesa, 66
 suwakami, 179
rzutowanie, 447, 456
rzutowanie argumentu sender, 90

S

schowek, 906
SDK, Software Development Kit, 9
segmenty
 Béziera, 1027
 linii łamanej, 1027
sensor Inclinometer, 981
sensory, 959
serializacja, 605
Silverlight, 191, 235
siła, 964
siła nacisku, 647
sinusoida, 377
skalowanie, 416, 560, 675, 998, 1002
 centrowane, 686
 mapy bitowej, 741, 773
składanie transformacji, 430
składnia
 elementu Binding, 132
 element-właściwość, 52, 53, 203
 Path Markup Syntax, 74
 XAML, 49
skrypt Windows PowerShell, 835
słownik
 Dictionary, 583, 633
 pages, 580, 583

 pageState, 582, 587
 Resources, 63
słowo kluczowe
 abstract, 138
 async, 252, 273
 Auto, 169
 await, 251
 extern, 794
 override, 94
 partial, 23, 42
 private, 824
 protected, 138
 ref, 822
 StaticResource, 62
 var, 41
sortowanie, 804, 924
specyfikacja XML, 862
spirala Archimedesesa, 66
splajn Béziera, 1025
sprawdzanie kolizji, 976
stała
 AngleIncrement, 663
 PROCESSOR_ARCHITECTURE_ARM, 796
 StrokeThickness, 651
stan
 FullScreenPortrait, 598
 Portrait, 601
 Pressed, 665
 Selected, 551
 SelectedDisabled, 551
 SelectedPointerOver, 551
 SelectedPressed, 551
 SelectedUnfocused, 551
 Snapped, 598, 600
 Unselected, 551
 zawieszenia, 267
stany
 aplikacji, 583
 strony, 580
 widoku, 612
 wizualne, 528, 536, 551
sterowanie układem, 117, 161
stoper, 788
stos powrotu, 583
stosy poziome, 125
strefy czasowe, 798, 818
stronicowanie, 878, 922, 928, 955
struktura
 ArcSegment, 738
 DWRITE_FONT_METRICS, 838

- struktura
 - DYNAMIC_TIME_ZONE_INFORMATION, 801
 - catalogów, 777
 - LineSegment, 737
 - PrintPageDescription, 919, 921
 - RoundCappedPath, 740
 - SYSTEM_INFO, 792
 - SYSTEMTIME, 800
 - TIME_ZONE_INFORMATION, 799–805
 - union, 793
 - Vector2, 655, 737, 975
 - Vector3, 992
 - strumień
 - JPEG, 706
 - pamięciowy, 707
 - styl
 - AppBarButtonStyle, 293
 - domniemany, 545
 - domyślny kontrolki, 536
 - HomeAppBarButtonStyle, 294
 - SaveAppBarButtonStyle, 309
 - SettingsAppBarButtonStyle, 308
 - style, 78
 - niejawne, 81, 83
 - przycisków, 293, 616
 - TextBlock, 83
 - suwak, 163, 166, 179
 - suwak pionowy, 167
 - symbol saksofonu, 303
 - symulacja klawiatury, 185
 - szablon
 - Blank App, 601
 - Blank Page, 601
 - ContentControl, 680
 - ControlTemplate, 518–522, 529, 544
 - DataTemplate, 479, 482, 493
 - Grid App, 597, 600
 - ItemsPanelTemplate, 516
 - ItemTemplate, 513
 - Split App, 597
 - TemplateBinding, 523, 539
 - URI, 1007, 1011
 - szablony, 465
 - domyślne, 549
 - kontrolki, 537
 - Visual Studio, 596
 - współdzielone, 470
 - szerokość
 - elementu, 401
 - kontrolki, 903
 - Grid, 538
 - Thumb, 539
 - szkielet
 - pliku generic.xaml, 544
 - procedury obsługi zdarzenia, 89
- ## Ś
- ściananie, shear, 424
 - środek
 - obrotu, 691, 850
 - przekształcenia, 1002
- ## T
- tablica
 - masek, 732
 - pikseli, 698, 726, 908
 - TAP, Task-based Asynchronous Pattern, 244
 - technika Platform Invoke, 791
 - technologia
 - IntelliSense, 33
 - WPF, 24
 - tekst, 30
 - sformatowany, 855
 - wygrawerowany, 407
 - wytłoczony, 407
 - testowanie programów, 14
 - timer, 255
 - tło, 111
 - tło LightGray, 292
 - transformacja, 397
 - Composite, 436
 - MatrixTransform, 434
 - RenderTransformOrigin, 425
 - RotateTransform, 414, 433, 694
 - ScaleTransform, 416, 429, 436
 - TranslateTransform, 414, 433, 442
 - transformacje
 - afiniczne, 454, 462
 - nieafiniczne, 457, 462
 - rzutów, 447
 - związujące, 432, 463
 - translacja, 407
 - tryb
 - Auto, 180
 - Filled, 567, 568, 598
 - łagodzenia, 372

- pionowy, 617
- snap, 176
- Snapped, 566, 613
- Stretch, 958
- wideo, 785
- tworzenie
 - bibliotek, 147
 - BitmapDecoder, 783, 909
 - BitmapEncoder, 908
 - dekodera, 730
 - e-booka, 140
 - efektów startowych, 427
 - egzemplarze elementów, 39
 - elementu TextBlock, 137
 - FileOpenPicker, 719, 730
 - FileSavePicker, 723
 - InkFileManager, 1055
 - ISurfaceImageSourceNative, 843
 - klasy GradientButton, 196
 - kodera, 731
 - kodera losowego, 725
 - kontrolki, 146
 - losowego koloru, 648, 658
 - mapy bitowej, 726, 768
 - modeli widoku, 625
 - notatek, 1043
 - obiektu do rysowania linii, 773
 - Path, 654
 - PathFigure, 1028
 - PathGeometry, 657, 1026
 - pędzli, 49, 111
 - pliku dziennika, 269
 - PointerInfo, 658, 744
 - Polyline, 644
 - PopupMenu, 814
 - PrintTask, 941
 - projektu, 21
 - RichTextBlock, 882
 - SolidColorBrush, 844
 - StrokeStyle, 844
 - strumienia pamięciowego, 707
 - SurfaceImageSource, 849
 - TimeZoneClock, 815
 - układu, 138
 - wektora, 740
 - własnych kontroltek, 163
 - WriteableBitmap, 712, 722, 731, 783
 - wykresu słupkowego, 513
 - zegara analogowego, 418

- typ
 - MIME, 717, 724, 894
 - TappedEventHandler, 88
 - wyliczeniowy
 - ApplicationViewState, 565
 - DisplayOrientations, 569, 960
 - FontStretch, 825
 - InkManipulationMode, 1029
 - LayoutKind, 793
 - ManipulationModes, 670, 675, 690
 - ParagraphAlignment, 892
 - PreviewPageCountType, 955
 - SimpleOrientation, 960
 - StyleSimulations, 861
 - VirtualKey, 589
- typy
 - inercji, 675
 - Windows Runtime, 374
 - żądań, 1007

U

- układ współrzędnych, 966, 986
- unieszkodliwianie znaków, 34
- uruchamianie
 - animacji, 351
 - zdarzeń, 850
 - programu, 45
 - na symulatorze, 27
 - pod kontrolą debugera, 26
 - z ekranu startowego, 27
- ustalanie pozycji kropek, 156
- ustawianie
 - aplikacji, 329
 - czasu lokalnego, 814
 - elementu Image, 772
 - fokusu, 643
 - koloru, 1047
 - pędzla, 249
 - piksela, 742, 783, 908
 - właściwości obiektu, 53
- usuwanie linii zaznaczenia, 1039, 1053
- UTC, Universal Coordinated Time, 798

V

- Visual Studio, 9

W

- walidacja danych, 233
- warstwa
 - View, 216
 - ViewModel, 216, 228, 329
- wartości
 - typu double, 359
 - typu HRESULT, 823
- wartość
 - BITRES, 1012
 - NaN, 677
 - null, 85
 - quadkey, 1013
 - WCHAR, 799
 - WORD, 792
- wątek, 243
 - interfejsu użytkownika, 734, 955
 - pomocniczy, 272
- wektor
 - Accelerometer, 971
 - przyspieszenia, 974
- wektory linii stycznych, 657
- wersje systemu, 9
- wiązanie danych, 83, 106, 128, 216, 875
- widok, 215
 - Snapped, 598
 - typu zatrask, 129
- widok-model, 215
- Windows
 - Runtime, 10
 - Runtime Component, 819
 - Runtime Library, 821
 - Store, 820
- WinRT, 10, 215
- wirtualizacja, 496
- własne metody asynchroniczne, 272
- właściwości, 652
- właściwości
 - atramentu, 1023
 - drukowania, 929
 - elementu Grid, 168
 - elementu TextBlock, 90
 - funkcji łączącej, 372
 - klasy
 - AccelerometerReading, 964
 - CompositeTransform, 436
 - Control, 162
 - DoubleAnimation, 357
 - FrameworkElement, 161
 - InkStrokeRenderingSegment, 1025
 - ManipulationDeltaRoutedEventArgs, 676
 - Panel, 501
 - Paragraph, 868
 - PlaneProjection, 450
 - Pointer, 632
 - RangeBase, 163
 - Slider, 167
 - obiektu, 53
 - publiczne, 238
 - statyczne, 191, 317, 993
 - typu Point, 380
 - typu TransitionCollection, 394
 - zależności, 32, 189, 317, 545
 - złączone, attached properties, 153, 366
- właściwość
 - ActualHeight, 928
 - ActualWidth, 129
 - Alignment, 892
 - AlignmentX, 70
 - All, 489
 - AllowDrop, 611
 - Angle, 400
 - Application.Current, 65
 - ApplicationViewState, 566
 - AutomationProperties.Name, 294
 - AutoParsing, 337
 - AutoReverse, 355, 368
 - Background, 49, 100, 379, 767
 - BackStackDepth, 579, 582
 - BasedOn, 80
 - BeginTime, 356, 378
 - BezierControlPoint1, 1025
 - BitmapAlphaMode, 722
 - BitmapPixelFormat, 722
 - BorderBrush, 209, 520, 523
 - BorderThickness, 119, 523
 - Build Action, 29
 - By, 354
 - CalculateCommand, 234, 237
 - CanGoBack, 576
 - CanPasteFromClipboard, 1042
 - CanRecorderItems, 611
 - Canvas.Left, 367
 - Canvas.Top, 367
 - Canvas.ZIndex, 158, 453
 - Center, 380
 - Children, 41
 - Color1, 193
 - Colors.Transparent, 774

Command, 234
 CommandParameter, 234, 235
 Completed, 245
 ContactRect, 647
 Container, 690
 Content, 44, 117, 163, 466, 524, 579
 ContentProperty, 55, 59
 ContentTemplate, 471, 518, 525
 ContentTransitions, 528
 ControlTemplate, 519
 Converter, 129
 CornerRadius, 524
 Cumulative, 676
 Data, 73
 DataContext, 226, 234, 468, 496, 618
 Delta, 673, 676
 DesiredDeceleration, 677
 DesiredDisplacement, 677
 DesiredSize, 507
 DisplayGrid, 778
 DisplayInformation, 804
 DisplayMemberPath, 494
 DisplayProperties.AutoRotationPreferences, 963, 964, 972
 DisplayProperties.CurrentOrientation, 106, 961
 DisplayProperties.NativeOrientation, 570, 963
 DrawingAttributes, 1024, 1025
 Duration, 359
 EasingFunction, 357
 EasingMode, 377
 EditOrientation, 330, 332
 EditOrientationRadioButton, 343
 EnableDependentAnimation, 351, 367
 EndPoint, 51
 ErrorText, 934
 ExportedTypes, 136
 Fill, 390
 FillBehavior, 354
 FitToCurve, 1024
 FontFamily, 41, 837, 841, 855
 FontRenderingEmSize, 861
 FontSize, 32, 38, 110, 114, 523, 918
 FontSizeProperty, 32
 FontStyle, 41, 42
 FontUri, 861, 862
 Foreground, 31, 32, 509
 FriendlyName, 723
 From, 369
 GlobalOffset, 451
 GradientOrigin, 712
 GradientStops, 54
 GroupName, 184
 Handled, 96
 HasOverflowContent, 871, 878
 HeaderTemplate, 597
 HeadingMagneticNorth, 981
 Height, 905
 HorizontalAlignment, 26, 98, 125, 526
 HorizontalChange, 211
 HorizontalScrollBarVisibility, 133
 ID, 905
 Ignorable, 25
 IgnorePressure, 1024
 ImageableRect, 922
 ImageSource, 70, 712
 Indices, 862
 InitializeColor, 593
 InkFileManager, 1050
 Inlines, 59, 867
 IsBarrelButtonPressed, 1020
 IsChecked, 183
 IsDirectionReversed, 166
 IsEnabled, 629
 IsHitTestVisible, 629
 IsHoldingEnabled, 646
 IsInContact, 641, 663, 1023
 IsInRange, 641
 IsLargeArc, 74
 IsLightDismissEnabled, 289, 905
 IsOpen, 293, 719
 IsReadOnly, 206
 IsSticky, 292
 IsTextSelectionEnabled, 871
 ItemContainerStyle, 550
 ItemsPanel, 509
 ItemsPanelTemplate, 517
 ItemsSource, 488, 625
 ItemTemplate, 518, 598
 KeyTime, 387
 LoadedFilename, 747
 LoadedFilePath, 747
 LocalFolder, 256
 LocalSettings, 256
 Location, 817
 ManipulationMode, 670, 673–675, 690
 Margin, 120, 180, 368, 919

MaxLength, 206
 MaxWidth, 141
 Minimum, 165
 MinimumReportInterval, 965
 Mode, 690, 1029
 NamedColor.All, 492
 NavigationCacheMode, 577, 581, 595
 NavigationMode, 580
 NewValue, 195
 Offset, 113, 366
 Opacity, 34, 364
 Orientation, 125, 151, 318, 512, 960
 OriginalSource, 92, 101, 104, 165, 491, 647, 673
 OriginX, 861
 Padding, 120, 958
 PageMapping, 948
 PageNumber, 917
 PageSize, 919
 Parameter, 579
 Path, 85
 PenTip, 1024
 Pivot, 690
 PixelHeight, 44, 698
 PixelWidth, 44
 PointerId, 632, 634
 PointerPointProperties, 647
 Position, 632, 686, 1025
 Pressure, 647, 1020, 1025
 PrintTaskOptions, 919
 Projection, 399
 ProjectionMatrix, 455
 Properties, 588
 Relative, 440
 RelativeSource, 495
 RelativeTransform, 440
 RenderingTime, 110
 RenderTransform, 397, 402, 686, 776
 RenderTransformOrigin, 404, 438, 841
 RepeatBehavior, 355, 367
 ReportInterval, 965
 RequestedTheme, 33, 292
 Resources, 61
 ReturnColor, 595
 RoamingFolder, 256
 RoamingSettings, 256
 Rotation, 674
 RotationY, 453
 RowDefinition, 176
 Scale, 674
 ScaleTransform, 400
 ScrollView, 151
 Selected, 1035
 SelectedIndex, 492, 884, 1054
 SelectedItem, 492, 1058
 SelectedText, 871
 SelectedValue, 492
 SelectedValuePath, 494
 SelectionLength, 205
 SelectionStart, 205, 871
 Shape, 199
 SolidColorBrush, 173, 179
 Source, 30, 44, 70, 623, 701
 SourcePageType, 579, 594
 SpeedRatio, 357, 426
 SpreadMethod, 442
 StartPoint, 51
 StepFrequency, 165
 Storyboard.TargetProperty, 366
 Stretch, 28, 70, 158, 999
 StretchDirection, 76
 Stroke, 66
 StrokeDashCap, 362
 StrokeDashOffset, 363
 StrokeEndLineCap, 650
 StrokeLineJoin, 199
 StrokeStartLineCap, 199, 650
 StrokeThickness, 362
 StudentBody, 607
 Students, 608, 620
 SwapChildren, 319
 TabSpaces, 328
 Tag, 188, 198, 781
 TargetType, 81, 130, 520
 Template, 161, 518, 519
 TemporaryFolder, 256
 Text, 35, 59, 108, 205
 TextAlignment, 35
 TextIndent, 868
 TextWrapping, 35, 311
 ThumbToolTipValueConverter, 177
 TiltX, 1025
 TimeSpan, 359
 Timestamp, 960
 TimeZoneKey, 817
 Title, 620
 To, 367
 ToolTipService.ToolTip, 515
 TotalScore, 217, 218
 Transform, 440, 725

- TranslateTransform, 366, 400
- Translation, 673
- Triggers, 379
- Twist, 1025
- UnconsumedBufferLength, 260
- UnicodeString, 861
- UriSource, 44, 1000
- Value, 391
- Velocities, 676
- VerticalAlignment, 37
- Visibility, 390, 478, 532, 629, 667
- Width, 541
- x:Class, 24
- YawDegrees, 985
- ZIndex, 157, 210
- włączenie przycisków, 722
- WPF, Windows Presentation Foundation, 24, 191, 235, 402
- wprowadzanie
 - danych, 491
 - danych z klawiatury, 205
 - wprowadzanie tekstu, 895
- wskazówki
 - ekranowe, 177
 - zegara, 807
- wskazywanie plików, 256
- wskaźnik, 636, 792
- wskaźnik na strukturę, 800
- współczynnik
 - interpolacji, 712, 1003
 - proporcji ekranu, 173
 - translacji, 431
- współdzielenie
 - danych, 590, 901, 906
 - ekranu, 565
 - pędzli, 61
- współrzędne
 - homogeniczne, 432
 - horyzontalne, 996
 - sferyczne, 992
 - wyświetlacza, 976
- wstawianie grafik, 43
- wstępnie pomnożona alfa, 705
- wybór
 - koloru HSL, 759
 - platformy, 835
- wygładzanie przewężień, 651
- wyjątek, 262, 685, 721, 754
 - COMException, 823
 - TaskCanceledException, 254, 255
- wykres słupkowy, 515
- wyskakujące okna dialogowe, 287
- wysokość elementu, 991
 - ScrollView, 139
 - TextBlock, 839
- wyświetlacz, 33
- wyświetlanie
 - bitmap, 27, 65, 788, 847
 - daty, 943
 - dokumentu, 872
 - drzewa, 136, 924
 - elementu TextBlock, 158
 - menu, 815
 - miniatury, 779, 780
 - okna MessageDialog, 246, 254
 - tekstu, 65, 866
 - wartości Slider, 177
 - zegarów, 810
- wywołania
 - asynchroniczne, 243, 263, 782
 - rekurencyjne, 925
 - zwrotne, 250, 915, 930
- wywołanie
 - FileIO.ReadTextAsync, 270
 - FileIO.WriteTextAsync, 270
- wyzwalanie zdarzeń wejściowych, 91
- wznawianie aplikacji, 268
- wzorec MVVM, 83, 233
- wzór na
 - koło, 736
 - prostą, 736

X

- XAML, Extensible Application Markup Language, 11, 23
- XML, Extensible Markup Language, 12, 23
- XPS, XML Paper Specification, 862

Z

- zadanie asynchroniczne, 955
- zagnieżdżanie elementów Grid, 170
- zakres
 - stron, 954
 - współrzędnych, 773
- zaokrąglone obramowanie, 524
- zapisywanie
 - do pliku, 660
 - mapy bitowej, 755

- zapisywanie
 - pikseli, 725
 - plików, 260, 264, 749
 - plików obrazów, 716
 - rozdzielczości, 721
 - rysunku, 734
 - stanu strony, 583
 - ustawień, 755, 1048
 - zawartości InkFileManager, 1055
- zarządzane języki programowania, 11
- zasoby
 - współdzielone, 65
 - XAML, 61
- zatrzask, 9, 129
- zawieszanie aplikacji, 268
- zawijanie tekstu, 862
- zaznaczanie, 491
 - pociągnięć, 1035
 - w RichTextBlock, 871
- zdarzenia, 87
 - dotykowe, 629
 - generowane przez pióro, 630
 - inklinometru, 989
 - Key, 87
 - Manipulation, 87, 630, 674, 685
 - nawigacji, 579
 - Pointer, 87, 630, 636, 663, 668
 - routowane, routed events, 90, 104, 165
 - w klasie Control, 629
 - w klasie UIElement, 629, 669
- zdarzenie
 - AcceleratorKeyActivated, 588
 - AddPages, 918, 942
 - CanExecuteChanged, 239
 - CharacterReceived, 205, 895, 897
 - Checked, 201, 287, 297
 - Click, 162, 213, 250–253, 279, 358, 581, 618
 - Closed, 314, 347
 - CollectionChanged.ObservableCollection, 604
 - CommandsRequested, 904
 - Completed, 595
 - CompositionTarget, 851
 - CompositionTarget.Rendering, 110–114, 272, 349, 472, 814
 - ContextMenuOpening, 871
 - DisplayProperties.LogicalDpiChanged, 557
 - DisplayProperties.OrientationChanged, 570
 - DoubleTapped, 87
 - DragCompleted, 208
 - DragDelta, 208, 211
 - DragEnter, 88
 - DragLeave, 88
 - DragOver, 88
 - DragStarted, 208
 - Drop, 88
 - GotFocus, 88
 - GetPreviewPage, 917, 942
 - Holding, 87
 - ItemClick, 611, 618
 - KeyDown, 205, 329, 895
 - KeyUp, 205, 895
 - LayoutUpdated, 104
 - Loaded, 105, 202, 262, 295, 313, 334, 378, 496, 1000, 1010, 1055
 - LostFocus, 88
 - ManipulationDelta, 669, 678, 682
 - ManipulationInertiaStarting, 669, 680
 - ManipulationStarted, 669, 682, 685
 - ManipulationStarting, 669, 690
 - OnCharButtonClick, 188
 - OnColorChanged, 769
 - OnFileOpenButtonClick, 264
 - OnPageSizeChanged, 106
 - OnPageTapped, 97
 - OnPointerPressed, 647
 - OnPointerReleased, 634
 - OnSuspending, 584
 - OnTextBlockTapped, 96
 - Opened, 314, 891
 - OptionChanged, 932, 951, 954
 - OrientationChanged, 571, 960
 - Paginate, 916–922, 942, 949
 - Pointer, 1021
 - PointerCaptureLost, 631, 642
 - PointerEntered, 642, 663
 - PointerExited, 631, 663
 - PointerMoved, 631, 651, 735, 850
 - PointerPressed, 588, 631, 645, 1035
 - PointerReleased, 631, 642
 - PrintTaskRequested, 912–915, 931, 940, 951, 957
 - PropertyChanged, 217, 220, 231, 472
 - ReadingChanged, 965
 - Rendering, 813
 - Resuming, 268
 - RightTapped, 87, 289, 645
 - SelectionChanged, 610, 841, 871
 - SizeChanged, 102–106, 175, 211, 313, 565, 877, 881, 972

- Suspending, 252, 268, 309, 586, 813
- Tapped, 87–97, 446, 491
- TextChanged, 233, 329
- Tick, 109
- UnhandledException, 338
- ValueChanged, 162, 165, 173, 178, 222, 302, 375, 684, 692
- zegar, 418, 423
- zmiana
 - głębi kolorów, 727
 - gradientu, 708
 - orientacji, 101, 569, 941
 - rozmiaru, 101
 - rozmiaru czcionki, 880
 - właściwości, 193
 - właściwości Color, 769
- zmienna exception, 263
- znacznik
 - Binding, 489
 - BOM, 894
 - Computerstan, 672
 - RichEditBox, 886
 - Setter, 519
 - StaticResource, 489
 - TemplateBinding, 522
 - VisualState, 531

- znak
 - #, 858
 - ampersand, 34
- znaki specjalne, 34
- zrzutowanie argumentu, 173
- zwalnianie zasobów, 847
- zwracanie danych, 590

Ż

- źródło
 - wiązania, 216
 - zdarzenia, 93

Ż

- żądanie
 - GetImageryMetadataAsync, 1007
 - GetMapUriAsync, 1007

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>

Twórz najlepsze aplikacje dla systemu Windows 8®



Sztandarowy produkt giganta z Redmond — Windows 8 — na dobre zagościł na rynku komputerowym. Już po pierwszym uruchomieniu rzuca się w oczy nowy interfejs użytkownika. Kafelki, bo o nich mowa, wzbudziły skrajne emocje. Część użytkowników wyjątkowo polubiła nowy wygląd ekranu, a inni mają problemy z przyzwyczajaniem się do niego. Jedno jest pewne — twórcy aplikacji powinni podążać za trendami wyznaczonymi przez Windows 8 i tworzyć spójne, przejrzyste aplikacje, wykorzystujące potencjał platformy.

Właśnie trzymasz w rękach najlepszą książkę poświęconą programowaniu dla platformy Windows. Jej najnowsze wydanie zostało zaktualizowane o wszystkie nowości, które pojawiły się w Windows 8. W trakcie lektury dowiesz się, co to jest XAML, poznasz jego składnię oraz możliwości. W kolejnych rozdziałach nauczysz się obsługiwać zdarzenia, tworzyć animacje, korzystać z wywołań asynchronicznych oraz wykonywać operacje natywne. Druga część książki zawiera informacje o funkcjach specjalnych: obsłudze urządzeń dotykowych, kamer, drukarek i rysików. Książka ta jest kompletnym przewodnikiem dla wszystkich programistów C# i osób zainteresowanych tym językiem.

Sięgnij po tę książkę i:

- poznaj nowości systemu Windows 8
- zobacz, jak tworzyć interfejs użytkownika Metro
- projektuj aplikacje na ekrany dotykowe
- odnieś sukces na rynku aplikacji dla platformy Windows 8

helion.pl
księgarnia
internetowa

Nr katalogowy: 15233

Księgarnia internetowa:
<http://helion.pl>

Zamówienia telefoniczne:
0 801 339900
0 601 339900



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

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



ISBN 978-83-246-7383-4



9 788324 673834

Cena 129,00 zł