Jarosław Cisek

Helion

Tworzenie nowoczesnych aplikacji graficznych w WPF

Dobry interfejs graficzny aplikacji Windows? Tylko z WPF!

- Poznaj metody tworzenia nowoczesnych i spójnych GUI
- Naucz się korzystać z możliwości WPF i języka XAML
- Dowiedz się, jak łączyć atrakcyjne interfejsy z logiką programów

Redaktor prowadzący: Michał Mrowiec

Projekt okładki: Jan Paluch

Fotografia na okładce została wykorzystana za zgodą Shutterstock.com

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

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie?twnoap
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Kody źródłowe wybranych przykładów dostępne są pod adresem:
ftp://ftp.helion.pl/przyklady/twnoap.zip

ISBN: 978-83-246-3500-9

Printed in Poland.

Spis treści

Rozdział 8.

WPF i Windows 7

W tym rozdziale:

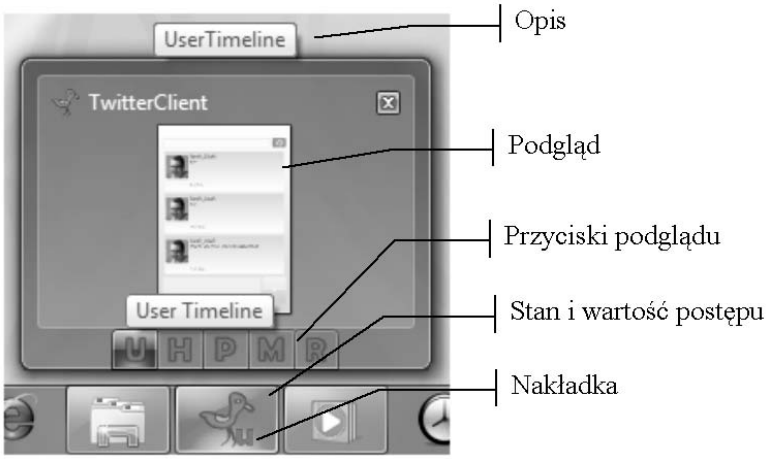
- dostosowywanie paska zadań,
- listy szybkiego dostępu,
- Aero Glass.

Windows 7 posiada nowe funkcje w porównaniu ze swoimi poprzednikami — należą do nich na przykład listy szybkiego dostępu. WPF w związku z tym posiada API, za pomocą którego możemy w naszych aplikacjach wykorzystywać te nowe funkcje — o tym właśnie będzie ten rozdział. Zobaczysz, jak można wpływać na informacje pokazywane w pasku zadań, jak w aplikacji WPF można manipulować tym, co jest wyświetlane w podglądzie aplikacji, oraz nauczysz się umieszczać zawartość w liście szybkiego dostępu. Na końcu rozdziału poruszę kwestie związane z interfejsem Aero Glass — zmienimy naszą aplikację TwitterClient tak, by całe okno wraz z listą wiadomości było przezroczyste.

8.1. Dostosowywanie paska zadań

Pasek zadań w systemie Windows 7 ma trochę inny wygląd niż jego poprzednicy, posiada również pewne nowe funkcje. Zapewne każdy zauważył, że w momencie najechania kursorem na pasek zadań poszczególne ikony oznaczające aplikacje są kolorowane dominującą barwą. Dodatkowo w przypadku niektórych programów można zaobserwować, że jeżeli wystąpi błąd ich działania, ich ikona w pasku zadań staje się czerwona. Jest jeszcze jedna funkcjonalność paska zdań, z którą na pewno każdy miał okazję się spotkać — jeżeli pobieramy jakieś dane z internetu, ikona przeglądarki w pasku zdań wskazuje poziom postępu procesu pobierania — jest to również jedna z nowych możliwości manipulacji zachowaniem paska zadań. Wszystkie te funkcjonalności z wyjątkiem zmiany koloru podświetlania ikony w pasku zadań (w tym przypadku jedyną opcją zmiany koloru podświetlania aplikacji jest zmiana kolorów ikony) da się zaimplementować w aplikacji WPF.

Zacznijmy od tego, co nowego możemy zaobserwować w związku z paskiem zadań (pomijamy na razie kwestie list szybkiego dostępu omówionych w kolejnych podrozdziałach). Rysunek 8.1 pokazuje podgląd aplikacji oraz wygląd paska zadań wraz z jego poszczególnymi elementami (jako przykład w tym rozdziale posłuży nam aplikacja TwitterClient).



Rysunek 8.1. *Pasek zadań wraz z podglądem programu TwitterClient oraz opisem poszczególnych jego elementów*

Klasą, za pomocą której można manipulować paskiem zadań, jest `TaskbarItemInfo`. Poniższy fragment kodu pokazuje, jak zdefiniowano opis, który widoczny jest na rysunku 8.1 (*UserTimeline*).

```
<Window.TaskbarItemInfo>
    <TaskbarItemInfo  Description="{Binding ElementName=Window, Path=TimeLine}"/>
</Window.TaskbarItemInfo>
```

Opis jest ustawiany przez wpisanie do właściwości `Decription` obiektu `TaskbarItemInfo` odpowiedniego napisu. W powyższym kodzie napis jest zdefiniowany za pomocą wiązania danych z właściwością `TimeLine` przechowującą typ wyliczeniowy zawierający wartości `User Timeline`, `Home Timeline`, `Private Timeline`, `Mentions` i `Retweete By Me` — czyli grupy wiadomości. Ponieważ wartości tego typu są wpisywane do zmiennej przy każdej zmianie wartości właściwości `TimeLine` , utworzone wiązanie danych zapewni samoczynne odświeżanie opisów.

„Grupy wiadomości" to wiadomości serwisu Twitter, które są zgrupowane tematycznie, przykładowo *UserTimeLine* zawiera tylko wiadomości napisane przez nas, *HomeTimeLine* to wiadomości nasze i osób, które śledzimy. W języku angielskim nazywane są one *time lines*. Niestety tej nazwy nie da się poprawnie przetłumaczyć na język polski — jedyne, co przyszło mi do głowy, to właśnie „grupy wiadomości". Ponieważ tłumaczenie wydaje mi się niezbyt trafione, a niestety lepszego nie jestem w stanie wymyślić (Google też ;)), proponuję stosowanie angielskiego określenia *time line*.

Właściwości zależne

Aby powyższe wiązania zadziałały, właściwość `TimeLine` nie może być zwykłą właściwością, ponieważ technika wiązania danych wymaga, aby źródło (w tym przypadku właściwość `TimeLine`) powiadamiało cel wiązania (właściwość `Description`) o zmianie swojej zawartości — w przeciwnym razie właściwość `Description` przechowywałaby ciągle tę samą, wpisaną za pierwszym razem wartość właściwości `TimeLine`. Właściwość `TimeLine`, aby mogła powiadamiać o swojej zmianie, musiała zostać zdefiniowana jako właściwość zależna, co przedstawia poniższy fragment kodu.

```
DependencyProperty.Register("TimeLine", typeof(TimeLines), typeof(MainWindow));
      public TimeLines TimeLine
      {
          get { return (TimeLines)GetValue(TimeLineProperty); }
          set { SetValue(TimeLineProperty, value); }
      }
```

Zagadnienie właściwości zależnych omówione jest dokładnie w rozdziale „Podstawy WPF".

Zagadnienie wiązania danych jest omówione w rozdziale „Wiązanie danych", jeśli więc zapis w powyższym kodzie nie jest dla Ciebie zrozumiały, nie przejmuj się — wystarczy wiedzieć, że dzięki niemu samoczynnie będą się zmieniały podpowiedzi po przełączeniu time line na inny — rysunek 8.2 pokazuje opis po zmianie *time line* na *HomeTimeLine*.

Rysunek 8.2. *Zmiana opisu po zmianie time line*

Zamiast używać wiązania danych, moglibyśmy zdefiniować wartość w następujący sposób:

```
Description="Aplikacja TwitterClient"
```

co spowodowałoby wyświetlenie napisu „Aplikacja TwitterClient" zamiast napisu widocznego na rysunku 8.2.

Następną w kolejności informacją prezentowaną w pasku zadań, a pokazaną na rysunku 8.1, jest podgląd. W tym przypadku nie mamy dużego wpływu na to, jak może być w nim prezentowana nasza aplikacja — jedyna możliwość manipulacji to ustawienie odpowiednich marginesów tak, by tylko część okna była widoczna. Takiego ustawienia dokonuje się za pomocą właściwości `ThumbnailClipMargin` typu `Thickness`, czyli za

pomocą czterech liczb typu `Double` określamy odstęp z każdej ze stron prostokąta (kolejno od lewej, od góry, od prawej i od dołu) — przykład definicji poniżej.

```
<TaskbarItemInfo  ThumbnailClipMargin="5,5,5,480" …/>
```

Działanie marginesów i jednocześnie powyższego fragmentu kodu pokazuje rysunek 8.3 — widać, że tylko górna część aplikacji jest pokazywana — dolna część stanowiąca 480 jednostek została „obcięta".

Rysunek 8.3. *W wyniku zastosowania właściwości ThumbnailClip ↪Margin w podglądzie prezentowana jest tylko część aplikacji*

Kolejną z udostępnionych możliwości dostosowywania wyglądu paska zadań jest umieszczanie w nim przycisków. Realizuje się to przy wykorzystaniu właściwości `ThumbButtonInfos`, przechowującej kolekcję obiektów `ThumbButtonInfo`. Ważne, że nie są to przyciski dziedziczące po klasie `IUElement`. Posiadają jednak podstawowe funkcjonalności przycisków. Niestety (a może i „na szczęście") zawartością takiego przycisku może być jedynie obrazek — nie można więc po prostu wpisać do niego zawartości tekstowej, która zostanie zaprezentowana. Sposób definiowania powyższych przycisków przedstawia poniższy fragment kodu. Jego działanie możemy obserwować na wszystkich powyższych rysunkach.

```
<TaskbarItemInfo  …>
    <TaskbarItemInfo.ThumbButtonInfos>
        <ThumbButtonInfo Description="User Timeline"
        ↪ImageSource="Images\UGreen.png"  Click="user_Click"/>
        <ThumbButtonInfo Description="Home Timeline"
        ↪ImageSource="Images\HGreen.png" Click="home_Click"/>
        <ThumbButtonInfo Description="Public Timeline"
         ↪ImageSource="Images\PGreen.png" Click="public_Click"/>
        <ThumbButtonInfo Description="Mentions" ImageSource="Images\MGreen.png"
         ↪Click="mentions_Click"/>
        <ThumbButtonInfo Description="Retweeted By Me"
        ↪ImageSource="Images\RGreen.png" Click="rbm_Click"/>
</TaskbarItemInfo.ThumbButtonInfos>
 </TaskbarItemInfo>
```

Jak widzimy, opisy, które pojawiały się na rysunkach nad przyciskami, zdefiniowane były za pomocą właściwości `Description`. Dla każdego z przycisków można również definiować uchwyt dla zdarzenia kliknięcia.

Pod przyciskami widoczny jest już tylko sam pasek zadań. Tam mamy do dyspozycji dwie możliwości zmiany jego wyglądu — zdefiniowanie nakładki dla ikony aplikacji w pasku oraz wskazanie stanu i postępu danego procesu.

Element `TaskbarItemInfo` oferuje dodatkowe możliwości informowania użytkownika o stanie aplikacji i wykonania operacji, odpowiednio za pomocą właściwości `Progress↪Value` i `ProgressState`. `ProgressValue` jest właściwością typu `Double` przyjmującą wartości od `0` do `1` — oznaczające procent wykonania czynności. Na wszystkich dotychczasowych rysunkach właściwość ta była ustawiona na wartość `1`, co oznaczało 100% wykonania danego procesu. Druga właściwość — `ProgressState` — może przyjmować następujące wartości[1]:

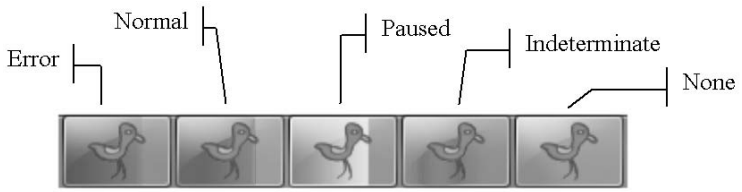
- `Error` — oznacza błąd, charakteryzuje się czerwonym wypełnieniem dookoła ikony aplikacji;
- `Normal` — stan normalny — zielone wypełnienie dookoła ikony aplikacji;
- `Paused` — pauza — pomarańczowe wypełnienie dookoła ikony aplikacji;
- `Indeterminate` — nieokreślony — stosowany w przypadku monitorowania procesu, o którego zaawansowaniu nie mamy żadnych informacji — prezentowany jest w postaci zielonego paska, który jest animowany w kierunku od lewej do prawej;
- `None` — domyślna wartość, stan postępu wykonania czynności nie jest prezentowany w pasku zadań.

Za jej pomocą możemy informować użytkownika o tym, w jakim stanie znajduje się w danym momencie aplikacja.

Na wszystkich powyższych rysunkach mogliśmy zobaczyć następujące ustawienia powyższych właściwości:

```
<TaskbarItemInfo ProgressState="Normal" ProgressValue="1" …/>
```

Rysunek 8.4 prezentuje natomiast wygląd aplikacji TwitterClient w pasku zadań w zależności od różnych wartości właściwości `ProgressState` oraz `ProgressValue` równej `0.75`.



Rysunek 8.4. *Wygląd paska zadań aplikacji TwitterClient dla właściwości ProgressValue ='0.75' oraz różnych wartości właściwości ProgressState*

Na koniec zostały nam nakładki. Nakładka również była widoczna na każdym z poprzednich obrazków — jest to mały obrazek prezentowany w pasku zadań w prawym dolnym rogu obok ikony aplikacji. Działa tylko wtedy, gdy pasek zadań pokazuje duże ikony. Nakładki stosowane są do przekazywania dodatkowych informacji użytkownikowi. Na powyższych przykładach widoczne były naprzemiennie literki *u* i *h* — odpowiednio skróty od *UserTimeLine* i *HomeTimeLine*. Służyły więc w tym przypadku do przekazywania informacji użytkownikowi o tym, jaki time line aktualnie ogląda

[1] Ibidem, s. 246.

w pasku zadań. Definicja nakładki jest bardzo prosta — do właściwości `Overlay` klasy `TaskbarItemInfo` należy wstawić nazwę obrazka, który ma być prezentowany w nakładce, co przedstawia poniższy fragment kodu.

```
<TaskbarItemInfo Overlay="Images\URed.png" … />
```

Dynamiczna zmiana nakładki

Rysunki z tego podrozdziału nie pokazywały ciągle tej samej nakładki — zmieniała się ona wraz ze zmianą time line — nakładki były zamieniane w trakcie działania aplikacji. Jednym ze sposobów zmiany obrazka w nakładce jest zdefiniowanie wiązania danych pomiędzy właściwością `Overlay` a zmieniającą się właściwością `TimeLine` (podobnie jak to miało miejsce w przypadku zmiany opisów aplikacji). `TimeLine` przechowuje jednak jedynie oznaczenia time line, podczas gdy `Overlay` chce dostać dane obrazka, który ma być umieszczony w nakładce. Konieczne więc byłoby użycie konwertera (zagadnienie konwerterów jest dokładnie omówione w rozdziale „Wiązanie danych"), który zamieniłby wartość `TimeLine` na określoną reprezentację obrazka.

Inną możliwością jest zdefiniowanie zasobu, na przykład w oknie `Window`, jak w poniższym kodzie (zagadnienie zasobów zostanie omówione w rozdziale „Zasoby").

```
<Window.Resources>
  <DrawingImage x:Key="HRedOverlay">
      <DrawingImage.Drawing>
          <ImageDrawing ImageSource="Images\HRed.png" Rect="0,0,16,16"/>
      </DrawingImage.Drawing>
  </DrawingImage>
  <DrawingImage x:Key="URedOverlay">
      <DrawingImage.Drawing>
          <ImageDrawing ImageSource="Images\URed.png" Rect="0,0,16,16"/>
      </DrawingImage.Drawing>
  </DrawingImage>
…
</Window.Resources>
```

Następnie w momencie zmiany time line, czyli na przykład po naciśnięciu przycisku z paska zadań, należałoby odnaleźć konkretny zasób i wstawić jego wartość do właściwości `Overlay`:

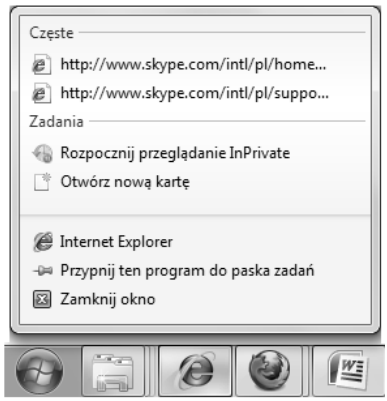
```
this.taskBarItemInfo.Overlay = (DrawingImage)FindResource("HRedOverlay");
```

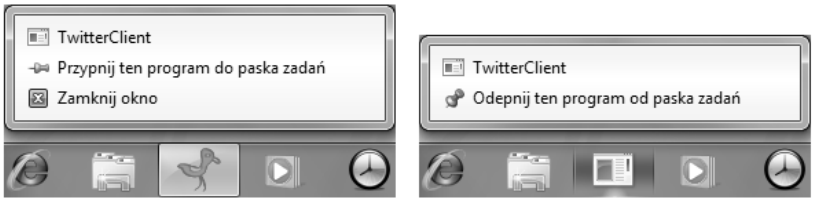
8.2. Listy szybkiego dostępu

Lista szybkiego dostępu jest tym, co pokazuje się po naciśnięciu prawym przyciskiem myszy na kontrolce aplikacji w pasku zadań lub przeciągnięciu kursora myszy do góry z jednocześnie przyciśniętym lewym przyciskiem myszy. Przykład listy szybkiego dostępu programu Internet Explorer pokazuje rysunek 8.5.

Domyślna lista szybkiego dostępu jest dołączana do każdej aplikacji w systemie Windows 7. Aplikacja może mieć listę szybkiego dostępu zarówno wtedy, kiedy działa, jak i wtedy, gdy jest wyłączona. Przykład domyślnej listy szybkiego dostępu dla aplikacji TwitterClient przedstawia rysunek 8.6.

Rysunek 8.5.
Lista szybkiego dostępu programu Internet Explorer w Windows 7





Rysunek 8.6. *Lista szybkiego dostępu dla programu TwitterClient, gdy aplikacja jest uruchomiona (po lewej) oraz gdy aplikacja jest zamknięta (po prawej)*

Oczywiście aby lista po prawej stronie rysunku 8.6 mogła być pokazana, konieczne było przypięcie aplikacji do paska zadań.

Listę szybkiego dostępu można zmieniać za pomocą właściwości dołączonej `JumpList.JumpList` ustawionej na wartość `JumpList`. Lista jest dołączana do powłoki jeden raz, zaraz po uruchomieniu aplikacji, więc umieszczamy jej definicję w klasie `App`. Alternatywną możliwością ustawienia listy szybkiego dostępu jest wywołanie metody `JumpList.SetJumpList` w kodzie proceduralnym.

Klasa `JumpList` posiada właściwość `JumpItems`, do której można wpisać zawartość listy. Mogą nią być dwie klasy dziedziczące po `JumpItem` — `JumpTask` i `JumpPath`[2]. W skrócie można powiedzieć, że `JumpTask` to połączenie kierujące do programu, natomiast `JumpPath` to połączenie prowadzące do pliku. W kolejnych podrozdziałach omówię każdą z tych klas.

8.2.1. JumpTask

Element `JumpTask` reprezentuje zadania do wykonania. Na rysunku 8.5 zadaniami są *Rozpocznij przeglądanie InPrivate* i *Otwórz nową kartę*. W praktyce służą one do uruchomienia aplikacji z parametrami linii poleceń. Przykład definicji elementu `JumpList` w aplikacji TwitterClient przedstawia poniższy fragment kodu:

[2] A. Nathan, *WPF 4 Unleashed*, *op. cit.*, s. 234, *http://msdn.microsoft.com/en-us/library/system.windows.shell.jumplist.jumpitems.aspx*, 2012-01-22.

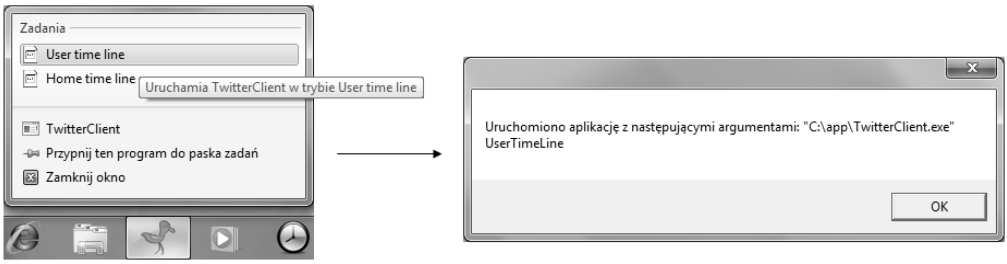
```
<Application x:Class="TwitterClient.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             StartupUri="MainWindow.xaml">
    <JumpList.JumpList>
        <JumpList>
            <JumpTask Title="User time line" Arguments="UserTimeLine"
                Description="Uruchamia TwitterClient w trybie User time line"/>
            <JumpTask Title="Home time line" Arguments="HomeTimeLine"
                Description="Uruchamia TwitterClient w trybie Home time line"/>
        </JumpList>
    </JumpList.JumpList>
…
</Application>
```

W powyższym kodzie utworzono dwa elementy `JumpTask`. Pierwszy z nich uruchamia kolejną instancję aplikacji TwitterClient z argumentem `UserTimeLine`. Drugi uruchamia tę aplikację z parametrem `HomeTimeLine`. Co to za argumenty? Są to po prostu argumenty linii poleceń, które możemy odczytać na przykład tak:

```
MessageBox.Show("Uruchomiono aplikację z następującymi argumentami: "+
↪Environment.CommandLine);
```

i od wyniku uzależnić działanie aplikacji (więcej informacji o metodach odczytywania parametrów przekazanych w linii poleceń znajduje się w rozdziale „Struktura i rodzaje aplikacji WPF”).

Rysunek 8.7 pokazuje zarówno wygląd listy zdefiniowanej w powyższym fragmencie kodu, jak i to, co przekazywane jest do aplikacji po wybraniu pierwszego zadania. Do odczytania przekazanych parametrów została wykorzystana klasa `Environment` i jej właściwość `CommandLine` (jej użycie widać w pokazanym powyżej fragmencie kodu). `CommandLine` przechowuje dwie wartości — pierwsza wartość to ścieżka do aplikacji, natomiast druga to argument przekazany do aplikacji. Argumenty te zostały wyświetlone w konstruktorze klasy `MainWidow` za pomocą okienka `MessageBox`.



Rysunek 8.7. *JumpList ze zdefiniowanym elementem JumpTask i argumenty przekazane do aplikacji wyświetlone za pomocą MessageBox*

Co do samego wyglądu listy, to widać, że podpowiedź wyświetlana do każdego z zadań definiowana jest za pomocą właściwości `Desctiption` klasy `JumpTask`. Element `Title` określa natomiast treść, która pokaże się jako odnośnik do danego zadania.

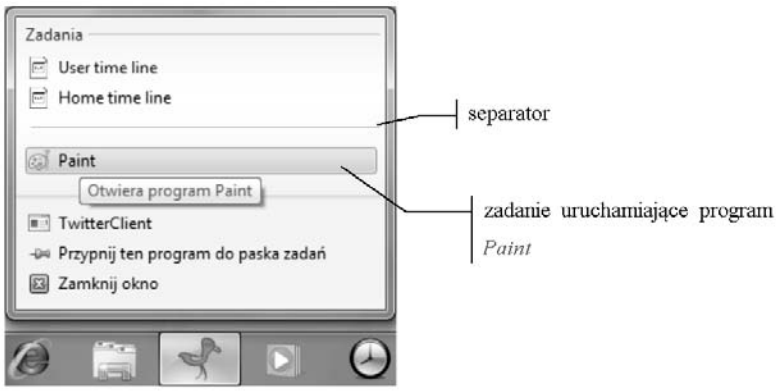
Przeanalizujmy krótko, jak zachowuje się TwitterClient po wybraniu zadania z listy. W przypadku gdy aplikacja jest zamknięta, przekazywane są odpowiednie argumenty, po czym TwitterClient jest uruchamiany. Spójrzmy teraz na działanie listy szybkiego dostępu dla programu Internet Explorer. Po wybraniu zadania *Otwórz nową kartę* gdy aplikacja jest wyłączona, uruchamiany jest Internet Explorer z nową kartą — czyli analogicznie do naszego przykładu. A teraz spójrzmy, co się stanie, gdy dla już uruchomionej aplikacji wybierzemy to samo zadanie — nowa karta otwierana jest w tej samej instancji aplikacji. Co natomiast dzieje się w naszym przykładzie? Za każdym razem otwierana jest nowa aplikacja. Ten problem można rozwiązać przy użyciu klasy `Microsoft.VisualBasic.ApplicationServices.Windows ↪FormsApplicationBase` — umożliwia ona zaimplementowanie komunikacji z parametrami przekazywanymi przez linię poleceń w czasie działania jednej instancji aplikacji.

Istnieją dodatkowe możliwości definiowania listy szybkiego dostępu. Przykład definicji zadania, które uruchamia aplikację *mspaint.exe*, oraz ikony dla tego zdania przedstawia poniższy fragment kodu definiujący kolejny element `JumpTask`:

```
<JumpTask/>
<JumpTask Title="Paint"
  Description="Otwiera program Paint"
  ApplicationPath="%WINDIR%\system32\mspaint.exe"
  IconResourcePath="%WINDIR%\system32\mspaint.exe"/>
<JumpTask/>
</JumpList>
```

Na rysunku 8.8 przedstawiono efekt działania tej definicji — widać, że w liście pojawiła się dodatkowa pozycja wraz z ikoną (odpowiednio właściwości `ApplicatonPath` oraz `IconResourcePath`).

Biblioteki *%WINDIR%\System32\shell32.dll* i *%WINDIR%\System32\imageres.dll* zawierają wiele ikon przydatnych do zastosowania w listach szybkiego dostępu.



Rysunek 8.8. *Program mspaint.exe uruchamiany za pomocą jednego z zadań*

Po wybraniu tego zadania zostanie uruchomiony program *mspaint.exe*. Widać, że zarówno ikony, jak i program lokalizowane są za pomocą zmiennych systemowych. Dodatkowo przed i po zadaniu `Paint` umieszczone zostały definicje elementów `JumpTask`

bez żadnych atrybutów — służą one do zdefiniowania separatorów również widocznych na rysunku 8.8.

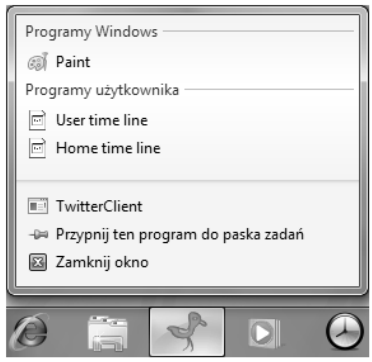
Windows 7 umożliwia również zgrupowanie elementów znajdujących się na liście szybkiego dostępu. Służy do tego właściwość `CustomCategory` typu `string` klasy `JumpItem`. Przykład definicji przedstawia poniższy fragment kodu:

```
<JumpTask CustomCategory="Programy użytkownika" Title="User time line"
↪Arguments="UserTimeLine"
 Description="Uruchamia TwitterClient w trybie User time line"/>
<JumpTask CustomCategory="Programy użytkownika" Title="Home time line"
↪Arguments="HomeTimeLine"
 Description="Uruchamia TwitterClient w trybie Home time line"/>
<JumpTask/>
<JumpTask  CustomCategory="Programy Windows" Title="Paint"
   Description="Otwiera program Paint"
   ApplicationPath="%WINDIR%\system32\mspaint.exe"
   IconResourcePath="%WINDIR%\system32\mspaint.exe"/>
<JumpTask/>
```

Grupy elementów są więc identyfikowane za pomocą nazwy grupy, do której chcą przynależeć. Co by się stało, gdyby dla jednego z elementów właściwość `CustomCategory` została niezdefiniowana? W takim przypadku zostałaby utworzona dla tego elementu domyślna grupa o nazwie „Zadania", którą mogliśmy zobaczyć na przykład na rysunku 8.7.

Wygląd listy po zdefiniowaniu grup tematycznych przedstawia rysunek 8.9.

Rysunek 8.9.
Poszczególne zadania przydzielone do grup

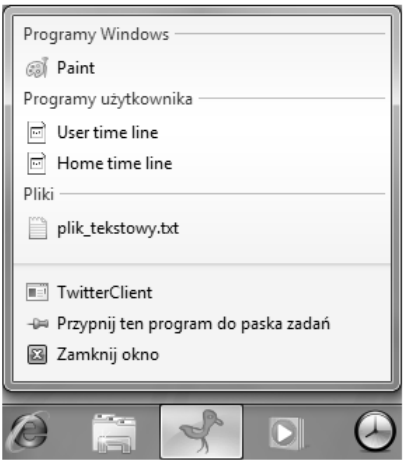


8.2.2. JumpPath

Za pośrednictwem `JumpPath` określa się elementy wskazujące na ścieżki do plików, które mogą być uruchomione przez daną aplikację. Przykład definicji i działania tego elementu przedstawiony jest w poniższym fragmencie kodu.

```
<JumpList.JumpList>
    <JumpList>
        <JumpPath Path="C:\plik_tekstowy.txt" CustomCategory="Pliki"/>
…
    </JumpList>
</JumpList.JumpList>
```

Jak widać, definicja `JumpPath` wygląda analogicznie do `JumpTask`. Ścieżka do pliku tekstowego została zdefiniowana za pomocą właściwości `Path` oraz umieszczona została w grupie *Pliki* — właściwość `CustomCategory` jest dziedziczona po klasie `JumpItem`. Na rysunku 8.10 przedstawiono wygląd elementu `JumpPath` w liście szybkiego dostępu.



Rysunek 8.10. *Element JumpPath w liście szybkiego dostępu*

Aby zdefiniowany plik *plik_tekstowy.txt* mógł się pokazać na liście, konieczne jest, by znajdował się we wskazywanej przez nas lokalizacji. Ponieważ nasza aplikacja Twitter ↪Client nie posiada funkcjonalności otwierania plików tekstowych, po wybraniu pliku *plik_tekstowy.txt* z listy zostanie uruchomiona druga instancja aplikacji TwitterClient.

W pokazanym fragmencie kodu definiującego element `JumpPath` widać ścieżkę bezwzględną do pliku. Jest to jedyna możliwość wskazania lokalizacji plików w przypadku elementu `JumpPath`[3]. Dlatego też aplikacje, które wykorzystują ten element, najczęściej definiują go w kodzie proceduralnym, co pozwala wykorzystać bieżące ustawienia, np. zmienne środowiskowe.

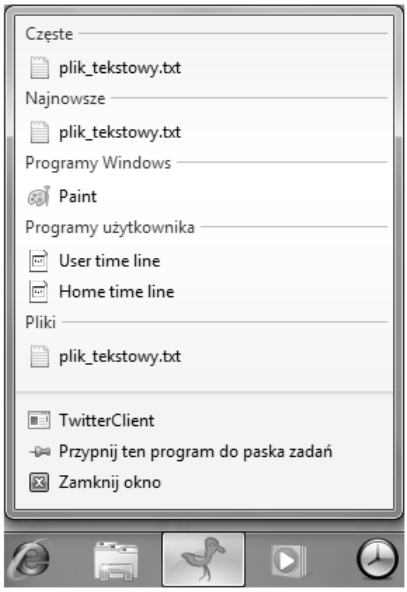
Na koniec tego podrozdziału spójrzmy jeszcze raz na listę `JumpList` programu Internet Explorer. Zawiera ona grupę o nazwie *Częste*. Pokazywana jest ona często zamiennie z inną grupą o nazwie *Najnowsze* — czyli po prostu listą często i ostatnio otwieranych plików i zadań. Na liście tej znajdują się pliki i zadania uruchamiane często lub ostatnio za pomocą okna dialogowego do otwierania plików lub konkretnej aplikacji, jeżeli dany rodzaj pliku został z nią skojarzony.

Aby taka grupa pokazała się na naszej liście, wystarczy ustawić dwie właściwości klasy `JumpList` odpowiedzialne za pokazywanie grupy najczęściej otwieranych plików i ostatnio otwieranych plików — odpowiednio `ShowFrequentCategory` i `ShowRecent` ↪`Category` — co przedstawia kod zamieszczony poniżej.

```
<JumpList ShowFrequentCategory="True" ShowRecentCategory="True">
…
</JumpList>
```

[3] Ibidem, s. 241.

Efekt widoczny jest na liście `JumpList` programu TwitterClient — rysunek 8.11.



Rysunek 8.11. *Element JumpList ze zdefiniowanymi właściwościami ShowFrequentCategory i ShowRecentCategory*

Na rysunku widać, że zarówno w kategorii *Częste*, jak i *Najnowsze* prezentowane są te same pliki — dlatego też zazwyczaj lista ostatnio otwieranych plików jest stosowana zamiennie z listą najczęściej otwieranych. Plik *plik_tekstowy.txt* znalazł się na liście najczęstszych i ostatnich, ponieważ pliki *.txt* zostały skojarzone w moim systemie z aplikacją TwitterClient, po czym otwarte przy jej użyciu. Generalnie tym, które pliki znajdą się na liście, zajmuje się powłoka Windows, ale możliwe jest też wymuszenie dodania elementów do omawianych grup poprzez wywołanie funkcji `JumpList.AddToRecent`
`↪Category`, której argumentami są albo zadania — `JumpTask`, albo pliki — `JumpPath`.

8.3. Aero Glass

Aero Glass jest terminem określającym okno, którego ramki są przezroczyste. Część będąca obramowaniem może być rozciągnięta na część użytkową aplikacji, co pozwala na uzyskanie efektu przezroczystej powłoki. Przykłady aplikacji wykorzystujących Aero Glass to Internet Explorer i Windows Media Player. Istnieją dwa ograniczenia co do możliwości definiowania przezroczystego obszaru[4]:

- Obszar ten musi być prostokątem — nie może to być inny kształt.
- Obszar przezroczysty jest rozszerzeniem ramki okna, więc nie może być zdefiniowany wewnątrz okna — jedyne opcje to albo przezroczysta ramka dookoła okna, albo całe przezroczyste okno.

[4] M. MacDonald, *Pro WPF Windows Presentation Foundation in .NET 3.0*, *op. cit.*, s. 242.

Efekt rozciągnięcia obramowania okna Aero Glass można uzyskać, wykorzystując API biblioteki *dwmapi.dll*, konkretnie funkcję `DwmExtendFrameIntoClientArea`. Za jej pomocą można rozszerzyć obramowanie okna na cały obszar roboczy lub jego część. Aby było możliwe użycie funkcji pochodzącej z kodu niezarządzanego, konieczne jest skorzystanie z mechanizmu `PInvoke` (ang. *Platform Invocation Service*). Umożliwia on przy wykorzystaniu atrybutu `[DllImport]` importowanie funkcji zapisanych w kodzie niezarządzanym w bibliotekach DLL. Poniżej znajduje się przykład definicji klasy `AeroGlass`, w której zdefiniowana jest metoda `ExtendGlass` używająca metody `DwmExtend`
`↪FrameIntoClientArea`. Na początku definiujemy strukturę, która będzie przechowywać odpowiednie marginesy. Następnie za pomocą Platform Invocation Services deklarujemy użycie metod z biblioteki. Jedna z nich to wspomniana `DwmExtendFrameInto`
`↪ClientArea`, druga będzie nam służyła do sprawdzenia, czy kompozycja pulpitu jest włączona.

Kompozycja pulpitu

Kompozycja pulpitu (ang. *Desktop composition*) to funkcjonalność zaprezentowana pierwszy raz w Windows Vista. Zmienia ona sposób, w jaki aplikacje pokazują piksele na ekranie, mianowicie jeżeli kompozycja jest włączona, nie rysują one okien bezpośrednio na ekranie, lecz ich instrukcje odnośnie do rysowania obiektów są przekierowywane do urządzenia graficznego. Dopiero to urządzenie renderuje ostateczną zawartość. Kompozycją pulpitu zarządza DMW, czyli Desktop Windows Manager. Funkcjonuje on jako usługa systemowa i może być wyłączony za pomocą narzędzi administracyjnych. Stosując kompozycję pulpitu, DMW udostępnia szereg efektów na ekranie, do których należą między innymi wsparcie dla wysokich rozdzielczości czy omawiane okna typu „Glass”[5].

Rozszerzenie ramki okna nastąpi jedynie w przypadku, gdy kompozycja pulpitu jest włączona oraz wersja systemu nie jest niższa niż 6 (wersje systemów operacyjnych Windows Vista i późniejsze).

W kolejnych liniach widzimy: pobranie uchwytu dla okna, zdefiniowanie jego tła na przezroczyste, pobranie rozdzielczości ekranu (aby marginesy zostały poprawnie obliczone) oraz wywołanie metody `DwmExtendFrameIntoClientArea` z odpowiednio przeliczonymi wartościami struktury `MARGIN`.

```
using System;
using System.Runtime.InteropServices;
using System.Windows;
using System.Windows.Interop;
using System.Windows.Media;

    [StructLayout(LayoutKind.Sequential)]
    public struct MARGINS
    {
        public MARGINS(Thickness thickness, float dpiX, float dpiY)
        {
            Left = (int)(thickness.Left * dpiX);
            Right = (int)(thickness.Right * dpiX);
            Top = (int)(thickness.Top * dpiY);
            Bottom = (int)(thickness.Bottom * dpiY);
```

[5] *http://msdn.microsoft.com/en-us/library/windows/desktop/aa969540%28v=vs.85%29.aspx*, 2012-01-22.

```
        }
        public int Left;
        public int Right;
        public int Top;
        public int Bottom;
    }

    public class AeroGlass
    {
        //deklaracja metod z biblioteki dwmapi.dll
        [DllImport("dwmapi.dll", PreserveSig = false)]
        static extern void DwmExtendFrameIntoClientArea(
            IntPtr hWnd, ref MARGINS pMarInset);
        [DllImport("dwmapi.dll", PreserveSig = false)]
        static extern bool DwmIsCompositionEnabled();

        public static void ExtendGlass(Window window, Thickness margin)
        {
            //jeżeli kompozycja jest nieustawiona lub system jest w starszej wersji, nic nie robimy
            if (DwmIsCompositionEnabled() && Environment.OSVersion.Version.Major > 5)
            {
                //uchwyt dla okna
                WindowInteropHelper helper = new WindowInteropHelper(window);
                HwndSource mainWindowSrc = (HwndSource)HwndSource.
                    FromHwnd(helper.Handle);

                // ustawienie tła okna na przezroczysty
                window.Background = Brushes.Transparent;
                mainWindowSrc.CompositionTarget.BackgroundColor = Colors.Transparent;
                //pobieramy rozdzielczość
                System.Drawing.Graphics desktop =
                   System.Drawing.Graphics.FromHwnd(mainWindowSrc.Handle);
                float dpiX = desktop.DpiX / 96;
                float dpiY = desktop.DpiY / 96;
                //ustawiamy marginesy
                MARGINS margins = new MARGINS(margin, dpiX, dpiY);
                //zwiększamy przezroczystą ramkę o stosowny rozmiar
                DwmExtendFrameIntoClientArea(mainWindowSrc.Handle, ref margins);
            }
            else
            {
                window.Background = SystemColors.WindowBrush;
            }
        }
    }
```

Teraz trzeba jeszcze w naszej klasie `MainWindow` wywołać metodę `ExtendGlass` klasy `AeroGlass`. Poniżej zamieszczono przykład użycia w nadpisanej metodzie `OnSource ↪Initializes`. Przekazany argument `new Thickness(-1)` oznacza, że ramka okna zostanie rozszerzona na całą jego powierzchnię.

```
protected override void OnSourceInitialized(EventArgs e)
        {
            base.OnSourceInitialized(e);
            AeroGlass.ExtendGlass(this, new Thickness(-1));
            IntPtr hwnd = new WindowInteropHelper(this).Handle;
            HwndSource.FromHwnd(hwnd).AddHook(new HwndSourceHook(WndProc));
        }
```

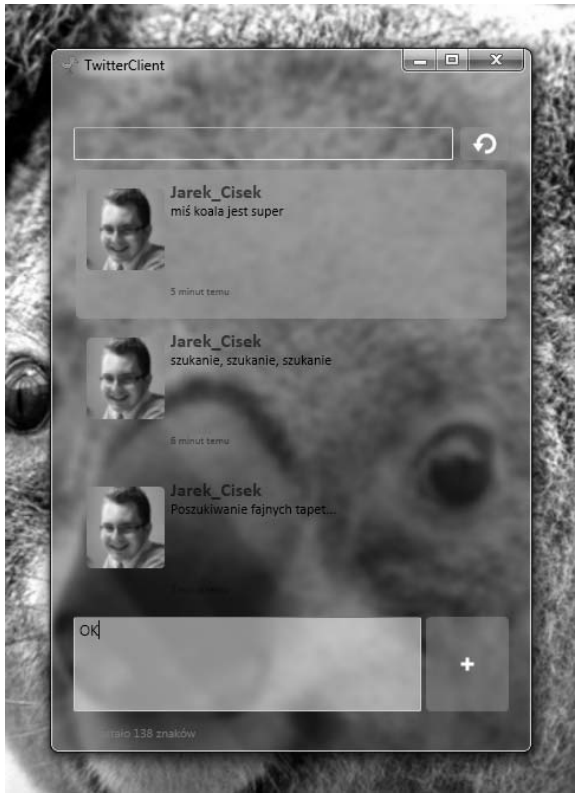
Do uruchomienia pozostało jeszcze jedno — ponieważ użytkownik może włączyć lub wyłączyć kompozycję pulpitu, należy zadbać o włączenie lub wyłączenie w zależności od sytuacji naszego okna. Można do tego wykorzystać metodę `WndProc`, która przetwarza wszystkie wiadomości wysyłane do okna.

```
private IntPtr WndProc(IntPtr hwnd, int msg, IntPtr wParam, IntPtr lParam, ref bool
        handled)
        {
            if (msg == WM_DWMCOMPOSITIONCHANGED)
            {
                AeroGlass.ExtendGlass(this, new Thickness(-1));
                handled = true;
            }
            return IntPtr.Zero;
        }
```

`WM_DWMCOMPOSITIONCHANGED` to stała zdefiniowana w klasie, przechowująca kod wiadomości o zmianie okna (0x031E), co pokazano poniżej.

```
private const int WM_DWMCOMPOSITIONCHANGED = 0x031E;
```

Spójrzmy więc, jak wygląda „szklany” TwitterClient — rysunek 8.12.



Rysunek 8.12. *TwitterClient z poszerzoną ramką Aero Glass*

Pytania testowe

1. Klasą, która umożliwia zmianę koloru tła okienka w pasku zadań po najechaniu kursorem myszy na ikonę aplikacji, jest:

   a) `TaskbarItemInfo`,

   b) `ThumbButtonInfo`,

   c) `JumpList`.

2. Zawartość którego z elementów wyświetlanych w oknie podpowiedzi i w pasku zadań może być zdefiniowana jak obrazek?

   a) nakładka,

   b) opis,

   c) zawartość przycisku `ThumButtonInfo`.

3. Która z poniższych wartości właściwości `ProgressState` wskazuje stan, w którym proces jest wykonywany, lecz nie da się określić stopnia jego zaawansowania?

   a) `Normal`,

   b) `None`,

   c) `Indeterminate`.

4. Wybierz zdanie, które jest prawdziwe dla poniższej definicji elementu `TaskbarItemInfo`:

```
<TaskbarItemInfo ProgressState="Error" ProgressValue="50"/>
```

   a) Okno programu w pasku zadań zostanie zakolorowane na pomarańczowo, postęp wykonania procesu to 50%.

   b) Okno programu w pasku zadań zostanie zakolorowane na czerwono, postęp wykonania procesu to 100%.

   c) Zdefiniowanie wartości 50 spowoduje błąd kompilacji, ponieważ wartość wychodzi poza dopuszczalne granice.

5. Wybierz prawdziwe zdanie.

   a) `JumpTask` służy do definiowania akcji uruchomienia programu z parametrami przekazywanymi przez linię poleceń.

   b) `JumpPath` służy do definiowania akcji uruchomienia programu z parametrami przekazywanymi przez linię poleceń.

   c) Lista `JumpList` jest podłączana raz, po uruchomieniu programu.

6. Wybierz prawdziwe zdanie opisujące poniżej zdefiniowaną listę `JumpList`.

```
<JumpList.JumpList>
  <JumpList>
      <JumpTask/>
      <JumpTask/>
```

```
        <JumpTask/>
        <JumpTask CustomCategory="Programy użytkownika" Title="Testowy program"
                Description="Uruchamia bieżącą aplikację"/>
      </JumpList>
    </JumpList.JumpList>
```

a) Do standardowej listy aplikacji zostaną dodane trzy separatory i nazwana grupa zatytułowana „Programy użytkownika", przechowująca element o nazwie „Testowy program".

b) Do standardowej listy aplikacji zostanie dodana grupa zatytułowana „Programy użytkownika", przechowująca element o nazwie „Testowy program", separatory zostaną zignorowane, ponieważ nowa grupa zawiera już swój separator.

c) Do standardowej listy aplikacji zostanie dodany jeden separator, następnie zostaną utworzone dwie puste pozycje, po czym zostanie dodana nazwana grupa zatytułowana „Programy użytkownika", przechowująca element o nazwie „Testowy program".

7. Wybierz poprawne zdania charakteryzujące klasę `JumpPath`.

a) Jest elementem wskazującym na ścieżki do plików, które mogą być uruchomione przez daną aplikację.

b) Do właściwości `Path` może zostać wpisana jedynie ścieżka bezwzględna do pliku.

c) `JumpPath` dziedziczy po `JumpItem`.

8. Aby w liście `JumpList` pokazywane były ostatnio otwierane pliki, należy:

a) Ustawić właściwość `ShowRecentCategory` klasy `JumpList` na wartość `True`.

b) Dodać pliki, które mają być pokazywane, za pomocą metody `AddToRecentCategory` klasy `JumpList`.

c) Dodać pliki, które mają być pokazywane, za pomocą metody `AddToRecentCategory` tylko wówczas, gdy ich pokazywanie ma być wymuszone.

9. Aby poszerzyć okno Aero Glass, należy użyć funkcji:

a) `WndProc`,

b) `DwmIsCompositionEnabled`,

c) `DwmExtendFrameIntoClientArea`.

Odpowiedzi do pytań

1. a

2. a, c

3. c

4. b

5. a, c

6. b

7. a, b, c

8. a, c

9. c

Skorowidz

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

V

W

X

Z

Ź

Windows Presentation Foundation to nowoczesny system graficzny, umożliwiający tworzenie i wyświetlanie atrakcyjnych wizualnie aplikacji okienkowych dla środowiska Windows. Technologia ta powstała w odpowiedzi na stale rosnącą potrzebę dostarczania wysokiej jakości grafiki, której prezentacja opiera się na wykorzystaniu akceleracji sprzętowej i nie wiąże się z dużym obciążeniem zasobów systemowych. Oddzielenie warstwy interfejsu użytkownika od logiki aplikacji umożliwia definiowanie interfejsów przy użyciu deklaratywnego języka XAML. Upraszcza tym samym proces tworzenia przejrzystych i elastycznych GUI.

Każdy szanujący się autor programów działających pod kontrolą systemów Windows powinien dobrze opanować WPF, zwłaszcza jeśli ma na celu produkowanie naprawdę interesujących aplikacji okienkowych przy możliwie niewielkim nakładzie sił i środków. Pomoże mu w tym odpowiednie źródło wiedzy takie jak książka *Tworzenie nowoczesnych aplikacji graficznych w WPF*. W ciekawy i przystępny sposób zostały w niej zaprezentowane najważniejsze informacje na temat korzystania z Windows Presentation Foundation, tworzenia nowoczesnych interfejsów użytkownika, zastosowania języka XAML, a także wyświetlania grafiki i animacji oraz obsługi zdarzeń związanych z używaniem różnego rodzaju urządzeń wskazujących.

- Zasady tworzenia graficznych interfejsów użytkownika
- Charakterystyka platformy WPF i języka XAML
- Struktura i typy aplikacji WPF oraz metody ich tworzenia
- Używanie paneli i zarządzanie układem ich składników
- Korzystanie z różnego rodzaju kontrolek
- Definiowanie interfejsów aplikacji systemu Windows 7
- Obsługa zdarzeń pochodzących z urządzeń wejściowych
- Używanie zasobów i wiązanie danych
- Stosowanie stylów, szablonów, skórek i motywów
- Prezentacja grafiki 2D, 3D i animacji

A wszystko to zilustrowane wieloma praktycznymi i ciekawymi przykładami!

Sięgnij po jedną z nielicznych książek poświęconych technologii WPF i twórz nowoczesne aplikacje graficzne!

helion.pl
księgarnia internetowa

Nr katalogowy: 7932

Księgarnia internetowa
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

Helion

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

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

Informatyka w najlepszym wydaniu

Cena 89,00 zł

ISBN 978-83-246-3500-9

9 788324 635009