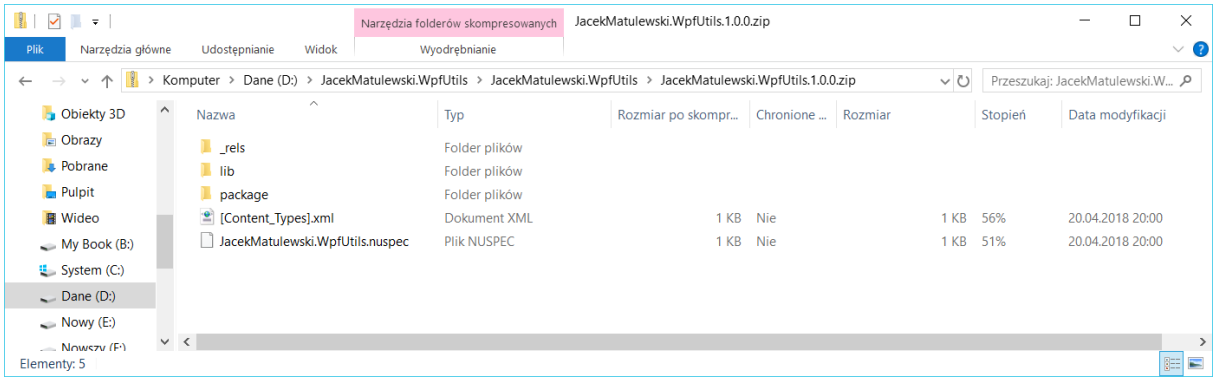
Pakiety NuGet

Rozdział z książki *Visual Studio 2017. Tworzenie aplikacji Windows w języku C#* Helion 2018

[1]Pakiety NuGet pozwalają na łatwe dzielenie się kodem z innymi programistami. Mechanizm ten nie przewiduje opłat, ale możliwe jest ustalenie warunków licencji. Witryna głównego repozytorium znajduje się pod adresem *http://nuget.org*[2]. Umożliwia przeglądanie i pobieranie pakietów, choć te czynności łatwiej wykonać korzystając z klienta wbudowanego w Visual Studio (w tej książce wykorzystywaliśmy go już wielokrotnie). Tylko poprzez stronę można natomiast wysyłać do serwisu utworzone przez siebie pakiety.

Pakiety NuGet, czyli pliki z rozszerzeniem *.nupkg* to tak naprawdę archiwa ZIP zawierające opis pakietu oraz pliki binarne np. biblioteki DLL. Te ostatnie umieszczone są w podkatalogu *lib*. Rysunek B.1. pokazuje zawartość przykładowego pakietu, którą łatwo można obejrzeć w Windows, jeżeli zmienimy rozszerzenie pliku pakietu na *.zip*.



Rysunek B.1. Zawartość pakietu NuGet

Samo tworzenie pakietów NuGet zostało doskonale rozwiązane w Visual Studio 2017, ale tylko dla projektów przeznaczonych dla platformy .NET Standard. Wystarczy w ustawieniach takiego projektu, na zakładce *Package* zaznaczyć pole opcji *Generate NuGet package on build*, aby pakiet został utworzony w momencie kompilacji projektu. Nie wymaga to żadnych dodatkowych narzędzi. Ten pakiet można następnie „uploadować" na stronie *nuget.org*. W przypadku innych platform, w tym UWP i .NET, już nie jest to tak wygodne[3].

Instalacja nuget.exe

Aby stworzyć pakiet dla platformy .NET, należy zainstalować *NuGet Command-Line Interpreter* (w skrócie NuGet CLI)[4], który można pobrać ze strony *https://www.nuget.org/downloads*. Visual Studio już od wersji 2012

---

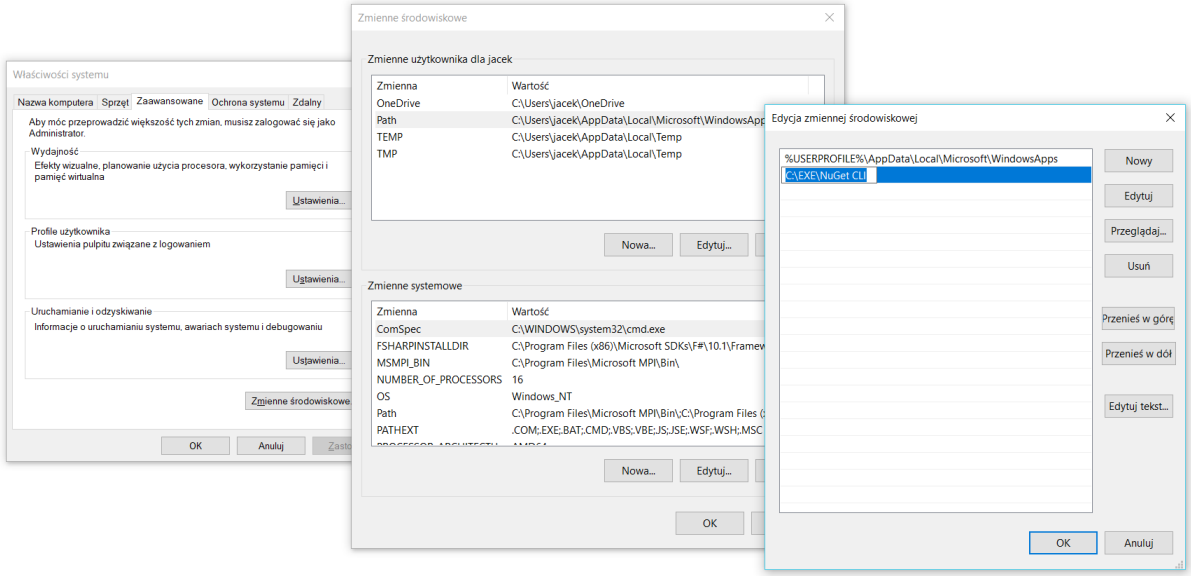
[1] Dziękuję studentom Krystianowi Jabłońskiemu oraz Marcinowi Burakowi za zebranie informacji dotyczących tworzenia pakietów NuGet.

[2] Możliwe jest też lokalne przechowywanie pakietów NuGet (lokalne repozytorium Visual Studio), a nawet własny hosting pakietów.

[3] Strona MSDN zawierająca informacje o tworzeniu pakietów NuGet to *https://docs.microsoft.com/en-us/nuget/create-packages/creating-a-package*

[4] Istnieją dwa narzędzia: NuGet CLI i .NET Core 2.0 CLI. Nas interesuje to pierwsze. Drugie przeznaczone jest dla .NET Core.

zawiera *NuGet Package Manager*, ale bez pliku *nuget.exe*, który jest wymagany do tworzenia własnych pakietów. Po wejściu na powyższą stronę, należy pobrać najnowszą rekomendowaną wersję pliku *nuget.exe*. W kwietniu 2018 roku była to wersja 4.6.2. Co ciekawe, pobiera się bezpośrednio plik wykonywalny, który nie jest instalatorem – zamiast uruchamiać, należy go wobec tego zapisać, a katalog, w którym go zapisaliśmy dodać do ścieżki przeszukiwania. Dzięki temu będzie można go uruchamiać z linii poleceń bez specjalnych zabiegów. Ja zapisałem go do katalogu *c:\EXE\NuGet CLI*. Następnie dodałem ten katalog do zmiennej środowiskowej `PATH` (rysunek B.2).



Rysunek B.2. Modyfikowanie zmiennej środowiskowej PATH

Tworzenie pakietu z projektu biblioteki klas

Uruchommy Visual Studio i wczytajmy dowolny projekt biblioteki DLL klas dla platformy .NET. Ja użyłem projektu biblioteki *JacekMatulewski.WpfUtils*, która zawiera różne fragmenty kodu opisane w tej książce, w tym klasę `Font`, klasę okien dialogowych, `RelayCommand` itp.[5].

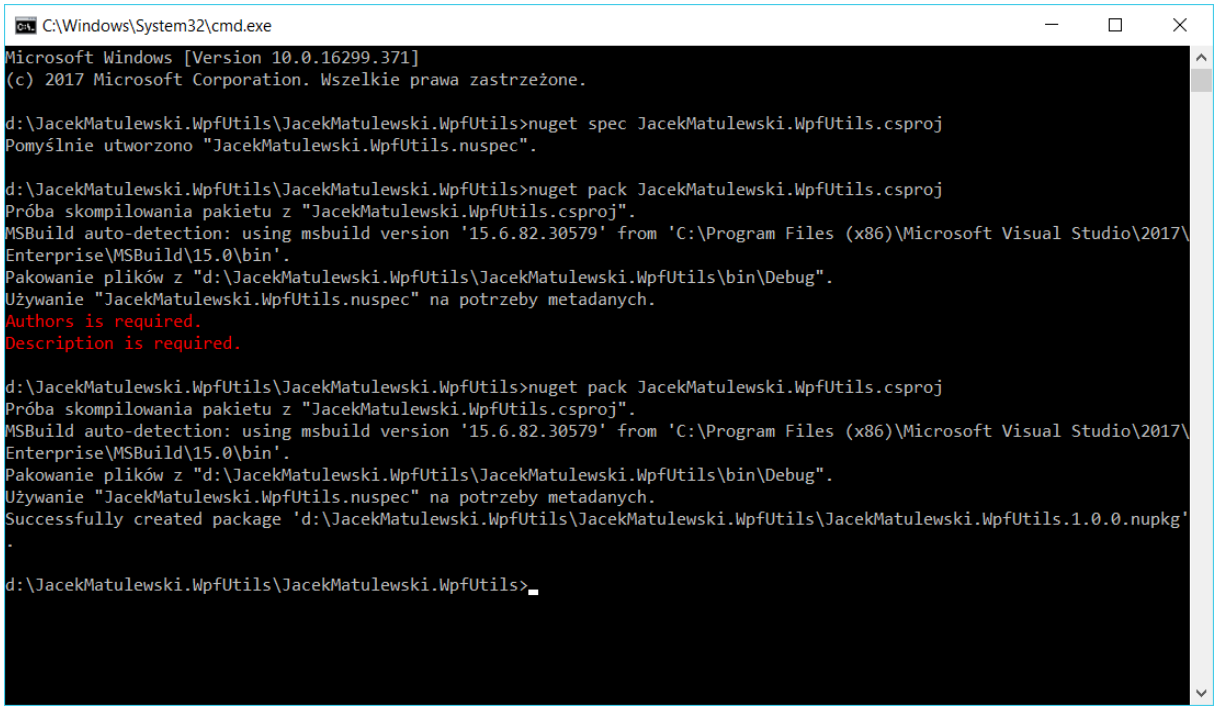
Warto zmniejszyć wersję platformy .NET, dla której przeznaczona jest biblioteka, żeby jak najbardziej poszerzyć krąg odbiorców. W moim przypadku jest to .NET Framework 4.5. Należy również uzupełnić opis projektu (pozwala na to przycisk *Assembly Information...* w ustawieniach projektu) – te dane posłużą do stworzenia opisu paczki. Następnie zadbajmy o to, żeby przebudować projekt w trybie *Release* tak, żebyśmy na pewno udostępniali najnowszą wersję kodu.

Kolejną czynnością jest utworzenie pliku *.nuspec* z opisem paczki, który również jej zawartość. Aby go utworzyć należy przejść do katalogu projektu (katalog, w którym jest plik *.csproj*) i uruchomić w nim wiersz poleceń (rysunek B.3). Następnie wpisujemy polecenie `nuget spec JacekMatulewski.WpfUtils.csproj` (nazwa projektu może być oczywiście inna)[6]. W efekcie utworzony zostanie plik *JacekMatulewski.WpfUtils.nuspec*.

Następnie możemy spróbować wykonać polecenie `nuget pack JacekMatulewski.WpfUtils.csproj`, ale w moim przypadku zakończyło się to błędem (rysunek B.3). Do utworzenia paczki wymagane jest bowiem określenie autora i opisu projektu. Jeżeli zajrzymy do pliku *.nuspec* (to plik w formacie XML), zobaczymy, że dane te powinny być pobierane z projektu. Jeżeli to się nie udało, a jest to błąd, który na forach jest często przywoływany, otrzymujemy właśnie taki błąd.

[5] Kod źródłowy tej biblioteki dołączony jest do kodów źródłowych tej książki.

[6] Warta wspomnienia jest możliwość wykonania tej czynności również na skompilowanym pliku DLL.



Rysunek B.3. Częsty błąd podczas tworzenia paczki

Najprostszym rozwiązaniem jest wstawienie w odpowiednich znacznikach pliku *.nuspec* właściwych wartości bez odwoływania do danych odczytanych z projektu (listing B.1). Przy okazji należy uzupełnić lub skasować linie dotyczące licencji, czy opisu wydania.

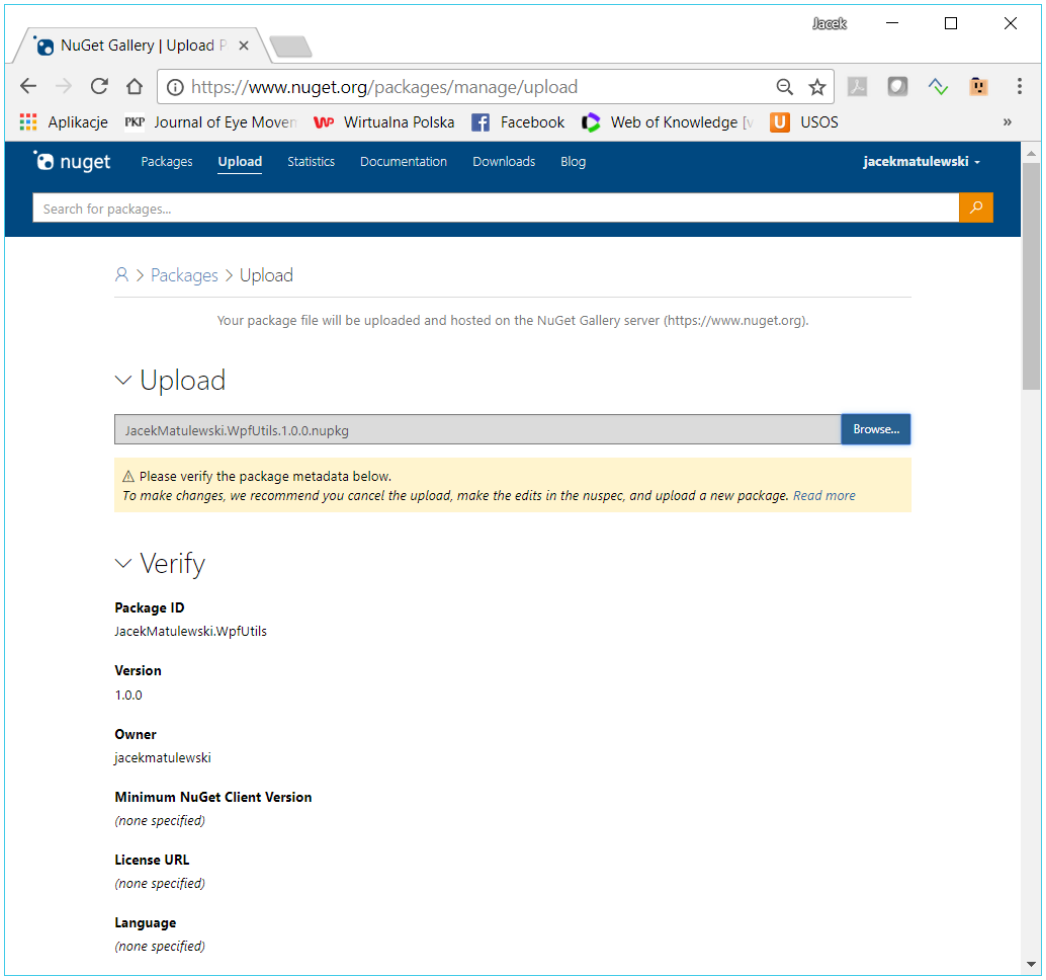
Listing B.1. Ręcznie zmodyfikowany plik .nuspec

```
<?xml version="1.0"?>
<package >
  <metadata>
    <id>$id$</id>
    <version>$version$</version>
    <title>$title$</title>
    <authors>Jacek Matulewski</authors>
    <owners>$author$</owners>
    <licenseUrl>http://LICENSE_URL_HERE_OR_DELETE_THIS_LINE</licenseUrl>
    <projectUrl>http://www.fizyka.umk.pl/~jacek</projectUrl>
    <iconUrl>http://www.fizyka.umk.pl/~jacek/index.ico</iconUrl>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>Some useful code from the book by Jacek Matulewski</description>
    <releaseNotes>Summary of changes made in this release of the package.</releaseNotes>
    <copyright>Copyright Jacek Matulewski 2018</copyright>
    <tags>WPF Utils</tags>
  </metadata>
</package>
```

Po wprowadzeniu zmian, ponówmy polecenie `nuget pack JacekMatulewski.WpfUtils.csproj`. Tym razem utworzenie paczki powinno się udać. W efekcie powstanie plik ZIP, ale z rozszerzeniem *.nupkg* (druga próba pakowania widoczna na rysunku B.1), w którym w podkatalogu *lib/net45/* znajduje się biblioteka DLL.

Publikacja pakietu

Kolejnym krokiem jest opublikowanie pakietu. Otwórzmy w przeglądarce stronę *http://nuget.org*. Publikowanie pakietów wymaga zarejestrowania w serwisie – można się jednak zalogować korzystając z istniejącego konta Microsoft. Po potwierdzeniu adresu e-mail, należy przejść na zakładkę *Upload*. Możemy plik pakietu przeciągnąć na widoczne na tej stronie pole tekstowe (działa w Chrome, ale nie działa w Edge) lub klikając przycisk *Browse...* – wybrać plik korzystając ze standardowego okna dialogowego. Następnie warto przejrzeć wyświetlone dane dotyczących pakietu, w tym adresy stron WWW, dodać tekst dokumentacji przygotowanej w języku Markdown[7] i jeżeli wszystko jest w porządku kliknąć przycisk *Submit* na dole strony. Pakiet nie zostanie od razu opublikowany, a wpierw będzie poddany procesowi automatycznej walidacji oraz indeksowania. W moim przypadku walidacja zajęła tylko 3 minuty, ale pakiet został zindeksowany, co jest warunkiem umożliwiającym jego wyszukanie w serwisie, dopiero po kilkudziesięciu minutach. Zasięg odbiorców jest ogromny – nawet tak nierozpoznawalny kod po jednym dniu został pobrany dwadzieścia parę razy.



Rysunek B.5. Umieszczanie pakietów w serwisie nuget.org

Opublikowanego pakietu nie można usunąć z serwera. Można go jednak ukryć z wyników wyszukiwania. Zawsze pozostanie jednak możliwość „ręcznej” instalacji z konsoli.

[7] Można ją wygenerować automatycznie na podstawie komentarzy w kodzie. Pozwala na to rozszerzenie do Visual Studio o nazwie Markdown Editor.

Test pakietu

Aby przetestować nowododany do repozytorium pakiet, stwórzmy nowy projekt aplikacji WPF, najlepiej w nowym rozwiązaniu. Jeżeli pakiet został już zindeksowany możemy uruchomić wyszukiwarkę pakietów NuGet korzystając z polecenia *Manage NuGet Packages...* dostępnego w menu kontekstowym projektu w oknie *Solution Explorer*. Jeżeli nie – z menu *Tools*, *NuGet Package Manager* wybierzmy pozycję *Package Manager Console* i wpiszmy w niej polecenie `Install-Package JacekMatulewski.WpfUtils -Version 1.0.0` (oczywiście nazwa pakietu i numer wersji odnosi się tylko do powyższego przykładu). Korzystając z konsoli zwróćmy uwagę, na rzecz którego projektu wykonywane są polecenia. Decyduje o tym pozycja w rozwijanej liście *Default project* w górnej części podokna konsoli. Po chwili nasz pakiet zostanie zainstalowany w bieżącym projekcie. Możemy to zweryfikować sprawdzając, czy odwołanie do biblioteki znajduje się w gałęzi *References* w podoknie *Solution Explorer*. Od tego momentu możemy korzystać z klas zdefiniowanych w bibliotece np. tworząc instancję klasy `Font` lub wykorzystując znaczniki okien dialogowych.

Zależności między pakietami

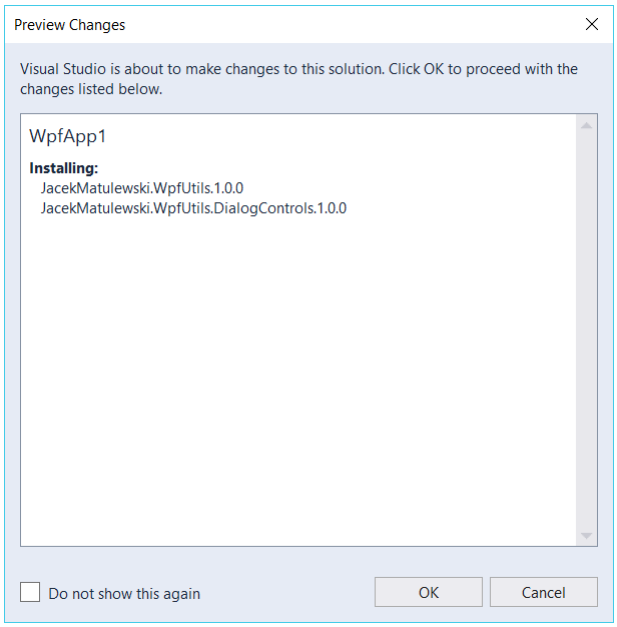
Biblioteka *JacekMatulewski.WpfUtils* nie zależy od żadnych bibliotek spoza platformy .NET. Dlatego po umieszczeniu jej w pakiecie NuGet, nie trzeba określać żadnych zależności względem innych pakietów. Inaczej będzie w przypadku biblioteki *JacekMatulewski.WpfUtils.DialogControls*, która odwołuje się do klas z biblioteki *JacekMatulewski.WpfUtils*.

Zacznijmy od stworzenia dla tego projektu pliku *.nuspec*. Służy do tego polecenie `nuget spec JacekMatulewski.WpfUtils.DialogControls.csproj`. Następnie plik ten należy uzupełnić, podobnie jak miało to miejsce w przykładzie opisanym powyżej (listing B.2). Dodatkowo, aby określić zależności między pakietami, do znacznika `metadata` wstawiamy znacznik `dependencies`, w którym w znaczniku `dependency` wskazujemy identyfikatory pakietów od których zależy bieżący projekt. Żadne zależności nie zostaną dopisane automatycznie, nawet jeżeli projekt odwołuje się do innych bibliotek z tego samego rozwiązania. Następnie tworzymy pakiet poleceniem `nuget pack JacekMatulewski.WpfUtils.DialogControls.csproj`.

Listing B.2. Uzupełniony plik opisu pakietu

```
<?xml version="1.0"?>
<package >
  <metadata>
    <id>$id$</id>
    <version>$version$</version>
    <title>$title$</title>
    <authors>Jacek Matulewski</authors>
    <owners>$author$</owners>
    <projectUrl>http://www.fizyka.umk.pl/~jacek</projectUrl>
    <iconUrl>http://www.fizyka.umk.pl/~jacek/index.ico</iconUrl>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>Some useful code from the book by Jacek Matulewski</description>
    <copyright>Copyright Jacek Matulewski 2018</copyright>
    <tags>WPF Utils Dialog Controls</tags>
    <dependencies>
      <dependency id="JacekMatulewski.WpfUtils" version="1.0.0" />
    </dependencies>
  </metadata>
</package>
```

Dzięki tak określonym zależnościom, podczas instalacji pakietu *JacekMatulewski.WpfUtils.DialogControls*, automatycznie zainstalowany zostanie także pakiet *JacekMatulewski.WpfUtils* (rysunek B.6).



Rysunek B.6. Informacja o instalowanych pakietach przez NuGet Package Manager

Warto zwrócić uwagę na narzędzie *NuGet Package Explorer*, które ułatwia edycję plików *.nuspec*. Dostępne jest na stronie *https://github.com/NuGetPackageExplorer*.

Tworzenie pakietu z zestawem bibliotek

Czy zamiast tworzyć osobną bibliotekę *JacekMatulewski.WpfUtils.DialogControls* zależną od *JacekMatulewski.WpfUtils*, nie można by tej drugiej biblioteki dołączyć do tej pierwszej i dystrybuować obie w jednym pakiecie? Oczywiście, że można, choć nie zawsze to jest dobre rozwiązanie. Pewnie nie warto tworzyć wielkich zbiorów bibliotek, ale zdecydowanie warto łączyć te, które ściśle ze sobą współpracują. Nie można niestety tworzyć pakietów dla całych rozwiązań, co byłoby wygodne, można jednak dodawać biblioteki do jednego pakietu.

Aby przetestować tę możliwość, wróćmy do folderu projektu *JacekMatulewski.WpfUtils* i jeszcze raz zmodyfikujmy plik *.nuspec*. Przede wszystkim, jeżeli chcemy nowy pakiet opublikować, należy zmienić wersję pakietu. A następnie do znacznika `package` (nie do `metadata`) dodajemy znacznik `files` widoczny na przykładzie z listingu B.3. Po instalacji takiego pakietu, w gałęzi *References* zobaczymy obie biblioteki.

Listing B.3. Plik .nuspec z dodatkowymi plikami bibliotek

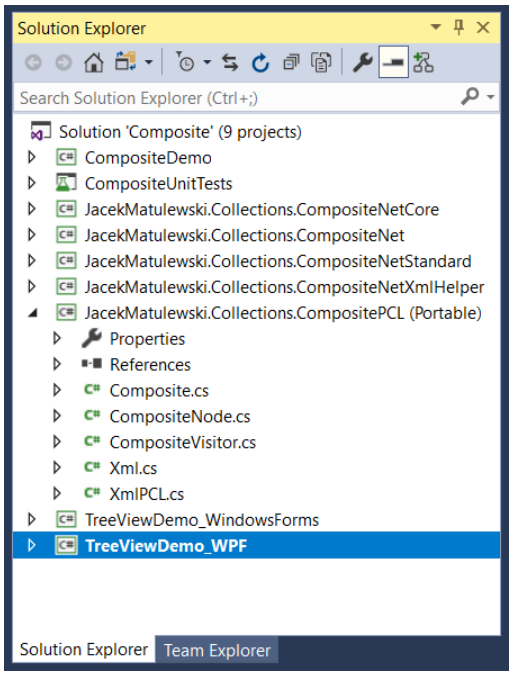
```
<?xml version="1.0"?>
<package >
  <metadata>
    <id>$id$</id>
    <version>1.0.1</version>
    <title>$title$</title>
    <authors>Jacek Matulewski</authors>
    <owners>$author$</owners>
    <projectUrl>http://www.fizyka.umk.pl/~jacek</projectUrl>
    <iconUrl>http://www.fizyka.umk.pl/~jacek/index.ico</iconUrl>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>Some useful code from book by Jacek Matulewski</description>
    <copyright>Copyright Jacek Matulewski 2018</copyright>
```

```
    <tags>WPF Utils</tags>
  </metadata>
  <files>
    <file
src="..\JacekMatulewski.WpfUtils.DialogControls\bin\Release\JacekMatulewski.WpfUtils.Dia
logControls.dll" target="lib\net45" />
  </files>
</package>
```

Pakiet dla wielu platform

W obecnej wersji pakietu, w jego podkatalogu *lib* znajduje się tylko jeden katalog o nazwie *net45*, co oznacza że pakiet zawiera skompilowane pliki przeznaczone dla platformy .NET 4.5. Możemy jednak przygotować i umieścić w jednym pakiecie biblioteki dla kilku platform. Podczas instalacji pakietu w projekcie wyodrębniane są tylko te części pakietu, które są zgodne z typem projektu i tylko one trafiają do katalogu z wynikiem kompilacji.

Jako przykładu użyję zbioru projektów implementujących wzorce projektowe *Kompozyt* (drzewo z cyklami) i *Odwiedzający*. Bazą dla nich jest biblioteka PCL, która przeznaczona jest dla platformy .NET 4.5, ASP.NET Core 1.0, Windows 8 (i automatycznie Windows 10) oraz różnych odmian Xamarin. Oprócz tego stworzyłem osobne projekty, które przeznaczone są dla platformy .NET 3.5 Client Profile, .NET Core 2.0 oraz .NET Standard 2.0 (rysunek B.6), ale w nich niemal wszystkie pliki są „podlinkowane" do plików z projektu PCL. Wszystkie te cztery projekty dzielą wspólną przestrzeń nazw *JacekMatulewski.Collections* i w efekcie ich kompilacji powstają pliki o takiej samej nazwie *JacekMatulewski.Collections.Composite.dll*. Poza tym jest jeszcze projekt *JacekMatulewski.CompositeNetXmlHelper* przeznaczony dla platformy .NET 4.5, który zawiera metody rozszerzające, które nie mogły być umieszczone w projekcie PCL (związane z zapisem kolekcji do pliku).



Rysunek B.6. Okno rozwiązania

Spróbujmy utworzyć pakiet NuGet o nazwie *JacekMatulewski.Collections.Composite*, który zawiera zarówno bibliotekę PCL, jak również jej wersję dla platformy .NET 3.5[8]. Sporym problemem jest znalezienie właściwych nazw podkatalogów katalogu *lib* prawidłowo identyfikujących poszczególne platformy. Problem ten szczególnie

[8] Pakietów dla .NET Core i .NET Standard nie zbudujemy tym samym narzędziem. Pierwsza platforma ma własne narzędzie, a druga – wbudowane w Visual Studio (zob. opis wyżej).

dotyczy biblioteki PCL, dlatego to dla tego projektu utworzymy początkowy plik *.nuspec*, dzięki czemu nazwa folderu dla tej biblioteki będzie ustalana automatycznie (właściwa nazwa to *lib\portable45-net45+win8*). Sposób dołączenia do pakietu różnych wersji biblioteki jest zasadniczo taki sam, jak opisany wyżej przepis na umieszczanie w pakiecie wielu bibliotek DLL, tj. należy dodać do pliku *.nuspec* odpowiednie znaczniki `file`. Tym razem różnić je będą nie tylko ścieżki do plików bibliotek DLL, ale także foldery w pakiecie podawane w atrybucie `target`. W przypadku opisywanego projektu zmodyfikowany plik *.nuspec* z dodanymi znacznikami powinien wyglądać tak, jak przykład pokazany na listingu B.4.

Listing B.4. Plik opisujący pakiet z dwoma wersjami biblioteki i biblioteką pomocniczą

```
<?xml version="1.0"?>
<package >
  <metadata>
    <id>JacekMatulewski.Collections.Composite_</id>
    <version>1.0.1</version>
    <title>Jacek</title>
    <authors>Jacek Matulewski</authors>
    <owners>Jacek Matulewski</owners>
    <projectUrl>http://www.fizyka.umk.pl/download/nuget.html</projectUrl>
    <iconUrl>http://www.fizyka.umk.pl/~jacek/index.ico</iconUrl>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>Composite (with cycles) and Visitor design patters implemented together
with wrapper implementing IEnumerable interface</description>
    <releaseNotes></releaseNotes>
    <copyright>(c) Jacek Matulewski 2018</copyright>
    <tags>Composite Tree Cycles Collection Visitor</tags>
  </metadata>
  <files>
    <file src="..\CompositeNet\bin\Release\JacekMatulewski.Collections.Composite.dll"
target="lib\net35-client" />
    <file
src="..\CompositeNetXmlHelper\bin\Release\JacekMatulewski.Collections.CompositeXmlHelper
.dll" target="lib\net45" />
  </files>
</package>
```