

Maciej Grabek

WCF OD PODSTAW

Komunikacja sieciowa nowej generacji

Poznaj i wykorzystaj przyszłość komunikacji internetowej!
Wprowadzenie do Windows Communication Foundation
Konfiguracja usług sieciowych i nowości WCF 4.0
Narzędzia oraz wykorzystanie WCF

Helion 

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

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

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

Redaktor prowadzący: Ewelina Burska

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

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?wfcodp>
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Kody źródłowe wykorzystane w książce można znaleźć pod adresem:
<ftp://ftp.helion.pl/przyklady/wfcodp.zip>

ISBN: 978-83-246-3094-3

Copyright © Helion 2012

Printed in Poland.

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

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

Spis treści

Wstęp	7
Część I Wprowadzenie do Windows Communication Foundation ...	11
Rozdział 1. E = A + B + C = WCF	13
Adres	13
Wiązanie	14
Kontrakt	17
Punkt końcowy	18
Warto zapamiętać	18
Rozdział 2. Definiowanie kontraktu usługi	19
Warto zapamiętać	30
Podsumowanie	31
Rozdział 3. Hostowanie usługi	33
Self hosting	33
Serwis Windows	41
IIS/WAS	47
Wady i zalety	58
Warto zapamiętać	58
Podsumowanie	58
Rozdział 4. Tworzenie klienta	59
Channel Factory	59
Service Reference	61
Wady i zalety	67
Warto zapamiętać	67
Podsumowanie	67
Część II Konfiguracja usług sieciowych	69
Rozdział 5. Podstawowa konfiguracja	71
Wspólna konfiguracja	71
Konfiguracja serwisu	82
Konfiguracja klienta	85
Warto zapamiętać	87
Podsumowanie	88

Rozdział 6. Zaawansowane aspekty konfiguracji	89
<behaviors>	89
Rozszerzenia, czyli <extensions>	90
Podsumowanie	96
Część III Nowości WCF 4.0	97
Rozdział 7. Simplified Configuration	99
Warto zapamiętać	103
Podsumowanie	103
Rozdział 8. Routing service	105
Prosty routing	108
Most	113
Lista backupowa	116
Broadcast	121
Własny filtr	125
Podsumowanie	128
Rozdział 9. Service discovery	129
Połączenia bezpośrednie	133
Połączenia przez proxy	137
Podsumowanie	143
Rozdział 10. Workflow w WCF	145
Projekt usługi	146
Korzystanie z referencji	154
Debugowanie	158
Podsumowanie	159
Część IV Narzędzia	161
Rozdział 11. Visual Studio	163
Rozdział 12. WCF Service Host	171
Rozdział 13. WCF Test Client	175
Rozdział 14. SVC Util	181
Rozdział 15. Microsoft Service Configuration Editor	187
Podsumowanie	191
Część V Wykorzystanie WCF	193
Rozdział 16. Callback contract	195
Podsumowanie	199
Rozdział 17. OData i WCF	201
Model Entity Framework	201
Dowolna kolekcja	209
Klient	211
Podsumowanie	216

Rozdział 18. WCF RIA Services	217
WCF + RIA	217
Walidacja	224
Podsumowanie	226
Rozdział 19. Wydajność	227
Porównanie wydajności wiązań	227
Wybór kodowania	234
Monitorowanie	241
Podsumowanie	243
Podsumowanie	245
Skorowidz	247

Rozdział 4.

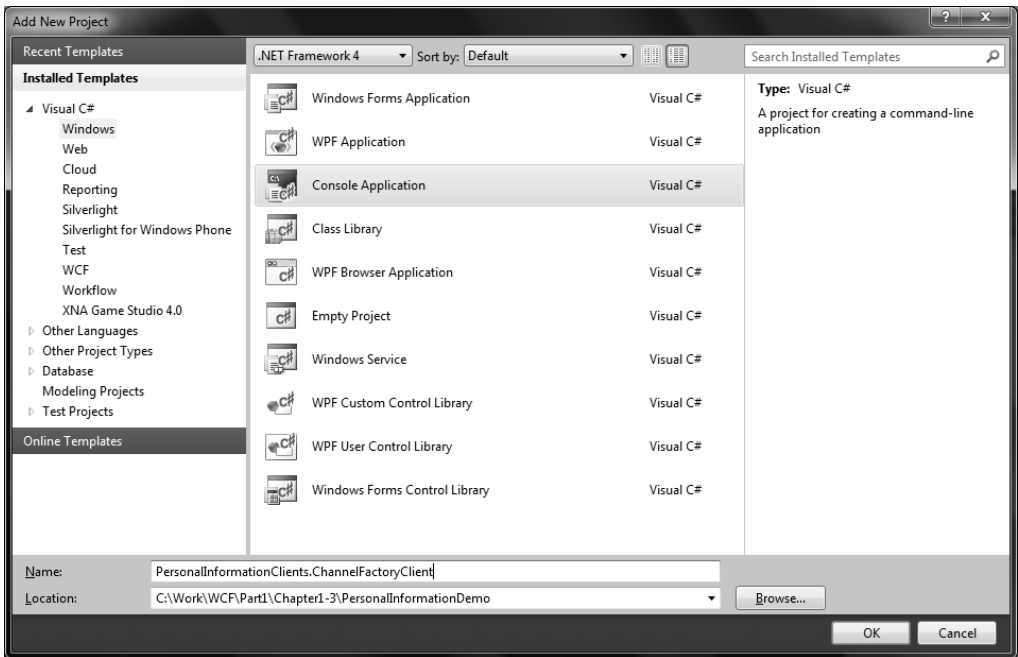
Tworzenie klienta

Trzecim elementem, którym zajmiemy się w tej części książki, jest tworzenie klienta dla naszych usług. Rolę tę pełniło do tej pory narzędzie WCF Test Client, jednakże oczywiste jest, że nie może ono realizować zadań biznesowych stawianych przed systemem, ponieważ służy tylko do testów. Działanie usług bez elementu, który by się do nich odwoływał, mija się z celem, dlatego można pokusić się o stwierdzenie, że aplikacje klienckie są równoprawną częścią wzorca projektowego Service Oriented Architecture. Czym właściwie jest klient usługi? Oczywiście, najczęściej jest to aplikacja, która prezentuje dane użytkownikowi końcowemu. Równie popularnym klientem może być proces działający w naszym systemie, a także odpowiedzialny za cykliczne wywoływanie odpowiednich procesów i zadań biznesowych udostępnianych przez nasze usługi. Klientem może być również inna usługa — mamy wówczas do czynienia z wielowarstwową architekturą, w której usługodawca jest też usługobiorcą. W tym rozdziale prześledzimy proces tworzenia klienta oraz zastanowimy się nad różnymi sposobami podejścia do tego zagadnienia.

Channel Factory

Pierwszym sposobem tworzenia klienta, jakim się zajmiemy, jest wykorzystanie `ChannelFactory<T>`. Klasa ta jest w stanie wygenerować odpowiednie proxy do połączenia z serwisem, jednakże ma pewne ograniczenie. Jest nim konieczność posiadania przez klienta definicji klas opisujących kontrakt usługi. W tworzonym przez nas projekcie taki dostęp jest zapewniony, zatem możemy pokusić się o przetestowanie tego sposobu. W związku z tym do solucji dodamy nowy projekt o nazwie *PersonalInformationClients.ChannelFactoryClient*; skorzystamy z szablonu aplikacji konsolowej, co widać na rysunku 4.1.

Klasa `ChannelFactory` znajduje się w przestrzeni nazw `System.ServiceModel`, zatem do projektu dołączamy taką właśnie referencję. Jednocześnie musimy również umożliwić aplikacji klienckiej dostęp do definicji kontraktu usługi, co wymaga dodania referencji do projektu *PersonalInformationService*. Hostem, do którego będziemy się podłączać,



Rysunek 4.1. Nowy projekt klienta oparty na aplikacji konsolowej

będzie *PersonalInformationHosts.WebHost*. Można też skorzystać z pozostałych projektów hostujących usługę, jednakże ten jest uruchamiany domyślnie przy starcie procesu debugowania tworzonego przez nas klienta, zatem nie będzie trzeba pamiętać o wystartowaniu hosta. Dlaczego tak się dzieje? Visual Studio, w momencie gdy rozpoczynane jest debugowanie, przegląda rozwiązanie w poszukiwaniu projektów, które bazują na serwerze WWW, a następnie uruchamia je za pomocą wbudowanego serwera WWW, co widać w pasku zadań obok zegara. W naszym przypadku dotyczyć to będzie projektu *PersonalInformationHosts.WebHost*.

Korzystanie z *ChannelFactory* jest stosunkowo proste. Jednym z konstruktorów, których możemy użyć, jest konstruktor przyjmujący dwa parametry: typ wiązania oraz adres punktu końcowego. Aby zainicjować te wartości w poprawny sposób, musimy odwołać się do pliku konfiguracyjnego usługi. Widoczny na listingu 3.18 plik konfiguracyjny zawiera informacje o tym, że usługa jest dostępna poprzez wiązanie *WsHttp* ↪ *Binding*. Adres, na jakim jest dostępna, to *http://localhost:9000/PersonalInformationService.svc*. W związku z tym w celu uruchomienia klienta możemy użyć kodu widocznego na listingu 4.1. Kod ten na podstawie znanego nam interfejsu (w tym przypadku jest to *IPersonalInformationService*) tworzy obiekt fabryki (*ChannelFactory*), który inicjuje kanał komunikacji z serwisem. Tak utworzone proxy daje następnie możliwość wywoływania potrzebnych metod udostępnianych przez serwis. Oczywiście, jest to model w pełni obiektowy, dzięki czemu korzystanie z serwisów jest intuicyjne i wygodne.

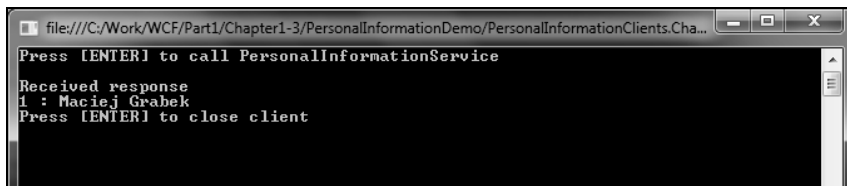
Listing 4.1. Wykorzystanie *ChannelFactory*

```
Uri baseAddr = new Uri("http://localhost:9000/PersonalInformationService.svc/");
ChannelFactory<IPersonalInformationService> factory =
```



```
new ChannelFactory<IPersonalInformationService>(new WSHttpBinding(),
    ↪new EndpointAddress(baseAddr));
IPersonalInformationService proxy = factory.CreateChannel();
var response = proxy.GetPersonalInformation(
    new PersonalInformationService.Messages.PersonalInformationRequest()
    {
        PersonId = 1
    });
```

Po uruchomieniu tak przygotowanego kodu otrzymujemy wynik widoczny na rysunku 4.2.



Rysunek 4.2. Wynik wywołania klienta opartego na *ChannelFactory*

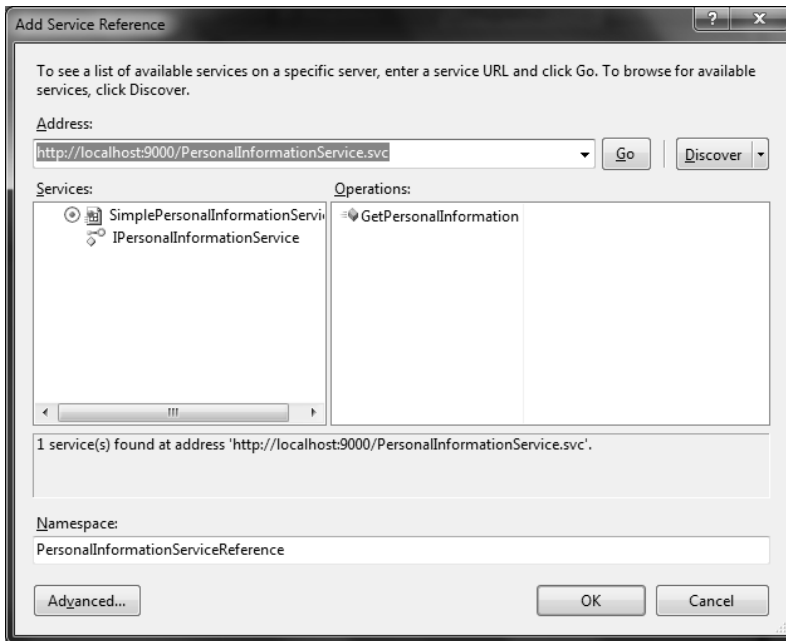
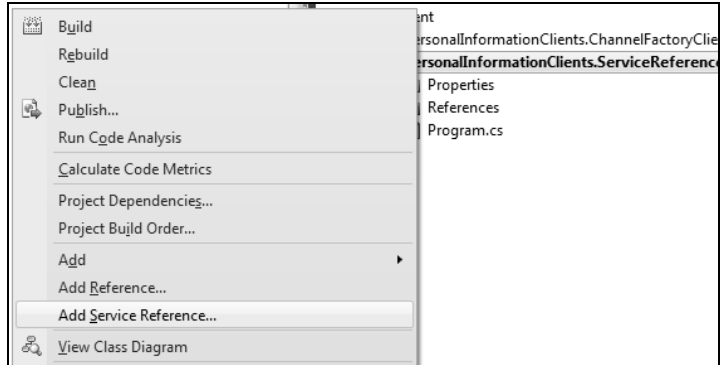
Service Reference

Wykorzystanie *ChannelFactory* do generowania klienta ma jedną wadę: podczas tworzenia projektu niezbędne jest posiadanie biblioteki zawierającej kontrakt. Taka sytuacja z przyczyn oczywistych nie zawsze jest możliwa. Wyobraźmy sobie, że rozważany przez nas serwis jest udostępniany przez firmę trzecią na jej serwerach. W takim przypadku nie ma szansy na uzyskanie dostępu do bibliotek opisujących kontrakty usług, a nawet jeżeli taki dostęp jest możliwy, to z dużymi trudnościami. W związku z tym istnieje druga możliwość utworzenia klienta. Polega ona na skorzystaniu z wbudowanych w Visual Studio narzędzi, które generują klasy niezbędne do komunikacji z usługą na podstawie metadanych. Podobnie jak poprzednio, do solucji dodamy nowy projekt będący aplikacją konsolową, który posłuży do utworzenia klienta w bardziej elastyczny sposób. Nowy projekt nazwiemy *PersonalInformationClients.ServiceReferenceClient*. Aby można było odwołać się do usługi, dodajemy do niej referencję, wybierając w menu kontekstowym projektu opcję *Add Service Reference...*, tak jak widać to na rysunku 4.3.

Po jej wybraniu pojawi się okno widoczne na rysunku 4.4.

W tym momencie mamy dwie możliwości. Pierwsza z nich polega na użyciu przycisku *Discover*, który powoduje przeanalizowanie solucji w poszukiwaniu zdefiniowanych serwisów. Podobnie jak w przypadku *ChannelFactory*, potrzebny jest kod źródłowego usługi, a takiej sytuacji nie rozważamy. W związku z tym posłużymy się drugą opcją, która polega na podaniu znanego nam adresu usługi (*http://localhost:9000/PersonalInformationService.svc*), a następnie wciśnięciu przycisku *Go*. W tym momencie zostaje nawiązane połączenie z usługą oraz pobranie informacji na jej temat poprzez zdefiniowany po stronie serwisu punkt końcowy MEX. Po przetworzeniu tych danych zostają wyświetlone struktura usługi oraz oferowane przez nią kontrakty i metody. Warto

Rysunek 4.3.
Opcja dodawania referencji do usługi



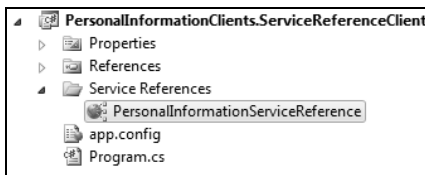
Rysunek 4.4. Okno dodawania referencji

jeszcze wspomnieć, że bez dostępnego punktu końcowego odpowiedzialnego za meta-dane metoda ta nie może zostać zastosowana. W wielu sytuacjach stosuje się wręcz polityki bezpieczeństwa nakazujące usuwanie na serwerach produkcyjnych punktów końcowych tego typu, co w pewnym stopniu umożliwiłoby dodatkowo ich zabezpieczenie przed utworzeniem „niechcianych” aplikacji klienckich. Przed zatwierdzeniem dodawania referencji warto również zmienić domyślną przestrzeń nazw, którą jest `ServiceReference1`, na bardziej znaczącą, np. `PersonalInformationServiceReference`. W wyniku tej operacji do projektu jest dodawany nowy element widoczny na rysunku 4.5.

W pliku konfiguracyjnym `app.config` pojawiły się również wpisy potrzebne do nawiązania komunikacji z usługą. Istotną w tym momencie częścią pliku jest sekcja `client`, która zawiera informacje i konfiguracje punktów końcowych, które są zdefiniowane w serwisie. Przykładowa treść takiej sekcji widoczna jest na listingu 4.2.

Rysunek 4.5.

Referencja do serwisu
dodana do projektu

**Listing 4.2.** Sekcja client w konfiguracji usługi po stronie klienta

```
<client>
  <endpoint address="http://localhost:9000/PersonalInformationService.svc"
    binding="wsHttpBinding" bindingConfiguration="WSHttpBinding_
    ↪IPersonalInformationService"
    contract="PersonalInformationServiceReference.IPersonalInformationService"
    name="WSHttpBinding_IPersonalInformationService">
  </endpoint>
</client>
```

Teraz nic nie stoi na przeszkodzie, aby odwołać się do usługi. W tym celu korzystamy z kodu widocznego na listingu 4.3.

Listing 4.3. Korzystanie z referencji do serwisu

```
using (PersonalInformationServiceClient client = new
PersonalInformationServiceClient())
{
    var response = client.GetPersonalInformation(
        new PersonalInformationRequest()
        {
            PersonId = 1
        });

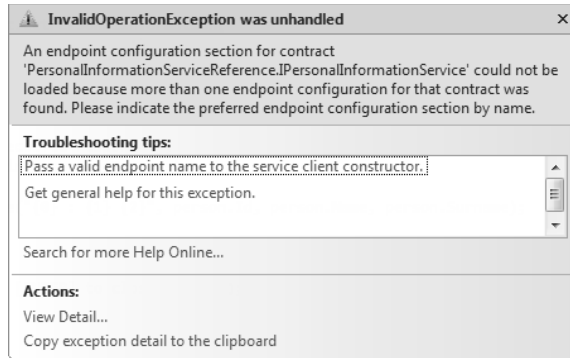
    Console.WriteLine("Received response");
    foreach (var person in response.Persons)
    {
        Console.WriteLine("{0} : {1} {2}", person.Id, person.Name, person.Surname);
    }
}
```

Klasą, która w tym przypadku służy do komunikacji, jest `PersonalInformationService ↪Client`. Wykorzystanie jej wewnątrz klauzuli `using` daje pewność, że na koniec pracy z usługą nawiązane połączenie zostanie poprawnie zamknięte. Wynik działania aplikacji klienckiej jest w tym momencie dokładnie taki sam jak na rysunku 4.2, zmienia się tylko sposób działania, a nie sposób prezentacji wyników.

Gdy określono więcej niż jeden punkt końcowy usługi, z której będziemy korzystać, należy podać jego nazwę jako parametr konstruktora klasy klienta. Inaczej próba uruchomienia tworzonego projektu skończy się komunikatem o błędzie działania widocznym na rysunku 4.6.

Aby wygenerować taki błąd, wystarczy dodać do serwisu nowy punkt końcowy widoczny na listingu 4.4, a do konfiguracji klienta dołączyć zawartość listingu 4.5.

Rysunek 4.6.
*Błąd niezdefiniowanej
nazwy konfiguracji
punktu końcowego*



Listing 4.4. *Dodatkowy punkt końcowy po stronie serwera usługi*

```
<endpoint address="basic" binding="basicHttpBinding" contract=
↳ "PersonalInformationService.ServiceContracts.IPersonalInformationService" />
```

Listing 4.5. *Dodatkowy punkt końcowy po stronie klienta usługi*

```
<endpoint address="http://localhost:9000/PersonalInformationService.svc/basic"
↳ binding="basicHttpBinding" contract="PersonalInformationServiceReference.
↳ IPersonalInformationService" name="BasicHttpBinding_IPersonalInformationService">
↳ </endpoint>
```

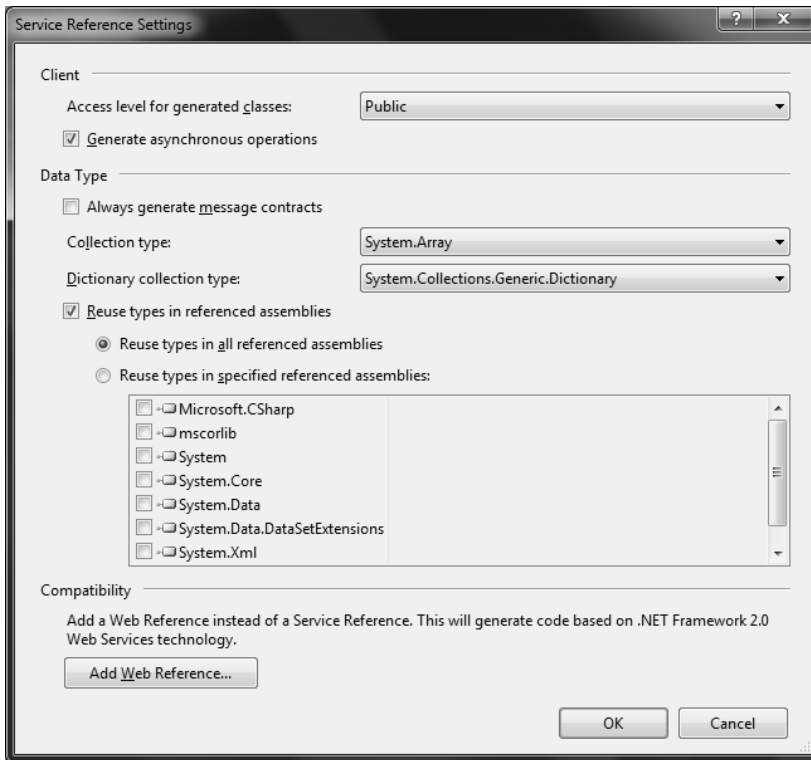
Aby po takich modyfikacjach poprawnie uruchomić klienta usługi, należy w konstruktorze wygenerowanej klasy `PersonalInformationServiceClient` podać nazwę konfiguracji, z której będziemy korzystać. W tym przypadku mamy dwie możliwości: `WSHttpBinding_IPersonalInformationService` lub `BasicHttpBinding_IPersonalInformationService`. Po zastąpieniu konstruktora bezparametrowego wywołaniem widocznym na listingu 4.6 uzyskujemy poprawne działanie aplikacji klienckiej.

Listing 4.6. *Konstruktor proxy wykorzystujący parametr nazwy konfiguracji punktu końcowego*

```
new PersonalInformationServiceClient("BasicHttpBinding_
↳ IPersonalInformationService")
```

Wywołania asynchroniczne

Kolejną zaletą zaprezentowanego podejścia do tworzenia klienta jest fakt, że w prosty sposób generowane proxy może również udostępnić asynchroniczne wywołania metod usługi. Dzięki nim nie będzie konieczności blokowania interfejsu użytkownika, czy też głównego wątku aplikacji, do czasu otrzymania odpowiedzi z serwisu. W celu zaprezentowania wywołań asynchronicznych utworzymy nowy projekt aplikacji konsolowej o nazwie `PersonalInformationClients.AsyncClient`. Podobnie jak poprzednio, dodajemy referencję do serwisu, jednakże w oknie widocznym na rysunku 4.4 wybieramy opcję *Advanced*. W rezultacie otwiera się kolejne okno dialogowe widoczne na rysunku 4.7.



Rysunek 4.7. Okno zaawansowanych ustawień referencji do serwisu

Istotna jest tu opcja *Generate asynchronous operations*. Dzięki niej tworzone automatycznie klasy będą zawierać również logikę niezbędną do asynchronicznego wywoływania metod usługi. Po jej zaznaczeniu i zatwierdzeniu wyboru w projekcie, podobnie jak poprzednio, pojawiają się nowe elementy odpowiedzialne za komunikację z usługą.

Po takim utworzeniu i skonfigurowaniu proxy każda z metod oferowanych przez serwis posiada również swój odpowiednik z postfiksem *Async* oraz zdarzenie **Completed*, gdzie gwiazdka oznacza nazwę metody, która należy do kontraktu. Zdarzenie to pozwala na wpięcie się w moment, gdy aplikacja kliencka otrzyma odpowiedź od serwisu w celu jej dalszego przetworzenia. W związku z tym przed wywołaniem odpowiedniej metody musimy dodać również obsługę takiego zdarzenia. Pełny kod asynchronicznego odwołania do serwisu prezentuję na listingu 4.7.

Listing 4.7. Asynchroniczne wywołanie metody udostępnionej przez serwis

```

Console.WriteLine("Press [ENTER] to call PersonalInformationService");
Console.ReadLine();
using (PersonalInformationServiceClient client = new PersonalInformationService
↳ Client("BasicHttpBinding_IPersonalInformationService"))
{
    client.GetPersonalInformationCompleted += new EventHandler
↳ <GetPersonalInformationCompletedEventArgs>
↳ (client_GetPersonalInformationCompleted);
}

```

```

client.GetPersonalInformationAsync(
    new PersonalInformationRequest()
    {
        PersonId = 1
    });

Console.WriteLine("Press [ENTER] to close client");
Console.ReadLine();

```

W przypadku komunikacji dwukierunkowej (pytanie — odpowiedź, ang. *request* — *reply*) niezbędne może okazać się też przetworzenie odpowiedzi otrzymanej od serwisu. W tym celu należy odpowiednio obsłużyć dane dostępne w metodzie podpiętej pod zdarzenie `Completed`. Niezbędny do tego kod widoczny jest na listingu 4.8.

Listing 4.8. Obsługa odebranej asynchronicznie odpowiedzi

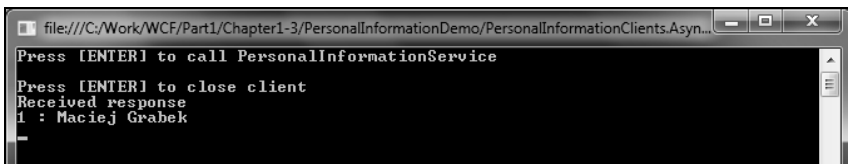
```

static void client_GetPersonalInformationCompleted(object sender,
↳GetPersonalInformationCompletedEventArgs e)
{
    Console.WriteLine("Received response");
    foreach (var person in e.Result.Persons)
    {
        Console.WriteLine("{0} : {1} {2}", person.Id, person.Name, person.Surname);
    }
}

```

Aby uzyskać dostęp do danych zwracanych przez serwis, należy odwołać się do argumentu `e` oraz jego pola `Result`, które przechowuje wartość odpowiedniego typu; w tym przypadku będzie to typ `PersonalInformationResponse`.

Uruchomienie tak przygotowanego kodu generuje wynik widoczny na rysunku 4.8.



Rysunek 4.8. Wynik działania asynchronicznego klienta

Zgodnie z oczekiwaniami główny wątek programu nie został zatrzymany na czas komunikacji z serwisem i dotarł do komunikatu końcowego. Odpowiedź, którą otrzymał klient, została przetworzona i wyświetlona dopiero na samym końcu. Z jednej strony, zapewnia to dużo większą responsywność interfejsu użytkownika, z drugiej jednak, wymaga kilku dodatkowych zabiegów w celu poprawnego obsłużenia i dobrej prezentacji wyników komunikacji z usługą.

Istnieje również możliwość dodania referencji w sposób znany z wcześniejszych odsłon .NET Framework, czyli jako referencji `Web`, jednakże daje ona dużo mniejsze pole do manewru w porównaniu z WCF. Jest to możliwe po wybraniu opcji `Add Web Reference` widocznej na rysunku 4.7.

Wady i zalety

Podstawową wadą ChannelFactory jest konieczność posiadania dostępu do skompilowanej biblioteki zawierającej kontrakt usługi. Powoduje to zmniejszenie elastyczności projektu w systemach opartych na wielu dostawcach. Problem ten znika w przypadku wykorzystania referencji do serwisu. Dodatkową przewagą drugiego rozwiązania jest łatwość dodania metod asynchronicznych do komunikacji z usługą.

Warto zapamiętać

Gdy istnieje kilka punktów końcowych danej usługi, w kodzie aplikacji klienckiej należy użyć konstruktora z parametrem będącym nazwą punktu końcowego wybranego do komunikacji.

Korzystanie z referencji do serwisu daje możliwość prostego wygenerowania metod asynchronicznych odwołujących się do usługi.

Podsumowanie

W rozdziale zademonstrowałem dwa sposoby korzystania z serwisów utworzonych przy użyciu technologii WCF. W zależności od konkretnego zastosowania mamy do wyboru odmienne metody, które dają różne możliwości. Oczywiście, nie są to jedyne metody, bo istnieje możliwość odwołania do usług bez konieczności dodawania jakichkolwiek referencji od nich. Możemy odwołać się do usługi, korzystając z JavaScriptu lub odpowiedniej konstrukcji adresu URL, jak ma to miejsce m.in. w przypadku standardu OData i WCF Data Services. Temat ten podejmę jeszcze w części piątej dotyczącej wykorzystania WCF.

Skorowidz

A

Action, 27, 111, 119
Adobe Flash, 217
adres, 13, 18, 21, 36
 bazowy, 21
 modyfikacja, 39
 określenie
 pliki *.config, 13
 System.Uri, 13
 warstwy transportu, 14
 wyświetlenie, 38
Adventure Works, 201
AdventureWorksDomainService,
 218
AdventureWorksEntities, 204,
 212
AJAX-enabled WCF Service, 167
algorithmSuite, 73
And, 111
AnnouncementEndpoint, 139
app.config, 20, 28, 62, 95
 domyślna zawartość pliku, 21
architektura zorientowana na
 usługi, *Patrz* SOA
ASP.NET Empty Web
 Application, 201
aspnet_regiis.exe, 57
AsQueryable, 211
AsyncPattern, 27
AsyncResult, 139
 implementacja, 139
ATOM, 165
 dane pobrane, 167
atomytrybuty kontraktu
 parametry, 27
AZU, *Patrz* SOA

B

baseAddr, 39, 112
baseAddresses, 40
BasicHttpBinding, 15, 16
BasicHttpContextBinding, 15, 16
behavior, 90, 102
behaviorConfiguration, 107
BehaviorExtensionElement,
 92, 93
behaviors, 89
 clientBehaviors, 89
 serviceBehaviors, 89
bindingConfiguration, 99
BindingElement, 95
bindings, 72, 94, *Patrz także*
 wiązania
 parametry, 73
błędy, 72, 75
 protokołu komunikacji, 80
 przekroczenia czasu
 wywołania żądania, 76
 zerwania połączenia, 77
broadcast
 definicja filtra, 123
 definicja serwisu, 122
 kod wywołania serwisu, 124
 kontrakt usługi, 122
 punkty końcowe, 123
 schemat, 121
 tablica routingu, 123
 wynik działania, 124
Buffered, 73

C

callback contract, 195
 klasa implementująca, 198
 komunikacja z usługą, 198
 treść, 197

uruchomienie, 197
 wynik działania
 hosta, 199
 klienta, 199
 zastosowanie, 197
CallbackContract, 27
ChannelFactory, 59, 67, 109
 wynik wywołania klienta, 61
client, 85
clientBehaviors, 89
clientCredentialType, 73
CodeBehind, 51
COM+, 7
config, 13, 71, 90
ConfigurationManager, 86
ConfigurationName, 27
Console Application, 33
contract, *Patrz* kontrakt
CreateFeed, 165
Custom, 111
CustomBinding, 17, 95

D

DataContract, 23, 26, 30
DataGrid, 213, 220
DataMember, 23, 30
DataServiceCollection, 213
DCOM, 7
DiscoveryClient, 133, 134
DiscoveryEndpoint, 139
DiscoveryProxy, 137
DisplayName, 44
DTO, 24
duplex, 195

E

echo, 108
 definicja, 108
 implementacja, 108

- echo
 - kod hosta, 108
 - kod klienta, 109
 - wynik działania, 110
 - EmitDefaultValue, 27
 - endpoint, *Patrz* punkt końcowy
 - EndpointAddress, 111
 - EndpointAddressPrefix, 111
 - EndpointName, 111
 - Entity Set Name, 205
 - EnumMember, 26, 30
 - Envelope, 119
 - extensions, 90, 93
- F**
- filterData, 111, 125
 - filterTables, 111
 - filterType, 111
 - FindCompleted, 136
 - FindCriteria, 134
 - FindProgressChanged, 136
 - Flash, 217
- G**
- GenderDto, 26
 - GetCallbackChannel, 197
 - GetPersonalInformation, 29
 - GreetingService, wywołanie usługi, 117
- H**
- Header, 119
 - hostowanie, 33
 - IIS, 47
 - samohostowanie, 33
 - serwis Windows, 41
 - uruchomienie, 46
 - usunięcie, 47
 - WAS, 47, 57
 - zatrzymanie, 46
 - HTTP, 7, 8
 - HttpBinding, 81
 - HttpGetEnabled, 37
- I**
- IAsyncResult, 139
 - IBroadcastDemoService, 153
 - IDuplexSessionRouter, 106
 - IFeed, 165
 - IGreetingService, kontrakt, 117
 - IIS, 47
 - IIS Manager, 55
 - tworzenie nowej witryny, 55
 - InitializeService, 204
 - installutil, 44, 46
 - InstallUtil.InstallLog, 45
 - IPersonalInformationService, 24
 - IQueryable, 211
 - IRequestReplyRouter, 106
 - ISampleContractCallback, 198
 - IService, 23
 - IService1, domyślna zawartość pliku, 21
 - IServiceBehavior, 90, 91
 - ISimplexDatagramRouter, 106, 122
 - ISimplexSessionRouter, 106
 - IsInitiating, 27
 - IsOneWay, 27, 122
 - IsReference, 27
 - IsRequired, 27
 - IsTerminating, 27
- J**
- jQuery, 214
 - JSON, 215
- K**
- klient
 - projekt, 60
 - tworzenie, 59
 - kodowanie
 - binarne, 234
 - MTOM, 235
 - tekstowe, 235
 - wydajność, 234
 - komunikacja
 - dwukierunkowa, 196
 - jednokierunkowa, 195
 - poprzez HTTP, 7
 - poprzez TCP/IP, 7, 8
 - żądanie-odpowiedź, 195
 - konfiguracja, 69
 - behaviors, 89
 - bindings, 72
 - parametry, 73
 - extensions, 90
 - klienta, 85
 - rozszerzenia, 90
 - serwisu, 82
 - uproszczona, *Patrz* simplified configuration
 - wiadomości, 78
 - wspólna, 71
 - zaawansowana, 89
- L**
- LINQ, 212
 - lista backupowa
 - błędy, 121
 - konfiguracja, 118
 - schemat działania, 117
 - serwisy zapasowe, 119
 - wpis tablicy routingu, 119
 - wynik działania, 120
 - LoadCompleted, 213
 - LocalService, 44
 - LocalSystem, 44
- M**
- MatchAll, 111
 - maxArrayLength, 73
 - maxBufferPoolSize, 73
 - maxBufferSize, 73
 - maxBytesPerRead, 73
 - maxConcurrentCalls, 90
 - maxConcurrentInstances, 90
 - maxConcurrentSessions, 90
 - maxReceivedMessageSize, 73
 - maxStringContentLength, 73
 - MessageFilter, 125
 - MEX, 37, 43, 61, 100
 - Microsoft Service Configuration Editor, 187
 - Contract Type Browser, 190
 - host, 188
 - konfiguracja, 187
 - narzędzie, 188
 - kontrakt, 190
 - punkt końcowy, 189
 - typy wiązań, 189
 - Microsoft Transaction Server, 7

Model Entity Framework, 201
 aplikacja, 202
 baza danych, 201
 model danych, 202
 zawartość, 203
 serwisy
 dodawanie, 203
 uruchomienie, 204, 205
 zapytanie, 206, 207, 208
 monitorowanie wydajności, 241
 liczniki, 242
 most między protokołami
 konfiguracja, 115
 schemat, 113
 wynik działania, 116
 MSMQ, 8
 MsmqIntegrationBinding, 15, 17
 MTOM, 235
 serializator, 237

N

Name, 27
 Namespace, 27
 narzędzia, 161
 Microsoft Service
 Configuration Editor, 187
 SVC Util, 181
 Visual Studio, 163
 WCF Service Host, 171
 WCF Test Client, 175
 NetMsmqBinding, 15, 17, 81
 NetNamedPipeBinding, 15, 16
 NetPeerTcpBinding, 15, 16
 netsh, 38, 39
 NetTcpBinding, 15, 16, 82, 100
 NetTcpContextBinding, 15, 16
 NetworkService, 44
 nowości WCF 4.0, 97

O

OData, 201
 dowolna kolekcja, 209
 klasy, 209, 210
 serwis, 211
 klient, 211
 .NET, 214
 jQuery, 214
 Silverlight, 211
 Model Entity Framework, 201

odwracanie łańcucha znaków
 host usługi, 114
 implementacja usługi, 114
 kontrakt usługi, 114
 wywołanie usługi, 114
 OnBeginFind, 138
 OnBeginOfflineAnnouncement,
 137
 OnBeginOnlineAnnouncement,
 137
 OnBeginResolve, 138
 OnEndFind, 138
 OnEndOfflineAnnouncement,
 137
 OnEndOnlineAnnouncement, 137
 OnEndResolve, 138
 open/closeTimeout, 73
 OperationContract, 22, 24, 30,
 195
 Order, 27

P

parametry atrybutów kontraktu,
 27
 PersonalInformationRequest,
 25
 PersonalInformationResponse,
 25, 66
 PersonalInformationService, 41,
 51
 właściwości projektu, 23
 zawartość pliku, 50
 PersonalInformationServiceClie
 nt, 63, 64
 PersonalServiceRequest, 149
 PersonDto, 25
 ProcessData, 122
 ProductCategoryValidator, 224
 Program.cs, 34, 38
 ProjectInstaller, 43
 prosty routing
 błędy, 112
 filtr, 110
 typy, 111
 sekcja kliencka, 110
 tablica routingu, 111
 wynik działania, 113
 ProtectionLevel, 27
 przepływ wiadomości, 18
 punkt końcowy, 13, 18, 21, 63, 83
 domyślny, 99, 100
 składowe, 18

R

readerQuotas, 73
 maxArrayLength, 73
 maxBytesPerRead, 73
 maxStringContentLength, 73
 receive/sendTimeout, 73
 ReceiveRequest, 146
 ReplyAction, 27
 request-reply, 150, 195
 RIA Services, 217
 nowy serwis, 219
 serwis domenowy, 219
 walidacja, 224
 klasa, 224
 wynik działania aplikacji,
 222, 223
 XAML, 220, 222
 źródła danych, 220
 routing, 111
 routing service, 105, 128
 broadcast, 121
 konfiguracja, 107
 lista backupowa, 116
 most między protokołami, 113
 prosty routing, 108
 rozdzielanie wiadomości, 105
 schemat działania, 105
 uruchomienie, 106
 RoutingService, 106
 rozszerzenia, 90
 RSS, 165
 dane pobrane, 166
 RunAdditionalHosts.bat, 153
 RunHost, 38
 RunInlineConfiguredHost, 36,
 38, 43

S

samohostowanie, 33
 SampleReferenceServiceLibra
 ry, 157
 SayHello, 119
 security, 73
 algorithmSuite, 73
 clientCredentialType, 73
 SendAndReceiveReply, 148, 150
 SendResponse, 146
 service discovery, 129
 konfiguracja, 133
 połączenia bezpośrednie,
 129, 133
 punkt końcowy, 133
 schemat, 130

service discovery
 połączenia przez proxy,
 129, 137
 modyfikacja hostów, 140
 modyfikacja klienta, 141
 schemat, 131
 uruchomienie proxy, 139
 wynik działania klienta, 142
 wynik uruchomienia
 proxy, 140
 wyrejestrowanie usługi, 142
 Service Reference, 61, 67
 dodawanie referencji, 62
 ustawienia zaawansowane, 65
 wywołania asynchroniczne, 64
 Service1, 23
 serviceBehaviors, 89, 107
 ServiceContract, 22, 24, 30, 149
 ServiceHost, 33
 serviceHostingEnvironment, 84
 ServiceInstaller, 44
 ServiceMetadataBehavior, 37
 ServiceMetadataEndpoint, 37
 ServiceModelReg.exe, 56
 ServiceName, 44
 ServiceProcessInstaller, 44
 ServiceProxy, 182
 services, 82
 ServicesDependedOn, 44
 serviceThrottling, 89, 96
 parametry, 90
 serwer IIS, 47
 działanie usługi, 57
 tworzenie nowej witryny, 56
 serwis Windows, 41
 serwis XML, 7, 8, 14
 SessionMode, 27
 SetEntitySetAccessRule, 204
 Silverlight, 211, 217
 włączenie RIA Services, 218
 Silverlight-enabled WCF
 Service, 167
 SimpleIntroduceService, 126
 SimplePersonalInformationSer
 vice, 27, 35
 simplex, 195
 simplified configuration, 36, 99
 usługa, 102
 SOA, 7, 9, 11, 59
 SOAP, 8, 118
 standardEndpoints, 84
 StartType, 44
 Streamed, 73
 StreamedRequest, 73

StreamedResponse, 73
 svc, 48, 52
 działanie pliku, 52
 SVC Util, 181
 /async, 185
 /config, 181
 /language, 185
 /mergeConfig, 185
 /output, 181
 przełączniki, 181, 185
 uruchomienie, 181
 svcconfigeditor, 190
 svcutil, 181
 Syndication Service Library, 164
 zawartość wygenerowanej
 aplikacji, 165
 System.Runtime.Serialization, 12
 System.ServiceModel, 11, 12,
 23, 33
 Channels, 17
 Configuration, 12
 Description, 12
 IContractBehavior, 90
 IEndpointBehavior, 90
 IOperationBehavior, 90
 IServiceBehavior, 90
 Discovery, 132, 133, 137
 Dispatcher, 125
 DomainServices.Client, 220
 MsmqIntegration, 12
 Routing, 106
 Security, 12
 System.Uri, 13
 szablon WCF Service
 Application, 48, 51
 szablon WEB, 48

T

TCP/IP, 7, 8
 transferMode, 73
 Buffered, 73
 Streamed, 73
 StreamedRequest, 73
 StreamedResponse, 73

U

UdpAnnouncementEndpoint, 140
 UdpDiscoveryEndpoint, 140
 User, 44

V

Visual Studio, 163
 darmowa wersja, 10
 Syndication Service
 Library, 164
 WCF Service Application, 164
 WCF Service Library, 164
 WCF Workflow Service
 Application, 164
 zestaw projektów, 163

W

warstwy transportu, 14
 WAS, 47, 57
 WCF
 kompatybilność, 8
 nowości w wersji 4.0, 97
 przykłady wykorzystania, 193
 WCF Data Service, 167, 201
 WCF Service, 167
 WCF Service Application, 48,
 51, 164
 zawartość wygenerowanej
 aplikacji, 164
 WCF Service Host, 171
 ikona, 172
 konfiguracja argumentów, 173
 okno z uruchomionym
 serwisem, 172
 parametry uruchomienia, 172
 właściwości projektu, 171
 WCF Service Library, 20, 164
 zawartość wygenerowanej
 aplikacji, 164
 WCF Test Client, 175
 dodanie testowanej usługi, 175
 konfiguracja, 178
 okno, 176
 plik konfiguracyjny, 179
 przegląd wiadomości XML,
 178
 uruchomienie, 175
 wywołanie metod
 tablice obiektów, 178
 typy proste, 177
 typy złożone, 177
 WCF Workflow Service
 Application, 146, 164
 zawartość wygenerowanej
 aplikacji, 164
 wcfstestclient, 35, 38, 52, 175

- WEB, 48
 - Web Services, Patrz* serwis XML
 - web.config, 50
 - WebHttpBinding, 15, 16
 - WebMethod, *Patrz*
 - OperationContract
 - WebService, *Patrz*
 - ServiceContract
 - wiadomość
 - przepływ, 18
 - wiązania, 13, 14, 18, 21
 - domyślne, 99
 - oparte na HTTP, 16
 - oparte na MSMQ, 17
 - oparte na TCP, 16
 - schemat wyboru, 234
 - wydajność, 227
 - Windows
 - serwis, 41
 - Windows Workflow
 - Foundation, 145
 - własny filtr
 - definicja, 126
 - host usługi, 126
 - implementacja, 125
 - kontrakt, 126
 - reguły, 126
 - wynik działania, 128
 - workflow, 145, 158
 - debugowanie, 158
 - okno zmiennych, 150
 - parametr wywołania, 149
 - projekt usługi, 146
 - przykład użycia, 146
 - pułapka debugowania, 158
 - referencje, 154, 156
 - sekwencja połączenia z usługą, 148
 - szablon, 146
 - wartość odpowiedzi, 151, 152
 - wygląd procesu
 - biznesowego, 155
 - wynik, 149
 - wywołanie usługi, 147
 - żądanie, 149
 - WorkflowServiceHost, 159
 - WS-Discovery, 129
 - specyfikacja, 129
 - WSDualHttpBinding, 15, 16
 - WSFederationHttpBinding, 15, 16
 - WSHttpBinding, 15, 16
 - WSHttpContextBinding, 15, 16
 - wydajność, 227
 - kodowania, 234
 - zestawienie, 240
 - monitorowanie, 241
 - liczniki, 242
 - wiązań, 227
 - schemat wyboru, 234
 - zestawienie, 234
 - wywołania asynchroniczne, 64
 - wynik działania, 66
- X**
- XML
 - serwis, 7, 8, 14
 - XmlSerializer, 240
 - XPath, 111, 118

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



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

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

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

GRUPA WYDAWNICZA

 **Helion SA**

Windows Communication Foundation to technologia umożliwiająca komunikację pomiędzy praktycznie wszystkimi systemami wymiany danych, przeznaczona przede wszystkim dla tych użytkowników, których główną potrzebą jest zachowanie dużej elastyczności usług sieciowych. Technologia ta jest nadrzędna wobec wszystkich wcześniejszych rozwiązań Microsoftu w tej dziedzinie, zapewnia zgodność z innymi standardami, pozwala też używać różnych protokołów (m.in. HTTP, TCP) bez konieczności ingerowania w sam kod aplikacji. Jeśli chcesz uniknąć problemów związanych z koniecznością ciągłego dostosowywania się do wymagań klientów bądź współpracowników używających różnych kanałów komunikacji sieciowej, pora zapoznać się z możliwościami oferowanymi przez technologię WCF.

Książka WCF od podstaw. Komunikacja sieciowa nowej generacji składa się z pięciu części zawierających najważniejsze zagadnienia związane z technologią WCF. Znajdziesz w niej informacje na temat samej technologii i jej elementów oraz poczytasz o rozmaitych aspektach i wariantach konfiguracji. Dowiesz się, jakie nowości oferuje WCF 4.0, a także poznasz narzędzia używane podczas pracy z tą technologią, w tym Visual Studio, WCF Service Host i WCF Test Client. Ostatnim, choć wcale nie najmniej ważnym omówionym obszarem będzie kwestia wykorzystania serwisów WCF w kontekście standardu OData, biblioteki jQuery i aplikacji Silverlight. Lektura tej pozycji uświadomi Ci, jak wiele tracisz, nie używając na co dzień technologii WCF, a przede wszystkim, jak wiele możesz zyskać!

- Czym jest WCF i dlaczego warto stosować SOA?
- Definiowanie kontraktu usługi i hostowanie usługi
- Tworzenie klienta
- Podstawowa i zaawansowana konfiguracja
- Routing Service, Service Discovery, Workflow w WCF
- Visual Studio, WCF Service Host, WCF Test Client
- SVC Util, Microsoft Service Configuration Editor
- OData i WCF, WCF RIA Services
- Wydajność

Twórz przejrzysty kod, wykorzystując nowoczesne, elastyczne i wygodne w użyciu rozwiązanie WCF!

helion.pl
księgarnia internetowa

Nr katalogowy: 6819



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



Zamówienia telefoniczne:
0 801 339900



0 601 339900



Helion

Sprawdź najnowsze promocje:

- <http://helion.pl/promocje>
 - <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>

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

ISBN: 978-83-246-3094-3



Cena: 49,00 zł

Informatyka w najlepszym wydaniu