

Sławomir Orłowski

Maciej Grabek

C#

Tworzenie aplikacji sieciowych

GOTOWE PROJEKTY

Wykorzystaj rewolucję sieciową i twórz nowatorskie aplikacje!

Helion



Autorstwo: Sławomir Orłowski (wstęp, rozdziały 1-8), Maciej Grabek (rozdział 9).

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: Jan Paluch

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock.

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

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie?cshta2>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-246-2910-7

Copyright © Helion 2012

Printed in Poland.

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

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

Spis treści

Wstęp	7
Rozdział 1. Język C# i platforma .NET	9
Technologia .NET. Krótki wstęp	9
Elementy języka C# i programowanie zorientowane obiektowo	11
Przestrzenie nazw	15
Kolekcje	16
Zdarzenia i metody zdarzeniowe	17
Delegacje	17
Wyjątki	17
Interfejsy	19
Rozdział 2. Visual C# 2010 Express Edition. Opis środowiska	21
Projekt 1. Budujemy interfejs pierwszej aplikacji. Projekt Windows Forms	22
Projekt 2. Poznajemy pliki projektu pierwszej aplikacji	25
Projekt 3. Interakcja aplikacji z użytkownikiem. Metody zdarzeniowe	29
Rozdział 3. Visual Web Developer 2010 Express Edition. Opis środowiska	33
Projekt 4. Pierwsza strona ASP.NET. Tworzymy interfejs	33
Projekt 5. Pierwsza strona ASP.NET. Poznajemy pliki projektu	37
Projekt 6. Pierwsza strona ASP.NET. Metody zdarzeniowe	40
Rozdział 4. Programowanie sieciowe	43
Sieci komputerowe	43
Protokoły TCP i UDP	46
Protokół IP i adresy MAC	48
Programowanie klient-serwer i peer-to-peer	49
Popularne protokoły sieciowe	50
Protokół ICMP	50
Protokół HTTP	51
Protokół FTP	51
Protokół POP3	52
Rozdział 5. Aplikacje TCP i UDP	53
Projekt 7. Połączenie TCP. Klient	53
Projekt 8. Połączenie TCP. Serwer	56
Projekt 9. Odczytanie adresu IP przyłączonego hosta	60
Projekt 10. Połączenie UDP. Klient	61

Projekt 11. Połączenie UDP. Serwer	62
Projekt 12. Asynchroniczne połączenie TCP	64
Projekt 13. Prosty skaner otwartych portów hosta zdalnego	67
Projekt 14. Skaner otwartych portów lokalnego hosta	68
Projekt 15. Sprawdzenie adresu IP naszego komputera	69
Projekt 16. Komplet informacji na temat połączeń sieciowych	72
Projekt 17. Ping	74
Projekt 18. Ping. Przeciwdziałanie zablokowaniu interfejsu	77
Projekt 19. NetDetect. Sprawdzanie dostępnych komputerów w sieci	79
Projekt 20. Traceroute. Śledzenie drogi pakietu ICMP	81
Projekt 21. Protokół HTTP. Sprawdzanie dostępnych uaktualnień	85
Projekt 22. Pobieranie pliku z użyciem protokołu HTTP	86
Projekt 23. Pobranie źródła strony z serwera WWW	88
Projekt 24. Przeglądarka WWW	89
Projekt 25. Edytor HTML. Budowanie interfejsu	91
Projekt 26. Edytor HTML. Obsługa plików tekstowych	92
Projekt 27. Edytor HTML. Współpraca ze schowkiem	95
Projekt 28. Edytor HTML. Wprowadzanie tagów	95
Projekt 29. Edytor HTML. Podgląd bieżącej strony	98
Projekt 30. Wysyłanie wiadomości e-mail bez uwierzytelniania	99
Projekt 31. Wysyłanie sformatowanej wiadomości e-mail z załącznikami	102
Projekt 32. Wysyłanie poczty za pomocą serwera wymagającego uwierzytelnienia	105
Projekt 33. Masowe wysyłanie wiadomości e-mail	106
Projekt 34. Klient FTP. Interfejs aplikacji	110
Projekt 35. Klient FTP. Definiowanie pól i własności klasy FTPClient	112
Projekt 36. Klient FTP. Listowanie katalogów serwera FTP	116
Projekt 37. Klient FTP. Zmiana katalogu	119
Projekt 38. Klient FTP. Metoda pobierająca plik asynchronicznie	122
Projekt 39. Klient FTP. Wywołanie metody pobierającej plik asynchronicznie	125
Projekt 40. Klient FTP. Metoda wysyłająca plik asynchronicznie	127
Projekt 41. Klient FTP. Wywołanie metody wysyłającej plik asynchronicznie	129
Projekt 42. Klient FTP. Kasowanie pliku	131
Projekt 43. Menedżer pobierania plików w tle. Budowa interfejsu	133
Projekt 44. Menedżer pobierania plików w tle. Pobieranie pliku	135
Projekt 45. Menedżer pobierania plików w tle. Przerwanie pobierania pliku	137
Projekt 46. Serwer Uśmiechu. Budowa interfejsu	138
Projekt 47. Serwer Uśmiechu. Lista kontaktów	140
Projekt 48. Serwer Uśmiechu. Wysyłanie danych do wielu odbiorców	143
Projekt 49. Klient Uśmiechu. Umieszczenie ikony w zasobniku systemowym	144
Projekt 50. Klient Uśmiechu. Oczekiwanie na połączenie w osobnym wątku	147
Projekt 51. Klient Uśmiechu. Bezpieczne odwoływanie się do własności kontrolek formy z poziomu innego wątku	149
Projekt 52. Komunikator. Serwer. Budowa interfejsu	150
Projekt 53. Komunikator. Serwer. Bezpieczne odwoływanie się do własności kontrolek formy z poziomu innego wątku	153
Projekt 54. Komunikator. Serwer. Obsługa rozmowy	154
Projekt 55. Komunikator. Klient	159
Projekt 56. Zdalny screenshot. Klient. Zrzut ekranu	162
Projekt 57. Zdalny screenshot. Klient	162
Projekt 58. Klient. Wysyłanie informacji o dostępności klienta	165
Projekt 59. Serwer screenshot. Budowa interfejsu	166
Projekt 60. Serwer screenshot. Bezpieczne odwoływanie się do własności kontrolek formy z poziomu innego wątku	167
Projekt 61. Serwer screenshot. Lista aktywnych klientów	168

Projekt 62. Serwer screenshot. Pobranie zrzutu ekranu	169
Projekt 63. Serwer Czat. Budowanie interfejsu	171
Projekt 64. Serwer Czat. Bezpieczne odwoływanie się do własności kontrolek formy z poziomu innego wątku	173
Projekt 65. Serwer Czat. Klasa formy oraz pętla główna programu	174
Projekt 66. Serwer Czat. Obsługa wątków związanych z klientami	179
Projekt 67. Serwer Czat. Rozłączenie klienta	180
Projekt 68. Czat. Klient	181
Rozdział 6. Remoting	187
Projekt 69. Serwer HTTP	188
Projekt 70. Klient HTTP	193
Projekt 71. Serwer TCP	195
Projekt 72. Klient TCP	197
Projekt 73. Serwer TCP. Plik konfiguracyjny	199
Projekt 74. Klient TCP. Plik konfiguracyjny	202
Projekt 75. Czat. Klasa serwera	203
Projekt 76. Czat. Serwer	205
Projekt 77. Czat. Klient	206
Rozdział 7. ASP.NET i ADO.NET	211
Projekt 78. Pozycjonowanie kontrolek na stronie	212
Projekt 79. Ping	217
Projekt 80. Wysyłanie wiadomości e-mail	218
Projekt 81. Pobieranie plików na serwer	220
Projekt 82. Wysyłanie wiadomości e-mail z załącznikami	221
Projekt 83. Księga gości. Współpraca z plikiem XML	222
Projekt 84. Księga gości. Wyświetlanie zawartości pliku XML	226
Projekt 85. Księga gości. Sprawdzanie poprawności wpisywanych danych	228
Projekt 86. Księga gości. Liczba gości online	230
Projekt 87. Wielojęzyczny serwis internetowy. Zasoby lokalne	232
Projekt 88. Wielojęzyczny serwis internetowy. Zasoby globalne	237
Projekt 89. Wielojęzyczny serwis internetowy. Wybór języka przez użytkownika	239
Projekt 90. Identyfikacja użytkowników	241
Projekt 91. Rejestrowanie nowych użytkowników	245
Projekt 92. Identyfikacja użytkowników, część II	246
Projekt 93. Baza książek. Stworzenie bazy danych	247
Projekt 94. Baza książek. Przyłączenie się do bazy danych	250
Projekt 95. Baza książek. Prezentacja danych	251
Rozdział 8. Web Services	255
Projekt 96. Pierwsza usługa sieciowa	256
Projekt 97. Korzystanie z usługi sieciowej	259
Projekt 98. Usługa Maps Account Center wyszukiwarki bing. Rejestracja usługi	261
Projekt 99. Bing Maps. Klient	262
Projekt 100. Bing Maps. Modyfikacja klienta	265
Rozdział 9. WCF — ponad transportem	267
Wstęp	267
Podstawy działania	269
WCF = E = A + B + C	269
A jak address	270
B jak binding	270
C jak contract	273
Punkt końcowy	273

Projekt 101. Definiowanie kontraktu 273

Projekt 102. Udostępnianie usługi 280

 Self hosting 281

 IIS 285

 Serwis Windows 287

Projekt 103. Tworzenie klienta 290

 ChannelFactory 290

 Referencja 292

Skorowidz 299

Rozdział 6.

Remoting

Technologia Remoting pozwala na komunikację pomiędzy aplikacjami środowiska .NET niezależnie od tego, czy znajdują się one na tym samym komputerze, czy w sieci. Technologia ta jest składową platformy .NET. Jej odpowiednikiem w języku Java jest technologia RMI (ang. *Remote Method Invocation*). Innym przykładem może być rozwiązanie CORBA (ang. *Common Object Request Broker Architecture*), które zapewnia komunikację między obiektami znajdującymi się w różnych systemach komputerowych. Przodkiem Remotingu była technologia DCOM (ang. *Distributed Component Object Model*), która obecnie nie jest już rozwijana. Dobra definicja .NET Remoting znajduje się na stronach MSDN:

„Microsoft® .NET Remoting zapewnia bogatą i rozszerzalną infrastrukturę komunikacyjną dla obiektów znajdujących się w różnych domenach aplikacji AppDomains, w różnych procesach i na różnych komputerach. .NET Remoting oferuje silny, a jednocześnie prosty model programistyczny i wsparcie w czasie wykonania, co sprawia, że infrastruktura komunikacyjna jest niewidoczna”.

Myślę, że bardzo dobrze oddaje ona ideę tej składowej platformy .NET. Zwykle budowa rozproszonej aplikacji, opartej na technologii Remoting, składa się z trzech faz:

- ◆ Stworzenie obiektu odpowiedzialnego za funkcjonalność rozproszonej aplikacji.
- ◆ Utworzenie serwera, który będzie odpowiadał za rejestrację, udostępnienie oraz zarządzanie zdalnym obiektem.
- ◆ Napisanie aplikacji klienckich korzystających z udostępnionej przez serwer usługi.

Wszystkie projekty zebrane w tym rozdziale będą budowane w trzech opisanych wyżej krokach. Ogromną zaletą technologii Remoting jest to, że nie musimy dbać o połączenia i wymianę danych, tak jak miało to miejsce w projektach z rozdziału 5. Tutaj dzielony obiekt jest automatycznie udostępniany przez serwer. Do dyspozycji mamy trzy rodzaje połączeń: TCP, IPC i HTTP. Pierwsze dwa połączenia przesyłają dane w formacie binarnym. Połączenie HTTP przesyła informacje w formacie tekstowym. Udostępniany obiekt może być przekazywany przez referencję bądź wartość. W przypadku referencji obiekt zdalny istnieje wyłącznie na serwerze. Podczas komunikowania się z klientem otrzymuje on jedynie referencje do obiektu. Aby klasa zdalna przekazywała dane przez

referencję, musi dziedziczyć po klasie `MarshalByRefObject`. Klasa ta zapewnia niezbędne mechanizmy udostępniania. Jeżeli zdecydujemy się na przekazywanie przez wartość, wówczas klient będzie miał pełną wiedzę na temat obiektu zdalnego oraz będzie posiadał kopię przeznaczoną tylko dla niego. W definicji klasy obiektu zdalnego musimy wstawić wtedy atrybut `[Serializable]`. W naszych projektach będziemy używać przekazywania przez referencję. Z racji tego, że technologia Remoting jest częścią składową platformy .NET, może być używana jedynie przez aplikacje stworzone dla tej platformy. Jeżeli chcielibyśmy używać pewnych usług z poziomu różnych systemów komputerowych, wówczas lepszym rozwiązaniem wydają się być usługi sieciowe (ang. *web services*) opisane w rozdziale 8. Technologia .NET Remoting jest obecnie zastąpiona przez *Windows Communication Foundation* (rozdział 9.). Stanowi więc technologię, która powoli schodzi ze sceny. Nie zaleca się używania jej w nowych projektach. Projekty, które już są w użyciu, powinny migrować w stronę WCF. Jeśli to niemożliwe, powinny przynajmniej integrować się z WCF.

W .NET Remoting istnieją trzy podstawowe typy obiektów zdalnych, które pośredniczą w transakcjach pomiędzy klientem a serwerem:

- ◆ **SingleCall.** Obiekt, który obsługuje jedno żądanie. Nie przechowuje on informacji pomiędzy wywołaniami. Nie zapamiętuje więc stanu.
- ◆ **Singleton.** Zgodnie z wzorcem projektowym o nazwie Singleton jest to obiekt wspólny dla wielu klientów. Jedna instancja przechowująca stan dla wszystkich wywołań. Umożliwia jawne współdzielenie danych.
- ◆ **CAO (Client Activated Object).** Obiekt aktywowany na żądanie klienta, działający po stronie serwera. Do komunikacji wykorzystuje klasę pośredniczącą. Zapamiętuje stan dla danego klienta, nie dla wszystkich klientów.

Rozdział ten stanowi jedynie bardzo krótki wstęp, który ma za zadanie zaznajomić Czytelnika z podstawami technologii Remoting. Brak tu takich zagadnień, jak szyfrowanie danych, ukrywanie implementacji obiektu zdalnego przed klientami oraz zarządzanie czasem życia. Myślę jednak, że w opisie podstaw programowania sieciowego dla platformy .NET nie mogło zabraknąć choćby podstawowego opisu technologii Remoting. Istnieje jeszcze wiele projektów, które z niej korzystają. Warto ją więc poznać w celu zakonserwowania dotychczasowych rozwiązań i ich przeniesienia do WCF. Jeśli stoisz przed wyborem: .NET Remoting czy WCF, to należy wybrać WCF.

Projekt 69. Serwer HTTP

Napiszemy teraz serwer oparty na połączeniu HTTP. Jako pierwszą utworzymy klasę odpowiedzialną za usługę zdalną. Nasza usługa będzie polegała na wyświetlaniu odpowiedniego komunikatu w przypadku połączenia się z klientem. Będzie to bardzo prosty projekt, odpowiednik projektu „witaj świecie”. Jego zadaniem jest pokazanie, w jaki sposób należy budować projekty .NET Remoting. Nie będziemy skupiać się w tym miejscu na jakimś skomplikowanym projekcie.

1. Z menu *File* wybieramy opcję *New project...*
2. Zaznaczamy ikonę *Class Library*. W polu *Name* wpisujemy *RemotingExample*. Przypominam, że jest to nazwa przestrzeni nazw, w której znajdą się nasze przykładowe klasy. Do nazywania zmiennych będziemy używać języka angielskiego. Myślę, że jest to dobry nawyk. Warto już teraz przyzwyczajać się do anglojęzycznych nazw zmiennych w programie.
3. Piszemy prostą klasę *RemotingObject* z jedną publiczną metodą, która zwraca pewien tekst. Konstruktor bezparametrowy zostanie pusty. Kod klasy jest pokazany na listingu 6.1. Należy zwrócić uwagę na fakt, że nasza klasa ma dziedziczyć z klasy *MarshalByRefObject*. Dla aplikacji, które obsługują Remoting, oznacza to udostępnienie obiektu tej klasy nie tylko w ramach jednej domeny aplikacji.

Listing 6.1. Klasa odpowiedzialna za usługę

```
using System;
using System.Text;

namespace RemotingExample
{
    public class RemotingObject: MarshalByRefObject
    {
        public RemotingObject() { }

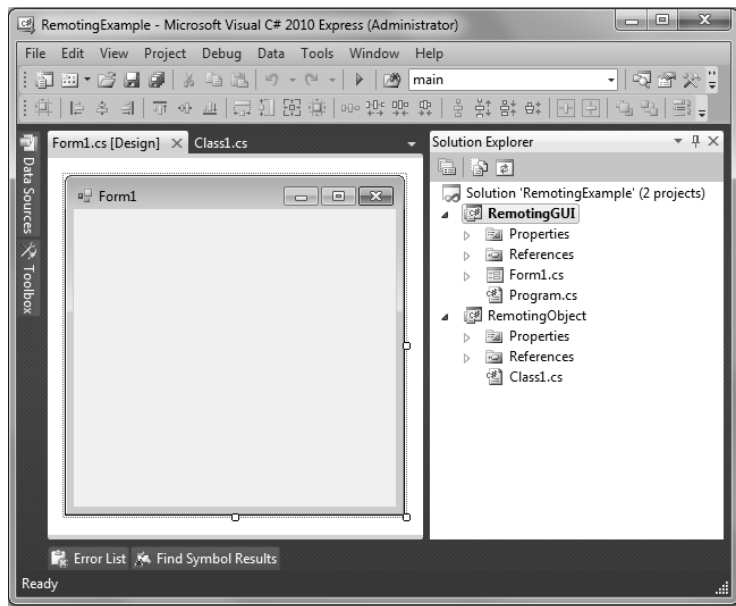
        public string Test()
        {
            return ".NET Remoting test";
        }
    }
}
```

4. Zapisujemy teraz nasz projekt pod nazwą *RemotingObject* oraz kompilujemy go (klawisz *F6*). W wyniku kompilacji otrzymujemy bibliotekę *RemotingObject.dll*.
Naszą klasę nazwaliśmy *RemotingObject*. Dziedziczy ona po klasie *MarshalByRefObject*, co zapewnia przesyłanie do przyłączonych klientów jedynie referencji obiektu *RemotingObject*. Stworzyliśmy pusty konstruktor oraz jedną publiczną, bezparametrową metodę *Test*, zwracającą tekst *.NET Remoting test*. Jak widać, przykład ten nie jest skomplikowany. Przejdźmy teraz do napisania aplikacji serwera, która będzie korzystała ze stworzonej przed chwilą biblioteki.
5. Do naszego rozwiązania dołączymy nowy projekt. Przypominam, że najszybciej można to zrobić, klikając prawym klawiszem w oknie *Solution Explorer* nazwę naszego rozwiązania (ang. *solution*). Z menu podręcznego wybieramy opcję *Add* oraz *New Project...* Rozpoczynamy nowy projekt *Windows Forms Application*. W polu *Name* wpisujemy *RemotingGUI*. Dzięki temu poprzednia biblioteka i nasz projekt będą znajdowały się w tej samej przestrzeni nazw. Oczywiście nie jest to konieczne, ale wygodne.

6. Ustawiamy projekt *RemotingGUI* jako startowy. Aby to zrobić, klikamy prawym klawiszem nazwę projektu, a następnie wybieramy opcję *Set as StartUp Project*. Po tych czynnościach okno Visual Studio powinno przypominać to z rysunku 6.1.

Rysunek 6.1.

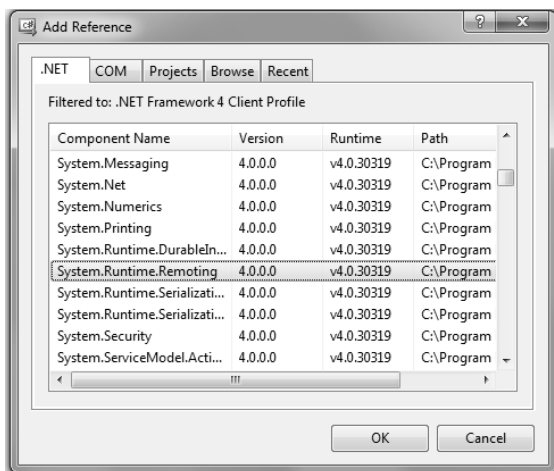
Dwa projekty w ramach jednego rozwiązania



7. Do projektu dodajemy referencję do biblioteki *System.Runtime.Remoting.dll*, która nie jest standardowo zamieszczana w projektach typu *Windows Forms Application*. W tym celu z menu głównego wybieramy opcję *Project*, a następnie klikamy pozycję *Add Reference....* Otworzy się okno z wyborem referencji (rysunek 6.2). Należy pamiętać, aby wybrać zakładkę *.NET*.

Rysunek 6.2.

Okno Add Reference



8. Na zakładce *.NET* wyszukujemy i zaznaczamy komponent o nazwie *System.Runtime.Remoting*. Klikamy przycisk *OK*. W ten sposób referencja została dodana. Aby to sprawdzić, możemy w oknie *Solution Explorer* rozwinąć gałąź *References*. Powinien się tam znajdować wpis *System.Runtime.Remoting*.
9. Przełączmy się do widoku kodu projektu *RemotingGUI* (klawisz *F7*). W sekcji odpowiedzialnej za użyte w projekcie przestrzenie nazw dodajemy nowe wpisy:

```
using System.Runtime.Remoting;  
using System.Runtime.Remoting.Channels;  
using System.Runtime.Remoting.Channels.Http;
```
10. W oknie widoku projektu na formę wrzucamy kontrolkę *button1*. Jej własność *Text* ustawiamy na *Start*.
11. Do projektu dodajemy kontrolkę *textBox1*. Własność *ReadOnly* zmieniamy na *true*.
12. Musimy dodać jeszcze referencje do stworzonej wcześniej biblioteki *RemotingObject.dll*. Robimy to podobnie jak w przypadku dodawania referencji *System.Runtime.Remoting*. Po otwarciu okna *Add Reference* zmieniamy zakładkę na *Projects* i wybieramy projekt *RemotingObject*. Klikamy przycisk *OK*. Biblioteka została dodana do referencji naszego projektu.
13. Serwer powinien wystartować w momencie kliknięcia przycisku *Start*. Wobec tego piszemy metodę zdarzeniową *Click* (listing 6.2).

Listing 6.2. *Metoda uruchamiająca serwer*

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Windows.Forms;  
using System.Runtime.Remoting;  
using System.Runtime.Remoting.Channels;  
using System.Runtime.Remoting.Channels.Http;  
  
namespace RemotingExample  
{  
    public partial class Form1 : Form  
    {  
        public Form1()  
        {  
            InitializeComponent();  
        }  
  
        private void button1_Click(object sender, EventArgs e)  
        {  
            try  
            {  
                HttpChannel channel = new HttpChannel(25000);  
                ChannelServices.RegisterChannel(channel, false);  
            }  
        }  
    }  
}
```

```

RemotingConfiguration.RegisterWellKnownServiceType(typeof(RemotingObject),
"Remoting", WellKnownObjectMode.SingleCall);
        textBox1.Text = "Serwer oczekuje .. ";
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Błąd");
    }
}
}
}
}

```

Dla naszej aplikacji rezerwujemy port **25000**, na którym będzie ona nasłuchiwać wywołań opartych na protokole HTTP. Używając metody `RegisterChannel` klasy `ChannelServices`, rejestrujemy nasz kanał HTTP. Pierwszym argumentem jest kanał. Drugi argument odpowiada za zabezpieczenie (`true` — zabezpieczony, `false` — niezabezpieczony). Jeżeli spróbowałibyśmy jeszcze raz zarejestrować ten kanał, zostałby zgłoszony wyjątek informujący nas, że kanał już jest zarejestrowany. Dalej korzystamy z metody `RegisterWellKnownServiceType` klasy `RemotingConfiguration`. Metoda ta rejestruje nasz serwis w usłudze *Remoting*. Jej pierwszym argumentem jest typ klasy, która będzie dostępna jako usługa. Drugi argument to nazwa `Uri`, czyli unikalna nazwa łącząca usługę z obiektem. Celowo użyliśmy nazwy *Remoting*, aby pokazać, iż klient wcale nie musi znać nazwy klasy odpowiedzialnej za usługę. Wystarczy, że zna adres i port serwera oraz nazwę samej usługi. W ten sposób możemy również udostępniać wiele usług na jednym porcie. Usługi te muszą mieć inne `Uri`. Ostatni parametr mówi, w jaki sposób obiekt ma być tworzony po stronie serwera. Mamy do dyspozycji dwie opcje: `SingleCall` i `Singleton`. Jeżeli wybieramy pierwszą, wówczas każda przychodząca wiadomość jest obsługiwana przez nową instancję obiektu. W drugim przypadku wszystkie przychodzące wiadomości są obsługiwane przez tę samą instancję obiektu. Całość jest zamknięta w bloku ochronnym `try/catch`, ponieważ metody tu użyte mogą generować wyjątki. Serwer będzie działał, dopóki nie wyłączymy aplikacji.

14. Uruchamiamy teraz nasz projekt. Włączamy serwer. Musimy pamiętać o tym, aby ustawić odpowiednio firewall.
15. Sprawdzimy teraz, czy nasz serwer rzeczywiście działa. Otwieramy przeglądarkę internetową i jeżeli nie jesteśmy podłączeni do sieci, wpisujemy w niej adres `http://localhost:25000/Remoting?wsdl`. W przypadku gdy nasz komputer posiada adres IP, zamiast `localhost` wpisujemy ten adres. Powinna ukazać się strona XML z informacjami o naszym obiekcie *RemotingObject*. Język WSDL (ang. *Web Services Description Language*) opisuje usługi sieciowe. Odpowiedź w przypadku lokalnego hosta zawiera listing 6.3.

Listing 6.3. Odpowiedź serwera

```

<definitions name="RemotingObject"
targetNamespace="http://schemas.microsoft.com/clr/nsassem/RemotingExample/RemotingO
bject%2C%20Version%3D1.0.0.0%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull">

```

```

<message name="RemotingObject.TestInput"></message>

<message name="RemotingObject.TestOutput"><part name="return"
type="xsd:string"/></message>

<portType name="RemotingObjectPortType"><operation name="Test"><input
name="TestRequest" message="tns:RemotingObject.TestInput"/><output
name="TestResponse"
message="tns:RemotingObject.TestOutput"/></operation></portType>

<binding name="RemotingObjectBinding"
type="tns:RemotingObjectPortType"><soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/><suds:class
type="ns0:RemotingObject" rootType="MarshalByRefObject">
  </suds:class><operation name="Test"><soap:operation
soapAction="http://schemas.microsoft.com/clr/nsassem/RemotingExample.RemotingObject
/RemotingObject#Test"/><suds:method attributes="public"/><input
name="TestRequest"><soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://schemas.microsoft.com/clr/nsassem/RemotingExample.RemotingObject/
RemotingObject"/></input><output name="TestResponse">
<soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://schemas.microsoft.com/clr/nsassem/RemotingExample.RemotingObject/
RemotingObject"/></output></operation>
</binding>

<service name="RemotingObjectService"><port name="RemotingObjectPort"
binding="tns:RemotingObjectBinding"><soap:address
location="http://localhost:25000/Remoting"/></port></service></definitions>

```

W odpowiedzi opisane są wszystkie szczegóły naszego serwisu .NET Remoting. Na listingu 6.3 wyróżniono kilka najważniejszych.

Projekt 70. Klient HTTP

Nasz klient będzie miał za zadanie połączyć się z usługą (z serwerem), uruchomić metodę i wyświetlić jej wyniki.

1. Tworzymy nowy projekt *Windows Forms*. W polu *Name* wpisujemy *RemotingClient*.
2. Podobnie jak w poprzednim projekcie tutaj także dodajemy referencje do bibliotek *RemotingObject.dll* oraz *System.Runtime.Remoting.dll*. W tym celu można użyć zakładki *Recent* z okna *Add Reference*, gdzie powinny się już znajdować wcześniej dodane referencje.
3. Zapisujemy projekt pod nazwą *RemotingClient*. Zmieniamy przestrzeń nazw projektu na *RemotingExample*. Można to zrobić np. za pomocą narzędzia *Refactor*.
4. W podglądzie kodu dodajemy odpowiednie przestrzenie nazw:

```
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
```

5. Na formę wrzucamy pole edycyjne `textBox1`. Tutaj będziemy wprowadzać adres serwera.
6. Do projektu dodajemy kontrolkę `numericUpDown1`— będzie ona przechowywała numer portu, na którym nasłuchuje serwer.
7. Formę uzupełniamy o kontrolkę `listBox1`. Będziemy tu wypisywać wszystkie komunikaty.
8. Na formę wrzucamy przycisk `button1`. Właśność `Text` zmieniamy na *Połącz*.
9. Dla domyślnej metody zdarzeniowej kontrolki `button1` piszemy kod z listingu 6.4.

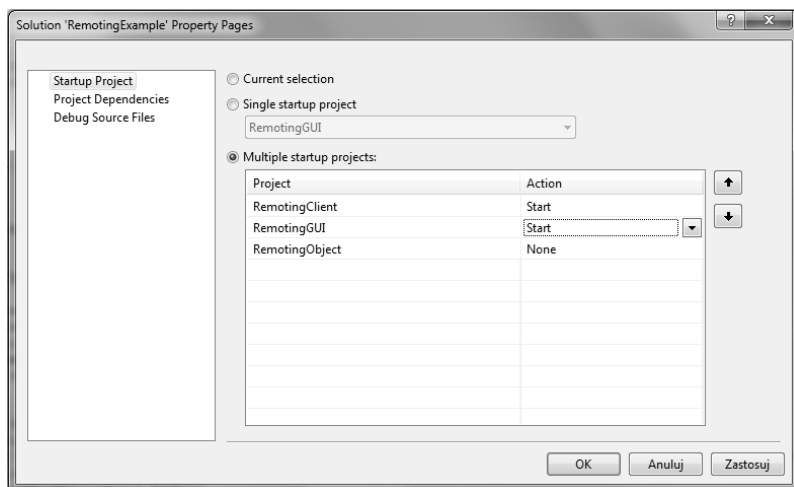
Listing 6.4. *Uzyskanie połączenia z serwerem*

```
private void button1_Click(object sender, EventArgs e)
{
    string address = textBox1.Text;
    if (!address.StartsWith("http://"))
        address = "http://" + textBox1.Text;
    int port = (int)numericUpDown1.Value;
    HttpClientChannel channel = null;
    try
    {
        channel = new HttpClientChannel();
        ChannelServices.RegisterChannel(channel, false);
        RemotingObject remotingObject = (RemotingObject)Activator.GetObject
        ↵(typeof(RemotingObject), address + ":" + port.ToString() +
        ↵"/Remoting");
        listBox1.Items.Add("Połączenie: " + address + ":" + port.ToString()
        ↵+ "/Remoting");
        listBox1.Items.Add(remotingObject.Test());
        ChannelServices.UnregisterChannel(channel);
        listBox1.Items.Add("Połączenie zakończone");
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Błąd");
        listBox1.Items.Add("Połączenie przerwane");
        ChannelServices.UnregisterChannel(channel);
    }
}
```

W tej metodzie tworzymy kanał typu `HttpClientChannel`. Następnie za pomocą metody `GetObject` klasy `Activator` tworzony jest obiekt typu `RemotingObject`. Pierwszym argumentem tej metody jest typ obiektu. Drugi argument to opisany wcześniej obiekt `Uri`. Jeżeli aktywacja się powiedzie, wówczas mamy dostęp do metody `Test()`. Jej wywołanie zostało wyróżnione na listingu 6.4. Należy pamiętać, że w tym przypadku mamy do czynienia z obiektem zdalnym, kontrolowanym przez serwer. Po wywołaniu metody `Test()` zamykamy kanał metodą `UnregisterChannel`. Całość musimy umieścić w bloku ochronnym, ponieważ w trakcie działania aplikacji mogą być zgłaszane wyjątki.

Pora teraz przetestować nasze rozwiązanie. W jego skład wchodzi dwa projekty, które powinny być uruchomione jednocześnie. W Visual Studio poradzimy sobie z tym bez problemu. W oknie *Solution Explorer* klikamy prawym klawiszem nazwę rozwiązania. Otworzy się podręczne menu, z którego wybieramy opcję *Properties*. Możemy również użyć skrótu klawiszowego *Alt+Enter*. Wybieramy teraz opcję *Startup Project*. W kolejnym kroku zaznaczamy opcję *Multiple startup projects*. Możemy wskazać, które projekty mają wystartować w momencie kompilacji i uruchomienia rozwiązania. Wybieramy projekty *RemotingClient* oraz *RemotingGUI* (rysunek 6.3). Po naciśnięciu klawisza *F5* uruchomią się wskazane projekty. W projekcie serwera klikamy przycisk *Start*. W polu adresu na formie klienta wpisujemy *localhost* oraz port *25000*. Teraz klikamy *Połącz*. Klient powinien nawiązać połączenie i wyświetlić wynik działania metody *Test()*.

Rysunek 6.3.
Okno właściwości
rozwiązania



Projekt 71. Serwer TCP

Protokół TCP również może służyć do udostępniania usług za pomocą technologii .NET Remoting. W ramach obiektu zdalnego napiszemy prostą metodę dodającą dwie liczby i zwracającą wynik. Tak jak w poprzednich projektach z tego rozdziału tutaj również nie zależy nam na rozbudowanych przykładach. W tym i kolejnym projekcie skupimy się na zasadzie tworzenia rozwiązania opartego na protokole TCP.

1. Rozpoczynamy nowy projekt *Class Library*. W polu *Name* wpisujemy *RemotingTCP*.
2. Naszą klasę nazwiemy, trochę na wyrost, *RemotingMath*. Dodamy bezparametrowy konstruktor oraz publiczną metodę *Add*, która zwróci wynik dodawania dwóch liczb przekazanych jako argumenty. Metodę przeciążymy dla typów *double* i *int*. Oczywiście to nie koniec. Powinno się jeszcze bardziej przeciążyć metodę *Add*. Nie będziemy się tym zajmować teraz, ponieważ co innego jest celem tego ćwiczenia. Cały kod tej klasy zawiera listing 6.5. Klasa ta nie różni się znacząco od klasy napisanej w projekcie 69. Tutaj także dziedziczymy po klasie *MarshalByRefObject*.

Listing 6.5. *Klasa Matematyka*

```

using System;
using System.Collections.Generic;
using System.Text;

namespace RemotingTCP
{
    public class RemotingMath: MarshalByRefObject
    {
        public RemotingMath() { }

        public double Add(double a, double b)
        {
            return a + b;
        }
        public int Add(int a, int b)
        {
            return a + b;
        }
    }
}

```

3. Zapisujemy projekt pod nazwą *RemotingMath*, a rozwiązanie pod nazwą *RemotingTCP*, następnie całość kompilujemy. W ten sposób otrzymamy bibliotekę *RemotingMath.dll*.
4. Tworzymy nowy projekt *Windows Forms* o nazwie *RemotingServer* i umieszczamy go w tym samym rozwiązaniu. Sposób został opisany w poprzednich projektach z tego rozdziału (projekt 69. i 70.).
5. Do projektu dodajemy referencje do *System.Runtime.Remoting.dll* oraz do projektu *RemotingMath* (patrz projekt 70.).
6. Przenosimy się do widoku kodu (klawisz *F7*). W bloku definiującym przestrzeń nazw dodajemy wpis:

```

using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;

```
7. Na formę wrzucamy kontrolkę *textBox1*. Jej własność *ReadOnly* zmieniamy na *true*.
8. Do projektu dodajemy przycisk *button1*. Tworzymy dla niego metodę zdarzeniową *Click* i wpisujemy kod z listingu 6.6. Komentarz z projektu 69. odnosi się również do tego listingu. Jedyną różnicą jest użycie obiektu klasy *TcpServerChannel*.

Listing 6.6. *Inicjacja serwera TCP*

```

private void button1_Click(object sender, EventArgs e)
{
    try
    {
        TcpServerChannel channel = new TcpServerChannel(20000);
    }
}

```

```

        ChannelServices.RegisterChannel(channel, false);

        RemotingConfiguration.RegisterWellKnownServiceType(typeof(RemotingMath), "Math",
        WellKnownObjectMode.SingleCall);
        textBox1.Text = "Serwer oczekuje na połączenia ...";
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Błąd");
    }
}

```

Projekt 72. Klient TCP

Idea klienta .NET Remoting została już opisana w projekcie 70. Ten projekt będzie nieco bardziej rozbudowany. Rozważymy różne przypadki użycia tej małej aplikacji. Zaczynamy się więc od razu do kodowania:

1. Tworzymy nowy projekt *Windows Forms*. W polu *Name* wpisujemy *RemotingClient*. Jak poprzednio projekt ten powinien być dodany do tego samego rozwiązania. Ułatwi nam to zarządzanie projektami i testowanie ich.
2. Dodajemy referencję do *System.Runtime.Remoting.dll* oraz *RemotingMath* (patrz projekt 69.).
3. Projekt uzupełniamy o następujące definicje przestrzeni nazw:

```

using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;

```

4. Na formę wrzucamy kontrolki *textBox1*, *textBox2* i *textBox3* (rysunek 6.4).

Rysunek 6.4.
Widok formy klienta
.Net Remoting TCP



5. Właśność *ReadOnly* kontrolki *textBox3* ustawiamy na *true*.
6. Do projektu dodajemy przycisk *button1*. Właśność *Text* zmieniamy na *=*.
7. Dodajemy dwa prywatne pola klasy formy (listing 6.7).

Listing 6.7. Klient TCP. Pola prywatne

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;

```

```

using System.Windows.Forms;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;

namespace RemotingTCP
{
    public partial class Form1 : Form
    {
        private TcpClientChannel channel;
        private RemotingMath remotingMath;

        public Form1()
        {
            InitializeComponent();
        }
    }
}

```

- 8.** Tworzymy metodę zdarzeniową `Load` formy. Zainicjalizujemy w niej obiekty, których dalej użyjemy w połączeniu .NET Remoting (listing 6.8). Należy zwrócić uwagę, że inicjalizacja obiektów nie oznacza nawiązania połączenia. Innymi słowy, nawet jeśli serwer nie jest włączony, metoda `Load` nie spowoduje błędów.

Listing 6.8. Klient TCP. Metoda inicjalizująca połączenie .NET Remoting

```

private void Form1_Load(object sender, EventArgs e)
{
    channel = new TcpClientChannel();
    ChannelServices.RegisterChannel(channel, false);
    WellKnownClientTypeEntry config = new WellKnownClientTypeEntry(typeof
    ↪(RemotingMath), "tcp://localhost:20000/Math");
    RemotingConfiguration.RegisterWellKnownClientType(config);
    remotingMath = new RemotingMath();
}

```

- 9.** Tworzymy nową metodę zdarzeniową `Click` kontrolki `button1` i wpisujemy do niej kod z listingu 6.9. Metoda ta użyje zdalnego obiektu i wykona na nim metodę `Add`.

Listing 6.9. Klient TCP

```

private void button1_Click(object sender, EventArgs e)
{
    if (textBox1.Text == null && textBox2.Text == null && textBox3.Text == null)
        return;

    double a = 0.0;
    double b = 0.0;

    if (!Double.TryParse(textBox1.Text, out a))
    {
        textBox3.Text = "Błąd!";
        return;
    }
}

```

```
    }
    if (!Double.TryParse(textBox2.Text, out b))
    {
        textBox3.Text = "Błąd!";
        return;
    }

    try
    {
        textBox3.Text = (remotingMath.Add(a, b)).ToString();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Błąd");
        ChannelServices.UnregisterChannel(channel);
    }
}
```

Jak to często bywa, kod osłaniający ewentualne złe użycie metody jest bardziej rozbudowany niż samo użycie metody. Ta aplikacja jest odpowiednikiem programu z listingu 6.4. Istnieją jednak pewne różnice. Aplikacja klienta używa metody `RegisterWellKnown` $\hookrightarrow$ `ClientType` klasy `RemotingConfiguration` do połączenia się z serwerem. Dalej tworzona jest instancja klasy `RemotingMath`. Pomimo że dzieje się to w klasie klienta, instancja ta nie jest w pełni lokalnym obiektem. Zarządzana jest przez aplikację serwera. Będzie to dobrze widoczne w projektach 75 – 77. W ten sposób pokazaliśmy dwa rozwiązania, które możemy zastosować do aktywacji dostępu do zdalnego obiektu po stronie klienta.

Projekt 73. Serwer TCP. Plik konfiguracyjny

W poprzednich projektach aplikacje serwerowe były konfigurowane w kodzie programu. Każdorazowa zmiana nazwy udziału, portu lub innego parametru wiąże się z ponownym skompilowaniem projektu. Zamiast konfigurować serwer w kodzie, możemy skonfigurować go za pomocą pliku konfiguracyjnego. Jest on standardu XML. Takie rozwiązanie jest o wiele bardziej elastyczne.

1. Tworzymy nowy projekt *Windows Forms*.
2. Do projektu dodajemy referencje *System.Runtime.Remoting.dll* i *RemotingMath.dll*.
3. Dodamy teraz do projektu plik XML. Z menu *Project* wybieramy opcję *Add New Item....*
4. Zaznaczamy ikonę *XML File* i w polu *Name* wpisujemy *config.xml*. Klikamy *Add*.
5. W otwartym pliku XML wpisujemy kod z listingu 6.10.

Listing 6.10. *Plik konfiguracyjny serwera*

```
<configuration>
  <system.runtime.remoting>
    <application>
      <service>
        <wellknown mode="SingleCall" objectUri="Math"
          type="RemotingTCP.RemotingMath, RemotingMath" />
        <activated type=" RemotingTCP.RemotingMath, RemotingMath " />
      </service>
    </application>
  </system.runtime.remoting>
</configuration>
```

Rozpoczynamy kluczem <configuration>. Kolejny klucz to definicja przestrzeni nazw. W kluczu <service> definiujemy parametry serwisu. W znaczniku <wellknown> podajemy tryb inicjalizacji obiektu (SingleCall), Uri oraz przestrzeń nazw i nazwę klasy obiektu. Znacznik <activated> zapewnia możliwość tworzenia przez klienta lokalnej kopii obiektu. Na końcu zdefiniowaliśmy typ transmisji oraz numer portu, na którym serwer będzie prowadził nasłuch.

6. Teraz przechodzimy do kodu *Form1.cs*. Jak zwykle dodajemy trzy przestrzenie nazw:

```
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
```

7. Do projektu dodajemy pole edycyjne `textBox1`.

8. Na formę wrzucamy przycisk `button1`. Dla jego domyślnej metody zdarzeniowej piszemy kod z listingu 6.11.

Listing 6.11. *Inicjacja serwera na podstawie pliku konfiguracyjnego*

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        RemotingConfiguration.Configure("config.xml", false);
        textBox1.Text = "Serwer nasłuchuje ...";
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

Inicjacja serwera polega jedynie na użyciu metody `Configure` klasy `RemotingConfiguration`, za pomocą której wskazujemy plik konfiguracyjny. Jeżeli do naszego serwisu dodaliśmy plik *config.xml*, wówczas standardowo będzie się on znajdował w katalogu projektu. W momencie uruchomienia

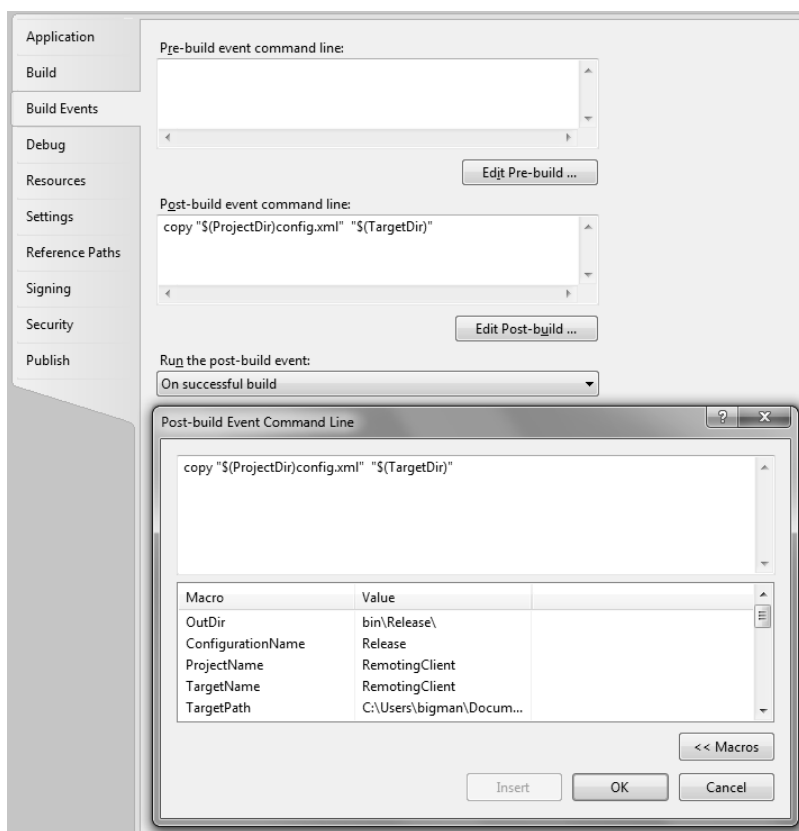
serwera jesteśmy przeniesieni do katalogu *katalog_projektu/bin/Debug*. Aby plik *config.xml* był kopiowany do katalogu wynikowego aplikacji, musimy zmienić konfigurację budowania aplikacji. Otwieramy okno własności (*Properties*) projektu. Odnajdujemy zakładkę *Build Events*. W miejscu *Post-build event command line* wpisujemy następujące polecenie:

```
copy "$(ProjectDir)config.xml" "$(TargetDir)"
```

Spowoduje to skopiowanie pliku *config.xml* do katalogu wynikowego aplikacji. Posłużyliśmy się w tym miejscu makrodefinicjami. Warto sprawdzić opcję *Edit Post-build....* Jest to edytor, który zawiera wszystkie makrodefinicje dostępne w Visual Studio (rysunek 6.5). Napisaną aplikację możemy teraz przetestować.

Rysunek 6.5.

Okno własności projektu z otwartym edytorem Post-build event



9. Zapisujemy i uruchamiamy serwer.

10. W katalogu projektu 72. odnajdujemy plik wykonywalny klienta TCP. Uruchamiamy tę aplikację. W polach edycyjnych wpisujemy dowolne liczby i klikamy przycisk =. Połączenie z nowo utworzonym serwerem powinno zostać ustanowione i odpowiednie działanie powinno być wykonane.

Projekt 74. Klient TCP.

Plik konfiguracyjny

Klient również może zostać skonfigurowany za pomocą pliku XML zawierającego odpowiednie wpisy.

1. Tworzymy nowy projekt *Windows Forms*. W polu *Name* wpisujemy, tak jak poprzednio, *RemotingTCP*. Dzięki temu będziemy się znajdować w tej samej przestrzeni nazw co serwer. Nazwę projektu możemy zmienić potem w oknie *Solution Explorer*.
2. Interfejs naszej aplikacji będzie identyczny z interfejsem klienta TCP z projektu 71.
3. Dodajemy również te same referencje (*RemotingMath.dll* i *System.Runtime.Remoting.dll*).
4. Projekt uzupełniamy o definicje następujących przestrzeni nazw:

```
using System.Runtime.Remoting;  
using System.Runtime.Remoting.Channels;  
using System.Runtime.Remoting.Channels.Tcp;
```
5. Do projektu dodajemy plik XML (patrz punkt 3. w projekcie 73.). Nazwiemy go *config.xml*.
6. W utworzonym pliku wpisujemy kod z listingu 6.12.

Listing 6.12. Plik konfiguracyjny klienta TCP

```
<?xml version="1.0" encoding="utf-8" ?>  
<configuration>  
  <system.runtime.remoting>  
    <application>  
      <channels>  
        <channel ref="tcp">  
          <clientProviders>  
            <formatter ref="binary"/>  
          </clientProviders>  
        </channel>  
      </channels>  
    <client>  
      <wellknown url="tcp://localhost:20000/Math"  
type="RemotingTCP.RemotingMath, RemotingMath" />  
    </client>  
  </application>  
</system.runtime.remoting>  
</configuration>
```

W pliku tym użyliśmy tych samych tagów co w projekcie 73. Znacznik `<formatter>` definiuje format przesyłania danych. W tym przypadku dane zostają przesyłane binarnie.

Projekt 75. Czat. Klasa serwera

Program, który teraz napiszemy, będzie odpowiadał projektowi *czat* z rozdziału 5. Różnica będzie polegać na zastosowaniu technologii Remoting. W projekcie 72. przekonywałem, że obiekt stworzony na podstawie klasy `RemotingMath` nie jest obiektem wyłącznie lokalnym, choć samo stworzenie go za pomocą operatora `new` wskazywałoby na to, iż jest to obiekt lokalny, kontrolowany przez aplikację klienta. W tym projekcie i dwóch kolejnych będziemy mieli szansę sprawdzić, że tak utworzony obiekt jest dzielony przez wszystkich klientów, a jego stan jest kontrolowany przez serwer.

1. Tworzymy nowy projekt *Class Library* i nazywamy go `ChatRemoting`.
2. Stworzoną klasę nazywamy `Chat`. Jako klasę bazową dla nowej klasy wybierzemy `MarshalByRefObject` (listing 6.13). Do klasy dodamy jeszcze bezparametrowy konstruktor i przestrzeń nazw `System.Collections`.

Listing 6.13. Szablon klasy *Czat*

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Text;

namespace ChatRemoting
{
    public class Chat : MarshalByRefObject
    {
        private ArrayList userList = new ArrayList();
        private String talk = String.Empty;

        public Chat() { }
    }
}
```

Jeżeli zlikwidowalibyśmy dziedziczenie po klasie `MarshalByRefObject`, to każdy klient na czas sesji posiadałby własną, niezależną kopię obiektu. Kopia ta nie byłaby uzupełniana o nowe wiadomości bądź nowych klientów. Dziedziczenie z klasy bazowej `MarshalByRefObject` umożliwia współdzielenie obiektu klasy pochodnej `Chat`.

3. W klasie `Chat` umieszczamy dwa prywatne pola: `usersList` i `talk` (listing 6.13). Pierwsze z nich będzie przechowywało listę aktualnie przyłączonych użytkowników. Zastosowaliśmy tutaj dynamiczną tablicę klasy `ArrayList`. Pole `talk` będzie przechowywało całą rozmowę, począwszy od startu serwera. Pola inicjalizujemy w miejscu definicji.
4. Do klasy `Chat` dodajemy metodę umożliwiającą dodawanie nowych użytkowników (listing 6.14). Zabezpieczymy się w niej przed dodawaniem użytkowników o tych samych pseudonimach. Ponieważ pole `usersList` jest de facto dzielone pomiędzy aplikacje klienckie wykorzystujące obiekt klasy `Chat`, dodawanie umieszczamy w sekcji krytycznej. W przypadku nieudanej próby dodania użytkownika metoda zwróci `false`.

Listing 6.14. *Dodawanie nowych użytkowników serwisu*

```
public bool AddUser(string userName)
{
    if (usersList.Contains(userName))
        return false;

    if (userName != null && userName.Trim() != null)
    {
        lock (usersList)
        {
            usersList.Add(userName);
        }
        return true;
    }
    return false;
}
```

- 5.** Jeżeli użytkownik się odłączy, wówczas powinien zostać usunięty z listy `usersList`. Odpowiedni kod realizujący to zadanie został umieszczony na listingu 6.15.

Listing 6.15. *Usunięcie użytkownika z listy użytkowników*

```
public void RemoveUser(string userName)
{
    lock (usersList)
    {
        usersList.Remove(userName);
    }
}
```

- 6.** Do naszej klasy dodajemy metodę, która będzie umieszczała w polu `rozmowa` nową wiadomość wysłaną przez klienta (listing 6.16). Metoda ta sprawdza również, czy w podanym ciągu nie znajdują się znaki formatujące tabulacji i końca linii. Na potrzeby tego przykładu jest to wystarczające, ale powinno się zwiększyć kontrolę wprowadzanych znaków.

Listing 6.16. *Dodawanie nowej wiadomości*

```
public void AddMessage(string newMessage)
{
    if (newMessage != null && newMessage.Trim() != null)
    {
        newMessage.Replace("\n", " ");
        newMessage.Replace("\t", " ");
        lock (talk)
        {
            talk += newMessage + "\n";
        }
    }
}
```

7. Aby uzyskać dostęp do prywatnych pól klasy *Czat*, tworzymy własności tylko do odczytu (listing 6.17). Pola te nie mogą być modyfikowane bezpośrednio z zewnątrz klasy.

Listing 6.17. Własności wyświetlające listę użytkowników i rozmowę

```
public ArrayList UsersList
{
    get { return usersList; }
}

public string Talk
{
    get { return talk; }
}
```

8. Projekt zapisujemy (np. pod nazwą *ChatRemoting*) i kompilujemy.

W ten sposób stworzyliśmy klasę, której następnie użyjemy do połączenia z klientami. Ponieważ jednocześnie kilku użytkowników usługi mogłoby chcieć uzyskać dostęp do pól klasy poprzez metody *AddUser*, *RemoveUser* i *AddMessage*, zastosowaliśmy metodę *lock*. Blokuję ona określone zasoby, tak aby metody z innych wątków programu nie miały do nich dostępu tak długo, jak długo aktualny wątek musi korzystać z tych zasobów. Jest to sposób często używany w synchronizowaniu wątków.

Projekt 76. Czat. Serwer

Aplikacja serwera będzie niezwykle prosta. Ograniczymy się w niej tylko do uruchomienia usługi związanej ze stworzoną wcześniej biblioteką *ChatRemoting.dll*. Ponieważ będziemy przysyłać wiadomości tekstowe, najwygodniejszym rozwiązaniem jest zastosowanie połączenia HTTP.

1. Rozpoczynamy nowy projekt *Windows Forms*. Dodajemy go do już istniejącego rozwiązania. W polu *Name* wpisujemy *ChatServer*. Niech przestrzeń nazw będzie ta sama co w projekcie 75., czyli *ChatRemoting*. Najszybciej można zamienić nazwę przestrzeni nazw, używając narzędzia *Refactor*.
2. Do projektu dodajemy referencję *System.Runtime.Remoting.dll* oraz referencję do projektu 75.
3. Tradycyjnie dodajemy następujące przestrzenie nazw:

```
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
```

4. Na formę wrzucamy przycisk *button1*. Dla jego domyślnej metody zdarzeniowej piszemy kod z listingu 6.18.

Listing 6.18. Główna metoda serwera

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        HttpChannel channel = new HttpChannel(1978);
        ChannelServices.RegisterChannel(channel, false);
        RemotingConfiguration.RegisterWellKnownServiceType(typeof(Chat),
            ↵ "Chat", WellKnownObjectMode.Singleton);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

Połączenie będzie nawiązywane w trybie *Singleton*. Oznacza to, że każdy z klientów będzie miał dostęp do tego samego (jednego) obiektu. Zmiany wprowadzone przez jednego klienta będą widoczne z poziomu każdego innego (nawet nowo przyłączonego). W naszym przypadku jest to naturalny wybór, ponieważ chcemy, aby wszystkie aplikacje klienckie mogły ze sobą rozmawiać. Stąd również w metodach modyfikujących klasę *ChatRemoting* znajdują się sekcje krytyczne.

Projekt 77. Czat. Klient

Pora na aplikację klienta. Aplikacja ta będzie nieco bardziej skomplikowana od poprzedniej, choć jej napisanie nie przysporzy nam większych kłopotów.

1. Tworzymy nowy projekt *Windows Forms*. W polu *Name* wpisujemy *ChatClient*. Jak poprzednio musimy zadbać, aby klasa znajdowała się w tej samej przestrzeni nazw co pozostałe klasy projektu. Jest to oczekiwana własność, która uchroni nas przed dodawaniem w sekcji *using* referencji do nowych przestrzeni nazw.
2. Podobnie jak w projekcie 76. tutaj także dodajemy referencję do biblioteki *System.Runtime.Remoting.dll* oraz do projektu *CzatRemoting*.
3. Do projektu dodajemy nowe przestrzenie nazw:

```
using System.Collections;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
```

4. Klasę *Form1* uzupełniamy o nowe pola:

```
private Chat chat;
private HttpChannel channel;
private bool isConnected = false;
```

5. Zaprojektujemy teraz interfejs aplikacji. Na formę wrzucamy pole edycyjne `textBox1`. Tutaj użytkownik będzie mógł wprowadzać wiadomość do wysłania.
6. Do projektu dodajemy pole edycyjne `textBox2`, które będzie przechowywało nasze imię.
7. Formę uzupełniamy o kontrolkę `textBox3`. W niej będzie widoczna rozmowa. Własność `ReadOnly` ustawiamy na `true`. Własność `Multiline` ustawiamy na `true`.
8. Na formie umieszczamy listę `listBox1`, która będzie zawierała aktualnie podłączonych użytkowników.
9. Na formę wrzucamy przycisk `button1`. Jego własność `Text` zmieniamy na *Połącz*.
10. Dodajemy kontrolkę `timer1` klasy `Timer`. Jest to „stoper” odmierzający czas w milisekundach. Własność `Interval` ustawiamy na 1000. Oznacza to, że w odstępach czasu równych 1000 ms, czyli 1 s, będzie następowało zdarzenie `Tick`. Jest to wygodne narzędzie do oprogramowania periodycznie powtarzających się funkcji aplikacji. Jego użycie zostanie przedyskutowane niżej.
11. Tworzymy metodę zdarzeniową `Click` kontrolki `button1` i umieszczamy w niej kod z listingu 6.19.

Listing 6.19. *Połączenie z serwerem*

```
private void button1_Click(object sender, EventArgs e)
{
    if (isConnected)
    {
        MessageBox.Show("Już podłączony", "Błąd");
        return;
    }
    try
    {
        channel = new HttpChannel();
        ChannelServices.RegisterChannel(channel, false);
        RemotingConfiguration.RegisterWellKnownClientType(typeof(Chat),
            ↪ "http://localhost/Chat");
        chat = new Chat();
        bool status = chat.AddUser(textBox2.Text.Trim());
        if (status == false)
        {
            MessageBox.Show("Użytkownik już istnieje", "Błąd");
            ChannelServices.UnregisterChannel(channel);
            timer1.Enabled = false;
            isConnected = false;
            button1.Enabled = true;
            textBox2.ReadOnly = false;
            return;
        }
        chat.AddMessage("Użytkownik [" + textBox2.Text + "] dołączył
            ↪ do rozmowy \n\n");
        timer1.Enabled = true;
        isConnected = true;
        button1.Enabled = false;
        textBox2.ReadOnly = true;
    }
}
```

```

        catch (Exception ex)
        {
            MessageBox.Show(ex.Message, "Błąd");
            textBox3.Text = "Nie udało się nawiązać połączenia ...";
            ChannelServices.UnregisterChannel(channel);
            timer1.Enabled = false;
            isConnected = false;
            button1.Enabled = true;
            textBox2.ReadOnly = false;
            channel = null;
        }
    }
}

```

Kod, za pomocą którego łączymy się z serwerem, jest prawie taki sam jak w projekcie 70. Jedyna różnica polega na tym, że po podłączeniu do serwera wysyłamy wiadomość, używając metody `AddUser`. W momencie przyłączenia się do serwera i utworzenia obiektu klasy `Chat` uzyskujemy dostęp do dzielonego obiektu. Jeżeli rozmowa będzie się już toczyć jakiś czas, a nowy klient dopiero się podłączy, wówczas dostanie on wszystkie wiadomości od momentu rozpoczęcia usługi (włączenia serwera).

- 12.** Do projektu dodajemy przycisk `button2`. Jego kliknięcie spowoduje dodanie wiadomości do pola `talk`. Odpowiedni kod zawiera listing 6.20.

Listing 6.20. *Dodanie nowej wiadomości*

```

private void button2_Click(object sender, EventArgs e)
{
    if (isConnected)
        if (textBox2.Text != null && textBox2.Text.Trim() != null)
            chat.AddMessage("[ " + textBox2.Text + " ] " + textBox1.Text);
}

```

- 13.** Dla komponentu `timer1` tworzymy metodę zdarzeniową `Tick` (listing 6.21).

Listing 6.21. *Metoda pobierająca nowe wiadomości i nowych użytkowników*

```

private void timer1_Tick(object sender, EventArgs e)
{
    ArrayList users = chat.UsersList;
    listBox1.Items.Clear();
    foreach (string user in users)
        listBox1.Items.Add(user);
    textBox3.Clear();
    textBox3.Text = chat.Talk;
}

```

Jak sprawdzić, czy ktoś coś napisał lub czy ktoś nowy dołączył do rozmowy? Pomysł jest bardzo prosty. Co jedną sekundę odświeżamy zawartość pola edycyjnego `textBox3`, które przechowuje stan rozmowy oraz zawartość listy gości (kontrolka `listBox1`). Pomysł ten jest od razu najsłabszym punktem rozwiązania. Musimy mieć na uwadze, jakie wprowadza ograniczenia. Wielkość

pamięci rozmowy nie może być zbyt duża. Spowodowałoby to przeciążenie sieci. Dodatkowo po co od nowa pobierać rozmowę, którą już pobraliśmy? Proponuję, aby problemy te potraktować jako ciekawe ćwiczenie. Omówiony projekt jest jedynie początkiem. Należy go zmodyfikować i rozszerzyć.

14. Do projektu dodajemy jeszcze przycisk button3. Jego własność Text zmieniamy na *Rozłącz*.
15. Kod klasy uzupełniamy o metodę zdarzeniową Click kontrolki button3 (listing 6.22).

Listing 6.22. *Rozłączenie użytkownika*

```
private void button3_Click(object sender, EventArgs e)
{
    if (isConnected)
    {
        chat.AddMessage("Użytkownik [" + textBox2.Text + "] opuścił rozmowę");
        chat.RemoveUser(textBox2.Text);
        listBox1.Items.Clear();
        timer1.Enabled = false;
        ChannelServices.UnregisterChannel(channel);
        isConnected = false;
        button1.Enabled = true;
        textBox2.ReadOnly = false;
    }
}
```

Aby rozłączyć użytkownika, należy usunąć go z listy rozmówców oraz użyć metody `UnregisterChannel` klasy `ChannelServices`, która zakończy połączenie z serwerem. Można również wysłać odpowiedni komunikat do wszystkich użytkowników usługi.

Rozdział ten, jak wspomniałem na początku, jest szybkim wstępem do technologii .NET Remoting, która... w zasadzie zanika. Jeśli nie musisz jej używać, po prostu jej nie używaj. Wsparcie dla niej może w jednej z kolejnych wersji platformy .NET zostać ograniczone. Niemniej jednak jest to ciekawa (moim zdaniem) technologia, która wprowadziła pewną warstwę abstrakcji pomiędzy gniazda sieciowe a wyższe warstwy aplikacji. Projekt już nie jest rozwijany, ale z pewnością przyczynił się do powstania nowych koncepcji w *Windows Communication Foundation*.

Skorowidz

#endregion, 27, 113
#region, 27, 113
 , 213
.NET, 7, 9
.NET Framework, 9, 272
.NET Remoting, 188
@ Page, 38
[Serializable], 188
[STAThread], 28
<!>, 38
<% %>, 38
<body>, 36

, 213
<div>, 36
, 36
<form>, 39
<h1>, 36
<head>, 38
<td>, 214
<title>, 36

A

A jak address, 270
Active Server Pages, 211
Active Template Library, 21
ActiveX Data Object, 212
Address, 269
ADO.NET, 10, 33, 211
adres
 IP, 48, 60, 80
 MAC, 44, 48
 schemat adresu, 270
 sieci, 49
 ścieżka do serwisu, 270
anonimowy użytkownik, 246
aplikacje
 hostujące, 280
 konsolowe, 72

ASP.NET, 33
 bazodanowe, 212
 rozproszone, 187
 sieciowe, 43, 45, 53
 WWW, 33
App.config, 274
AppendChild(), 225
Application.EnableVisualStyles, 28
Application_Start(), 231
ARP, 44
ARPANET, 46
ArrayList, 16, 117
ASP.NET, 10, 33, 211
 baza danych, 247
 identyfikacja użytkowników, 241
 księga gości, 222
 ping, 217
 pliki zasobów, 232
 pobieranie plików na serwer, 220
 pozycjonowanie kontrolek na stronie, 212
 sesja, 230
 Web Site, 212
 wielojęzyczny serwis internetowy, 232
 wysyłanie wiadomości e-mail, 218
 wysyłanie wiadomości e-mail
 z załącznikami, 221
 XML, 222
 zasoby globalne, 237
 zasoby lokalne, 232
asynchroniczne odwołanie do usługi, 295
asynchroniczne połączenie TCP, 64
asynchroniczne wysyłanie
 wiadomości e-mail, 108
ATL, 21
Attachment, 104, 222
AutoEventWireup, 38
AutoPostBack, 240
AutoResetEvent, 78

B

B jak binding, 270
 BackgroundWorker, 147
 BallonTipIcon, 144
 BallonTipTitle, 144
 baseAddress, 274
 BasicHttpBinding, 270, 271
 BasicHttpContextBinding, 271
 baza danych, 211, 247

- połączenie, 250
- SqlDataSource, 250
- tabele, 249

 baza książek, 247

- baza danych, 247
- połączenie z bazą danych, 250
- prezentacja danych, 251

 bazowy adres, 274
 bezpieczne odwoływanie się

- do własności kontrolek formy
- z poziomu innego wątku, 149, 153

 biblioteka

- RemotingObject.dll, 189
- System.Runtime.Serialization.dll, 269
- System.ServiceModel.dll, 267
- Windows Forms, 22

 binarna reprezentacja danych, 272
 BinaryReader, 45, 155
 BinaryWriter, 45, 155
 Binding, 269
 blocking socket, 58
 blok programu, 27
 blokowanie gniazda, 58
 błędy, 14, 17
 błyskawiczne tworzenie aplikacji, 21
 Button, 111, 216

C

C jak kontrakt, 273
 C#, 7, 11
 C# generics, 11
 C++, 11, 12
 callbacks, 17
 CancelAsync(), 137
 CancelDestinationPageUrl, 245
 CAO, 188
 CarRentalHost.SelfHost, 281
 CarRentalServices, 288
 catch, 18
 CGI, 211
 ChannelFactory, 290
 ChannelServices, 192

checkIP, 72
 chronione zasoby, 241
 ciało metody Main, 282
 CIL, 9
 class, 13
 Click, 61, 94
 Close(), 55, 93
 CLR, 9
 code-behind, 37
 CodeFile, 38
 ComboBox, 111, 151
 Common Intermediate Language, 9
 Common Language Runtime, 9
 CompareTo, 19
 CompareValidator, 230
 Configure Data Source, 250
 ContextMenuStrip, 144, 151
 ContinueDestinationPageUrl, 245
 Contract, 269
 ControlToCompare, 230
 ControlToValidate, 228, 229
 CORBA, 187
 CreateElement(), 225
 CreateUserText, 245
 CreateUserUrl, 245
 CreateUserWizard, 245
 Culture, 235
 CurrentDirectory, 99
 czat (klient), 181

- odbieranie wiadomości od serwera, 182
- połączenie z serwerem, 183
- Remoting, 206
- wysyłanie wiadomości, 184

 czat (serwer), 171

- bezpieczne odwoływanie się do własności
 - kontrolek formy z poziomu innego wątku, 173
- interfejs, 171
- klasa formy, 174
- kończenie pracy, 178
- obsługa połączeń, 176, 179
- obsługa wątków związanych z klientami, 179
- pętla główna programu, 174
- pseudonimy, 175
- Remoting, 205
- rozłączenie klienta, 180
- rozsyłanie wiadomości, 176
- wątki, 177
- wysyłanie wiadomości, 178

 CzatRemoting, 203

- dodawanie użytkowników, 204
- dodawanie wiadomości, 204, 208
- klient, 206
- lista użytkowników, 205
- pobieranie wiadomości, 208

połączenie z serwerem, 207
rozłączenie użytkownika, 209
serwer, 203, 205
usuwanie użytkownika, 204

D

Database Explorer, 248
DataSourceID, 253
DataTextField, 253
DataValueField, 253
DateTime, 40
DCCP, 44
DCOM, 187
DCompletedEventHandler, 124
Debug, 29, 39
debugger, 39
Default.aspx, 34, 37
Default.aspx.cs, 37
definicja kontraktów i danych, 269
definiowanie przestrzeni nazw, 16
deklaracja klasy, 13
delegacje, 17, 153
delegate, 17
DeleteFile(), 131–133
Delphi, 12
Delphi VCL, 10
DestinationPageUrl, 242
DetailsView, 253
DNS, 56
Document.Write(), 154
DocumentSource, 227
DoD, 43
dodawanie komponentów, 27
dokumenty XML, 225
domyślna metoda zdarzeniowa, 30
dostęp do bazy danych, 211
dotGNU, 9
DoubleClick, 107
DownloadFileAsync(), 122
DoWork(), 147, 149
DProgressChanged, 124
DProgressChangedEventHandler, 124
DropDownList, 239, 253
dynamiczna konfiguracja, 285
dyrektywa #endregion, 113
dyrektywa #region, 113
dyrektywa @ Page, 38
dyrektywa Language, 38
dziedziczenie, 14

E

e.Result, 297
ECMA, 9
Edytor HTML, 91
 interfejs, 91
 obsługa plików tekstowych, 92
 odczytywanie pliku, 93
 podgląd strony, 98
 schowek, 95
 wprowadzanie tagów, 95
edytor stylów, 215
Email, 100
Encoding, 64
Endpoint, 269, 273
ErrorMessage, 228, 229
EventArgs, 31
exception, 17
Expressions, 238

F

FileUpload, 220
finally, 18
firewall, 51
focus, 27
FolderBrowserDialog, 111
Font, 24
Form, 23
Form1.cs, 25
Form1.Designer.cs, 25
Form1.resx, 29
forma, 23, 26
 konstruktor, 26
 przezroczystość, 30
formatowanie tekstu, 152
FormClosing(), 145, 185
FTP, 45, 51
 kasowanie pliku, 131
 klient, 110
 pobieranie pliku, 122
 wysyłanie pliku, 127
FTPClient, 112
FtpWebRequest, 117
FtpWebResponse, 117
funkcja, 13
 Main(), 28
 RunWorkerAsync, 147
 wypisująca listę plików, 118
WyslijPing, 76
zapisująca tekst do pliku, 92

G

garbage collector, 12
 GC, 12
 GDI, 28
 GDI+, 28
 Generate asynchronous operations, 295
 GetCars, 295
 GetCarsAsync, 295
 GetDirectories(), 119, 121
 GetHostEntry(), 71
 GetHostName(), 71
 GetLocalResourceObject(), 237
 GetObject(), 194
 Global Application Class, 230
 Global.asax, 230
 Google Web API, 261
 klient, 262, 265
 podpowiedzi, 265
 rejestracja usługi, 261
 wyszukiwanie, 264
 green pages, 256
 GridView, 251
 GroupBox, 139

H

HeaderText, 252
 host, 269
 host lokalny, 68
 hostowanie usługi WCF
 IIS, 285
 Self hosting, 281
 serwis Windows, 287
 HTTP, 45, 51, 85, 270
 BasicHttpBinding, 271
 BasicHttpContextBinding, 271
 klient, 193
 pobieranie pliku, 86, 135
 pobieranie pliku tekstowego, 85
 pobranie źródła strony, 88
 serwer, 188
 WebHttpBinding, 271
 WSDualHttpBinding, 271
 WSFederationHttpBinding, 271
 WSHttpBinding, 271
 WSHttpContextBinding, 271
 HttpClientChannel, 194
 HTTPS, 270
 HyperLink, 227

I

ICarInformationService, 281
 ICMP, 44, 50, 74, 81
 ICMP Echo Reply, 74
 ICMP Echo Request, 74
 identyfikacja użytkowników, 241, 246
 anonimowy użytkownik, 246
 edycja zabezpieczeń, 243
 Login, 244
 Login.aspx, 241
 LoginView, 246
 IDisposable, 78
 IIS, 285
 modyfikowanie pliku konfiguracyjnego, 286
 ikona w zasobniku systemowym, 144
 informacje o połączeniach sieciowych, 72
 Inherits, 38
 inicjacja
 pobierania pliku, 125
 połączenia FTP, 130
 serwera, 196
 serwera TCP, 57
 usługi sieciowej, 261
 wysyłania pliku, 129
 inicjalizacja obiektu, 13
 inicjalizacja pracy serwera, 177
 InitializeComponent(), 26
 InitializeCulture(), 240
 In-line code, 37
 instalowanie usługi Windows, 290
 instalowanie serwisu, 289
 instalowanie usługi na maszynie docelowej, 289
 instancja klasy, 12, 14
 IntelliSense, 13, 30, 36
 interakcja z użytkownikiem, 29
 interfejs aplikacji, 22
 interfejs graficzny, 22
 interfejs IComparable, 19
 interfejsy, 19
 internet, 7
 InvalidOperationException, 147
 Invoke(), 149
 InvokeRequired, 149
 IP, 44, 48
 IPEndPoint, 63
 IPX, 44
 IsAuthenticated(), 242
 IService1.cs, 274
 IsUserInRole(), 246
 IsValid(), 230
 IsWebBrowserContextMenuEnabled, 151, 181

J

Java, 19
 JavaScript, 38
 język
 C#, 7, 11
 C++, 12
 HTML, 211
 MSIL, 9
 Object Pascal, 12
 WSDL, 192, 255
 XML, 255

K

kanal HTTP, 192
 katalog
 Debug, 29
 Messages, 276
 Release, 29
 ServiceContracts, 276
 ServiceImplementations, 276
 KeyDown, 31
 KeyEventArgs, 31
 KeyPress, 158
 KeyPreview, 31
 klasa
 Activator, 194
 Attachment, 222
 AutoResetEvent, 78
 CarInformation (Request, 276
 CarInformationDTO, 278
 CarInformationResponse, 276
 ChannelFactory<T>, 290
 DateTime, 40
 Encoding, 64
 FtpClient, 113
 FtpWebResponse, 117
 GC, 12
 IDisposable, 78
 IPEndPoint, 63
 IPHostEntry, 71
 MailAddress, 101
 MailMessage, 101
 MarshalByRefObject, 188, 189, 195
 Matematyka, 196
 NetworkStream, 45
 Page, 230
 RemotingConfiguration, 192
 Server, 242
 SmtpClient, 108
 SMTPClient, 101
 StreamReader, 93
 System.Net.NetworkInformation, 76

TcpListener, 64
 UDPClient, 62
 WebClient, 86
 XmlElement, 225
 klasy, 12, 14
 deklaracja, 13
 dziedziczenie, 14
 instancja, 14
 tworzenie, 113
 klasy adresów IP, 48
 klient, 49, 269
 aplikacja konsolowa, 290
 aplikacja okienkowa, 290
 aplikacja webowa, 290
 Google Web API, 262
 wysyłanie informacji o dostępności, 165
 zdalny screenshot, 162
 klient FTP, 110
 asynchroniczne pobieranie pliku, 122
 asynchroniczne wysyłanie pliku, 127
 FTPClient, 112
 inicjacja wysyłania pliku, 129
 interfejs aplikacji, 110
 kasowanie pliku, 131
 listowanie katalogów serwera, 116
 obsługa połączenia, 112
 wywołanie metody pobierającej plik
 asynchronicznie, 125
 wywołanie metody wysyłającej plik
 asynchronicznie, 129
 zmiana katalogu, 119
 klient HTTP, 193
 połączenie z serwerem, 194
 klient TCP, 53, 55, 197, 198
 plik konfiguracyjny, 202
 poła prywatne, 197
 połączenie .NET Remoting, 198
 klient TCPClient, 163
 klient UDP, 61
 Klient Uśmiechu, 144
 oczekiwanie na połączenie
 w osobnym wątku, 147
 odwoływanie się do własności kontrolerek
 formy z poziomu innego wątku, 149
 umieszczenie ikony w zasobniku
 systemowym, 144
 klient-serwer, 49
 kod ASP, 38
 kod usługi sieciowej, 256
 kod XHTML, 37
 kod zarządzany, 12
 kolejki MSMQ, 271, 272
 kolejność kontrolerek, 27
 kolekcje, 16
 ArrayList, 16

- komponenty, 27
 - BackgroundWorker, 147
 - Web Forms, 34
- komunikacja przez sieć, 7
- komunikator (klient), 159
 - inicjacja połączenia, 161
 - interfejs, 159
 - nawiązanie połączenia z serwerem, 160
 - odczytywanie wiadomości z serwera, 160
- komunikator (serwer), 150
 - adres IP, 155
 - bezpieczne odwoływanie się do własności
 - kontrolki formy z poziomu innego wątku, 153
 - formatowanie tekstu, 152
 - inicjacja pracy, 157
 - interfejs, 150
 - komunikacja, 155
 - obsługa rozmowy, 154
 - oczekiwanie na połączenie, 156
 - odczytywanie wiadomości, 157
 - strumienie, 155
 - wysyłanie wiadomości, 158
- konfiguracja klienta, 293
- konfiguracja referencji, 295, 296
- konfiguracja WCF, 274
- konfiguracja wiązania, 293
- konflikt nazw, 15
- konstruktory, 13, 115
- kontrakt danych, 276
- kontrakt usługi, 273
- kontrolka
 - comboBox1, 151
 - contextMenuStrip1, 144
 - DropDownList, 239, 253
 - FileUpload, 220
 - GridView, 251, 252
 - HyperLink, 227
 - listBoxFtpDir, 111
 - LoginStatus, 241
 - maskedTextBoxPass, 111
 - menuStrip, 92
 - numericUpDown1, 151
 - openFileDialog1, 112
 - RequiredFieldValidator, 228
 - saveFileDialog, 94
 - statusStrip1, 112
 - System.Web.UI.WebControls, 15
 - System.Windows.Forms, 15
 - TextBox, 95
 - textBoxLogin, 111
 - WebBrowser, 89, 150
 - webBrowserChat, 172
 - Web Forms, 39
 - Windows Forms, 23, 66

- księga gości, 222
- dodawanie wpisu, 224
- liczba gości online, 230
- sprawdzanie poprawności wpisywanych
 - danych, 228
- wyświetlanie zawartości pliku XML, 226
- kultura, 235

L

- Label, 241
- LAN, 46
- Language, 38
- Layer, 217
- liczba gości online, 230
- licznik gości, 231
- link, 227
- LINQ, 10
- lista kontaktów, 140
- ListBox, 15, 107, 111, 217
 - dodawanie elementu, 134
 - usuwanie dowolnej ilości elementów, 140
 - usuwanie elementu, 107, 134
- listowanie katalogów serwera FTP, 116
- listy, 16
- Login, 241, 244
- Login Task, 241, 242
- Login.aspx, 241
- LoginName, 241
- LoginStatus, 241
- LoginView, 246
- logowanie, 241
- loopback, 48

M

- MAC, 44, 48
- MailAddress, 101
- MailMessage, 101, 108, 222
- Main(), 28
- MarshalByRefObject, 188, 189, 195
- maska podsieci, 49
- maskarada, 49
- MaskedTextBox, 111
- masowe wysyłanie wiadomości e-mail, 106
- Maximum, 24
- menedżer pobierania plików w tle, 133
 - interfejs, 133
 - pobieranie pliku, 135
 - przerwanie pobierania pliku, 137
- menu Debug, 39
- menu główne, 92
- menu kontekstowe, 146

- MenuStrip, 92
 - MessageBox.Show(), 56, 87
 - Metadata Exchange, 279
 - metoda, 13, 14
 - AcceptTcpClient, 58
 - AppendChild, 225
 - asynchroniczna, 64
 - BeginAcceptTcpClient, 65
 - CancelAsync, 137
 - Click, 148
 - Close, 55, 93
 - CompareTo, 19
 - CreateElement, 225
 - DeleteFile, 131, 132
 - do usuwania wpisu, 168
 - Document.Write, 154
 - dodająca wpis, 140, 224
 - DownloadFile, 86
 - DownloadFileAsync, 122
 - DoWork, 149
 - EndAcceptTcpClient, 66
 - FormClosing, 145, 185
 - GetDirectories, 121
 - GetHostEntry, 71
 - GetHostName, 71
 - GetLocalResourceObject, 237
 - GetObject, 194
 - host.AddDefaultEndpoints, 285
 - inicjalizująca połączenie .NET Remoting, 198
 - InitializeComponent, 26
 - InitializeCulture, 240
 - Invoke, 149
 - IsValid, 230
 - kasująca plik, 132
 - KoniecPing, 79
 - Main(), 28
 - MessageBox.Show, 56, 87
 - nawiązująca połączenie z serwerem, 183
 - obsługująca połączenie z serwerem, 164
 - odbierająca wiadomości, 147, 182
 - odwołująca się do własności kontrolki, 173
 - pobierająca plik, 135
 - pobierająca wiadomości, 208
 - pobierająca zrzut ekranu, 162, 170
 - PrzerwijPobieranie, 138
 - przesyłająca wiadomość, 165, 176
 - Receive, 63, 147
 - Refresh, 99
 - RegisterChannel, 192
 - RegisterWellKnownServiceType, 192
 - Scroll, 30
 - SelectedIndex, 75
 - Send, 77
 - SendAsync, 77
 - SendPingAsync, 84
 - SetCompatibleTextRenderingDefault, 28
 - SetListBoxText, 66
 - SetTextHTML, 154
 - ToShortDateString, 41
 - ToString, 41
 - Transfer, 242
 - UnregisterChannel, 209
 - uruchamiająca serwer, 191
 - ustanawiająca połączenie, 118
 - wczytująca listę kontaktów, 141
 - wprowadzająca tagi, 97
 - Write, 99
 - WypiszTekst, 175
 - wysyłająca asynchronicznie plik, 127
 - zapisująca do pliku listę kontaktów, 142
 - zdarzeniowa, 17, 29, 30
 - ASP.NET, 40
 - zdarzeniowa Click, 61, 94
 - zdarzeniowa DoubleClick, 107
 - zdarzeniowa DoWork, 147
 - zdarzeniowa KeyDown, 121
 - zdarzeniowa MouseDoubleClick, 121
 - zdarzeniowa Page_Load, 241
 - zdarzeniowa ProgressChanged, 147
 - zdarzeniowa RunWorkerCompleted, 147
 - zdarzeniowa UploadFileCompleted, 128
 - zdarzeniowa UploadProgressChanged, 128
 - zmieniająca język serwisu, 240
 - MEX, 279, 281
 - MFC, 21
 - Microsoft, 9
 - Microsoft Foundation Classes, 21
 - Microsoft Intermediate Language, 9
 - model DoD, 43
 - model klient-serwer, 49
 - model OSI, 43
 - model żądanie-odpowiedź, 86
 - modyfikacja Program.cs, 281
 - modyfikowanie własności kontrolki, 153, 159
 - Mono, 9
 - MouseDoubleClick, 121
 - MSIL, 9
 - MSMQ, 270
 - MsmqIntegrationBinding, 272
 - NetMsmqBinding, 272
 - MsmqIntegrationBinding, 272
 - MultiExtended, 140
 - Multiline, 88
- N**
- naciśnięcie przycisku, 29
 - nagłówek sieciowy, 45
 - namespace, 16
 - namespaces, 26

nasłuch, 283
 rejestracja nasłuchu, 283
 usuwanie nasłuchu, 283
 NAT, 49
 NavigateUrl, 227
 nazwy, 15
 net.msmq, 270
 net.pipe, 270
 net.tcp, 270
 NetDetect, 79
 NetMsmqBinding, 270, 272
 NetNamedPipeBinding, 272
 NetPeerTcpBinding, 272
 NetTcpBinding, 270, 272
 NetTcpContextBinding, 272
 NetworkInformation, 72
 NetworkStream, 45
 new, 12
 NotifyIcon, 144
 Now, 40
 NumericUpDown, 151
 numery portów, 47

O

obiekt int[], 16
 obiekt ServiceHost, 282
 obiekty, 11
 czas życia, 12
 inicjalizacja, 13
 tworzenie, 12
 Object Pascal, 12
 object-oriented programing, 13
 obsługa plików tekstowych, 92
 obsługa połączeń, 176
 obsługa wyjątków, 17
 oczekiwanie na pakiety UDP, 169
 oczekiwanie na połączenie w osobnym wątku, 147
 odczytywanie adresu IP, 60
 odczytywanie pliku tekstowego, 93
 odwołanie do usługi poprzez referencję, 292
 odwoływanie się do własności kontrolek formy z poziomu innego wątku, 149
 okno aplikacji, 25
 okno Login Tasks, 242
 okno openFileDialog, 94
 okno Solution Explorer, 23, 28, 35
 okno Toolbox, 23
 OLE DB, 212
 Opacity, 30
 opcja
 Inherits, 38
 Language, 38
 Place code in separate file, 37

runat, 38
 Start Debugging, 39
 xmlns, 38
 OpenFileDialog, 94, 112
 operacje na plikach, 92
 operator +, 14
 OSI, 43
 warstwa aplikacji, 44
 warstwa fizyczna, 44
 warstwa łącza danych, 44
 warstwa prezentacji, 44
 warstwa sesji, 44
 warstwa sieciowa, 44
 warstwa transportowa, 44
 otwarte porty, 67, 68
 override, 14

P

P2P, 50
 Page, 230
 Page_Load, 39, 40, 241
 pakiet ICMP, 81
 przerwanie wysyłania, 84
 rozpoczęcie wysyłania, 82
 pakiety ICMP, 218
 parametr
 AutoEventWireup, 38
 CodeFile, 38
 server, 38
 peer-to-peer, 49, 50
 PictureBox, 166
 ping, 74
 ASP.NET, 217
 tryb asynchroniczny, 78
 Place code in separate file, 37
 platforma .NET, 7
 plik
 App.config, 274
 CarInformationService.svc.cs, 286
 CarInformationWindowsService.cs, 287
 CarRentalHost.WinService.exe, 289
 Default.aspx, 37
 Default.aspx.cs, 37
 Form1.cs, 25
 Form1.Designer.cs, 26
 Form1.resx, 29
 Global.asax, 230
 ICarInformationService.cs, 286
 IService1.cs, 275
 Program.cs, 28, 70
 Service1.cs, 274, 287
 SVC, 286
 web.config, 286

- Web.Config, 40
 - welcome.aspx, 38
 - welcome.aspx.cs, 38, 39, 41
 - pliki
 - csproj, 29
 - odczytywanie, 93
 - projektu, 25
 - resx, 232
 - sln, 29
 - tekstowe, 92
 - tymczasowe, 99
 - XML, 222
 - XSL, 226
 - zasoby, 232
 - pliki konfiguracyjne, 200
 - *.config, 298
 - klienta TCP, 202
 - serwera, 199, 200
 - serwisu, 274
 - pobieranie plików
 - FTP, 122
 - HTTP, 85, 86, 135
 - na serwer, 220
 - pobranie źródła strony z serwera WWW, 88
 - poczta elektroniczna, 99
 - podejście obiektowe, 14
 - podgląd strony, 98
 - podpowiedzi, 265
 - pola, 13
 - pole protected, 13
 - polecenie using, 12
 - połączenia
 - hosta i klienta, 269
 - peer-to-peer, 50
 - sieciowe, 72
 - z bazą danych, 247, 250
 - z usługą Google Web API, 262
 - połączenie FTP, 112
 - połączenie TCP, 53
 - asynchroniczne, 64
 - klient, 53
 - serwer, 56
 - połączenie three-way handshake, 46
 - połączenie UDP, 61
 - klient, 61
 - serwer, 62
 - POP3, 45, 52
 - porty, 47
 - potoki, 270
 - pozycjonowanie kontrolek na stronie, 212
 - pozycja względna, 212
 - tabela, 214
 - pozycjonowanie relatywne, 213
 - prefiksy transportu, 270
 - prezentacja danych XML, 226
 - program Ping, 74
 - program Windows Form Designer, 27
 - Program.cs, 28
 - programowanie
 - aplikacji sieciowych, 43
 - klient-serwer, 49
 - obiektywne, 13
 - RAD, 17
 - w systemie Windows, 21
 - wielowątkowe, 147, 163
 - zorientowane obiektywne, 11
 - ProgressChanged(), 147
 - projekt
 - aplikacje Windows, 25
 - pliki, 25
 - strona ASP.NET, 37
 - projekt dotGNU, 9
 - projekt Mono, 9
 - projektowanie aplikacji, 15
 - Properties, 23
 - Properties Window, 23
 - protokoły, 45, 50
 - DNS, 45
 - FTP, 45, 51
 - HTTP, 51, 85, 271
 - ICMP, 50, 74
 - IP, 48
 - POP3, 52
 - SOAP, 255
 - SSL, 52
 - TCP, 45–46, 271, 272
 - UDP, 46, 47
 - przeciążanie operatora, 14
 - przeciwdziałanie zablokowaniu interfejsu, 77
 - przeglądarka WWW, 89
 - przejrzystość kodu, 15
 - przekierowanie do nowej strony, 227
 - przerwanie pobierania pliku, 137
 - przestrzeń nazw, 15, 26
 - System.Runtime.Serialization, 269
 - XML, 38
 - przesyłanie wiadomości, 165, 176
 - przezroczystość formy, 30
 - punkt końcowy, 269, 273, 284, 293
- R**
- RAD, 13, 21
 - RAD Studio, 13
 - Rapid Application Development, 21
 - RARP, 44
 - Receive(), 63, 147
 - referencja do serwisu, 293, 295

Refresh(), 99
 RegisterChannel(), 192
 RegisterWellKnownServiceType(), 192
 Regular Expression Editor, 229
 rejestracja usługi sieciowej, 261
 rejestracja użytkowników, 241, 245
 CreateUserWizard, 245
 Release, 29
 Remoting
 czat, 203
 klient HTTP, 193
 klient TCP, 197
 serwer HTTP, 188
 serwer TCP, 195
 RemotingConfiguration, 192
 RequiredFieldValidator, 228
 Resource File, 233
 RMI, 187
 role użytkownika, 246
 Roles, 246
 router, 48
 routing, 44
 rozłączenie sesji FTP, 133
 RPC, 255
 RunWorkerAsync(), 147
 RunWorkerCompleted(), 147

S

SaveFileDialog, 94, 167
 schowek, 95
 Screen.PrimaryScreen, 162
 Scroll, 30
 ScrollBars, 88
 SelectedIndex(), 75
 Self hosting, 281
 Send(), 77
 SendAsync(), 77
 SendAsyncCancel(), 84
 SendPingAsync(), 84
 Service Oriented Architecture, 267
 serwer, 38, 49, 242
 serwer centralny, 50
 serwer HTTP, 188
 kanał HTTP, 192
 uruchamianie, 191
 serwer IIS, 285
 serwer screenshot, 166
 bezpieczne odwoływanie się do własności
 kontrolki formy z poziomu innego
 wątku, 167
 interfejs, 166
 lista aktywnych klientów, 168
 pobranie zrzutu ekranu, 169
 serwer TCP, 56, 195
 inicjacja, 196
 inicjacja na podstawie pliku
 konfiguracyjnego, 200
 plik konfiguracyjny, 199
 serwer TcpListener, 163
 serwer UDP, 62
 Serwer Uśmiechu, 138
 blokowanie użytkownika, 141
 interfejs, 138
 lista kontaktów, 140
 odblokowanie użytkownika, 141
 wysyłanie danych do wielu odbiorców, 143
 serwer WWW, 40
 serwis Windows
 kod serwisu, 288
 tworzenie instalatora usługi, 289
 sesja, 230
 Session_Start(), 231
 SetCompatibleTextRenderingDefault(), 28
 SetScroll(), 154, 182
 SetScrollCallBack(), 153, 159, 182
 SetText(), 154
 SetTextCallBack(), 153, 159
 SetTextHTML(), 154, 182
 SetTextHTMLCallBack(), 182
 ShowInTaskBar, 144
 sieć, 7, 43
 adres IP, 48
 maska podsieci, 49
 modele warstwowe, 44
 porty, 47
 protokoły, 45
 przepływ informacji pomiędzy dwoma
 hostami, 45
 WWW, 51
 wymiana danych, 45
 sieć ARPANET, 46
 Simple Object Access Protocol, 255
 SingleCall, 188
 Singleton, 188, 206
 skanowanie portów, 67, 69
 SMTP, 45, 99
 SMTPClient, 101, 108
 SOA, 267
 SOAP, 255
 solucja, 273
 Solution Explorer, 23, 35
 spam, 106
 sprawdzanie dostępnych komputerów w sieci, 79
 sprawdzanie dostępnych uaktualnień, 85
 sprawdzanie poprawności wpisywanych
 danych, 228

sprawdzenie adresu IP komputera, 69
 SPX, 44
 SQL Database, 247
 SqlDataSource, 250
 StatusStrip, 112
 sterta, 12
 stos, 12
 StreamReader, 45, 93
 StreamWriter, 45, 93
 strona ASP.NET, 33
 metody zdarzeniowe, 40
 pliki projektu, 37
 strona internetowa, 33
 strona usługi sieciowej, 257
 strona WWW, 211
 strumienie, 45
 Style Builder, 216
 style wizualne, 28
 suwak, 29
 System.Collections, 16
 System.Drawing.Imaging, 162
 System.IO, 92
 System.Net.Mail, 100
 System.Net.NetworkInformation, 76
 System.Net.Socket, 53, 55
 System.Runtime.Remoting, 191
 System.Runtime.Serialization, 269
 System.ServiceModel, 268, 281, 286, 288
 System.ServiceModel.Configuration, 268
 System.ServiceModel.Description, 268
 System.ServiceModel.MsmqIntegration, 268
 System.ServiceModel.Security, 269
 System.Threading, 147
 System.Uri, 270
 System.Web.Services, 257
 System.Web.Services.Protocols, 257
 System.Web.UI.Page, 39
 System.Web.UI.WebControls, 15
 System.Windows.Forms, 15
 systemy rozproszone, 298
 szablon CarInformationService.svc, 286
 szablon WCF Service Application, 287
 szablon WCF Service Library, 273

Ś

ścieżka dostępu, 111
 śledzenie drogi pakietu ICMP, 81
 środowisko
 .NET, 12
 .NET Framework, 9
 Visual C# 2010 Express Edition, 21
 Visual Web Developer 2010 Express Edition,
 41, 251

T

tabele, 214
 TabIndex, 27, 96
 Table, 219
 tablice, 16
 tablice dynamiczne, 16
 tagi HTML, 95
 TCP/IP, 43–44, 46, 270
 klient, 53, 197
 NetNamedPipeBinding, 272
 NetPeerTcpBinding, 272
 NetTcpBinding, 272
 NetTcpContextBinding, 272
 połączenie, 53
 połączenie asynchroniczne, 64
 serwer, 56, 195
 TCPClient, 53, 55, 163
 TCPListner, 57, 64, 163
 TcpServerChannel, 196
 TCPServer, 60
 technologia .NET, 9
 testowanie usługi, 279
 Text, 24
 TextBox, 95, 111, 217
 pozycja karetki, 152
 this, 30
 this.Controls.Add, 27
 three-way handshake, 46
 throw, 18
 Toolbox, 23
 ToolStripStatusLabel, 112
 ToShortDateString(), 41
 ToString(), 41
 Traceroute, 81
 transakcje klient serwer, 188
 CAO (Client Activated Object), 188
 SingleCall, 188
 Singleton, 188
 Transfer(), 242
 TransformSource, 227
 trójkąt SOA, 298
 try, 18
 try/catch, 17, 68
 TTL, 75, 81
 tworzenie
 aplikacja WWW, 33
 baza danych, 247
 interfejs aplikacji, 22
 klasa, 113
 metoda zdarzeniowa, 31
 obiekt, 12
 plik zasobu, 234
 projekt, 23

tworzenie
 strona ASP.NET, 33
 usługa sieciowa, 256
 zrzut ekranu, 162
 tworzenie klienta, 290
 ChannelFactory, 290
 referencja, 292
 tworzenie Windows Service, 288
 typy ogólne, 11

U

UDDI, 255
 udostępnianie usługi, 280
 UDP, 44, 46, 47
 klient, 61
 przesyłanie wiadomości, 165
 serwer, 62
 UDPCClient, 62
 UICulture, 235
 umieszczenie ikony w zasobniku
 systemowym, 144
 UnregisterChannel(), 209
 UploadFileChanged, 129
 UploadFileCompleted, 128
 UploadProgressChanged, 128
 URI, 86
 uruchamianie usługi, 279
 urządzenia przenośne, 10
 UseCompatibleTextRendering, 28
 using, 12
 usługi sieciowe, 10, 188, 255, 256
 dodawanie metody, 258
 HTTP, 255
 inicjacja, 261
 korzystanie, 259
 odpowiedź z serwera, 258
 opis, 255
 referencja, 260
 rejestracja usługi, 261
 SOAP, 255
 standardy, 255
 strona informacyjna, 258
 UDDI, 255, 256
 WSDL, 255
 wymiana informacji, 255
 wyszukiwanie, 256
 usuwanie plików *.svc.cs, 287
 uwierzytelnianie, 99

V

Validation, 228
 ValidationExpression, 229

Value, 24
 ValueToCompare, 230
 Visual Basic, 37, 38
 Visual C# 2010 Express Edition, 21
 ekran startowy, 22
 IntelliSense, 30
 pliki projektu, 25
 projekt, 24
 projekt Windows Forms, 22
 Properties, 23
 Properties Window, 23
 Solution Explorer, 23
 Toolbox, 23
 widok formy, 26
 widok kodu, 26
 właściwości, 23
 Visual C# Express Edition, 22
 Visual Studio, 27
 Visual Web Developer 2010 Express Edition, 33,
 211
 ASP.NET Web Site, 34
 Debug, 39
 IntelliSense, 36
 język programowania, 34
 New Web Site, 34
 obiekty HTML, 34
 pliki projektu, 37
 serwer WWW, 40
 Solution Explorer, 35
 strona ASP.NET, 33
 Toolbox, 34
 Web Forms, 34
 widok kodu HTML, 34
 widok projektu, 37
 wybór elementu serwisu, 35
 VWD, 251

W

walidacja danych, 228
 WAN, 46
 warstwa hostowania, 287
 warstwa implementacji, 287
 warstwy modelu OSI, 43
 wątek obsługujący połączenie, 179
 wątki, 28, 147
 WCF, 267, 269
 adres, 269
 definiowanie kontraktu, 273
 konfiguracja, 274
 kontrakt, 269
 tworzenie klienta, 290
 udostępnianie usługi, 280
 wiązanie, 269

- WCF = E = A + B + C, 269
- WCF Service Host, 279
- WCF Service Library, 273, 279
- WCF Test Client, 279, 287, 290
- wcftestclient, 279
- WEB, 285
- Web Forms, 10, 34, 39, 211
- Web Services, 10
- WebBrowser, 89, 90, 150
- WebBrowserShortcutsEnabled, 151, 181
- WebClient, 86
- WebHttpBinding, 271
- WebRequest, 86
- WebRequestMethods.Ftp, 117
- wersja językowa serwisu, 235
- węzeł <services>, 278
- white pages, 256
- wiadomości e-mail, 99
- wiązanie
 - kodowanie, 270
 - opis protokołu, 270
 - transport, 271
- wiązanie ze sobą zdarzeń, 97
- wielkość liter, 28
- wielojęzyczny serwis internetowy, 232
 - kultura, 235
 - plik zasobów, 234
 - wybór języka przez użytkownika, 239
 - zasoby globalne, 237
 - zasoby lokalne, 232
- wielowątkowość, 147
- Windows Application, 22
- Windows Communication Foundation, 269,
Patrz WCF
- Windows Form Designer generated code, 27
- Windows Forms, 10, 22, 23
- WindowState, 144
- wirtualna maszyna Javy, 39
- własność
 - AutoPostBack, 240
 - BalloonTipIcon, 144
 - BalloonTipTitle, 144
 - Bold, 24
 - CancelDestinationPageUrl, 245
 - ContextMenuStrip, 151
 - ContinueDestinationPageUrl, 245
 - ControlToCompare, 230
 - ControlToValidate, 228
 - CreateUserText, 245
 - CreateUserUrl, 245
 - Culture, 235
 - DataSourceID, 253
 - DataTextField, 253
 - DataValueField, 253
 - DocumentSource, 227
 - ErrorMessage, 228
 - HeaderText, 252
 - InvokeRequired, 149
 - IsWebBrowserContextMenuEnabled, 151
 - KeyPreview, 31
 - Maximum, 24
 - Multiline, 88
 - NavigateUrl, 227
 - Now, 40
 - Opacity, 30
 - ScrollBars, 88
 - SelectionMode, 140
 - ShowInTaskBar, 144
 - Text, 24, 144
 - TransformSource, 227
 - UICulture, 235
 - ValidationExpression, 229
 - Value, 24
 - ValueToCompare, 230
 - WebBrowserShortcutsEnabled, 151
 - WindowState, 144
- WMI, 10
- wprowadzanie tagów, 95
- Write(), 99
- WSDL, 192, 255
- WSDualHttpBinding, 271
- WSFederationHttpBinding, 271
- WsHttpBinding, 270
- WSHttpBinding, 271
- WSHttpContextBinding, 271
- wskazniki, 17
- WWW, 51
- wybór języka przez użytkownika, 239
- wybór ścieżki dostępu, 111
- wycieki pamięci, 12
- wyjątki, 17
 - catch, 18
 - finally, 18
 - obsługa, 18
 - throw, 18
 - try/catch, 17
 - zgłaszanie, 18
- wyłączenie MEX, 281
- wymiana plików, 50, 51
- wyrażenia regularne, 229
- wyróżniony blok programu, 27
- WYSIWYG, 212
- wysyłanie informacji o dostępności klienta, 165
- wysyłanie pakietu ICMP, 82, 218
- wysyłanie pliku na serwer FTP, 128
- wysyłanie wiadomości, 138, 178

wysyłanie wiadomości e-mail, 99
 ASP.NET, 218
 asynchroniczne wysyłanie, 108
 bez uwierzytelnienia, 99
 format HTML, 102
 masowe wysyłanie wiadomości, 106
 pole odbiorców ukrytych wiadomości, 102
 sformatowane wiadomości
 z załącznikami, 102
 uwierzytelnienie, 105
 załączniki, 104, 221
 wyświetlanie zawartości pliku XML, 226
 wywołania zwrotne, 17
 wywołanie asynchroniczne usługi, 295
 wywołanie metody, 125
 wywołanie synchroniczne, 297
 wywołanie usługi, 294

X

XHTML, 36
 XHTML 1.0 Transitional, 38
 XML, 10, 199, 227, 272
 dodawanie elementu, 224
 wyświetlanie zawartości pliku, 226
 XmlDocument, 225
 XmlElement, 225
 xmlns, 38
 XSL, 226

Y

yellow pages, 256

Z

zamknięcie aplikacji, 145
 zapisywanie rozmowy do pliku, 152
 zaporą sieciowa, 51
 zasobnik systemowy, 144

zasoby, 29, 232
 zasoby globalne, 237
 zasoby lokalne, 232
 zdalny screenshot, 162
 klient, 162
 połączenie z serwerem, 164
 serwer, 166
 wysyłanie informacji o dostępności
 klienta, 165
 zrzut ekranu, 162
 zdarzenie, 17
 DownProgressChangedEventHandler, 123
 EventArgs, 31
 KeyDown, 31
 KeyEventArgs, 31
 KeyUp, 97
 metody zdarzeniowe, 17
 MouseDown, 97
 Page_Load, 39, 40
 Scroll, 30
 UploadProgressChangedEventHandler, 128
 zgłaszanie wyjątku, 18
 zmiana języka serwisu internetowego, 240
 zmiana katalogu, 120
 zmienna
 downloadCompleted, 115
 ftpDirectory, 115
 uploadCompleted, 115
 znacznik transport, 270
 http, 270
 net.msmq, 270
 net.pipe, 270
 net.tcp, 270
 znaczniki XHTML, 36
 zrzut ekranu, 162

Ź

źródło strony WWW, 88

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**

Znaczna część współczesnego życia na stałe przeniosła się do Internetu. Nikt dziś nie wyobraża już sobie świata, w którym nie można sprawdzić pogody, zrobić zakupów czy szybko znaleźć potrzebnych informacji — właśnie w sieci. A skoro tak, wciąż wzrasta zapotrzebowanie na programistów mogących zaspokoić oczekiwania rzeszy klientów oraz dostarczyć im wygodnych, świetnie działających aplikacji sieciowych, skrojonych na potrzeby konkretnych użytkowników. To właśnie było źródłem niezwykle popularności poprzedniej książki Sławomira Orłowskiego, w praktyczny sposób przekazującej wiedzę na temat sposobów tworzenia takich aplikacji i spełniania różnych warunków związanych ze specyfiką określonych projektów programistycznych.

C#. Tworzenie aplikacji sieciowych. Gotowe projekty to podręcznik przeznaczony dla osób z choćby pobieżną znajomością zasad programowania, które chcą rozwinąć i wykorzystać w praktyce swoje umiejętności. Autorzy tłumaczą, dlaczego wybrali język C#, a także wskazują możliwości i zalety platformy .NET oraz środowiska Visual Studio. Następnie skupiają się na projektach obejmujących wszystkie aspekty komunikacji internetowej, od wykorzystania różnych protokołów sieciowych, przez budowę interfejsu, wysyłanie e-maili, obsługę FTP, komunikatorów, czatów, aż po kontaktowanie się z bazami danych, tworzenie wielojęzycznych serwerów i udostępnianie usługi. Jeśli chcesz szybko i bez kłopotu podszkolić się w zakresie aplikacji sieciowych, by w krótkim czasie osiągnąć spektakularne efekty, nie znajdziesz nic lepszego!

- Język C# i platforma .NET
- Visual C# 2010 Express Edition. Opis środowiska
- Visual Web Developer 2010 Express Edition. Opis środowiska
- Programowanie sieciowe
- Aplikacje TCP i UDP
- Remoting
- ASP.NET i ADO.NET
- Web Services
- WCF — ponad transportem

Aplikacje sieciowe — wykorzystaj potencjał Internetu!

Nr katalogowy: 6822



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>

helion.pl
księgarnia
internetowa

Cena: 49,00 zł

ISBN 978-83-246-2910-7



Informatyka w najlepszym wydaniu