

O'REILLY®

Wydanie VII



C# 7.0 w pigułce

Helion 

Joseph Albahari, Ben Albahari

Tytuł oryginału: C# 7.0 in a Nutshell: The Definitive Reference, 7th edition

Tłumaczenie: Łukasz Piwko, z wykorzystaniem fragmentów książki "C# 6.0 w pigułce. Wydanie VI" w tłumaczeniu Roberta Górczyńskiego i Jakuba Hubisza

ISBN: 978-83-283-4075-6

© 2018 Helion S.A.

Authorized Polish translation of the English edition of C# 7.0 in a Nutshell
ISBN 9781491987650 © 2018 Joseph Albahari, Ben Albahari

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

All rights reserved. No part of this book may be reproduced or transmitted in any form
or by any means, electronic or mechanical, including photocopying, recording or by
any information storage retrieval system, without permission from the Publisher.

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

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

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.

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/c7pig7>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

<ftp://ftp.helion.pl/przyklady/c7pig7.zip>

Printed in Poland.

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

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

Spis treści

Wstęp	11
1. Wprowadzenie do C# i .NET Framework	15
Obiektość	15
Bezpieczeństwo typów	16
Zarządzanie pamięcią	17
Platformy	17
Powiązania C# z CLR	17
CLR i .NET Framework	18
Inne platformy	18
Historia C# w pigułce	22
2. Podstawy języka C#	29
Pierwszy program w języku C#	29
Składnia	32
Podstawy typów	34
Typy liczbowe	42
Typ logiczny i operatory	50
Łańcuchy znaków i pojedyncze znaki	52
Tablice	54
Zmienne i parametry	58
Wyrażenia i operatory	67
Operatory null	72
Instrukcje	73
Przestrzenie nazw	82

3. Tworzenie typów w języku C#	89
Klasy	89
Dziedziczenie	104
Typ object	113
Struktury	117
Modyfikatory dostępu	118
Interfejsy	120
Wyliczenia	125
Typy zagnieżdżone	128
Typy generyczne	130
4. Zaawansowane elementy języka C#	143
Delegaty	143
Zdarzenia	151
Wyrażenia lambda	158
Metody anonimowe	162
Instrukcje try i wyjątki	163
Wyliczanie i iteratory	171
Typy wartościowe dopuszczające wartość null	176
Metody rozszerzające	181
Typy anonimowe	184
Krotki (C# 7)	185
Atrybuty	188
Atrybuty informacji wywołującego	190
Wiązanie dynamiczne	192
Przeciążanie operatorów	199
Niebezpieczny kod i wskaźniki	202
Dyrektywy preprocesora	206
Dokumentacja XML	208
5. Ogólny zarys platformy	213
.NET Standard 2.0	215
CLR i rdzeń platformy	217
Technologie praktyczne	221
6. Podstawowe wiadomości o platformie	229
Obsługa łańcuchów i tekstu	229
Data i godzina	242
Daty i strefy czasowe	249

Formatowanie i parsowanie	255
Standardowe łańcuchy formatu i flagi parsowania	260
Inne mechanizmy konwersji	267
Globalizacja	271
Praca z liczbami	272
Wyliczenia	276
Struktura Guid	279
Sprawdzanie równości	280
Określanie kolejności	289
Klasy pomocnicze	293
7. Kolekcje	297
Przeliczalność	297
Interfejsy ICollection i IList	304
Klasa Array	308
Listy, kolejki, stosy i zbiory	316
Słowniki	324
Kolekcje i pośredniki z możliwością dostosowywania	330
Dołączanie protokołów równości i porządkowania	336
8. Zapytania LINQ	343
Podstawy	343
Składnia płynna	345
Wyrażenia zapytań	351
Wykonywanie opóźnione	355
Podzapytania	361
Tworzenie zapytań złożonych	364
Strategie projekcji	368
Zapytania interpretowane	370
LINQ to SQL i Entity Framework	376
Budowanie wyrażeń zapytań	389
9. Operatory LINQ	395
Informacje ogólne	396
Filtrowanie	399
Projekcja	403
Łączenie	414
Porządkowanie	421
Grupowanie	424

Operatory zbiorów	427
Metody konwersji	428
Operatory elementów	431
Metody agregacyjne	433
Kwantyfikatory	437
Metody generujące	438
10. LINQ to XML	441
Przegląd architektury	441
Informacje ogólne o X-DOM	442
Tworzenie drzewa X-DOM	445
Nawigowanie i wysyłanie zapytań	448
Modyfikowanie drzewa X-DOM	452
Praca z wartościami	455
Dokumenty i deklaracje	458
Nazwy i przestrzenie nazw	461
Adnotacje	466
Projekcja do X-DOM	467
11. Inne technologie XML	473
Klasa XmlReader	474
Klasa XmlWriter	482
Typowe zastosowania klas XmlReader i XmlWriter	484
XSD i sprawdzanie poprawności schematów	488
XSLT	491
12. Zwalnianie zasobów i mechanizm usuwania nieużytków	493
IDisposable, Dispose i Close	493
Automatyczne usuwanie nieużytków	499
Finalizatory	501
Jak działa mechanizm usuwania nieużytków?	506
Wycieki pamięci zarządzanej	511
Słabe odwołania	515
13. Diagnostyka i kontrakty kodu	519
Kompilacja warunkowa	520
Debugowanie i klasy monitorowania	523
Integracja z debuggerem	526
Procesy i wątki procesów	528

Klasy StackTrace i StackFrame	529
Dziennik zdarzeń Windows	530
Liczniki wydajności	533
Klasa Stopwatch	538
14. Współbieżność i asynchroniczność	539
Wprowadzenie	539
Wątkowanie	540
Zadania	557
Reguły asynchroniczności	565
Funkcje asynchroniczne w języku C#	570
Wzorce asynchroniczności	586
Wzorce uznane za przestarzałe	594
15. Strumienie i wejście-wyjście	599
Architektura strumienia	599
Użycie strumieni	601
Adapter strumienia	614
Kompresja strumienia	622
Praca z plikami w postaci archiwum ZIP	624
Operacje na plikach i katalogach	625
Plikowe operacje wejścia-wyjścia w UWP	635
Mapowanie plików w pamięci	638
Odizolowany magazyn danych	641
16. Sieć	643
Architektura sieci	643
Adresy i porty	646
Adresy URI	647
Klasy po stronie klienta	649
Praca z HTTP	661
Utworzenie serwera HTTP	667
Użycie FTP	670
Użycie DNS	672
Wysyłanie poczty elektronicznej za pomocą SmtplibClient	673
Użycie TCP	673
Otrzymywanie poczty elektronicznej POP3 za pomocą TcpClient	677
TCP w środowisku uruchomieniowym Windows	679

17. Serializacja	681
Koncepcje serializacji	681
Serializacja kontraktu danych	685
Kontrakty danych i kolekcje	695
Rozszerzenie kontraktu danych	697
Serializacja binarna	700
Atrybuty serializacji binarnej	702
Serializacja binarna za pomocą ISerializable	706
Serializacja XML	709
18. Zestawy	719
Co znajduje się w zestawie?	719
Silne nazwy i podpisywanie zestawu	724
Nazwy zestawów	727
Technologia Authenticode	730
Global Assembly Cache	734
Zasoby i zestawy satelickie	736
Wyszukiwanie i wczytywanie zestawów	745
Wdrażanie zestawów poza katalogiem bazowym	750
Umieszczenie w pojedynczym pliku wykonywalnym	751
Praca z zestawami, do których nie ma odwołań	753
19. Refleksja i metadane	755
Refleksja i aktywacja typów	756
Refleksja i wywoływanie składowych	762
Refleksja dla zestawów	774
Praca z atrybutami	775
Generowanie dynamicznego kodu	781
Emitowanie zestawów i typów	787
Emitowanie składowych typów	791
Emitowanie generycznych typów i klas	796
Kłopotliwe cele emisji	798
Parsowanie IL	801
20. Programowanie dynamiczne	807
Dynamiczny system wykonawczy języka	807
Unifikacja typów liczbowych	809
Dynamiczne wybieranie przeciążonych składowych	810
Implementowanie obiektów dynamicznych	816
Współpraca z językami dynamicznymi	819

21. Bezpieczeństwo	821
Code Access Security	821
Tożsamość i role	822
Zabezpieczenia systemu operacyjnego	825
Kryptografia	827
Windows Data Protection	828
Obliczanie skrótów	829
Szyfrowanie symetryczne	831
Szyfrowanie kluczem publicznym i podpisywanie	835
22. Zaawansowane techniki wielowątkowości	839
Przegląd technik synchronizacji	840
Blokowanie wykluczające	840
Blokady i bezpieczeństwo ze względu na wątki	848
Blokowanie bez wykluczania	854
Sygnalizacja przy użyciu uchwytów zdarzeń oczekiwania	859
Klasa Barrier	867
Leniwa inicjalizacja	868
Pamięć lokalna wątku	871
Metody Interrupt i Abort	873
Metody Suspend i Resume	874
Zegary	875
23. Programowanie równoległe	879
Dlaczego PFX?	879
PLINQ	882
Klasa Parallel	895
Równoległe wykonywanie zadań	901
Klasa AggregateException	910
Kolekcje współbieżne	913
Klasa BlockingCollection<T>	916
24. Domeny aplikacji	921
Architektura domeny aplikacji	921
Tworzenie i likwidowanie domen aplikacji	923
Posługiwanie się wieloma domenami aplikacji	924
Metoda DoCallback	926
Monitorowanie domen aplikacji	927
Domeny i wątki	927
Dzielenie danych między domenami	929

25. Współdziałanie macierzyste i poprzez COM	935
Odwołania do natywnych bibliotek DLL	935
Szeregowanie	936
Wywołania zwrotne z kodu niezarządzanego	939
Symulowanie unii C	939
Pamięć współdzielona	940
Mapowanie struktury na pamięć niezarządzaną	943
Współpraca COM	946
Wywołanie komponentu COM z C#	948
Osadzanie typów współpracujących	952
Główne moduły współpracujące	952
Udostępnianie obiektów C# dla COM	953
26. Wyrażenia regularne	955
Podstawy wyrażeń regularnych	956
Kwantyfikatory	960
Asercje o zerowej wielkości	961
Grupy	964
Zastępowanie i dzielenie tekstu	965
Receptury wyrażeń regularnych	967
Leksykon języka wyrażeń regularnych	970
27. Kompilator Roslyn	975
Architektura Roslyn	976
Drzewa składni	977
Kompilacja i model semantyczny	991
Skorowidz	1003



Ogólny zarys platformy

Prawie wszystkie elementy funkcjonalności platformy .NET Framework są udostępniane przez szeroki wachlarz typów zarządzanych. Typy te są posegregowane w hierarchicznych przestrzeniach nazw i zapakowane w zbiór zestawów, które to zestawy, wraz z systemem wykonawczym CLR (ang. *Common Language Runtime*), stanowią platformę .NET.

Niektóre typy .NET są bezpośrednio wykorzystywane przez CLR i niezbędne do działania środowiska zarządzanego. Typy te znajdują się w zestawie o nazwie *mscorlib.dll* i obejmują np. wszystkie typy wbudowane języka C#, jak również: podstawowe klasy kolekcyjne, typy do przetwarzania strumieni, serializacyjne, refleksyjne, wątkowe oraz interoperacyjne (nazwa *mscorlib* to skrót od ang. słów *Multi-language Standard Common Object Runtime Library*).

Na wyższym poziomie znajdują się dodatkowe typy, na których opiera się funkcjonalność CLR i które zapewniają obsługę formatu XML, sieci oraz LINQ. Znajdują się one w zestawach: *System.dll*, *System.Xml.dll* oraz *System.Core.dll*. Wraz z zestawem *mscorlib* reprezentują bogate środowisko programistyczne, na bazie którego zbudowana jest cała platforma. Pozostała część książki jest w głównej mierze poświęcona właśnie temu „rdzeniowi platformy”.

Reszta platformy .NET Framework obejmuje praktyczne interfejsy API, których większość można zaliczyć do jednego z trzech obszarów funkcjonalności:

- technologie dotyczące interfejsu użytkownika;
- technologie zapleczone;
- technologie dotyczące systemów rozproszonych.

W tabeli 5.1 przedstawiono poziomy wzajemnej zgodności różnych wersji języka C#, CLR oraz .NET Framework.

W tym rozdziale znajduje się zwięzły przegląd wszystkich najważniejszych składników platformy .NET Framework — od podstawowych typów opisywanych w tej książce po technologie stosowane.

Tabela 5.1. Wersje C#, CLR i .NET Framework

Wersja C#	Wersja CLR	Wersja platformy
1.0	1.0	1.0
1.2	1.1	1.1
2.0	2.0	2.0, 3.0
3.0	2.0 (SP2)	3.5
4.0	4.0	4.0
5.0	4.5 (udoskonalony CLR 4.0)	4.5
6.0	4.6 (udoskonalony CLR 4.0)	4.6
7.0	4.6/4.7 (udoskonalony CLR 4.0)	4.6/4.7

Co nowego w .NET Framework 4.6

- Programista ma większe możliwości w zakresie kontrolowania procesu usuwania nieużytków dzięki nowym metodom klasy GC. Dodano też kilka opcji wywoływania GC.Collect.
- Dodano nowy, szybszy, 64-bitowy kompilator JIT.
- W przestrzeni nazw System.Numerics znalazły się obsługiwane sprzętowo typy macierzowe i wektorowe, BigInteger oraz Complex.
- Dodano klasę System.AppContext, dzięki której twórcy bibliotek mogą umożliwiać konsumentom włączanie i wyłączanie nowych funkcji API.
- Zadania podczas tworzenia pobierają kulturę bieżącego wątku i interfejsu użytkownika.
- Zwiększono liczbę typów kolekcyjnych implementujących interfejs IReadOnlyCollection<T>.
- Wprowadzono dalsze udoskonalenia w WPF, m.in. poprawiono obsługę ekranów dotykowych i o wysokiej rozdzielczości DPI.
- ASP.NET obsługuje HTTP/2 oraz Token Binding Protocol w Windows 10.



Zestawy i przestrzenie nazw należące do platformy .NET Framework są *pomieszane*. Najbardziej jaskrawymi przykładami są zestawy *mscorlib.dll* i *System.Core.dll* zawierające definicje typów należących do kilkudziesięciu przestrzeni nazw, z których żadna nie ma przedrostka *mscorlib* ani *System.Core*. Ale są też mniej oczywiste przykłady, które powodują więcej nieporozumień, np. typy w przestrzeni *System.Security.Cryptography*. Większość typów należących do tej przestrzeni znajduje się w zestawie *System.dll*, ale niektóre umieszczono w *System.Security.dll*. Na stronie internetowej oryginalnego wydania tej książki można znaleźć informacje (w języku angielskim) na temat tego, które przestrzenie nazw odpowiadają poszczególnym zestawom (<http://www.albahari.com/nutshell/namespaceReference.aspx>).

Wiele rdzennych typów jest zdefiniowanych w następujących zestawach: *mscorlib.dll*, *System.dll* oraz *System.Core.dll*. Zestaw *mscorlib.dll* zawiera typy wymagane przez samo środowisko wykonawcze. Zestawy *System.dll* i *System.Core.dll* zawierają dodatkowe typy potrzebne programiście. Powody istnienia tych dwóch zestawów zamiast jednego mają podłoże historyczne. Firma Microsoft wprowadziła platformę .NET Framework 3.5 jako *dodatek* w tym sensie, że działała ona jako warstwa nad istniejącym CLR 2.0. Dlatego prawie wszystkie nowe typy rdzenne (np. klasy obsługujące LINQ) dodano do nowego zestawu, któremu nadano nazwę *System.Core.dll*.

Co nowego w .NET Framework 4.7

Framework 4.7 to wersja zawierająca więcej udoskonaleń niż nowych funkcji. Poprawiono w niej wiele błędów i wprowadzono liczne usprawnienia, a ponadto:

- Struktura *System.ValueTuple* należy do Framework 4.7, więc w C# 7 można używać krotek bez odwoływania się do zestawu *System.ValueTuple.dll*.
- Poprawiono obsługę funkcji dotykowych w WPF.
- Poprawiono obsługę monitorów o wysokiej rozdzielczości w Windows Forms.

.NET Standard 2.0

W rozdziale 1. opisaliśmy trzy główne alternatywy dla .NET Framework w programowaniu wieloplatformowym:

- UWP dla urządzeń i komputerów z systemem Windows 10.
- .NET Core/ASP.NET Core dla Windows, Linux i MacOS.
- Xamarin dla urządzeń mobilnych (iOS, Android i Windows 10).

Dobra wiadomość jest taka, że w .NET Core 2.0 te środowiska — wraz z .NET Framework 4.6.1 i nowszymi — upodobniły się do siebie pod względem funkcjonalności i teraz wszystkie oferują podstawową bibliotekę klas (ang. *base class library* — BCL) zawierającą podobne typy i składowe. Podobieństwu tym nadano formalny kształt w postaci nowego standardu o nazwie **.NET Standard 2.0**.

Pisząc bibliotekę w Visual Studio 2017, jako platformę docelową można wybrać .NET Standard 2.0 zamiast któregośkolwiek spośród bardziej konkretnych środowisk. Dzięki temu biblioteka będzie przenośna i jeden zestaw będzie działał bez modyfikacji w (nowych wersjach) wszystkich czterech środowisk.



.NET Standard to nie środowisko, lecz specyfikacja opisująca minimalną podstawową funkcjonalność (typy i składowe), która gwarantuje zgodność z określonym zbiorem środowisk. Koncepcja ta jest podobna do koncepcji interfejsów w C#. .NET Standard jest jak interfejs, który może być implementowany przez konkretne typy (środowiska).

W tej książce znajduje się opis większości składników .NET Standard 2.0.

Starsze standardy .NET Standard

W użyciu znajdują się także starsze standardy .NET Standard, a konkretnie 1.1, 1.2, 1.3 oraz 1.6. Standard o wyższym numerze wersji zawsze jest nadzbiorem poprzedniego standardu. Gdybyśmy na przykład pisali bibliotekę dla .NET Standard 1.6, to obsługiwałaby ona nie tylko nowe wersje czterech najważniejszych środowisk, ale również .NET Core 1.0. A gdybyśmy pisali dla .NET Standard 1.3, obsługiwalibyśmy wszystko, o czym była już mowa, plus .NET Framework 4.6.0 (zobacz tabelę 5.2).

Tabela 5.2. Starsze standardy .NET Standard

Standard docelowy	Dodatkowa obsługa
Standard 1.6	.NET Core 1.0
Standard 1.3	Powyżej plus .NET 4.6.0
Standard 1.2	Do .NET 4.5.1, Windows Phone 8.1, WinRT dla Windows 8.1
Standard 1.1	Do .NET 4.5.0, Windows Phone 8.0, WinRT dla Windows 8.0



Standardy 1.x są pozbawione tysięcy API, które zawiera wersja 2.0, w tym wielu składników opisanych w tej książce. Dlatego pisanie programu dla standardu 1.x może okazać się znacznie trudniejsze, w szczególności jeśli dodatkowo trzeba będzie zintegrować już istniejący kod lub biblioteki.

Jeśli interesują Cię starsze środowiska, ale nie potrzebujesz zgodności międzyplatformowej, to lepszym rozwiązaniem jest obranie za cel starszej wersji konkretnego środowiska. W przypadku systemu Windows dobrym wyborem jest .NET Framework 4.5, ponieważ to środowisko jest szeroko rozpowszechnione (zainstalowane we wszystkich komputerach z systemem Windows 8 i nowszymi) i zawiera większość składników .NET Framework 4.7.

Standard .NET Standard można też traktować jako swego rodzaju najmniejszy wspólny mianownik. Jeśli chodzi o .NET Standard 2.0, to cztery środowiska, które go implementują, mają podobną bibliotekę Base Class Library, dzięki czemu ich najmniejszy wspólny mianownik jest duży i użyteczny. Gdybyśmy jednak dodatkowo potrzebowali zgodności z .NET Core 1.0 (o bardzo okrojonej BCL), to najmniejszy wspólny mianownik — .NET Standard 1.x — byłby jeszcze mniejszy i mniej użyteczny.

Zestawy referencyjne

Podczas kompilowania programu należy wskazać zestawy zawierające potrzebne w tym programie części środowiska. Na przykład prosty program konsolowy dla platformy .NET Framework, zawierający zapytanie LINQ do XML, potrzebowałby zestawów *mscorlib.dll*, *System.dll*, *System.Xml.dll*, *System.Xml.Linq.dll* oraz *System.Core.dll*.

W Visual Studio zestawy wskazuje się poprzez dodawanie odwołań do projektu (wymienione powyżej zestawy są dodawane automatycznie przy tworzeniu projektów przeznaczonych dla środowiska .NET Framework 4.x). Wskazane przez programistę zestawy są potrzebne tylko kompilatorowi i nie muszą być takie same jak te wykorzystywane w czasie działania programu. Dlatego można też używać specjalnych zestawów referencyjnych, które mają postać pustych skorup niezawierających

żadnego skompilowanego kodu. Tak właśnie działa .NET Standard: dodajemy zestaw referencyjny o nazwie *netstandard.dll*, który zawiera wszystkie dozwolone typy i składowe .NET Standard 2.0 (ale bez jakiegokolwiek skompilowanego kodu). Już w czasie działania programu następuje załadowanie „prawdziwych” zestawów za pomocą atrybutów przekierowania zestawów. (Wybór „prawdziwych” zestawów zależy od tego, w którym konkretnie środowisku nastąpi uruchomienie).

Zestawy referencyjne umożliwiają także pisanie programów przeznaczonych dla niższych wersji środowiska, niż jest zainstalowane w komputerze. Jeśli na przykład mamy zainstalowane .NET Framework 4.7 i Visual Studio 2017, to nadal możemy utworzyć projekt przeznaczony dla .NET Framework 4.0. Dzięki zbiorowi zestawów referencyjnych Framework 4.0 nasz projekt będzie używał tylko typów i składowych z Framework 4.0.

CLR i rdzeń platformy

Typy systemowe

Najbardziej podstawowe typy znajdują się bezpośrednio w przestrzeni nazw System. Zaliczają się do nich typy wbudowane C#, klasy bazowe Exception, Enum, Array i Delegate oraz Nullable, Type, DateTime, TimeSpan i Guid. Ponadto przestrzeń nazw System zawiera typy do wykonywania funkcji matematycznych (Math), generowania liczb losowych (Random) oraz dokonywania konwersji między różnymi typami (Convert i BitConverter).

Typy te zostały opisane w rozdziale 6. razem z interfejsami definiującymi standardowe protokoły wykorzystywane w .NET Framework do wykonywania takich czynności jak formatowanie (IFormatable) czy porównywanie w celu określenia kolejności (IComparable).

Dodatkowo w przestrzeni nazw System zdefiniowano interfejs IDisposable i klasę GC do współpracy z systemem usuwania nieużytków. Szerzej na te tematy piszemy w rozdziale 12.

Przetwarzanie tekstu

Przestrzeń nazw System.Text zawiera klasę StringBuilder (umożliwiającą edytowanie kuzynkę klasy string) i typy do pracy z różnymi technikami kodowania tekstu, np. UTF-8 (Encoding i podtypy). Szerzej na ten temat piszemy w rozdziale 6.

Przestrzeń nazw System.Text.RegularExpressions zawiera typy do wykonywania skomplikowanych operacji wyszukiwania i zastępowania tekstu na podstawie wzorców. Ich dokładniejszy opis zamieściliśmy w rozdziale 26.

Kolekcje

Platforma .NET Framework zawiera wiele klas do przechowywania kolekcji elementów. Można wśród nich znaleźć zarówno struktury listowe, jak i słownikowe, które współpracują z zestawem standardowych interfejsów unifikujących ich wspólne cechy. Wszystkie typy kolekcyjne, opisane w rozdziale 7., są zdefiniowane w następujących przestrzeniach nazw:

```
System.Collections // kolekcje niegeneryczne
System.Collections.Generic // kolekcje generyczne
System.Collections.Specialized // kolekcje silnie typizowane
System.Collections.ObjectModel // klasy bazowe do tworzenia kolekcji
System.Collections.Concurrent // kolekcje bezpieczne pod względem wątków (zob. rozdział 23.)
```

Zapytania

Technologia LINQ (ang. *Language Integrated Query*) została dodana w .NET Framework 3.5. Przy jej użyciu można wysyłać bezpieczne pod względem typów zapytania do lokalnych i zdalnych kolekcji (np. tabel serwera SQL). Opis tej technologii zajmuje rozdziały 8. – 10. Wielką zaletą LINQ jest spójny zapytaniowy interfejs API dla różnych domen. Podstawowe typy służące do rozpoznawania zapytań LINQ znajdują się w następujących przestrzeniach nazw i należą do .NET Standard 2.0:

```
System.Linq // LINQ dla obiektów i PLINQ
System.Linq.Expressions // do ręcznego tworzenia wyrażeń
System.Xml.Linq // LINQ dla XML
```

Pełne środowisko .NET zawiera również następujące typy, których szczegółowy opis znajduje się w podrozdziale „Technologie zaplecza”:

```
System.Data.Linq // LINQ dla SQL
System.Data.Entity // LINQ dla jednostek (Entity Framework)
```

XML

Format XML jest powszechnie wykorzystywany w .NET Framework, więc cieszy się też bardzo dobrą obsługą. Cały rozdział 10. jest poświęcony technologii LINQ dla XML, czyli lekkiemu obiektowemu modelowi dokumentu XML, który można tworzyć i odpytywać przez LINQ. W rozdziale 11. znajduje się opis starszego modelu DOM W3C, szybkich niskopoziomowych klas do odczytu i zapisu danych oraz sposobu obsługi przez platformę schematów XML, arkuszy stylów i technologii XPath. Przestrzenie nazw XML to:

```
System.Xml // XmlReader, XmlWriter + stary DOM W3C
System.Xml.Linq // DOM LINQ dla XML
System.Xml.Schema // obsługa XSD
System.Xml.Serialization // deklaratywna serializacja XML dla typów .NET
System.Xml.XPath // język zapytań XPath
System.Xml.Xsl // obsługa arkuszy stylów
```

Poniższe przestrzenie nazw są dostępne w profilach klasycznych .NET (tzn. nie dla Windows Store):

```
System.Xml.XPath // język zapytań XPath
System.Xml.Xsl // obsługa arkuszy stylów
```

Diagnostyka

W rozdziale 13. opisujemy narzędzie platformy .NET do obsługi dzienników i definiowania asercji. Ponadto w rozdziale tym dowiesz się, jak współpracować z innymi procesami i zapisywać informacje w dzienniku zdarzeń systemu Windows oraz jak monitorować wydajność za pomocą mierników. Odpowiednie do tego typy są zdefiniowane w przestrzeni nazw `System.Diagnostics`.

Współbieżność i asynchroniczność

Wiele nowoczesnych aplikacji musi pracować nad kilkoma rzeczami jednocześnie. Od C# 5.0 stało się to łatwiejsze dzięki dodaniu funkcji asynchronicznych i takich wysokopoziomowych konstrukcji jak zadania i kombinatory zadań. Szczegółowo zajmujemy się tym w rozdziale 14., po wprowadzeniu do podstaw wielowątkowości. Typy do pracy z wątkami i asynchronicznego wykonywania operacji znajdują się w przestrzeniach nazw `System.Threading` i `System.Threading.Tasks`.

Strumienie oraz operacje wejścia i wyjścia

Platforma .NET Framework zapewnia strumieniowe narzędzia do niskopoziomowej pracy z wejściem i wyjściem. Strumienie są zazwyczaj wykorzystywane do bezpośredniego odczytywania i zapisywania danych w plikach i połączeniach sieciowych oraz można je łączyć lub opakowywać w strumienie dekoracyjne w celu zastosowania kompresji lub szyfrowania. W rozdziale 15. znajduje się opis architektury strumieniowej platformy .NET oraz narzędzi do pracy z plikami i katalogami, kompresji, przechowywania danych w wydzielonej pamięci masowej, pracy z potokami oraz plikami mapowanymi na pamięć. Typy `Stream` oraz wejścia i wyjścia są zdefiniowane w przestrzeni nazw `System.IO` i jej przestrzeniach podrzędnych, a typy `WinRT` do obsługi wejścia i wyjścia plikowego znajdują się w przestrzeni nazw `Windows.Storage`.

Sieć

Za pomocą typów z przestrzeni nazw `System.Net` można bezpośrednio pracować z takimi protokołami, jak: HTTP, FTP, TCP/IP i SMTP. W rozdziale 16. pokazujemy sposoby komunikacji przy użyciu każdego z nich, zaczynając od prostej czynności pobierania strony internetowej, a na pobieraniu poczty POP3 bezpośrednio przy użyciu TCP/IP kończąc. Oto lista przestrzeni nazw, które opisujemy:

```
System.Net
System.Net.Http // HttpClient
System.Net.Mail // do wysyłania poczty przez SMTP
System.Net.Sockets // TCP, UDP i IP
```

Dwie ostatnie przestrzenie są niedostępne dla aplikacji dla Windows Store, jeśli docelowym systemem jest Windows 8/8.1 (WinRT), ale są dostępne dla aplikacji Windows 10 Store (UWP) w ramach kontraktu .NET Standard 2.0. W przypadku aplikacji WinRT do wysyłania poczty należy używać bibliotek zewnętrznych, a do pracy z gniazdami należy używać typów WinRT z przestrzeni nazw `Windows.Networking.Sockets`.

Serializacja

Platforma zawiera kilka systemów do zapisywania i przywracania obiektów do postaci binarnej lub tekstowej. Systemy te są potrzebne w technologiach do budowy aplikacji rozproszonych, np. WCF, usług sieciowych i programów do komunikacji zdalnej, a także do zapisywania obiektów w plikach i przywracania ich stamtąd. W rozdziale 17. opisujemy trzy najważniejsze mechanizmy serializacji — serializator kontraktu danych, serializator binarny oraz serializator XML (w .NET dostępny jest też serializator JSON). Typy do obsługi serializacji znajdują się w następujących przestrzeniach nazw:

```
System.Runtime.Serialization
System.Xml.Serialization
```

Zestawy, refleksja i atrybuty

Zestawy, do postaci których kompilowane są programy w języku C#, zawierają instrukcje wykonywalne (w postaci języka pośredniego zwanego skrótowo IL od ang. nazwy *intermediate language*) i metadane opisujące typy, składowe oraz atrybuty programu. Za pomocą technik refleksji można przeglądać te metadane w czasie działania programu i np. dynamicznie wywoływać metody. Za pomocą typów z przestrzeni nazw `Reflection.Emit` można tworzyć nowy kod w locie.

W rozdziale 18. opisujemy budowę zestawów oraz sposoby ich podpisywania, metody wykorzystania globalnego bufora zestawów (ang. *global assembly cache* — GAC) i zasobów zestawów oraz sposoby rozpoznawania odwołań do plików. W rozdziale 19. opisujemy refleksję i atrybuty — metody przeglądania metadanych, dynamicznego wywoływania funkcji, pisania własnych atrybutów, emitowania nowych typów oraz przetwarzania surowego kodu IL. Typy do posługiwania się technikami refleksji i do pracy z zestawami znajdują się w następujących przestrzeniach nazw:

```
System
System.Reflection
System.Reflection.Emit // tylko .NET Framework
```

Programowanie dynamiczne

W rozdziale 20. przedstawiamy niektóre typowe techniki programowania dynamicznego oraz metody wykorzystania systemu DLR (ang. *Dynamic Language Runtime*) wchodzącego w skład CLR od wersji 4.0 platformy. Pokazujemy sposoby implementacji wzorca wizytator (ang. *Visitor*), pisania własnych obiektów dynamicznych oraz współpracy z językiem IronPython. Typy do programowania dynamicznego znajdują się w przestrzeni nazw `System.Dynamic`.

Bezpieczeństwo

Platforma .NET Framework ma własną warstwę bezpieczeństwa, która pozwala programiście uruchamiać w piaskownicy zarówno inne zestawy, jak i ją samą. W rozdziale 21. opisujemy kwestie dotyczące uprawnień dostępu kodu, ról oraz tożsamości, jak również model transparentności wprowadzony w CLR 4.0. W dalszej części rozdziału opisujemy narzędzia kryptograficzne platformy, m.in. techniki szyfrowania, mieszania i ochrony danych. Odpowiednie do tego typy są zdefiniowane w następujących przestrzeniach nazw:

```
System.Security
System.Security.Permissions
System.Security.Policy
System.Security.Cryptography
```

Zaawansowane techniki pracy z wątkami

Funkcje asynchroniczne języka C# znacznie ułatwiają programowanie współbieżne, ponieważ zmniejszają zapotrzebowanie na techniki niskopoziomowe. Mimo to czasami nie można się obejść bez konstrukcji sygnalizacyjnych, pamięci wątkowej, blokad odczytu i zapisu itd. Szerzej na ten temat piszemy w rozdziale 22. Typy do pracy z wątkami znajdują się w przestrzeni nazw `System.Threading`.

Programowanie równoległe

W rozdziale 23. zamieściliśmy szczegółowy opis bibliotek i typów służących do pracy z procesorami wielordzeniowymi, m.in.: interfejsów API równoległego wykonywania zadań, imperatywnej równoległości na poziomie danych oraz równoległości na poziomie funkcji (PLINQ).

Domeny aplikacji

System CLR zapewnia dodatkowy poziom izolacji wewnątrz procesu, zwany **domeną aplikacji** (ang. *application domain*). W rozdziale 24. opisujemy te własności domeny aplikacji, z których można korzystać, i pokazujemy, jak tworzyć dodatkowe domeny aplikacji oraz jak ich używać w obrębie jednego procesu do takich celów jak testowanie jednostkowe. Ponadto pokazujemy sposoby użycia technik **pracy zdalnej** (ang. *remoting*) z tymi domenami.

Tworzenie osobnych domen aplikacji nie jest częścią .NET Standard 2.0, chociaż można odnieść się do bieżącej domeny za pośrednictwem klasy `AppDomain` z przestrzeni nazw `System`.

Interoperacyjność macierzysta i COM

Współpracować można zarówno z kodem macierzystym, jak i z COM. Interoperacyjność macierzysta umożliwia wywoływanie funkcji w niezarządzanych bibliotekach DLL, rejestrowanie funkcji zwrotnych, mapowanie struktur danych oraz współpracę z macierzystymi typami danych. Natomiast interoperacyjność COM umożliwia wywoływanie typów COM i udostępnianie typów .NET technologii COM. Typy dotyczące tej funkcjonalności znajdują się w przestrzeni nazw `System.Runtime.InteropServices` i zostały opisane w rozdziale 25.

Technologie praktyczne

API interfejsu użytkownika

Aplikacje z interfejsem użytkownika można podzielić na dwie kategorie: **klienty ubogie** (ang. *thin clients*), do których zaliczają się strony internetowe, i **klienty bogate** (ang. *rich clients*), do których zaliczają się programy pobierane na dysk komputera lub urządzenia przenośnego i na nim instalowane.

Do tworzenia aplikacji pierwszego typu na platformie .NET służą biblioteki ASP.NET i ASP.NET Core.

Natomiast aplikacje bogate, przeznaczone dla klasycznych komputerów z systemem Windows 7/8/10, są tworzone przy użyciu interfejsów API WPF i Windows Forms. Aplikacje przeznaczone dla systemów iOS, Android i Windows Phone można tworzyć przy użyciu Xamarin, a do tworzenia bogatych aplikacji dla systemu Windows 10 przeznaczona jest technologia UWP (zobacz tabelę 1.1 w rozdziale 1.).

Istnieje też technologia hybrydowa, o nazwie Silverlight, która właściwie straciła znaczenie z chwilą pojawienia się HTML5.

ASP.NET

Aplikacje napisane w technologii ASP.NET działają na serwerze Windows IIS i można uzyskać do nich dostęp z każdego innego serwera sieciowego. Oto zalety ASP.NET w porównaniu z technologiami „bogatymi”:

- brak konieczności wdrażania u klienta;
- klient może korzystać z innej platformy niż Windows;
- łatwość wdrażania aktualizacji.

Ponadto, dzięki temu, że większość kodu aplikacji ASP.NET działa na serwerze, warstwa dostępu do danych może działać w tej samej domenie aplikacji — bez ograniczania bezpieczeństwa i możliwości skalowania. Natomiast klienty bogate o podobnej funkcjonalności nie są generalnie tak bezpieczne ani skalowalne. (Rozwiązaniem jest dodanie **warstwy środkowej** między klientem i bazą danych. Warstwa ta działa na zdalnym serwerze aplikacji — często obok serwera baz danych — i komunikuje się z bogatymi klientami poprzez WCF, usługi sieciowe lub technologie pracy zdalnej).

Pisząc stronę internetową, można wybierać między tradycyjnymi API Web Forms (formularze internetowe) i nowszym MVC (ang. *Model-View-Controller*). Obie technologie bazują na infrastrukturze ASP.NET. Formularze internetowe są obecne na platformie od początku. Natomiast MVC dodano znacznie później jako odpowiedź na sukces Ruby on Rails i MonoRail. Generalnie MVC zapewnia lepszą abstrakcję programistyczną niż Web Forms i daje większą kontrolę nad generowanym kodem HTML. Porzucając Web Forms, tracimy natomiast projektanta, więc technologia ta nadal jest dobrym wyborem, jeśli trzeba utworzyć stronę zawierającą głównie statyczną treść.

Ograniczenia ASP.NET są zasadniczo odzwierciedleniem ogólnych ograniczeń systemów klientów ubogich:

- Choć dzięki technologiom HTML5 i AJAX przeglądarka internetowa jest w stanie zapewnić bogaty interfejs użytkownika, to wciąż pod względem funkcjonalności i szybkości działania ustępuje bogatym API macierzystym, takim jak WPF.
- Przechowywanie informacji o stanie na kliencie — lub w jego imieniu — może być kłopotliwe.

Typy służące do pisania aplikacji ASP.NET są zdefiniowane w przestrzeni nazw `System.Web.UI` i jej przestrzeniach podrzędnych oraz znajdują się w zestawie `System.Web.dll`. Technologia ASP.NET 5 jest dostępna w NuGet.

ASP.NET Core

Dość świeży dodatek, ASP.NET Core, jest podobny do ASP.NET, ale działa zarówno w .NET Framework, jak i .NET Core (dzięki czemu umożliwia obsługę wielu platform). ASP.NET Core ma lekką architekturę modułową, umożliwia samodzielne hostowanie we własnym procesie oraz jest dostępny na licencji open source. W odróżnieniu od poprzedników ASP.NET Core nie zależy od System.Web ani historycznego bagażu Web Forms. Najlepiej sprawdza się w mikrousługach i wdrożeniach w kontenerach.

Windows Presentation Foundation (WPF)

System WPF został wprowadzony w .NET 3 jako technologia do tworzenia aplikacji z bogatym interfejsem użytkownika. Zalety WPF w stosunku do poprzedniego systemu, Windows Forms, są następujące:

- Obsługa zaawansowanych operacji graficznych, takich jak: przekształcenia dowolnego rodzaju, renderowanie grafik trójwymiarowych, multimedia oraz prawdziwa przezroczystość. Skinning jest obsługiwany poprzez szablony i style.
- Podstawowa jednostka miary nie jest oparta na pikselu, więc aplikacje wyglądają prawidłowo przy każdych ustawieniach DPI (liczba punktów na cal).
- Dobra obsługa układów dynamicznych, dzięki czemu można dokonywać lokalizacji programów bez obawy, że elementy zaczną na siebie nachodzić.
- Szybkie renderowanie przy użyciu DirectX z wykorzystaniem akceleracji sprzętowej.
- Niezawodne wiązanie danych.
- Możliwość deklarowania interfejsu użytkownika aplikacji w plikach XAML, które mogą być przechowywane i modyfikowane niezależnie od „obsługującego je kodu” — w ten sposób lepiej oddziela się warstwę prezentacyjną od logiki.

Jednak rozmiar i złożoność WPF sprawiają, że nauka posługiwania się tym systemem jest dość trudna.

Typy do pisania aplikacji WPF znajdują się w przestrzeni nazw `System.Windows` i wszystkich jej przestrzeniach podrzędnych z wyjątkiem `System.Windows.Forms`.

Windows Forms

Windows Forms to interfejs API do tworzenia aplikacji „bogaty”. Jest tak stary jak sama platforma .NET Framework. W porównaniu z WPF Windows Forms jest dość prostą technologią zapewniającą większość funkcji potrzebnych do napisania typowej aplikacji dla systemu Windows. Ponadto jest potrzebny przy obsłudze serwisowej starych aplikacji. Ma jednak kilka wad w porównaniu z WPF:

- Położenie i rozmiar kontrolki są ustalane w pikselach, przez co aplikacja może źle wyglądać w urządzeniu mającym ekran o innych ustawieniach DPI niż ekran programisty (choć trochę to poprawiono we Framework 4.7).
- Do rysowania niestandardowych kontrolki wykorzystywana jest biblioteka GDI+, która wprawdzie jest całkiem elastyczna, ale powoli renderuje duże obszary (a bez podwójnego buforowania może powodować miganie).
- Kontrolki są pozbawione prawdziwej przezroczystości.
- Dla większości kontrolki nie ma możliwości komponowania z innymi kontrolkami. Na przykład nie można umieścić kontrolki obrazu w kontrolce karty. Modyfikowanie widoków list i pól kombi jest czasochłonne i żmudne.
- Trudno jest stworzyć niezawodny układ dynamiczny.

Ostatni punkt jest doskonałym argumentem przemawiającym na korzyść WPF, nawet jeśli ktoś pisze aplikację biznesową, w której potrzebny jest tylko interfejs użytkownika, bez zważania na kwestie „wrażeń estetycznych użytkownika”. Elementy układowe WPF, np. Grid, umożliwiają takie rozmieszczenie etykiet i pól tekstowych, że zawsze znajdują się w odpowiednim miejscu — nawet po zamianie języka — i aby to osiągnąć, nie trzeba się posługiwać skomplikowaną logiką i nie występuje żadne miganie. Ponadto nie trzeba sprowadzać projektu interfejsu do najniższego wspólnego mianownika pod względem rozdzielczości ekranu — elementy układu WPF od początku projektowano z myślą o dostosowywaniu rozmiaru do zmiennych warunków.

Jeśli chodzi o zalety Windows Forms, to technologia ta jest względnie łatwa do nauki i wciąż powszechnie wykorzystuje się ją w kontrolkach różnych producentów.

Typy Windows Forms znajdują się w przestrzeniach nazw `System.Windows.Forms` (w zestawie `System.Windows.Forms.dll`) i `System.Drawing` (w zestawie `System.Drawing.dll`). Druga z wymienionych przestrzeni zawiera też typy GDI+ do rysowania niestandardowych kontroltek.

Xamarin

Xamarin, już w posiadaniu Microsoftu, służy do pisania aplikacji mobilnych w języku C# przeznaczonych dla systemów iOS, Android i Windows Phone. Jest to wieloplatformowe rozwiązanie, które nie działa na bazie .NET Framework, lecz na bazie własnego środowiska (pochodnego platformy open source Mono). Więcej informacji znajduje się na stronie <https://www.xamarin.com>.

UWP (Universal Windows Platform)

UWP służy do pisania klasycznych aplikacji dla systemu Windows 10 i dla wyposażonych w ten system urządzeń aplikacji dystrybuowanych za pośrednictwem Windows Store. Ten bogaty API klienta jest przeznaczony do pisania interfejsów dotykowych. Powstał na podstawie inspiracji technologią WPF i do definiowania układu wykorzystuje XAML. Jego przestrzenie nazw to `Windows.UI` i `Windows.UI.Xaml`.

Silverlight

Silverlight także jest odrębną technologią od .NET Framework. Służy do pisania graficznych interfejsów użytkownika działających w przeglądarce internetowej, podobnie jak Adobe Flash. Rosnąca popularność HTML5 skłoniła firmę Microsoft do porzucenia projektu Silverlight.

Technologie zapleczone

ADO.NET

ADO.NET to zarządzany interfejs API dostępu do danych. Choć część jego nazwy pochodzi jeszcze z lat 90. (ang. *ActiveX Data Objects* — ADO), technologia ta jest już zupełnie inna. ADO.NET składa się z dwóch niskopoziomowych elementów:

Warstwa dostawcza

Model dostawcy definiuje wspólne klasy i interfejsy zapewniające niskopoziomowy dostęp do dostawców baz danych. Interfejsy te zawierają: połączenia, polecenia, adaptery i czytniki (kursory jednokierunkowe i tylko do odczytu baz danych). Platforma .NET Framework standardowo obsługuje Microsoft SQL Server i sterowniki do licznych baz danych innych producentów.

Model DataSet

DataSet to strukturalny bufor danych. Przypomina prymitywną przechowywaną w pamięci bazę danych, która definiuje takie konstrukcje SQL, jak: tabele, wiersze, kolumny, relacje, ograniczenia i widoki. Wykorzystując w programie taki bufor, można zredukować liczbę odwołań do serwera baz danych, zapewniając tym samym lepszą skalowalność i krótszy czas reakcji interfejsu użytkownika. Zbiory danych DataSet można serializować oraz przesyłać przez sieć między klientem a aplikacjami serwerowymi.

Nad warstwą dostawczą znajdują się trzy interfejsy API umożliwiające wysyłanie do baz danych zapytań LINQ:

- Entity Framework (tylko .NET Framework).
- Entity Framework Core (.NET Framework i .NET Core).
- LINQ to SQL (tylko .NET Framework).

Wszystkie te trzy technologie zawierają mapery ORM (ang. *object-relational mapper*), a więc automatycznie mapują obiekty (na podstawie zdefiniowanych przez programistę klas) na wiersze bazy danych. Pozwala to na przeszukiwanie tych obiektów za pomocą zapytań LINQ (nie trzeba pisać instrukcji SQL `select`) oraz modyfikowanie ich bez potrzeby pisania instrukcji SQL `insert`, `delete` czy `update`. Przyczynia się to do zmniejszenia objętości kodu źródłowego w warstwie dostępu do danych (zwłaszcza eliminowany jest kod „wypełniający”) oraz zapewnia wysoki poziom bezpieczeństwa statycznego typów. Dodatkowo technologie te eliminują konieczność używania obiektów DataSet jako odbiorników danych, chociaż te bufony zapewniają możliwość przechowywania i serializacji zmian stanu (coś, co jest bardzo ważne w aplikacjach wielowarstwowych). Technologie Entity Framework i LINQ to SQL można używać w połączeniu z obiektami DataSet, choć nie jest to może najwygodniejsze rozwiązanie, nie mówiąc już o tym, że obiekty DataSet z natury są dość nieporęczne. Innymi słowy: jak na razie w ofercie Microsoftu z zakresu ORM brak prostych gotowych rozwiązań do pisania aplikacji wielowarstwowych.

Technologia LINQ to SQL jest prostsza niż Entity Framework i od dawna wiadomo, że dostarcza lepszej jakości kod SQL (choć Entity Framework też znacznie poprawiono dzięki licznym aktualizacjom). Z kolei interfejs Entity Framework jest elastyczniejszy, ponieważ umożliwia przeprowadzanie wyrafinowanych mapowań między bazą danych a odpytywanymi klasami (Entity Data Model) i udostępnia model zapewniający możliwość obsługi innych baz danych niż SQL Server.

Entity Framework Core (EF Core) to Entity Framework z uproszczoną budową inspirowaną LINQ to SQL. Porzucono w nim skomplikowany Entity Data Model i może działać zarówno w .NET Framework, jak i .NET Core.



.NET Standard 2.0 zawiera wspólne interfejsy w warstwie dostawcy, jak również DataSets, ale jest pozbawiony typów specyficznych dla SQL Server i mapeń obiektowo-relacyjnych.

Windows Workflow (tylko .NET Framework)

Windows Workflow to system szkieletowy do modelowania i obsługi potencjalnie długotrwałych procesów biznesowych. Wykorzystuje standardową bibliotekę wykonawczą, więc nie ma obaw o spójność i współpracę z innymi rozwiązaniami. Ponadto system ten pozwala zredukować objętość źródłową dynamicznie kontrolowanych drzew decyzyjnych.

Windows Workflow nie jest, ściśle rzecz biorąc, technologią zapleczową, ponieważ systemu tego można używać wszędzie (np. do definiowania przepływu stron w interfejsie użytkownika).

Technologia ta została dodana w .NET Framework 3.0, a jej typy przypisano do przestrzeni nazw `System.WorkFlow`. W .NET Framework 4.0 Windows Workflow poddano gruntownym zmianom. Nowe typy znajdują się w przestrzeni nazw `System.Activities`.

COM+ i MSMQ (tylko .NET Framework)

Platforma .NET Framework może współpracować z technologią COM+ w zakresie takich usług, jak obsługa transakcji rozproszonych, za pośrednictwem typów znajdujących się w przestrzeni nazw `System.EnterpriseServices`. Dodatkowo obsługuje też technologię MSMQ (ang. *Microsoft Message Queuing*) do asynchronicznego jednokierunkowego przesyłania wiadomości za pomocą typów z przestrzeni nazw `System.Messaging`.

Technologie systemów rozproszonych

Windows Communication Foundation (WCF)

WCF to zaawansowana infrastruktura komunikacyjna, którą wprowadzono w .NET Framework 3.0. Jest tak elastyczna i zapewnia tyle możliwości konfiguracji, że przez nią starsze technologie — Remoting i (.ASMX) Web Services — stały się w *znacznej części* niepotrzebne.

Technologie WCF, Remoting i Web Services łączy to, że wszystkie implementują następujący podstawowy model komunikacji klienta z aplikacją serwerową:

- Na serwerze należy określić, które metody mają być dostępne do zdalnego wywoływania przez aplikacje klienckie.
- Na kliencie należy określić lub wydedukować *sygnatury* metod serwerowych, które mają być wywoływane.
- Zarówno na serwerze, jak i na kliencie wybiera się protokół przesyłania i komunikacji (w WCF dokonuje się tego przez *wiązanie*).
- Klient ustanawia połączenie z serwerem.
- Klient wywołuje zdalną metodę, która zostaje wykonana na serwerze.

Dzięki kontraktom usługowym i dotyczącym danych WCF jeszcze skuteczniej likwiduje zależność klienta od serwera. Zasadniczo współpraca przebiega w ten sposób, że klient wysyła wiadomość (w formacie XML lub binarnym) do odbiorcy w *usłudze* zdalnej, a nie bezpośrednio wywołuje zdalną *metodę*. Jedną z zalet takiego rozdziału jest to, że klienci nie są w żaden sposób uzależnieni od platformy .NET ani żadnego należącego do jakiejś firmy protokołu komunikacyjnego.

Możliwości konfiguracji WCF są bardzo szerokie. Prócz tego można korzystać z bardzo dobrej obsługi standardowych protokołów przesyłania wiadomości opartych na SOAP, w tym WS-*. Dzięki temu można się komunikować z jednostkami wykorzystującymi różne rodzaje oprogramowania — działającego na rozmaitych platformach — i jednocześnie nadal korzystać z zaawansowanych funkcji, takich jak szyfrowanie. Z drugiej strony, praktyczność tego rozwiązania ogranicza poziom złożoności protokołów i dlatego aktualnie najlepszym wyjściem w zakresie przesyłania wiadomości między niekompatybilnymi systemami jest skorzystanie z technologii REST i HTTP, które Microsoft obsługuje poprzez warstwę Web API na ASP.NET.

Natomiast do komunikacji między aplikacjami .NET w WCF zapewniono bogatszy wybór funkcji serializacji i innych narzędzi niż dla API REST. Ponadto procesy te powinny przebiegać szybciej, ponieważ pomijają HTTP i mogą wykorzystywać serializację binarną.

Typy do komunikacji z WCF znajdują się w przestrzeni nazw `System.ServiceModel` i przestrzeniach podrzędnych.

Web API

Web API działa na bazie ASP.NET/ASP.NET Core i pod względem architektury przypomina API MVC Microsoftu, z tym że służy do udostępniania usług i danych, a nie stron internetowych. Jego zaletą w porównaniu z WCF jest to, że pozwala na stosowanie popularnych technik REST-owych z wykorzystaniem HTTP, co zapewnia lepszą współpracę z szerszym spektrum platform.

Implementacje REST są prostsze niż protokoły SOAP i WS-, które WCF wykorzystuje do współpracy międzyplatformowej. Ponadto REST-owe interfejsy API mają elegantszą architekturę do pracy z luźno powiązanymi systemami, *de facto* bazują na standardach oraz doskonale wykorzystują możliwości protokołu HTTP.

Remoting i .ASMX Web Services (tylko .NET Framework)

Remoting i .ASMX Web Services to poprzednicy WCF. Wraz z pojawieniem się WCF technologia Remoting prawie przestała być potrzebna, a technologia .ASMX Web Services całkiem straciła rację bytu.

Technologia Remoting zachowała użyteczność w dziedzinie komunikacji między domenami aplikacji w obrębie tego samego procesu (rozdział 24.). Jest przeznaczona do pracy ze ściśle powiązanymi aplikacjami, więc typowym przykładem jej użycia jest aplikacja i serwer napisane w technologii .NET przez jedną firmę (albo różne firmy posługujące się tymi samymi zestawami). Komunikacja z reguły polega na wymianie potencjalnie złożonych niestandardowych obiektów .NET, które są następnie serializowane i deserializowane przez Remoting bez jakiegokolwiek zewnętrznej pomocy.

Typy technologii Remoting znajdują się w przestrzeni nazw `System.Runtime.Remoting` i przestrzeniach podrzędnych. Typy Web Services znajdują się pod przestrzenią nazw `System.Web.Services`.

.ASMX Web Services, 227
.NET Framework, 18
.NET Framework 4.6, 214
.NET Framework 4.7, 215
.NET Standard 2.0, 215

A

abstrakcyjne klasy, 109
adapter

 StringReader, 620
 StringWriter, 620
 strumienia, 614

adaptery

 binarne, 620
 tekstowe, 615

adnotacje, 466

ADO.NET, 224

adresy, 646

 URL, 647, 652

agregacje bez ziarna, 435

akcesory widoku, 640

aktualizacje, 387

aktywacja typów, 756

algorytm porównywania

 kulturowego, 235
 porządkowego, 235

aliasy typów, 86

analiza

 działających procesów, 528
 wątków w procesie, 528

anonimowe wywoływanie składowych, 772, 813

anulowanie zadań, 904

API, 389

API interfejsu użytkownika, 221

aplikacje

 UWP, 635
 Windows Store, 757

architektura

 domeny aplikacji, 921
 Roslyn, 976
 sieci, 643
 strumienia, 599, 600

archiwum ZIP, 624

argument NumberStyles, 263

argumenty, 30

 nazwane, 66, 949

asercja, 523, 961

 o zerowej długości, 972

asocjacje, 383

ASP.NET, 222

ASP.NET Core, 222

asynchroniczne wyrażenia lambda, 581

asynchroniczność, 219, 539, 586

asynchroniczny odczyt, 602

atak słownikowy, 830

atomowość, 844

atrapy, 655

atrybut, 188, 220, 775

 [Conditional], 521, 522

 [NonSerialized], 702

 [OnDeserialized], 703

 [OnDeserializing], 703

 [OnSerialized], 704

 [OnSerializing], 704

 [OptionalField], 704

 [Serializable], 699, 701

 [ThreadStatic], 871

 [XmlArrayItem], 715

 AttributeUsage, 777

- atrybut
 - Flags, 126
 - LoaderOptimization, 925
 - StructLayout, 938
- atrybuty
 - cele, 190
 - debuggera, 527
 - definiowanie, 778
 - informacji wywołującego, 190
 - klasy, 189
 - nazwane, 189
 - pobieranie w czasie działania, 779
 - pobieranie w kontekście refleksji, 780
 - pozycyjne, 189
 - pseudoniestandardowe, 777
 - przypisywanie, 795
 - serializacji binarnej, 702
 - warunkowe, 207
 - zestawu, 721
- automatyczne usuwanie nieużytków, 499
- autonomiczne zadania, 560

B

- bariera
 - pamięci, 844
 - wątku wykonawczego, 867
- bezpieczeństwo, 220, 821
 - deklaratywne, 823
 - imperatywne, 823
 - pliku, 628
 - typów, 16, 127
 - wątków, 546, 605, 848, 849
 - przy odczycie, 851
 - w serwerach aplikacji, 852
- białe znaki, 985
- biblioteki
 - DLL, 935
 - PFX, 880, 882
 - Roslyn, 975
- BitTorrent, 673
- blok
 - finally, 166
 - instrukcji, 73
 - kodu, 30
- blokada, 843, 848
 - odczytu, 855
 - z możliwością uaktualnienia, 857
 - zapisu, 855

- bloki try-catch-finally, 174
- blokowanie, 844
 - bez wykluczania, 840, 854
 - wykluczające, 840
- błędy
 - parsowania, 265
 - zaokrąglania liczb, 49
- buforowanie, 516
 - w miejscu wykonania, 808
- bufory, 205

C

- C# 2.0, 28
- C# 3.0, 27
- C# 4.0, 26
- C# 5.0, 26
- C# 6.0, 25
- C# 7.0, 22
- CA, certificate authority, 731
- CAS, Code Access Security, 821
- cele
 - delegatów, 146
 - emisji, 798
- certyfikat, 731
 - do podpisywania kodu, 732
- ciąg tekstowy zapytania, 662
- CLR, Common Language Runtime, 17, 28, 41, 213, 217
- COM, 27, 221, 935, 946
- COM+, 226
- COM-callable wrapper, 953
- cookies, 664
- cykl życiowy obiektu, 499
- czas letni, 253

D

- dane, 36
 - formularza, 663
 - hierarchiczne, 484
- data i godzina, 242
- deassembler, 801
- debugger, 526
- debugowanie, 523
- definiowanie
 - metod generycznych, 796
 - przestrzeni nazw, 463
 - równości, 285

- typów, 35
- typów generycznych, 797
- własnych atrybutów, 778
- deklarowanie, 458
 - kowariantnego parametru typu, 139
 - parametrów typów, 133
 - XML, 460
 - wielu pól, 90
- dekonstruktory, 23, 93
- dekonstruowanie krotek, 187
- dekrementacja, 46
- delegat, 143, 148, 390, 770
 - Action, 147
 - Func, 147
 - MatchEvaluator, 966
- delegaty
 - asynchroniczne, 595
 - multiemisji, 145
- deserializacja, 697
- deserializer, 702
- diagnostyka, 218, 519, 993
- diagnozowanie wycieku pamięci, 514
- DLR, dynamic language runtime, 807
- DNS, Domain Name Service, 645, 672, 680
- dokumentacja XML, 208
- dokumenty, 458
- dołączanie
 - debuggera, 527
 - protokołów równości, 336
- DOM, 391, 441
- domeny aplikacji, 221, 921
 - architektura, 921
 - dzielenie danych, 929
 - likwidowanie, 923
 - monitorowanie, 927
 - tworzenie, 923
 - wątki, 927
 - zastosowanie, 924
- domknięcie, 159
- domyślna wartość generyczna, 134
- dostawcy formatu, 255, 256
- dostęp do składowych niepublicznych, 770
- drobiazgi, 985
 - strukturalne, 986
- drzewo
 - obiektów, 681
 - składni Roslyn, 977
 - przeszukiwanie, 981
 - transformacja, 987
 - tworzenie, 980
 - węzły, 979
 - wyszukiwanie elementu, 983
 - wyrażeń, 28, 158, 370, 390
 - wywołań, 567
 - wywołań asynchronicznych, 579
 - X-DOM, 445
- dynamiczne
 - wybieranie przeciążonych składowych, 810
 - wywoływanie składowej, 768
- dynamiczny
 - odbiorca, 197
 - system wykonawczy języka, 807
- dyrektywa
 - fixed, 943
 - using, 83, 86
 - using static, 84
- dyrektywy preprocesora, 206, 520, 985
- dziedziczenie, 104, 111
- dzielenie
 - całkowitoliczbowe, 46
 - danych, 929
 - na części, 892
 - przy użyciu skrótów, 891
 - zakresowe, 892
- dziennik zdarzeń, 530
 - monitorowanie, 532
 - odczyt danych, 532
 - zapis danych, 531

E

- EAP, Event-based Asynchronous Pattern, 595
- egzemplarze, 36
 - kontekstu typizowanego, 379
 - nasłuchujące, 526
 - typów referencyjnych, 59
- element główny, 500
- elementy, 343
 - opcjonalne, 478
 - podklas kolekcji, 696, 716
 - puste, 478
 - składowe klasy
 - Assembly, 724
 - Stream, 601
 - TextReader, 616
 - TextWriter, 616
 - emisja podzespołu, 993

emitowanie
 generycznych typów i klas, 796
 konstruktorów, 794
 metod, 791
 pól i właściwości, 793
 składowych typów, 791
 typów, 787
 zestawów, 787
Entity Framework, 376
enumerator, 171
ewaluacja, 782

F

fabryka
 abstrakcyjna, 910
 zadań, 910
filtr, 385
filtrowanie, 399
 z indeksowaniem, 400
filtry wyjątków, 26, 165
finalizatory, 31, 102, 501, 505
flagi parsowania, 260
forma
 base64, 269
 XML, 218
formater SoapFormatter, 701
formatery, 684
 binarne, 688
formatowanie, 255
 złożone, 258
formularze, 663
FTP, File Transfer Protocol, 645, 670
funkcje, 36
 asynchroniczne, 26, 570
 fabryczne ziaren, 893
 klasy FileStream, 609
 operatorowe, 200
 pieczętowanie, 110
 tworzenie, 577
 wyrażeniowe, 25

G

GAC, Global Assembly Cache, 734
garbage collection, 493
generacje na stercie, 508
generator list z list, 351

generowanie
 dynamicznego kodu, 781
 IL, 781
 metod instancji, 792
 zmiennych lokalnych, 784
generyczne typy delegacyjne, 147
globalizacja, 271
globalna przestrzeń nazw, 83
głębokie klonowanie, 447
główne moduły współpracujące, 952
grupowanie, 424

H

hierarchia
 klas, 708
 obiektów, 404
HTTP, Hypertext Transfer Protocol,
 645, 661, 667

I

identyfikatory, 32
implementacja
 indeksatorów, 99
 interfejsu, 121
 interfejsów przeliczeniowych, 301
 obiektów dynamicznych, 816
 własności, 99
 zabezpieczeń, 824
indeksatory, 31, 99, 950
indeksowanie tablic, 310
inferencja typów literalów liczbowych, 43
informacje
 o postępie, 588
 o woluminie, 634
inicjalizacja
 pól, 90
 tablic, 57
inicjalizatory
 indeksów, 25
 kolekcji, 172
 obiektów, 27, 94, 368
 własności, 98
inkrementacja, 46
instalacja certyfikatu, 731
instancje typów, 760

- instrukcja, 29, 73
 - break, 81
 - continue, 81
 - fixed, 203
 - foreach, 172
 - goto, 77, 81
 - if, 75
 - if-else, 77
 - lock, 841
 - return, 82
 - switch, 76, 77
 - throw, 82
 - try, 163
 - using, 166
 - yield break, 174
- instrukcje
 - deklaracji, 73
 - iteracyjne, 79
 - skoku, 80
 - wyboru, 75
 - wyrażeń, 74
- interfejs, 125
 - Close, 493
 - COM, 947
 - Dispose, 493
 - EqualityComparer, 337
 - ICollection, 304, 305
 - ICollection<T>, 305
 - IComparable, 290
 - ICustomFormatter, 259
 - IDictionary, 326
 - IDictionary<TKey,TValue>, 325
 - IDispatch, 948
 - IDisposable, 300, 493, 497
 - IEnumerable, 298
 - IEnumerable<T>, 299, 300
 - IEnumerator, 298
 - IEnumerator<T>, 299
 - IEqualityComparer, 337
 - IEquatable<T>, 284, 288
 - IFormatProvider, 259
 - IList, 304, 306
 - IList<T>, 306
 - IOrderedEnumerable, 423
 - IOrderedQueryable, 423
 - IProducerConsumerCollection<T>, 914
 - IProgress<T>, 589
 - IReadOnlyList<T>, 307
 - ISerializable, 701, 706
 - IStructuralComparable, 341
 - IStructuralEquatable, 341
 - ISymbol, 996
 - IUnknown, 948
 - IXmlSerializable, 684, 700, 716
- interfejsy, 15, 120, 124, 148, 759
 - jawna implementacja, 121
 - niegeneryczne, 300
 - przeliczeniowe, 301
 - reimplementacja, 123
 - rozszerzanie, 121
 - wirtualna implementacja składowych, 122
- internacjonalizacja aplikacji, 271
- interoperacyjność, 700
 - macierzysta, 221
- interpolacja łańcuchów, 25, 54
- IP, Internet Protocol, 645, 680
- iteratory, 172, 174, 301
- izolowanie
 - typów, 931
 - zestawów, 931

J

- jednolity system typów, 15
- języki dynamiczne, 819

K

- katalog, 636
 - bazowy, 750
 - specjalny, 632
- klasa, 15, 31, 89, 125
 - AggregateException, 910
 - AppContext, 295
 - Array, 308, 314
 - ArrayList, 316
 - Assembly, 723, 749
 - AssemblyName, 728
 - AutoResetEvent, 859
 - BackgroundWorker, 596
 - Barrier, 867
 - BinaryReader, 620
 - BinaryWriter, 620
 - BitArray, 322
 - BitConverter, 270
 - BlockingCollection<T>, 916
 - BufferedStream, 613
 - Collection<T>, 331

klasa

- CollectionBase, 331, 333
- ConcurrentBag<T>, 915
- Console, 293
- Convert, 267
- CountdownEvent, 863
- CredentialCache, 659
- CryptoStream, 832–834
- CSharpSyntaxRewriter, 990
- CSharpSyntaxWalker, 984
- DataContext, 379, 381
- DataLoadOptions, 385
- DelegatingHandler, 656
- Dictionary<TKey,TValue>, 326
- DictionaryBase, 333
- Directory, 629
- DirectoryInfo, 630
- Dns, 672
- DynamicMethod, 781
- DynamicObject, 816
- Encoding, 240, 241
- Enumerable, 407, 420
- Environment, 294
- EqualityComparer, 337
- EventWaitHandle, 864
- ExpandoObject, 818
- File, 607, 626
- FileInfo, 630
- FileStream, 606, 609
- FileSystemWatcher, 634
- HashSet<T>, 322
- Hashtable, 326
- HttpClient, 652, 653, 662
- HttpClient, 649
- HttpMessageHandler, 655
- HybridDictionary, 328
- KeyedCollection<TKey,TItem>, 333
- Lazy<T>, 869
- LazyInitializer, 870
- LinkedList<T>, 318
- List<T>, 316
- ListDictionary, 328
- ManualResetEvent, 862
- Math, 273
- MemberInfo, 764
- MemoryStream, 609
- MessageBox, 935
- Monitor, 842
- Mutex, 847
- NetDataContractSerializer, 691
- Object, 117
- ObjectContext, 379, 381
- OrderedDictionary, 328
- Parallel, 880, 895
- ParallelLoopState, 898
- Path, 630
- PipeStream, 610
- Process, 294
- Queue<T>, 320
- Random, 275
- ReadOnlyCollection<T>, 335
- RSA, 836
- SortedSet<T>, 322
- Stack<T>, 321
- StackFrame, 529
- StackTrace, 529
- Stopwatch, 538
- StorageFolder, 636
- Stream, 601
- StreamReader, 617
- StreamWriter, 617
- StringBuilder, 238
- StringComparer, 340
- StringReader, 620
- StringWriter, 620
- SymbolInfo, 994, 995
- SyntaxNode, 978
- SyntaxNode, 979
- SyntaxTree, 981, 988
- System.Exception, 169
- System.Tuple, 188
- Task, 559
- TaskCompletionSource, 563
- TaskFactory, 910
- TcpClient, 676
- TcpListener, 676
- TextReader, 616
- TextWriter, 616
- ThreadLocal<T>, 871, 887
- TimeZone, 250
- TimeZoneInfo, 251
- TraceListener, 524
- TypeInfo, 997
- WebClient, 649, 659, 662
- WebRequest, 651, 662
- WebResponse, 651
- XElement, 486
- XmlConvert, 255, 269

- XmlReader, 474, 477, 481, 489
- XmlSerializer, 709, 714
- XmlWriter, 482, 484, 486, 488
- klasy
 - abstrakcyjne, 109
 - atrybutów, 189
 - bazowe, 105
 - jednostek Entity Framework, 378
 - jednostek LINQ to SQL, 377
 - kolekcji, 330
 - monitorowania, 523
 - po stronie klienta, 649
 - pochodne, 105
 - pomocnicze, 293
 - słownikowe, 325
 - statyczne, 37, 102
- klauzula
 - case, 77
 - catch, 164
 - else, 75
 - from, 352
 - inicjalizacji, 79
 - iteracyjna, 80
 - select, 404
 - warunkowa, 79
- klient, 649
- klienci bogate, 221
- klienci ubogie, 221
- klucze, 835
- kod niebezpieczny, 203
- kodowanie znaków, 239, 618
 - UTF-16, 241, 619
- kolejka, 316
 - komunikatów, 553
 - typu producent-konsument, 916
- kolejność
 - elementów, 885
 - składowych, 693
 - XML, 711
 - inicjalizacji pól, 102, 112
 - wykonywania działań, 69
- kolekcja pokoleniowa, 507
- kolekcje, 172, 217, 297, 330, 695
 - serializacja, 714
 - współbieżne, 913
- kolizje, 830
- komentarz, 34
 - dokumentacyjny, 208
- komparator, 338, 423
 - Comparer, 338
 - IComparer, 338
- kompilacja, 31, 991
 - emisja podzespołu, 993
 - sprawdzanie modelu semantycznego, 993
 - tworzenie, 991
 - warunkowa, 520, 521
- kompilator Roslyn, 975
- kompilowanie drzew wyrażań, 390
- komponenty bibliotek PFX, 880
- komponowanie sekwencji, 175
- kompresja, 627
 - strumienia, 622
- komunikaty
 - odpowiedzi, 654
 - żądania, 654
- konflikt nazw, 33
- konkatenacja
 - łańcuchów, 53
 - węzłów XText, 457
- konstrukcja
 - funkcyjna, 446
 - try-catch-finally, 549
- konstruktory, 31, 36, 111
 - bazowe, 795
 - egzemplarzy, 92
 - kolejność inicjalizowania pól, 93
 - niejawne bez parametrów, 93
 - niepubliczne, 93
 - przeciążanie, 92
 - statyczne, 101
- kontekst
 - asynchroniczności, 582
 - refleksji, 775
 - synchronizacji, 554
 - typizowany, 380
- konto użytkownika, 826
- kontrakty
 - danych, 695
 - kodu, 519
- kontrawariancja, 140, 150
- kontrola typów, 107
 - dynamiczna, 115
 - statyczna, 115
- kontynuacja zapytania, 366
- kontynuacje, 905
 - warunkowe, 907, 908
 - z wieloma przodkami, 909

- konwersje, 38, 136, 267
 - dynamiczne, 195, 268
 - dziesiętne, 45
 - jawne, 177, 201
 - liczb całkowitych, 44, 272, 278
 - liczb rzeczywistych, 44, 272, 268
 - łańcuchowe, 278
 - niejawne, 177, 201
 - typów zmiennoprzecinkowych, 45
 - typu logicznego, 50
 - wyliczeń, 126, 276
 - znaków, 53
- konwertery typów, 255, 270
- konwertowanie
 - na łańcuch, 255, 278
 - referencji, 106
 - uchwyty, 865
- kopiowanie tablic, 315
- kotwice, 962
- kowariancja, 137, 150
- krotki, 24, 185
 - dekonstruowanie, 187
 - nadawanie nazw, 186
 - porównywanie, 188
- kryptografia, 827
- kultury, 744
- kwantyfikatory, 437, 956, 960
 - leniwy, 961
 - zachłanny, 961

L

- lambda, 158, 348, 548
- LAN, Local Area Network, 645
- leniwa inicjalizacja, 868
- liczniki wydajności, 533
 - odczyt danych, 535
 - tworzenie, 536
 - wyświetlanie, 533
 - zapis danych, 536
- likwidowanie uchwytów oczekiwania, 861
- LINQ, Language Integrated Query, 27, 218, 343
 - filtrowanie, 399
 - GroupBy, 426
 - grupowanie, 424
 - kwantyfikatory, 437
 - metody agregacyjne, 433
 - metody generujące, 438
 - metody konwersji, 428
 - opakowywanie zapytań, 367
 - operatorzy, 395
 - operatorzy elementów, 431
 - operatorzy zbiorów, 427
 - podzapytania, 361, 405
 - porządkowanie, 421
 - porządkowanie naturalne, 350
 - projekcja, 403
 - przechwytywanie zmiennych, 357
 - składnia płynna, 345
 - wykonywanie opóźnione, 355, 358, 364
 - wykonywanie zapytania, 360
 - zapytania złożone, 364
 - złączenia, 405, 414
 - złączenia zewnętrzne, 412
- LINQ to SQL, 376
- LINQ to XML, 441
- listy, 316
- literały, 30, 34
 - liczbowe, 43
- lokalizacja, 271
- losowa kolejność elementów, 478

Ł

- ładowanie, 444
 - zestawu, 775
- łańcuch
 - null, 231
 - pusty, 231
- łańcuchy
 - dzielenie, 233
 - formatowania daty i godziny, 264
 - formatowania wyliczeń, 266
 - formatu, 256, 260
 - interpolacja, 54
 - klasa StringBuilder, 238
 - konkatenacja, 53, 233
 - łączenie dekoratorów, 359
 - łączenie operatorów zapytań, 346
 - modyfikowanie, 233
 - numeryczne formatu, 260
 - operatorów zapytań, 347
 - pobieranie znaków, 232
 - połączenia z jednostką, 379
 - porównywanie, 54, 235–237
 - przeszukiwanie, 232
 - strumienie szyfrowania, 833
 - tworzenie, 230

- zapisywanie deklaracji, 460
 - złożone formatu, 234
 - znaków, 52
 - łączenie
 - dekoratorów w łańcuchy, 359
 - łańcuchów, 233
 - operatorów zapytań, 346
 - łącznik zadań, 591, 593
 - łączność operatorów
 - lewostronna, 69
 - prawostronna, 69
- M**
- magazyn danych, 599, 638
 - odizolowany, 641
 - strumieni, 605
 - manifest
 - aplikacji, 719, 721
 - zestawu, 719, 720
 - mapowanie
 - plików, 638, 639
 - struktury, 943
 - mechanizm
 - cookies, 664
 - konwersji, 267
 - usuwania nieużytków, 493, 500, 506, 510
 - metadane, 21, 755
 - składowych, 765
 - metoda, 16, 30, 90
 - Abort, 873
 - Aggregate, 435
 - All, 438
 - Any, 437
 - AsEnumerable, 431
 - AsParallel, 884
 - AsQueryable, 431
 - Average, 434
 - Cast, 429
 - Close, 495
 - ComputeHash, 829
 - Concat, 427
 - Contains, 437
 - ContinueWhenAll, 909
 - ContinueWhenAny, 909
 - ContinueWith, 905
 - Count, 433
 - DefaultIfEmpty, 433
 - Delay, 565
 - Dispose, 498, 503
 - Distinct, 402
 - DoCallback, 926
 - ElementAt, 432
 - Empty, 438
 - Enqueue, 918
 - Enumerable.Where, 400
 - EqualityComparer<T>.Default, 338
 - Equals, 284, 285, 287, 290
 - Except, 428
 - First, 432
 - FirstNode, 449
 - Flatten, 911
 - ForAll, 891
 - get, 98
 - GetAmbiguousTimeOffsets, 251
 - GetAwaiter, 572
 - GetBuffer, 609
 - GetByteArrayAsync, 654
 - GetData, 872
 - GetHashCode, 286
 - GetMembers, 763
 - GetStreamAsync, 654
 - GetStringAsync, 654
 - GetType, 115
 - GroupBy, 424
 - GroupJoin, 414, 418
 - Handle, 912
 - IndexOf, 313
 - Interrupt, 873
 - Intersect, 428
 - IsAmbiguousTime, 251
 - IsControl, 231
 - IsDaylightSavingTime, 254
 - IsDigit, 231
 - IsInvalidTime, 251
 - IsLetter, 231
 - IsLetterOrDigit, 231
 - IsLower, 231
 - IsMatch, 956
 - IsNumber, 231
 - IsPunctuation, 231
 - IsSeparator, 231
 - IsSymbol, 231
 - IsUpper, 231
 - IsWhiteSpace, 231
 - Join, 414

metoda

- Last, 432
- LastIndexOf, 313
- LastNode, 449
- Load, 444
- LoadFile, 748
- LoadFrom, 748
- Lock, 609
- LongCount, 433
- Max, 433
- Min, 433
- Monitor.Enter, 841
- Monitor.Exit, 841
- MoveNext, 298
- Nodes, 449
- object.Equals, 283
- Object.Equals, 282
- object.ReferenceEquals, 283
- OfType, 429
- OperationCompleted, 583
- OperationStarted, 583
- Parallel.For, 896
- Parallel.ForEach, 896, 897
- Parallel.Invoke, 895
- Parse, 255, 444
- ParseText, 980
- Range, 439
- ReadInnerXml, 479
- ReadSubtree, 480
- ReadUInt32, 679
- Regex.Match, 956
- Repeat, 439
- Resume, 874
- Select, 403
- SelectMany, 407–412
- Send, 673
- SendAsync, 654, 656
- SequenceEqual, 438
- set, 98
- SetData, 872
- SignalAndWait, 866
- Single, 432
- Skip, 401
- SkipWhile, 402
- Stop, 495
- string.Format, 234
- Sum, 434
- Suspend, 874

- Take, 401
- TakeWhile, 402
- Task.WhenAll, 592
- Task.WhenAny, 591
- ToArray, 430
- ToList, 430
- ToLookup, 420, 430
- ToLower, 230
- ToString, 116, 248, 255, 445
- ToUpper, 230
- TryEnter, 842
- Union, 427
- ValueTuple.Create, 187
- WaitAll, 866, 904
- WaitAny, 866
- Where, 399
- WriteLine, 617

metody

- agregacyjne, 433
- anonimowe, 162
- asynchroniczne, 582
- częściowe, 28, 103
- dostępowe, 97, 98
- dostępowe zdarzenia, 152, 156
- egzemplarzy, 146, 183
- generujące, 438
- generyczne, 132, 772, 796
- instancji, 785, 792
- konwersji, 428
- lokalne, 23, 91, 161
- przeciążanie, 91
- rozszerzające, 27, 181, 348
- skrótów, 607
- statyczne, 146
- synchroniczne, 588
- wczytujące, 479
- wyraźniowe, 91
- miejsce wywołania, 808
- mieszanie operatorów, 179
- międzyprocesowa praca zdalna, 930
- MMC, Microsoft Management Console, 533
- model
 - DataSet, 225
 - obiektowy Reflection.Emit, 789
 - semantyczny, 991
 - wykonawczy PLINQ, 883
- moduł, 722, 775
 - równoległy, 886
 - sprawdzania pisowni, 886

modyfikator
 async, 571
 out, 63
 params, 64
 readonly, 90
 ref, 62
 modyfikatory
 dostępu, 118, 120
 zdarzeń, 157
 modyfikowanie
 drzewa X-DOM, 452
 łańcuchów, 233
 węzłów potomnych i atrybutów, 453
 monitorowanie, 523
 domen aplikacji, 927
 dziennika zdarzeń, 532
 MSMQ, 226
 multemisja, 145

N

nadawca, 151
 nadmierne przekazywanie, 586
 nagłówki, 662
 nakładanie blokad, 546
 nawigacja, 448
 do rodzica, 451
 na tym samym poziomie, 452
 po atrybutach, 452
 po węzłach potomnych, 448
 nazwy
 elementów krotek, 186
 plików, 607
 symbolów, 998
 typów, 758
 generycznych, 758
 osadzonych, 758
 parametrów ref i out, 759
 tablic, 759
 zestawów, 727
 niebezpieczny kod, 202
 niejawne
 określanie typów, 67
 parametry ref, 950
 typowane zmienne lokalne, 27
 wywoływanie konstruktora, 112
 numeryczne łańcuchy formatu, 261

obiekt typu
 DateTime, 245
 DateTimeOffset, 246
 obiektowość, 15
 obiekty, 59, 94
 niezmienne, 853
 potomne, 711
 stanu, 902
 synchronizacji, 842
 szyfrowania, 834
 obliczanie skrótów, 828, 829
 obsługa
 łańcuchów i tekstu, 229
 wyjątków, 549, 550, 660, 787
 oczekiwanie, 571
 na interfejs użytkownika, 573
 odczytywanie elementów, 477
 odrzucenia, 22, 64
 odwołania
 do obiektu, 690
 do samego siebie, 136
 do składowych, 211
 do typów, 211
 lokalne ref, 66
 ograniczenia
 dostępności, 120
 PLINQ, 886
 typów generycznych, 134
 opakowywanie zapytań, 367
 opcje wyrażeń regularnych, 958
 operacje
 asynchroniczne, 566
 na katalogach, 625
 na plikach, 625
 synchroniczne, 565
 wejścia-wyjścia, 219, 635, 638
 operator, 34, 70
 !=, 281
 &, 179
 |, 179
 <, 291, 401
 ==, 281, 284
 >, 291, 401
 adresowania, 203
 as, 107
 AsEnumerable, 375

operator

- AsQueryable, 391
- dereferencji, 203
- is, 107
- mnożenia, 31
- nameof, 104
- OrderBy, 422
- OrderByDescending, 422
- porównywania, 200
- przypisania, 34
- równości, 200
- sprawdzania null, 72
- ThenBy, 422
- ThenByDescending, 422
- typeof, 115, 133
- warunkowy, 51
- warunkowy null, 25, 72
- wskaźnika do składowej, 203, 204
- zapytania, 343

operatory

- arytmetyczne, 45
- bitowe, 47
- elementów, 351, 431
- inkrementacji i dekrementacji, 46
- LINQ, 395
- null, 179
- porównywania i równości, 50
- przypisania, 68
- relacyjne, 178
- równości, 178
- sprawdzania przepelnienia
 - całkowitoliczbowego, 46
- warunkowe, 51
- wyliczeń, 127
- zapytań, 396
 - sekwencja → element lub wartość, 398
 - sekwencja do sekwencji, 397
 - Void → sekwencja, 399
- zbiorów, 427
- zmieniające kształt, 397

opóźnienie podpisania, 726

optymalizacja, 507, 584

- agregacji, 893
- PLINQ, 890
- z wartościami lokalnymi, 900

osadzanie

- typów współpracujących, 952
- zasobów, 737

ostrzeżenia pragma, 208

P

pakowanie, 114, 124

- wartości typów, 177

pamięć, 41, 511

- lokalna wątku, 871
- niezarządzana, 943
- współdzielona, 639, 940

paralelizm strukturalny, 880

parametr, 58, 61

- out, 769
- ref, 769

parametry

- metod, 768
- niejawne ref, 950
- opcjonalne, 27, 65, 95, 949
- typów, 136

parsowanie, 255, 258, 265, 444

- argumentów, 783
- IL, 801
- liczb, 268

pary zastępcze, 241

pętle

- do-while, 79
- for, 79
- foreach, 80
- while, 79

pętle

- wewnętrzne, 897
- wychodzenie, 898
- zewnętrzne, 897

PFX, 879

pieczętowanie

- funkcji, 110
- klas, 110

pierwszy program, 29

planowanie zadań, 909

platforma .NET Framework, 18, 213

plik signtool.exe, 732

pliki, 637

- .resources, 739, 740
- .resx, 739, 740
- metadanych, 21

PLINQ, 880, 882

- anulowanie zapytania, 889
- czystość funkcyjna, 888
- ograniczenia, 886
- optymalizacja, 890
- ustawianie stopnia zrównoleglenia, 889
- zastosowanie, 888

- pobieranie
 - atrybutów
 - elementów, 449
 - elementów potomnych, 450
 - jednego elementu, 450
 - typów osadzonych, 757
 - typu, 756
 - w czasie działania, 779
 - w kontekście refleksji, 780
- poczta elektroniczna, 673, 677
- podklasy, 711
 - obiektów potomnych, 713
 - typów generycznych, 135
- podpisy cyfrowe, 837
- podpisywanie, 835
 - programu, 732
 - zestawu, 724
- podprocedura, 699
- podzapytania, 361, 364, 405
- poła, 36, 89
 - egzemplarzowe, 872
- polimorfizm, 105
 - wielokierunkowy, 813
- POP, Post Office Protocol, 645
- POP3, 677
- porównywanie
 - łańcuchów, 54, 235–237
 - strukturalne, 341
- porty, 646
- porządkowanie, 336
 - naturalne, 350
- pośredniki, 330
- potoki
 - anonimowe, 610, 612
 - nazwane, 610, 611
- pożyczanie operatorów, 177
- praca zdalna, 930
- precyzja, 45
- prefiksy adresów URI, 652
- procesy, 528
- programowanie
 - asynchroniczne, 566, 567, 594
 - dynamiczne, 220, 807
 - równoległe, 221, 879, 882
- projekcja, 368, 403, 467
 - do typów konkretnych, 406
 - z indeksowaniem, 404
- protokoły równości, 281, 336
- protokół
 - HTTP, 661
 - SMTP, 673
- proxy, 657
- przechwytywanie
 - stanu lokalnego, 572
 - zdarzeń, 634
 - zmiennych, 357, 548
 - iteracyjnych, 160
 - zewnętrznych, 159
- przeciążanie, 113
 - false, 202
 - konstruktorów, 92
 - metod, 91
 - operatorów, 199, 200, 287
 - true, 202
- przekazywanie
 - argumentów przez referencję, 62, 64, 91
 - argumentów przez wartość, 61, 91
 - danych, 654
 - stanu, 820
 - wyjątku, 583
- przeliczalność, 297
- zpełnienie całkowitoliczbowe, 46
- przesłanie metody, 286
- przestrzeń nazw, 31, 82, 297
 - import, 86
 - kontraktu danych, 688
 - kwalifikatory aliasów, 87
 - powtarzanie, 85
 - System.Reflection, 789
 - w XML, 461
 - zaawansowane właściwości, 86
 - zakres, 84
- przeszukiwanie
 - drzewa, 981
 - tablic, 312
- przetwarzanie tekstu, 217
- przewidywanie, 961
 - pozytywne, 961
 - wsteczne, 961
- przypisanie, 60
 - atrybutów, 795
 - ról, 824
 - użytkowników, 824
 - wielu atrybutów, 190
- przyrostki literalów liczbowych, 44
- przysłanie metod, 792

pseudotyp, 308
pula wątków, 555
punkt kontrolny, 527

R

referencja, 952
 this, 96
refleksja, 220, 755, 762
 dla zestawów, 774
 składowych, 763
reguły asynchroniczności, 565
rekurencja blokowania, 859
remoting, 227
repozytorium GAC, 735
REST, REpresentational State Transfer, 645
rodzina ReadXXX, 479
role, 822
Roslyn, 975
 architektura, 976
 drzewo składni, 977
rozgałęzianie, 785
rozpakowywanie, 114
rozpoznawanie, 113
rozszerzenie kontraktu danych, 697
równoległa biblioteka zadań, 879
równoległe wykonywanie zadań, 901
równoległość, 580
równość, 285
 referencyjna, 280
 wartościowa, 280
równoważenie obciążenia, 892
równoważność typów, 952
rzutowanie, 38, 106
 w dół, 106
 w górę, 106

S

schematy, 488
sekwencje, 343
 specjalne, 52
semafony, 854
semantyka iteratorów, 173
serializacja, 219, 445, 681, 697, 708
 binarna, 683, 700, 702, 706
 jawna, 685
 kolekcji, 714
 kontraktu danych, 683, 685

 na podstawie atrybutów, 709
 niejawna, 685
 obiektów potomnych, 712
 podklas, 689
 XML, 709
serializator
 DataContractSerializer, 686, 687
 NetDataContractSerializer, 686
serwer
 HTTP, 667
 proxy, 657
sieć, 219, 643
silne nazwy, 724
silnik serializacji, 682
 binarnej, 700
 XML, 684
Silverlight, 224
składnia, 32
 płynna, 345, 354
 SQL, 354
 zapytaniowa, 354
składniki PFX, 881
składowe, 109
 C#, 766
 CLR, 766
 egzemplarza, 37
 funkcyjne, 108
 niepubliczne, 770
 statyczne, 37, 851
 typów generycznych, 767
 typu, 36
 z wyrażeniami, 23
skrót, 829
skrypt, 820
słabe odwołania, 515
słowa kluczowe, 32
 kontekstowe, 33
słowniki, 324
 sortowane, 329
słowo kluczowe
 add, 152
 await, 571
 base, 111
 extern, 86
 fixed, 946
 into, 366
 let, 369
 new, 110, 785

- override, 110
- partial, 103
- public, 37
- remove, 152
- sealed, 110
- stackalloc, 204
- var, 67
- virtual, 108
- SMTP, Simple Mail Transfer Protocol, 645, 673
- sortowanie, 314, 423
- sprawdzanie
 - poprawności
 - dokumentu, 489
 - drzewa X-DOM, 490
 - schematów, 488
 - przepełnienia, 47
 - równości, 280
 - typów, 933
- SSL, 667
- stała, 35, 100
- statyczna kontrola typów, 16
- sterta, 59
- ogromnych obiektów, 508
- stopień zrównoleglenia, 889
- stos, 58, 316
 - ewaluacji, 782
- strategie projekcji, 368
- strefy czasowe, 249, 250
- struktura, 117
 - Area, 288
 - BigInteger, 273
 - Complex, 274
 - DateTime, 243–249, 253
 - DateTimeOffset, 243, 247, 250
 - DOM, 391
 - Guid, 279
 - Nullable<T>, 176, 177
 - SyntaxTree, 977
 - TextSpan, 983
 - TimeSpan, 242
- strumienie, 219, 599, 601
 - adaptery, 614
 - binarne, 620
 - tekstowe, 615
 - kompresja, 622
 - limit czasu, 605
 - magazyn danych, 605
 - odczyt, 603
 - opróżnienie, 604
 - wyszukiwanie, 604
 - zamknięcie, 604
 - zamykanie adapterów, 621
 - zapis, 603
- strumieniowanie projekcji, 470
- strumień
 - BufferedStream, 613
 - FileStream, 606
 - MemoryStream, 609
 - PipeStream, 610
- subkultury, 744
- subskrybenci, 151
- surogat, 242
- sygnalizowanie, 552, 840, 859
- sygnały dwustronne, 861
- sygnatura metody, 90
- sygnatury Func, 349
- symbole, 994
 - dostępność, 996
 - wyszukiwanie, 997
 - zadeklarowane, 996
 - zmiana nazwy, 998
- symulowanie unii, 939
- synchronizacja, 840
- system
 - plików, 634
 - typów COM, 947
- szablony, 141
- szeregowanie
 - In i Out, 938
 - klas i struktur, 937
 - typów wspólnych, 936
- szyfrowanie, 627, 828
 - kluczem publicznym, 835
 - symetryczne, 831
 - w pamięci, 832

Ś

- śledzenie obiektów, 381
- środowiska niszowe, 20

T

- tabela operatorów, 69
- tablice, 54, 204
 - bajtów, 241
 - indeksowanie, 310
 - indeksy, 58

- tablice
 - konwertowanie, 315
 - kopiowanie, 315
 - liczba wymiarów, 312
 - nieregularne, 56
 - odwracanie kolejności elementów, 315
 - prostokątne, 56
 - przeglądanie zawartości, 312
 - przeszukiwanie, 312
 - skrótów, 286
 - sortowanie, 314
 - tworzenie, 310
 - w pamięci, 309
 - wyrażenia inicjalizacji, 57
 - zmienianie rozmiarów, 315
- TCP, Transmission and Control Protocol, 645, 673, 676–679
- techniki
 - optymalizacji, 507
 - synchronizacji, 840
- technologia Authenticode, 730
- technologie
 - systemów rozproszonych, 226
 - XML, 473
 - zapleczone, 224
- tekst
 - dzielenie, 965
 - zastępowanie, 965
- testowanie zestawów satelickich, 743
- testy jednostkowe, 655
- token anulowania, 889
- tożsamość, 822
- TPL, Task Parallel Library, 879
- transformacja drzewa składni, 987
- tryb pliku, 607
- tworzenie
 - asercji, 524
 - deasemblera, 801
 - domen aplikacji, 923
 - drobiazgów, 988
 - drzewa X-DOM, 445
 - egzemplarzy, 36
 - funkcji asynchronicznych, 577
 - instancji obiektów, 785
 - instancji typów, 760
 - kompilacji, 991
 - literału krotki, 185
 - łańcuchów, 230
 - łańcuchów strumieni szyfrowania, 833
 - międzyprocesowego uchwytu, 864
 - podklas typów generycznych, 135
 - serwera HTTP, 667
 - struktur, 117
 - tablic, 310
 - tokenów, 988
 - typów, 89
 - wątku, 540
 - węzłów, 988
 - wyrażen lambda, 348
 - zadań, 902
 - zapytań złożonych, 364
 - zestawu satelickiego, 742
- typ, 34
 - bool, 60
 - char, 60, 229
 - CultureInfo, 257
 - DateTimeFormatInfo, 257
 - decimal, 49
 - delegacyjny, 143
 - double, 48, 49
 - dynamic, 195, 196
 - enum, 276
 - float, 48
 - logiczny, 50
 - NumberFormatInfo, 257
 - object, 78, 113
 - osadzony, 757
 - string, 53, 230
 - var, 196
 - wyliczeniowy RegexOptions, 957
 - zwrotny, 30, 143
 - zwrotny ref, 67
- TypeInfo, 763
- typizowanie elementów, 349
- typy, 16, 31
 - anonimowe, 27, 184, 368
 - bazowe, 759
 - całkowitoliczbowe, 48
 - częściowe, 103
 - definiowane, 285
 - delegacyjne, 147, 149
 - delegatów generycznych, 151
 - dopuszczające wartość null, 179
 - generyczne, 130, 141, 761, 797
 - niewiązane, 133
 - liczbowe, 42
 - unifikacja, 809

niezmiennie, 98
parametrów lambda, 159
predefiniowane, 35
referencyjne, 38, 42, 55, 60, 285
składowych, 764, 765
statyczne, 198
systemowe, 217
tablicowe, 139, 757
wartościowe, 38, 42, 55, 176, 285
węzłów, 979
własne, 36
współpracujące COM, 948
wyjątków, 169
wyrażeń, 392
zagnieżdżone, 128
zależności cykliczne, 799
zamknięte, 798
zadań sieciowych, 652

U

uchwyt oczekiwania
 na zadania, 865
 na zdarzenia, 840, 859
UDP, Universal Datagram Protocol, 645, 673
ukrywanie odziedziczonych składowych, 109
UNC, Universal Naming Convention, 645
unia, 939
Unicode, 239
unifikacja typów liczbowych, 809
uprawnienia, 822, 827
URI, Uniform Resource Identifier, 645
URL, Uniform Resource Locator, 645
uruchomienie zadania, 558, 902
urząd certyfikacji, 731
usługi P/Invoke, 935
usuwanie
 elementów składowych, 498
 nieużytków, 493, 499–501, 506, 510
 sekwencji węzłów, 454
uwierzytelnianie, 658, 660
 na podstawie formularzy, 666
UWP, Universal Windows Platform, 224, 635, 638

V

Visual Studio, 743
Voice over IP, 673

W

wariancja, 138
 parametrów, 151
 typów, 27
warstwa dostawcza, 224
wartości
 puste, 694
 skrótów, 286
 specjalne typów, 48
 zwrotne, 559
wartość null, 41, 176–180, 248, 694
wątki, 220, 528, 540, 927
 aktywne, 550
 bezpieczeństwo, 546, 605, 848
 blokowanie, 543
 dołączanie, 542
 działające w tle, 550
 interfejsu użytkownika, 554
 pamięć lokalna, 871
 priorytet, 551
 przekazywanie danych, 547
 pula, 555
 spinning, 543
 stan lokalny, 544
 stan współdzielony, 544
 sygnalizowanie, 552
 tworzenie, 540
 usypianie, 542
 w bogatych aplikacjach, 552
wczesne ładowanie, 386, 387
wczytywanie
 atrybutów, 480
 węzłów, 475
wdrożenie manifestu aplikacji, 722
Web API, 227
wejście-wyjście, 599, 635
wersjonowanie, 704, 735
weryfikacja Authenticode, 733
węzły, 979
 atrybutów, 480
 XML, 475
wiązanie
 dynamiczne, 26, 192, 951
 językowe, 194
 niestandardowe, 193
 statyczne, 192

- widok wyszukiwania, 419
- wielowątkowość, 839
- Windows Communication Foundation, 226
- Windows Data Protection, 828
- Windows Forms, 223
- Windows Presentation Foundation, 223
- Windows Runtime, 21
- Windows Workflow, 226
- WinRT, 501, 679
 - metody asynchroniczne, 582
- wirtualizacja, 827
- wirtualne składowe funkcyjne, 108
- własności, 16, 31, 96
 - automatyczne, 28, 98
 - obliczane, 97
 - tylko do odczytu, 97
 - wyrażeń, 97
- właściwość
 - CodeBase, 749
 - Location, 749
- wolumin, 634
- wpisywanie
 - atributów, 483
 - typów węzłów, 483
- wrapper CCW, 953
- wskazniki, 202, 759
 - do kodu niezarządzanego, 206
 - puste, 205
- wskrzeszenie, 504
- współbieżność, 219, 539, 676
 - drobnoziarnista, 567
 - gruboziarnista, 567, 576
- współdziałanie macierzyste, 935
- wtyczki, 144
- wybór trybu pliku, 608
- wychodzenie z pętli, 898
- wyciek pamięci, 511, 514
- wydajność, 770, 847
- wyjątek
 - IndexOutOfRangeException, 357
 - NullReferenceException, 560
 - OperationCanceledException, 588
 - RuntimeBinderException, 194
 - WebException, 660
- wyjątki, 163, 549, 560, 660, 787
 - klasa System.Exception, 169
 - ponawianie zgłoszenia, 168
 - typy, 169
 - zgłaszanie, 167
- wykonywanie opóźnione, 355, 358, 364, 384
- wyliczenia, 125, 171, 276, 278
 - BindingFlags, 771
 - DateTimeStyles, 266
 - TaskCreationOptions, 903
- wymazywanie typów, 186
- wyrażenia, 67
 - dekodowanie znaków, 970
 - dopasowanie liczb rzymskich, 968
 - dopasowanie numeru, 967
 - dynamiczne, 196
 - dzielenie tekstu, 965
 - granica słowa, 963
 - grupy, 964
 - kodowanie znaków Unicode, 969
 - kompilowane, 957
 - konstrukcje grupujące, 972
 - kotwice, 962
 - kwantyfikatory, 960
 - lambda, 16, 27, 158, 161, 348, 548
 - asynchroniczne, 581
 - leksykon, 970
 - licznik słów, 968
 - odwołania wsteczne, 973
 - opcje, 958, 973
 - pary nazwa=wartość, 967
 - podstawowe, 68
 - podział słowa, 969
 - prawidłowe nazwy pliku, 969
 - przetwarzanie
 - daty, 968
 - wyrażeń, 970
 - znacznika HTML, 969
 - przewidywanie, 961
 - przypisania, 68
 - puste, 68
 - receptury, 967
 - regularne, 955
 - alternatywy, 973
 - typy statyczne, 198
 - usunięcie powtórzonych słów, 968
 - wartości GUID, 969
 - weryfikacja silnego hasła, 967
 - wiersze, 968
 - zapytań, 16, 27, 351, 389
 - zastępowanie tekstu, 965
 - zestawy znaków, 959
 - znaki sterujące, 958
- wyrażenie throw, 24, 167

- wysyłanie
 - poczty elektronicznej, 673
 - zapytań, 448
- wyszukiwanie symboli, 997
- wywoływanie
 - anonimowe składowych, 772, 813
 - dynamiczne, 197
 - dynamiczne składowej, 768
 - komponentu COM, 948
 - konstruktorów bazowych, 795
 - metod
 - generycznych, 769
 - instancji, 785
 - rozszerzających, 182
 - składowych, 762
 - zwrotne, 144, 939
- wzorzec, 22
 - asynchroniczności, 586, 594
 - opartej na zdarzeniach, 595
 - oparty na zadaniu, 590
 - metod TryXXX, 170
 - Wizytator, 810
 - zdarzeń, 153

X

- Xamarin, 224
- XDocument, 458
- X-DOM, 442, 445
 - adnotacje, 466
 - aktualizowanie danych, 454
 - definiowanie treści, 446
 - deklaracje, 458
 - dokumenty, 458
 - domyślne przestrzenie nazw, 464
 - eliminowanie pustych elementów, 469
 - głębokie klonowanie, 447
 - klasa XmlReader, 486
 - klasa XmlWriter, 486
 - konkatenacja węzłów, 457
 - konstrukcja funkcyjna, 446
 - modyfikowanie
 - wartości, 453
 - drzewa, 452
 - węzłów potomnych, 453
 - nawigowanie, 448
 - nazwy, 461
 - pobieranie wartości, 456
 - projekcja, 467

- przekształcanie drzewa, 470
- przestrzenie nazw, 461
- sprawdzanie poprawności drzewa, 490
- strumieniowanie projekcji, 470
- ustawianie wartości, 455
- wartości, 457
- węzły z treścią mieszaną, 457
- wysyłanie zapytań, 448
- XML, 218, 473
- XSD, 488
- XSLT, Extensible Stylesheet Language Transformations, 491

Z

- zabezpieczanie
 - dostępu kodu, 821
 - systemu operacyjnego, 825
- zadania, 557
 - anulowanie, 904
 - autonomiczne, 560
 - długo wykonywane, 559
 - kontynuacje, 561, 905
 - kontynuacje warunkowe, 907
 - planowanie, 909
 - potomne, 903, 907
 - tworzenie, 902
 - uruchamianie, 558, 902
 - zimne, 918
- zagnieżdżanie blokad, 845
- zakleszczenia, 846
- zakres przestrzeni nazw, 84
- zaokrąglanie liczb, 49
- zapisywanie, 445
- zapytania, 218
 - interpretowane, 370, 372, 373
 - LINQ, 343
 - o składni mieszanej, 355
 - opakowane, 367
 - PLINQ, 882
 - złożone, 364
- zarządzanie
 - kluczami, 835
 - pamięcią, 17
- zasada pewnego przypisania, 60
- zasoby, 736
- zasób pack URI, 741
- zbieranie obiektów, 509
- zbiory, 316

- zdarzenia, 16, 31, 151, 516
 - metody dostępowe, 156
 - modyfikatory, 157
 - standardowy wzorzec, 153
 - systemu plików, 634
- zdarzenie AssemblyResolve, 746
- zegary, 513, 875
 - jednowątkowe, 877
 - wielowątkowe, 875
- zestawy, 31, 220, 719
 - emitowanie, 787
 - izolowanie, 931
 - ładowanie do kontekstu refleksji, 775
 - nazwy, 727
 - odwołania, 753
 - opóźnienie podpisania, 726
 - podpis, 724, 731
 - pojedynczy plik wykonywalny, 751
 - referencyjne, 216
 - refleksja, 774
 - satelickie, 736, 742
 - silne nazwy, 724
 - testowanie, 743
 - tworzenie, 742
 - ustalanie, 745
 - wczytywanie, 745, 746
 - wdrażanie, 750
 - wersja informacyjna, 729
 - wersja pliku, 729
 - wyszukiwanie, 745
 - zapisywanie, 789
 - zaprzyjaźnione, 119
 - znaków, 239
- zgłaszanie wyjątków, 167
- zgodność
 - delegatów, 149
 - parametrów, 150
 - typów, 149
 - typów zwrotnych, 150
- złączenia, 405
 - w składni płynnej, 417
 - z widokami wyszukiwania, 419
 - zewnętrzne, 412, 419
- złożone
 - łańcuchy formatu, 234, 248
 - operatory przypisania, 68
- zmiana definicji równości, 285
- zmiennne, 35, 58
 - delegacyjne, 144
 - iteracyjne, 160
 - lokalne, 30, 74, 784
 - wyjściowe, 22, 64
 - wzorcowe, 22, 108
 - zakresowe, 352, 409
 - zewnętrzne, 159
- znaczniki
 - dokumentacyjne XML, 209
 - niestandardowe, 211
- znaki
 - interpunkcyjne, 34
 - sterujące, 958
- znakowanie czasowe, 732
- zrównoleganie, 889, 901
 - wykonywania zadań, 880
- zużycie pamięci, 500
- zwalnianie zasobów, 493–497
- zwrot egzemplarza Task<TResult>, 578

Ż

- żądania sieciowe, 652

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**

C# w nowej odsłonie – nowoczesny język dla profesjonalistów!

C# jest uważany za flagowy projekt firmy Microsoft. Charakteryzuje się niezwykłą elastycznością i wszechstronnością. Udostępnia wysokopoziomowe abstrakcje, takie jak wyrażenia, zapytania i kontynuacje asynchroniczne, ale też pozwala na korzystanie z mechanizmów niskopoziomowych. W efekcie dzięki takim konstrukcjom jak własne typy wartościowe programiści czy opcjonalne wskaźniki można znacząco zwiększyć wydajność aplikacji.

Niniejsza książka jest kolejnym, uzupełnionym i zaktualizowanym wydaniem cenionego kompendium wiedzy o C#, CLR oraz o związanej z C# platformie. W zrozumieli, a równocześnie dogłębny sposób omówiono nowe składnie języka C# 6.0 i 7.0 oraz wyjaśniono takie trudne kwestie, jak współbieżność, bezpieczeństwo i domeny aplikacji. Znalazły się tu także informacje o nowym kompilatorze Roslyn. Książka jest przeznaczona dla średnio zaawansowanych programistów.

Najważniejsze zagadnienia ujęte w książce:

- składnia, typy oraz zmienne C#
- wskaźniki, przeciążanie operatorów i wiązanie dynamiczne
- programowanie współbieżne i asynchroniczne, praca z wątkami i programowanie równoległe
- kompilator Roslyn – jego architektura, struktura drzewa składni i model semantyczny

Joseph Albahari – jest autorem kilku książek dotyczących C# oraz LINQ. Napisał też LINQPad, popularny program do testowania zapytań LINQ.

Ben Albahari – był kierownik programowy w Microsoftzie; współtworzył takie projekty jak .NET Compact Framework i ADO.NET. Jeden z założycieli firmy Genamics zajmującej się produkcją narzędzi dla programistów C# i J++ oraz oprogramowania do analizy DNA i sekwencjonowania białek. Jest autorem i współautorem kilku książek dotyczących C#.

 Helion	<i>Sprawdź nasze szkolenia!</i>	KOD KORZYŚCI <i>Słęgnij po więcej!</i>	
 helion.pl	 SZKOLENIA	ISBN 978-83-283-4075-6	
 HELION SA ul. Kościuszki 1c 44-100 Gliwice tel.: 32 230 98 63 helion@helion.pl	AKADEMIA IT & BUSINESS WWW.SZKOLENIA.HELION.PL	9 788328 340756	
INFORMATYKA W NAJLEPSZYM WYDANIU		Cena: 129,00 zł	