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

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.

Spis treści

5

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 zapleczowe;
- 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. Podobieństwom 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 któregokolwiek 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 (`IFormat`
`↪table`) 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 odnosić się do bieżącej domeny za pośrednictwem klasy `AppDomain` z przestrzeni nazwy `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 „bogatych". 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 kontrolek są ustalane w pikselach, przez co aplikacja może źle wyglądać w urządzeniu mającym ekran o innych ustawieniach DPI niż ekran programisty (chociaż trochę to poprawiono we Framework 4.7).
- Do rysowania niestandardowych kontrolek 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 kontrolek 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 kontrolek.

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 zapleczowe

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 bufory zapewniają możliwość przechowywania i serializacji zmian stanu (coś, co jest bardzo ważne w aplikacjach wielowarstwowych). Technologii 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 maperów 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 jakiejkolwiek 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`.

Skorowidz

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

V

W

X

Z

Ż

O'REILLY®

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 programisty 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 zrozumiały, 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 — były kierownik programowy w Microsofcie; 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
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

ISBN 978-83-283-4075-6
9 788328 340756

INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 129,00 zł