

IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK

DODAJ DO KOSZYKA

CENNIK I INFORMACJE

ZAMÓW INFORMACJE
O NOWOŚCIACH

ZAMÓW CENNIK

CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

100 sposobów na Visual Studio

Autor: James Avery

Tłumaczenie: Bartłomiej Garbacz, Daniel Kaczmarek

ISBN: 83-246-0064-7

Tytuł oryginału: [Visual Studio Hacks](#)

Format: B5, stron: 472



Podnieś wydajność Visual Studio i przyspiesz swoją pracę

- Efektywne zarządzanie projektami i kodem źródłowym
- Tworzenie i wykorzystywanie makroinstrukcji
- Wyszukiwanie i usuwanie błędów w kodach

Visual Studio to jedno z najpopularniejszych środowisk programistycznych. Za jego pomocą można tworzyć programy w kilku językach, a łatwość obsługi sprawia, że już po kilku godzinach pracy jego użytkownik sprawnie porusza się po interfejsie i wykorzystuje większość jego możliwości. Visual Studio oferuje ogromną liczbę funkcji, pozwala na wszechstronną konfigurację, umożliwia automatyzację zadań i posiada wiele innych cech podnoszących komfort pracy programisty. Środowisko to posiada jednak wiele innych ciekawych cech i możliwości, których opisu nie znajdziemy w dokumentacji.

Książka „100 sposobów na Visual Studio” to unikatowy zbiór 100 wskazówek opracowanych przez programistów używających tego narzędzia w codziennej pracy i znających wszystkie jego tajniki. Opisuje funkcje, dodatki, makra oraz rozszerzenia, które pozwalają w jeszcze większym stopniu rozszerzyć funkcjonalność tego środowiska. Wykorzystywanie zawartych tu informacji pozwoli każdemu użytkownikowi Visual Studio na przyspieszenie swojej pracy i uczynienie jej bardziej efektywną dzięki zastosowaniu jego mniej znanych, a bardzo przydatnych funkcji.

- Zarządzanie plikami projektów
- Konfigurowanie edytora kodu źródłowego i przestrzeni roboczej
- Refaktoryzacja kodu
- Zapisywanie i przenoszenie ustawień środowiska
- Dostosowywanie procesu kompilacji i wykrywania błędów
- Wykorzystywanie szablonów i makr
- Stosowanie komentarzy XML
- Automatyzacja poleceń

Pisanie doskonałego oprogramowania wymaga opanowania wszystkich możliwości środowiska programistycznego. Dzięki tej książce Visual Studio odkryje wszystkie swoje tajemnice.



Spis treści

O autorach	7
Wstęp	11
Rozdział 1. Praca z projektami i rozwiązaniami	17
1. Zarządzanie projektami i rozwiązaniami	17
2. Praca z odwołaniami do podzespołów i projektów	26
3. Organizacja projektów i rozwiązań	31
4. Sposób na pliki projektów i rozwiązań	35
5. Usuwanie dowiązań do SourceSafe	42
Rozdział 2. Praca z edytorem	47
6. Praca ze schowkiem	47
7. Łatwe wklejanie tekstu do Visual Studio	50
8. Stosowanie IntelliSense	55
9. Stosowanie regionów	60
10. Dodawanie linii siatki w edytorze tekstu	63
11. Wybór najlepszego edytora	64
12. Dostosowywanie sposobu kolorowania składni	69
13. Edytowanie listy ostatnio używanych plików i projektów	71
14. Refaktoryzacja kodu	74
15. Używanie i współużytkowanie fragmentów kodu	82
Rozdział 3. Nawigacja w Visual Studio	85
16. Kontrola przestrzeni roboczej	85
17. Szybkie odnajdywanie plików	91
18. Szybkie przeszukiwanie plików	94
19. Wyszukiwanie wzorców w plikach	95
20. Nawigacja po kodzie źródłowym	101
21. Lista zadań do wykonania	103
22. Sposób na mysz	106
23. Wzbogacanie zakładek w Visual Studio	107

Rozdział 4. Dostosowywanie Visual Studio	109
24. Tworzenie własnych skrótów klawiszowych	109
25. Dostosowywanie menu i pasków narzędziowych	113
26. Tworzenie własnych układów okien	118
27. Dostosowywanie okna narzędziowego	120
28. Program wdrażający ustawienia okna narzędziowego	123
29. Dodawanie obsługi pliku z niestandardowymi rozszerzeniami	126
30. Sposób na rejestr	127
31. Zapisywanie i przenoszenie ustawień Visual Studio	131
32. Uruchamianie mechanizmu IntelliSense w dokumentach HTML i XML	134
33. Dodawanie narzędzi zewnętrznych	139
34. Dostosowywanie procesu kompilacji projektu	142
35. Modyfikacja wyników kompilacji i nawigacja po wynikach kompilacji	148
Rozdział 5. Debugowanie	153
36. Ustawianie punktów przerwań	154
37. Rozwiązywanie problemów z punktami przerwań	160
38. Dostosowywanie reakcji Visual Studio na występowanie wyjątków	163
39. Debugowanie kodu w językach skryptowych	165
40. Debugowanie kodu wykonywanego na serwerze SQL Server	168
41. Debugowanie uruchomionego procesu	171
42. Debugowanie nieprawidłowo działającej aplikacji	175
43. Tworzenie własnego wizualizatora	179
Rozdział 6. Sposoby na przyspieszanie pracy	185
44. Przyspieszanie pracy Visual Studio	185
45. Ładowanie plików z wiersza poleceń	187
46. Praca z oknem poleceń	189
47. Tworzenie specjalnego narzędzia	193
48. Rozszerzanie pliku konfiguracyjnego aplikacji	197
49. Generowanie zestawów danych o silnym typie	202
50. Definiowanie szablonów do generowania kodu	205
51. Tworzenie makra	210
52. Generowanie kodu za pomocą makr	215
53. Szybkie tworzenie ciągów połączeń	217
54. Szybkie podpisywanie podzestawów	219
55. Szybkie uaktualnianie odwołań w projektach	221
56. Automatyczne dodawanie instrukcji using i Imports	224
57. Automatyczne wstawianie często używanego kodu	228
58. Szybkie przełączanie się między programami kontroli kodu źródłowego	231

Rozdział 7. System pomocy i prace badawcze	233
59. System pomocy	233
60. Przeszukiwanie zasobów internetowych z poziomu Visual Studio	240
61. Przekierowanie systemu pomocy do wyszukiwarki Google	243
62. Łatwe korzystanie z mechanizmu P/Invoke	245
63. Badanie kodu IL generowanego na podstawie kodu wyższego poziomu	249
64. Badanie wewnętrznej struktury podzespołów	254
65. Zapewnianie przestrzegania reguł za pomocą programu FxCop	259
66. Generowanie statystyk dla kodu zapisanego w języku C#	267
67. Profilowanie przydziałów na stosie	270
Rozdział 8. Komentarze i dokumentacja	277
68. Obsługa komentarzy XML używanych w kodzie C#	278
69. Sprawniejsze tworzenie komentarzy	287
70. Tworzenie komentarzy XML w kodzie VB.NET	291
71. Tworzenie dokumentacji na podstawie komentarzy XML	296
72. Integracja utworzonej dokumentacji z systemem pomocy Visual Studio	298
Rozdział 9. Obsługa komponentu Server Explorer	303
73. Uzyskiwanie dostępu do liczników wydajności	304
74. Tworzenie diagramów i modyfikowanie baz danych	307
75. Tworzenie skryptów baz danych	314
76. Wyliczanie procesów, napędów, udziałów i innych elementów	320
Rozdział 10. Obsługa narzędzi Visual Studio Tools	325
77. Obsługa wiersza poleceń systemu Visual Studio	325
78. Uruchamianie Visual Studio z poziomu wiersza poleceń	329
79. Testy obciążeniowe aplikacji WWW	331
80. Zaciemnianie kodu	350
81. Generowanie kodu na podstawie diagramów UML	356
82. Generowanie diagramów UML na podstawie kodu	363
Rozdział 11. Obsługa mechanizmów Visual Studio Tools for Office	367
83. Instalacja Visual Studio Tools for Office 2003	367
84. Tworzenia paska poleceń w programie Word 2003	369
85. Wyświetlanie okna formularza z poziomu programu Excel 2003	375
Rozdział 12. Rozszerzanie Visual Studio	383
86. Automatyzacja Visual Studio	383
87. Uzyskiwanie dostępu do Visual Studio z poziomu niezależnych aplikacji	386
88. Tworzenie szablonu pliku	390

89. Tworzenie dodatku systemu Visual Studio	398
90. Znajdowanie nazwy paska poleceń	407
91. Zmiana ikony dodatku	410
Rozdział 13. Usprawnianie Visual Studio	417
92. Zarządzanie dodatkami	417
93. Uruchamianie testów jednostek z poziomu systemu Visual Studio	420
94. Sprawdzanie pisowni kodu i komentarzy	425
95. Przeglądanie struktury formantów	430
96. Konwersja kodu tworzego w Visual Studio na postać nadającą się do wstawienia do blogu	431
97. Zwijanie i rozwijanie kodu	435
98. Połączenie wiersza poleceń Visual Studio z oknem poleceń	437
99. Generowanie kodu usług sieciowych	438
100. Testowanie wyrażeń regularnych w środowisku Visual Studio	447
Skorowidz	451

Obsługa narzędzi Visual Studio Tools

Sposoby 77. – 82.

System Visual Studio zawiera wiele różnych narzędzi, których można używać zarówno z poziomu, jak i spoza środowiska IDE. Sposoby prezentowane w niniejszym rozdziale opisują wiersz poleceń systemu Visual Studio oraz niektóre z dostępnych narzędzi. Zostanie tu również omówiona metoda zapewnienia większej dostępności wiersza poleceń Visual Studio poprzez dodanie do niego różnych skrótów.

W rozdziale zawarto także sposób przeprowadzania testów obciążeniowych aplikacji, utrudniania dekompilacji tworzonych kodu oraz generowania diagramów UML na podstawie kodu (i vice versa).



SPOSÓB

77.

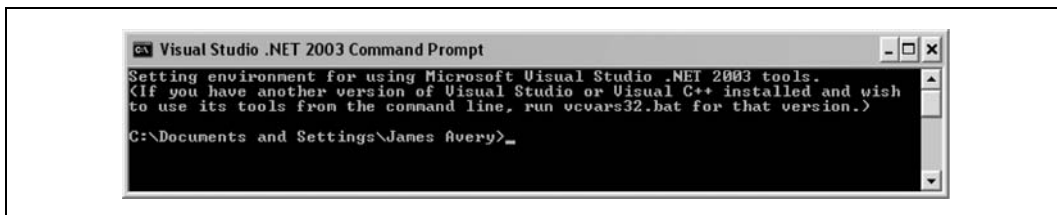
Obsługa wiersza poleceń systemu Visual Studio

Poniżej zostaną opisane metody wykonywania niemal wszystkich zadań z poziomu wiersza poleceń systemu Visual Studio.

Wiersz poleceń systemu Visual Studio (*Visual Studio command prompt*, VSCP) jest wierszem poleceń, w ramach którego są ładowane pewne ustawienia środowiskowe dla systemu Visual Studio oraz środowiska .NET Framework. Oznacza to, że użytkownik może uruchomić wiersz poleceń i wpisać polecenie dla Visual Studio lub pewnego narzędzia należącego do środowiska, co spowoduje jego uruchomienie bez konieczności podawania pełnej ścieżki dostępu lub przechodzenia do katalogu macierzystego narzędzia (VSCP definiuje także pewne zmienne środowiskowe, bez których wiele z tych narzędzi nie działałoby poprawnie).

W celu użycia VSCP należy uruchomić polecenie *Start/Wszystkie programy/Microsoft Visual Studio .NET 2003/Visual Studio .NET Tools/Visual Studio .NET 2003 Command Prompt* (w przypadku korzystania z innej wersji systemu Visual Studio ścieżka ta może mieć nieco inną postać).

Na rysunku 10.1 przedstawiono przykład uruchomienia wiersza poleceń Visual Studio.



Rysunek 10.1. Wiersz poleceń systemu Visual Studio .NET 2003

Z poziomu wiersza poleceń można uzyskiwać dostęp do wszystkich zwykle potrzebnych narzędzi. Poniżej wymieniono niektóre z najbardziej przydatnych dla programisty:

sn.exe

Narzędzie używane w celu silnego podpisywania podzespołów [**Sposób 54.**].

gacutil.exe

Narzędzie używane w celu dodawania podzespołów do globalnej pamięci podręcznej podzespołów.

xsd.exe

Narzędzie używane w celu tworzenia zbiorów danych (DataSets) o silnych typach na podstawie schematów XML [**Sposób 49.**].

ildasm.exe

Narzędzie używane w celu przeglądania kodu pośredniego generowanego na podstawie tworzonego kodu wysokopoziomowego [**Sposób 63.**].

wsdl.exe

Narzędzie używane w celu generowania kodu usług sieciowych w oparciu o pliki języka Web Service Description Language.

Jak widać, do wielu przydatnych narzędzi można szybko uzyskiwać dostęp z poziomu wiersza poleceń Visual Studio.

Dodanie wiersza poleceń Visual Studio jako narzędzia zewnętrznego

Ze względu na dostępność wszystkich wymienionych narzędzi z poziomu wiersza poleceń może okazać się, że użytkownik będzie często otwierał wiersz poleceń, a następnie przechodził do katalogu tworzonego rozwiązania lub projektu. Istnieje jednak lepszy sposób — można dodać VSCP do menu *Tools* w systemie Visual Studio i zapewnić, aby automatycznie był otwierany w katalogu odpowiedniego projektu lub rozwiązania:

1. Otwieramy okno konfiguracyjne *External Tools* poprzez wybranie polecenia *Tools/External Tools*.
2. Klikamy przycisk *Add*.

3. Wpisujemy nazwę (*Title*), taką jak:

```
Wiersz poleceń Visual Studio
```

4. Polecenie (*Command*) ustawiamy na wartość:

```
C:\WINDOWS\system32\cmd.exe
```

5. Argumenty (*Arguments*) ustawiamy na wartość:

```
/k ""C:\Program Files\Microsoft Visual Studio .NET 2003\Common7\Tools\  
vsvars32.bat""
```

6. Ustawiamy katalog roboczy (*Initial directory*) na jedną z dostępnych wartości (są to `$(SolutionDir)` oraz `$(ProjectDir)`), co zapewnia rozpoczęcie pracy w, odpowiednio, katalogu rozwiązania lub katalogu projektu.

7. Klikamy przycisk *OK*.

Na rysunku 10.2 przedstawiono przykład okna dialogowego *External Tools* [Sposób 33.] z wpisanymi podanymi powyżej wartościami.



Rysunek 10.2. Okno dialogowe *External Tools*

Od tego momentu VSCP znajduje się w menu *Tools*. Pozwala to na szybkie uruchamianie narzędzi przy jednoczesnym zapewnieniu ustawienia odpowiedniego katalogu roboczego.

Dodanie wiersza poleceń Visual Studio do Eksploratora Windows

Innym sposobem zapewnienia sobie łatwego dostępu do VSCP jest dodanie pewnego wpisu do rejestru, który doda w Eksploratorze Windows polecenie *Otwórz w tym miejscu*

wiersz poleceń VS. Najprostszym sposobem dodania odpowiednich wpisów w rejestrze jest utworzenie w dowolnym edytorze tekstu pliku *.reg* o następującej zawartości:

```
Windows Registry Editor Version 5.00

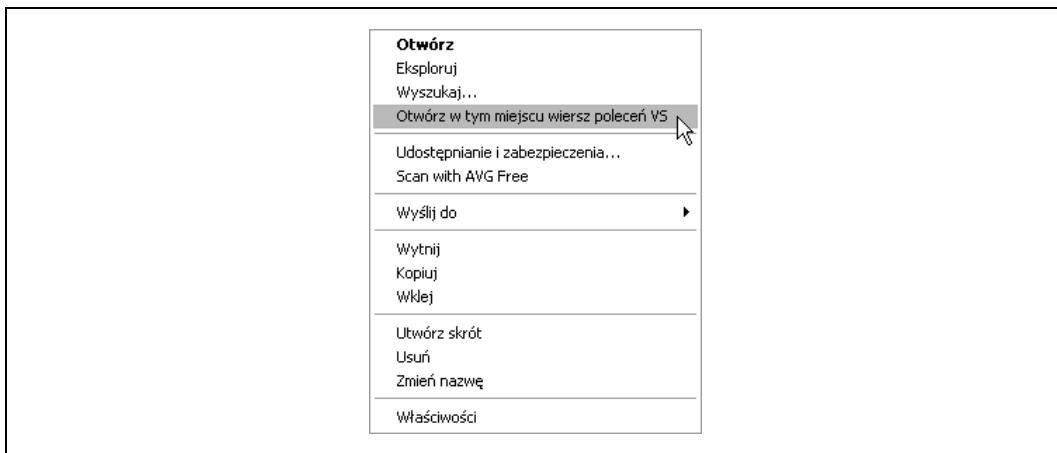
[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\Directory\shell\VSCP]
@="Otwórz w tym miejscu wiersz poleceń VS"

[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\Directory\shell\VSCP\command]
@="cmd.exe /k \"%C:\Program Files\Microsoft Visual Studio .NET 2003\Common7\Tools\vsvars32.bat\""
```

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\Drive\shell\VSCP]
@="Otwórz w tym miejscu wiersz poleceń VS"

[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\Drive\shell\VSCP\command]
@="cmd.exe /k \"%C:\Program Files\Microsoft Visual Studio .NET 2003\Common7\Tools\vsvars32.bat\""
```

Oczywiście użytkownik może być zmuszony zmienić podaną ścieżkę dostępu w celu dostosowania jej do zainstalowanej u siebie wersji oraz lokalizacji systemu Visual Studio. Następnie należy zapisać taki plik z rozszerzeniem *.reg*. Po dwukrotnym kliknięciu tego pliku pojawi się pytanie o dodanie zawartych w nim informacji do rejestru. Po ich dodaniu w Eksploratorze Windows stanie się dostępne nowe polecenie, co przedstawiono na rysunku 10.3.



Rysunek 10.3. Wybór polecenia *Otwórz wiersz poleceń VS w tym miejscu*

Od tego momentu użytkownik ma możliwość, po kliknięciu prawym przyciskiem myszy folderu i wybraniu nowo dodanego polecenia, otwierania VSCP w wybranym folderze.

Wiersz poleceń Visual Studio to wartościowe narzędzie i posiadanie łatwego dostępu do niego bardzo się przydaje.

Patrz również

- „Połączenie wiersza poleceń Visual Studio z oknem poleceń” [Sposób 98].

SPOSÓB
78.

Uruchamianie Visual Studio z poziomu wiersza poleceń

Jeżeli użytkownik chciałby mieć pełną kontrolę nad procesem uruchamiania systemu Visual Studio, z pewnością ucieszy się, że można to zapewnić z poziomu wiersza poleceń.

Plik wykonywalny środowiska IDE systemu Visual Studio nosi nazwę *devenv.exe* i obsługuje wiele opcji uruchomieniowych, które mogą okazać się bardzo przydatne. We wcześniejszym fragmencie książki omówiono już kilka z nich, jednak w ramach bieżącego sposobu Czytelnik pozna wszystkie te opcje oraz metody ich wykorzystywania.



Wpisywanie opcji uruchomieniowych przy każdorazowym wywołaniu aplikacji jest zadaniem czasochłonnym i po prostu niewydajnym. Należy pamiętać, że można tworzyć skróty wywołujące plik wykonywalny z wykorzystaniem opcji wiersza poleceń. Co więcej, można utworzyć kilka takich skrótów z różnymi zestawami opcji.

Ustawianie czcionek

Jedną z najprostszych, ale też najbardziej przydatnych rzeczy, jakie można osiągnąć, wykorzystując opcje wiersza poleceń, jest ustawienie czcionek i ich rozmiaru dla środowiska IDE. W celu określenia czcionki można skorzystać z opcji `/fn`, zaś w celu określenia jej rozmiaru — z opcji `/fs`. Istotną rzeczą jest pamiętać, że nie jest to rozmiar czcionki tekstu lub zawartości plików, lecz rozmiar tekstu środowiska IDE. Efekt wprowadzonych zmian nie będzie widoczny w przypadku normalnych menu, ale na przykład zakładek dokumentów, opcji wyboru itd.

Poniższe polecenie powoduje ustawienie czcionki środowiska Visual Studio na czcionkę Verdana o rozmiarze 14:

```
C:\> devenv /fn Verdana /fs 14
```

Nie trzeba go wywoływać za każdym razem — ustawienia te są zapamiętywane. Te same ustawienia można zmienić w oknie *Tools/Options/Fonts and Colors* po wybraniu z listy *Show Settings* opcji *Dialogs and Tool Windows*.

Wykonywanie polecenia

Używając opcji polecenia, można uruchamiać Visual Studio i automatycznie wywoływać wiersz poleceń środowiska. W tym celu wystarczy podać opcję `/command`, a po niej nazwę polecenia, które chce się wykonać. Są to te same polecenia, które omówiono w sposobie „Praca z oknem poleceń” [Sposób 46.]. W omawianym przykładzie wywołamy polecenie `File.OpenSolution` — niemal zawsze Visual Studio otwiera się w celu otwarcia rozwiązania, więc takie rozwiązanie pozwala zaoszczędzić nieco czasu:

```
C:\> devenv /command File.OpenSolution
```

Po uruchomieniu powyższego polecenia zostanie otworzone środowisko Visual Studio oraz okno dialogowe *New Solution*. Opcji `/command` można by także użyć w celu wykonania napisanego makro, które wykonuje pewne bardziej skomplikowane działania.

Uruchamianie rozwiązania

Istnieje możliwość automatycznego uruchamiania rozwiązań z poziomu wiersza poleceń przy użyciu opcji `/run`. Poniższe polecenie powoduje właśnie uruchomienie rozwiązania:

```
C:\> devenv /run HacksWinSample.sln
```

Po wykonaniu tego polecenia zostanie otwarte środowisko IDE i nastąpi automatyczne przejście do trybu uruchomieniowego z załadowaniem określonej aplikacji. Można także użyć opcji `/runexit`, która powoduje uruchomienie aplikacji i zminimalizowanie środowiska IDE. Po zamknięciu aplikacji zamykane jest również środowisko.

Konsolidacja projektów i rozwiązań

Projekty i rozwiązania można konsolidować przy użyciu opcji wiersza poleceń. Może stanowić to doskonałą alternatywę w sytuacji, gdy nie ma się czasu na skonfigurowanie narzędzia konsolidującego, takiego jak NAnt, a chce się obsługiwać ten proces poprzez plik wsadowy. W celu skonsolidowania rozwiązania używa się opcji `/build`, jak również `/project` oraz `/solution`. Poniżej podano adekwatny przykład:

```
C:> devenv HacksWinExample.sln /build release
```

Po opcji `/build` podaje się konfigurację rozwiązania, której chce się użyć podczas jego konsolidacji — w omawianym przykładzie została wykorzystana konfiguracja `release`. Uruchomienie powyższego polecenia powoduje skonsolidowanie rozwiązania bez uruchamiania środowiska IDE, zaś wyniki procesu konsolidacji zostaną zwrócone do okna wiersza poleceń. W tabeli 10.1 wymieniono inne opcje związane z procesem konsolidacji.

Tabela 10.1. Opcje konsolidacji

Opcja	Opis
<code>/clean</code>	Powoduje wyczyszczenie projektu lub rozwiązania, odpowiednio do jego konfiguracji.
<code>/rebuild</code>	Powoduje wyczyszczenie i skonsolidowanie projektu lub rozwiązania.
<code>/project</code>	Określa projekt poddawany procesowi konsolidacji.
<code>/projectconfig</code>	Określa konfigurację używaną w celu konsolidacji projektu.
<code>/deploy</code>	Określa, że Visual Studio ma wdrożyć rozwiązanie po jego skonsolidowaniu.
<code>/out</code>	Określa nazwę pliku, do którego będą przesyłane komunikaty o ewentualnych błędach.

Inne opcje

W celu wykonywania różnych działań w systemie Visual Studio można używać wielu innych opcji wiersza poleceń. Owe opcje wymieniono w tabeli 10.2.

Tabela 10.2. Opcje wiersza poleceń

Opcja	Opis
/lcid	Określa domyślny język używany w środowisku IDE, na przykład <code>devenv /lcid 1033</code> .
/mdi	Określa, że system Visual Studio powinien być uruchomiony w trybie MDI.
/mditabs	Określa, że system Visual Studio powinien być uruchomiony w trybie MDI z aktywowaną funkcją zakładek dokumentów.
/migratesettings	Określa, że system Visual Studio ma uruchomić proces migracji ustawień, z którego można korzystać w celu przenoszenia ustawień z jednej wersji Visual Studio do drugiej (ekran ten zwykle się pojawia, kiedy użytkownik pierwszy raz uruchomi nową instalację systemu Visual Studio).
/nologo	Powoduje uruchomienie Visual Studio bez wyświetlenia ekranu powitalnego.
/noVSIP	Dezaktywuje licencję VSIP na danej maszynie.
/safemode	Określa, że Visual Studio powinno być otworzone bez ładowania żadnych pakietów VSIP.
/setup	Powoduje przywrócenie wartości domyślnych niektórych elementów systemu Visual Studio („Zarządzanie dodatkami” [Sposób 92.]).
/resetskippgks	Aktywuje pakiety VSIP, czyszcząc zakładkę <i>SkipLoading</i> . Po uruchomieniu systemu w trybie bezpiecznym konieczne jest uruchomienie go z tą opcją w celu ponownego aktywowania pakietów, których chce się używać.
/rootsuffix	Pozwala na określenie przedrostka rejestru [Sposób 30.].
/?	Powoduje wyświetlenie opisu polecenia <code>devenv.exe</code> .

SPOSÓB

79.

Testy obciążeniowe aplikacji WWW

Użytkownik dysponujący tylko klawiaturą i myszą nigdy nie będzie w stanie wykonać należnych testów obciążeniowych aplikacji. Jednak narzędzie Application Center Test pozwala na symulowanie działania setek użytkowników.



W niniejszym sposobie skupimy się na użyciu narzędzia ACT z poziomu Visual Studio. W wielu przypadkach preferowanym rozwiązaniem jest jednak wykorzystanie ACT w wersji niezależnej, co zostanie omówione w dalszej części.

Jak każdy szanujący się programista rozwiązań sieciowych, chcemy się dowiedzieć, czy nasza aplikacja charakteryzuje się dobrą wydajnością — czy może sprostać spodziewanemu poziomowi obciążenia, czy osiągnięta przepustowość odpowiada potrzebom, ilu użytkowników jednocześnie możemy obsłużyć, ile żądań na sekundę jest w stanie obsłużyć, czy po upływie pewnego czasu nie występują wycieki pamięci?

Wszystko to są pytania istotne w kontekście testów wydajnościowych. Oczywiście można wymienić wiele innych pytań, jednak większość z nich pojawia się w momencie, gdy zaczynamy zbierać i analizować dane z pomiarów wydajności. W wielu przypadkach testy wydajnościowe są jednak wykonywane zbyt późno w ramach procesu opracowywania rozwiązania. W dużej mierze wynika to z ograniczeń czasowych — testy wydajnościowe odkłada się do schyłkowych etapów tworzenia aplikacji, kiedy jest już za późno na zaradzenie pojawiającym się problemom.

W celu przeprowadzenia pewnych testów wydajnościowych aplikacji WWW na wczesnym etapie jej tworzenia można wykorzystać narzędzie firmy Microsoft o nazwie Application Center Test (ACT). Stanowi ono element składowy systemu Visual Studio Enterprise Edition i jest wartościowym uzupełnieniem zestawu narzędzi programistycznych. Można go używać z poziomu Visual Studio lub jako niezależnego produktu. Choć nie jest ono w stanie zastąpić produktów w rodzaju LoadRunner firmy Mercury, to stanowi pożyteczne narzędzie, ponieważ może w wyraźny sposób wskazać na wczesnych etapach tworzenia aplikacji WWW występowanie ewentualnych problemów wydajnościowych, co pozwala zaradzić im przed wdrożeniem gotowego rozwiązania.



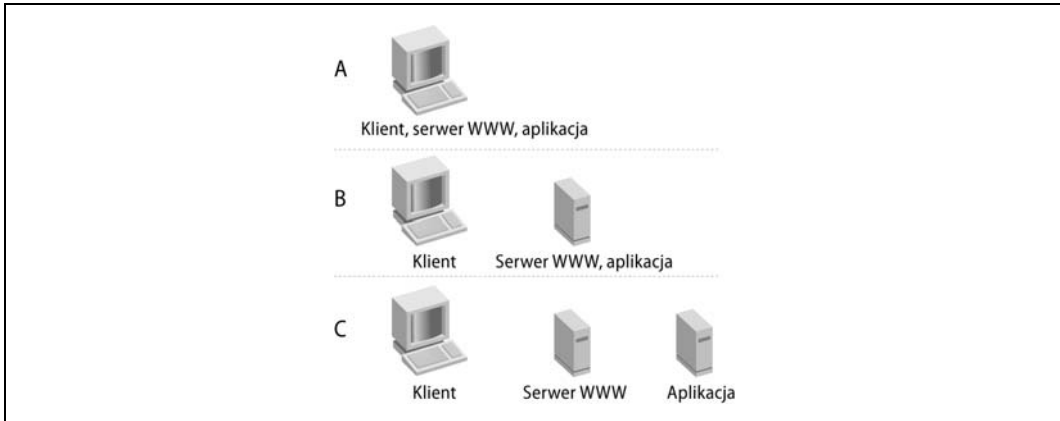
Jeżeli Czytelnik nie posiada wersji Visual Studio Enterprise Edition, wciąż może wykorzystać darmowe narzędzie Web Application Stress Tool. Chociaż nie oferuje ono takich samych możliwości jak ACT, pozwala wykonywać wiele podobnych zadań. ACT to następca Web Application Stress Tool, więc pod pewnymi względami są to narzędzia bardzo do siebie podobne. WAST można pobrać spod adresu <http://www.microsoft.com/downloads/details.aspx?FamilyID=e2c0585a-062a-439e-a67d-75a89aa36495&DisplayLang=en>.

Konfiguracje testów obciążeniowych

Narzędzie ACT generuje obciążenie, wykorzystując przeglądarkę internetową. W przypadku wszystkich konfiguracji klientem jest przeglądarka. Ze względu na ten fakt możliwe jest wykonywanie wywołań HTTP względem dowolnych serwerów WWW — lokalnych i zdalnych. Daje to możliwość tworzenia kilku różnych konfiguracji testów obciążeniowych. Typowa aplikacja WWW składa się z kilku warstw, które (zwykle) są wdrażane na wielu warstwach. Oznacza to, że konfiguracje testów obciążeniowych będą należeć do jednej z trzech kategorii konfiguracji, które przedstawiono na rysunku 10.4.

Poniżej przedstawiono opis tego rysunku:

- Konfiguracja A określa, że klient, serwer WWW oraz aplikacja działają na tej samej maszynie. Zwykle jest to stacja robocza programisty.
- Konfiguracja B pokazuje, że klient jest fizycznie oddzielony od serwera WWW i aplikacji, które wciąż są umieszczone w ramach tej samej maszyny. Sytuacja taka ma miejsce w przypadku, gdy aplikacja WWW działa na oddzielnym serwerze, zaś programista używa swojej maszyny jako klienta.



Rysunek 10.4. Konfiguracje testów obciążeniowych

- Konfiguracja C prezentuje, że klient, serwer WWW oraz logika aplikacji znajdują się na oddzielnych maszynach.

Należy również zauważyć, że produkty takie jak Virtual PC lub VMWare mogą być używane w celu emulowania dowolnej z opisanych konfiguracji na jeden maszynie. W rzeczywistości jest wskazane używanie takich produktów w celu tworzenia wirtualnych środowisk rozproszonych, tak by móc dopasować opracowywane rozwiązanie do środowiska, w którym będzie działała aplikacja WWW. Takie działania zwykle umożliwiają określanie bardziej realistycznych oczekiwań w toku procesu tworzenia aplikacji.

Identyfikacja przypadków testowych

Przed opracowaniem testów należy zidentyfikować przypadki testowe. Mogą to być zarówno proste żądania o strony WWW, jak i bardziej skomplikowane testy charakteryzujące się wieloetapowym przepływem sterowania, wieloma zadaniami i wieloma stronami WWW. Kluczowe znaczenie ma zidentyfikowanie punktów początkowego i końcowego każdego takiego testu. Pozwala to na określenie zakresu testu oraz momentu zaprzestania rejestracji jego wyników, co zostanie opisane poniżej.

Tworzenie projektu ACT

W celu wykorzystania narzędzia ACT w ramach testowania obciążeniowego aplikacji WWW w pierwszej kolejności należy utworzyć aplikację ASP.NET w systemie Visual Studio, a następnie uruchomić ją w celu upewnienia się, że działa zgodnie z oczekiwaniami. Następnie można utworzyć projekt ACT, postępując zgodnie z opisaną poniżej procedurą (zakładając, że użytkownik zaznaczył opcję instalacji narzędzia ACT w czasie procesu instalowania systemu Visual Studio .NET Enterprise Edition):

1. W oknie *Solution Explorer* klikamy prawym przyciskiem myszy nazwę rozwiązania i wybieramy polecenie *Add*, a następnie *New Project*.

2. W oknie dialogowym *Add New Project* na liście *Project Types* wybieramy opcję *Other Projects*, a następnie zaznaczamy *Application Center Test Projects*.
3. W polu *Templates* wybieramy jedyną dostępną możliwość — *ACT Project*.
4. Wpisujemy nazwę projektu, określamy jego lokalizację i klikamy przycisk *OK*.
5. W tym momencie do rozwiązania zostaje dodany projekt ACT.

Tworzenie testu

Po zidentyfikowaniu przypadków testowych i utworzeniu projektu ACT możemy tworzyć i dodawać kolejne testy do projektu, wykonując następujące działania:

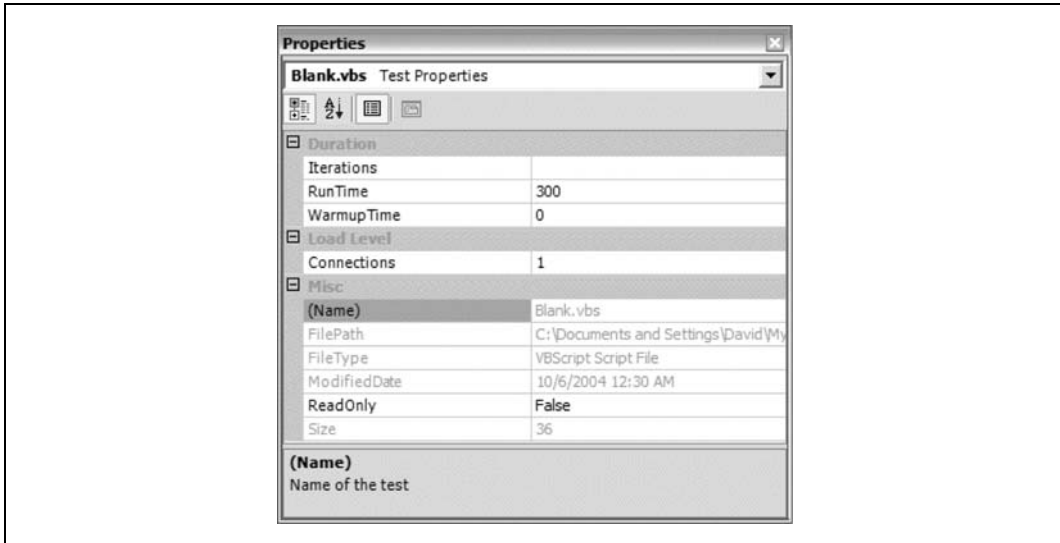
1. W oknie *Solution Explorer* klikamy prawym przyciskiem myszy nazwę projektu ACT, wybieramy polecenia *Add* oraz *Add New Item*.
2. Z listy *Templates* wybieramy opcję *Browser Recorded Test*.
3. Podajemy nazwę testu i klikamy przycisk *OK*.
4. Pojawia się okno dialogowe *Browser Record*. Klikamy przycisk *Start*.
5. Zostaje otwarte nowe okno przeglądarki Internet Explorer. Należy zauważyć, że ACT używa IE, nawet jeśli nie jest domyślną przeglądarką w systemie.
6. W przeglądarce wpisujemy adres URL naszej aplikacji. Na koniec „wykonujemy” zdefiniowane przypadki testowe i wreszcie zamykamy przeglądarkę.
7. W oknie dialogowym *Browser Record* możemy zauważyć, że w polu tekstowe *Request Details* zawiera listę zgłoszonych żądań. Klikamy przycisk *Stop*, a następnie *OK*.
8. Wygenerowany skrypt testowy (np. *NazwaTestu.vbs*) zostaje otworzony w oknie edytora i dodany do projektu ACT w oknie *Solution Explorer*.



Należy wspomnieć o jednej rzeczy: wszystkie testy narzędzia ACT generują skrypty VBScript lub JScript i użytkownik ma możliwość pozostawienia zadania generowania skryptów narzędziu ACT (rozwiązanie zalecane) lub samodzielnie napisać takie skrypty. W przypadku skryptów generowanych automatycznie są one tworzone w języku VBScript.

Właściwości testów

Po utworzeniu projektu ACT i dodaniu do niego pewnych testów należy ustawić ich właściwości. Okno *Properties* pozwala na określenie poziomów obciążenia, czasu trwania testu oraz liczby jego powtórzeń. W celu zapoznania się z tymi ustawieniami należy kliknąć prawym przyciskiem myszy test i wybrać polecenie *Properties*. Zostaje wówczas wyświetlone okno przedstawione na rysunku 10.5.



Rysunek 10.5. Okno właściwości testu

Domyślnie nowy test jest konfigurowany dla uruchomienia jednego połączenia poprzez przeglądarkę internetową przez okres pięciu minut (Connections = 1, RunTime = 300; wartość RunTime jest podawana w sekundach). Taka konfiguracja domyślnie zwykle stanowi odpowiedni punkt wyjścia do dalszych działań.

W celu zebrania większej liczby danych pomiarowych najlepiej jest wykonywać testy przy różnych obciążeniach co do czasu trwania i liczby powtórzeń. Test bazujący na czasie działania pokaże, ile żądań zostało obsłużonych w danym okresie. Z kolei test bazujący na powtórzeniach zaprezentuje, ile czasu zajmuje wykonanie testu określoną liczbę razy.

Przykład dla czasu działania

W celu skonfigurowania testu, który otworzy 10 połączeń i będzie trwał godzinę, należy podać wartość 10 dla parametru Connections oraz 3600 dla RunTime, zaś pole Iterations pozostawić puste.

Przykład dla powtórzeń

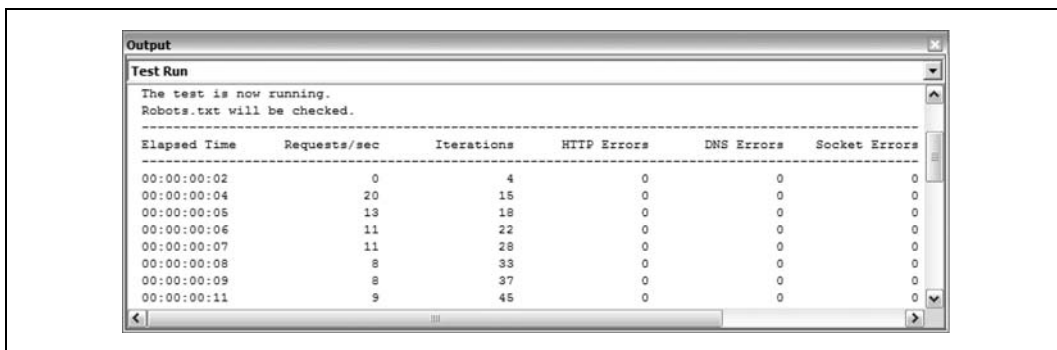
W celu skonfigurowania testu, który zostanie powtórzony 1000 razy z wykorzystaniem 5 połączeń, należy podać wartość 1000 dla parametru Iterations, wartość 5 dla parametru Connections, zaś pole RunTime pozostawić puste.

Uruchamianie testu

Ostatnią czynnością na tym etapie działań jest uruchomienie testu. W tym celu należy:

1. Kliknąć prawym przyciskiem myszy nazwę testu w oknie *Solution Explorer* i wybrać polecenie *Start Test*.

2. Zostanie wyświetlone okno *Output* ze statystykami dotyczącymi wykonywanego testu. Przykład takiego okna przedstawiono na rysunku 10.6.

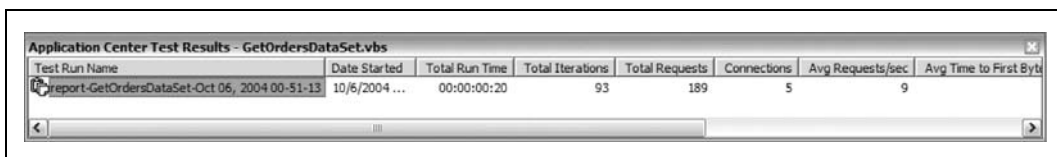


Rysunek 10.6. Okno *Output* w czasie wykonywania testu

3. Użytkownik ma w tym momencie dwie możliwości: pozwolić na wykonywanie testu lub zatrzymać go, co można osiągnąć, klikając prawym przyciskiem myszy nazwę testu i wybierając polecenie *Stop Test*.

Przeglądanie wyników testu

Po zakończeniu działania testu użytkownik może przejrzeć jego wyniki, klikając prawym przyciskiem jego nazwę i wybierając polecenie *View Results*. Zostaje wówczas wyświetlone okno podobne do przedstawionego na rysunku 10.7.



Rysunek 10.7. Okno wyników testu

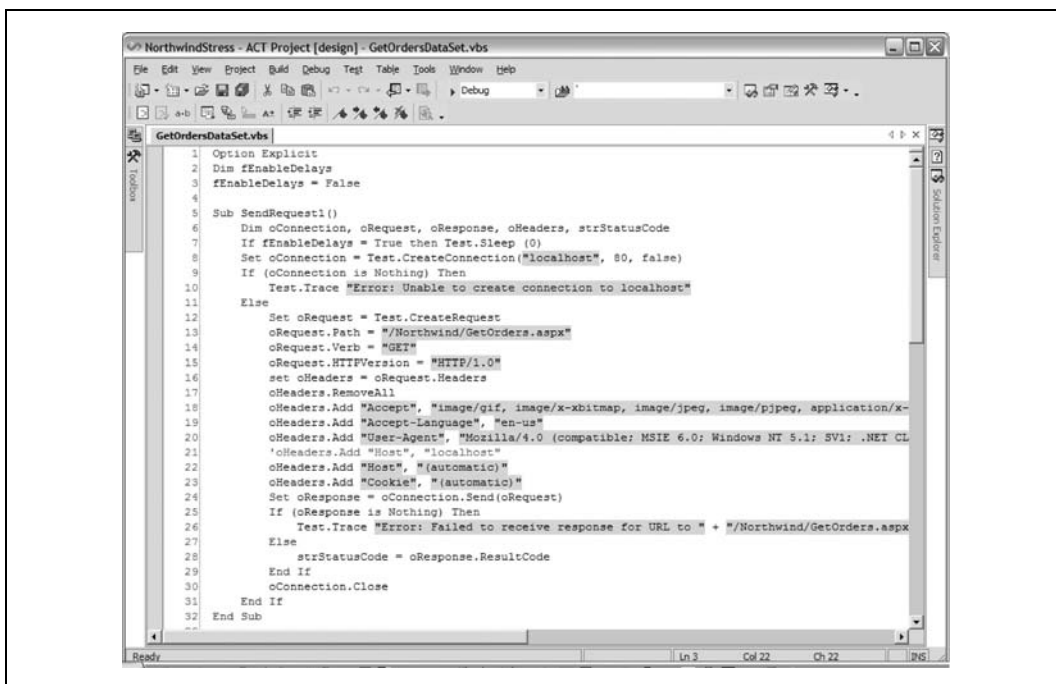
Okno wyników testu zapewnia szybki sposób przejrzania statystyk wykonanego testu. Są tu prezentowane takie elementy jak całkowity czas działania, całkowita liczba powtórzeń oraz średnia liczba obsłużonych żądań na sekundę. ACT automatycznie generuje nazwę raportu, wykorzystując nazwę testu oraz znacznik czasu. Za każdym razem, gdy test zostaje wykonany, ACT generuje raport, nawet jeśli test zostanie zatrzymany przed swoim normalnym zakończeniem. Jeżeli użytkownik uruchomi ten sam test dla tej samej konfiguracji wielokrotnie, ACT i tak wygeneruje oddzielne raporty dla każdego z takich uruchomień.

W celu usunięcia raportu testu należy kliknąć prawym przyciskiem myszy nazwę raportu w oknie wyników testu i wybrać polecenie *Delete*.

Jeżeli użytkownikowi nie podobają się konwencje nazewnictwa raportów stosowane przez ACT, może zmienić nazwę dowolnego raportu, klikając go prawym przyciskiem myszy, wybierając polecenie *Rename* i wpisując nową nazwę. Przykładowo, użytkownik może zechcieć nazywać raporty, opierając się na nazwie testu i jego konfiguracji, na przykład *Raport-ZamowieniaDataSet-1000Powtorzen* lub *Raport-ZamowieniaDataSet-10Polaczen-1Godzina*.

Dostosowywanie skryptów testowych do własnych potrzeb

Jak wspomniano wcześniej, skrypty testowe rejestrowane na podstawie użycia przeglądarki internetowej są automatycznie generowane w języku VBScript. Ma to swoje zalety, ponieważ pozwala dostosowywać i (lub) modyfikować wygenerowany skrypt w celu jego lepszego dopasowania do określonych wymagań. W większości przypadków generowany automatycznie skrypt to wszystko, czego potrzeba, ale warto wiedzieć, że jeżeli trzeba go zmienić, można to osiągnąć w bardzo prosty sposób. Wygenerowany skrypt można podejrzeć, klikając dwukrotnie lewym przyciskiem myszy dany test w projekcie ACT. Przykład takiego skryptu przedstawiono na rysunku 10.8.



```
1 Option Explicit
2 Dim fEnableDelays
3 fEnableDelays = False
4
5 Sub SendRequest()
6 Dim oConnection, oResponse, oHeaders, strStatusCode
7 If fEnableDelays = True then Test.Sleep (0)
8 Set oConnection = Test.CreateConnection("localhost", 80, false)
9 If (oConnection is Nothing) Then
10 Test.Trace "Error! Unable to create connection to localhost"
11 Else
12 Set oRequest = Test.CreateRequest
13 oRequest.Path = "/Northwind/GetOrders.aspx"
14 oRequest.Verb = "GET"
15 oRequest.HTTPVersion = "HTTP/1.0"
16 set oHeaders = oRequest.Headers
17 oHeaders.RemoveAll
18 oHeaders.Add "Accept", "image/gif, image/x-bitmap, image/jpeg, image/png, application/x-
19 oHeaders.Add "Accept-Language", "en-us"
20 oHeaders.Add "User-Agent", "Mozilla/4.0 (compatible: MSIE 6.0; Windows NT 5.1; SV1: .NET CL
21 'oHeaders.Add "Host", "localhost"
22 oHeaders.Add "Host", "(automatic)"
23 oHeaders.Add "Cookie", "(automatic)"
24 Set oResponse = oConnection.Send(oRequest)
25 If (oResponse is Nothing) Then
26 Test.Trace "Error! Failed to receive response for URL to " + P"/Northwind/GetOrders.aspx
27 Else
28 strStatusCode = oResponse.ResultCode
29 End If
30 oConnection.Close
31 End If
32 End Sub
```

Rysunek 10.8. Wygenerowany automatycznie skrypt w języku VBScript

Format skryptu jest prosty i ma charakter modułarny. Dla każdego zarejestrowanego żądania jest tworzona metoda `SendRequest()` — są one numerowane kolejnymi liczbami, rozpoczynając od 1. Istotną rzeczą jest zrozumienie, że metoda `SendRequest()` nie musi odpowiadać jednemu żądaniu dotyczącemu strony WWW. Pojedyncza strona może zawierać kilka obrazków lub odwołań do plików JavaScript i arkuszy stylów CSS. Każdy z tych

elementów wymaga odrębnego żądania, a stąd ACT generuje metodę `SendRequest()` dla każdego z nich. W zależności od sposobu zaprojektowania strony wygenerowany skrypt może osiągnąć spory rozmiar.

Na szczęście wszystkie informacje można znaleźć w danej metodzie `SendRequest()`. Każda z nich jest prostym wywołaniem podprogramu z poziomu programu głównego. Aby się o tym przekonać, wystarczy zajrzeć na koniec wygenerowanego skryptu. Jego postać będzie podobna do przedstawionej poniżej:

```
Sub Main()  
    call SendRequest1()  
    call SendRequest2()  
End Sub
```

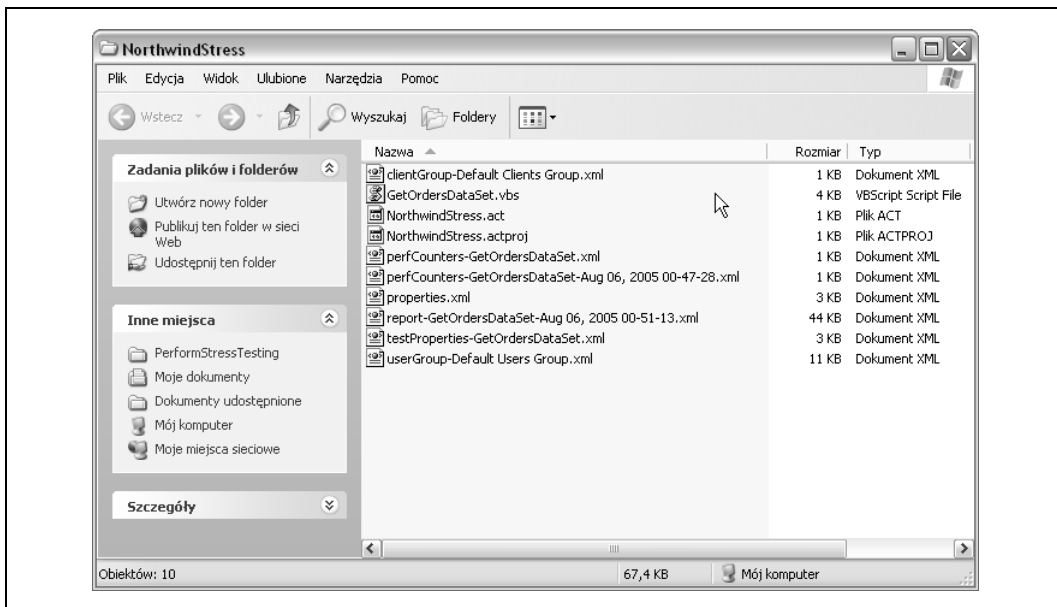
Podprogram `Main()` to nic innego jak szereg wywołań metod `SendRequest()`. W celu zademonstrowania sposobu dostosowywania skryptu do własnych potrzeb założymy, że przypadek testowy dotyczy pojedynczej strony WWW, która zawiera 10 obrazków i wszystko, czego chcemy, to przetestować czas jej ładowania. Narzędzie ACT wygeneruje 11 metod `SendRequest(): SendRequest1()` dla żądania dotyczącego strony oraz od `SendRequest2()` do `SendRequest11()` dla żądań dotyczących obrazków. Podprogram `Main()` będzie miał następującą postać:

```
Sub Main()  
    call SendRequest1()  
    call SendRequest2()  
    call SendRequest3()  
    call SendRequest4()  
    call SendRequest5()  
    call SendRequest6()  
    call SendRequest7()  
    call SendRequest8()  
    call SendRequest9()  
    call SendRequest10()  
    call SendRequest11()  
End Sub
```

Przeprowadzamy więc testy wydajnościowe, analizujemy dane i uświadamiamy sobie, że strona nie jest ładowana w akceptowalnym czasie. Na początek możemy zwiększyć wydajność poprzez oznaczenie jako komentarz niektórych wywołań metod `SendRequest()`. Następnie ponownie przeprowadzamy testy i analizujemy otrzymane wyniki. Tego rodzaju proste modyfikacje skryptu pozwalają na szybkie wprowadzanie zmian w skryptach testowych bez konieczności przeprojektowywania strony WWW i przygotowywania nowego testu.

Pliki projektu ACT

Omówiliśmy już sposób tworzenia projektów typu ACT Project w Visual Studio, dodawanie i konfigurowanie skryptów testowych, przeprowadzanie testów oraz przeglądanie otrzymywanych wyników. Kolejną wartościową cechą jest fakt, że ACT tworzy i zarządza wszystkimi plikami swojego projektu w formie kodu XML. Przykład takich plików przedstawiono na rysunku 10.9.



Rysunek 10.9. Część plików wygenerowanych przez narzędzie ACT

Oprócz pliku *.vbs* wszystkie inne widoczne pliki są w formacie XML — nawet pliki *.act* oraz *.actproj*. W czasie konfigurowania i wprowadzania zmian w projekcie ACT oraz testach narzędzie ACT w razie potrzeby modyfikuje te pliki w tle. Poniżej przedstawiono zawartość pliku *testProperties-GetOrdersDataSet.xml*:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE Settings>
<DefaultValues >
  <TestProperties >
    <ControllerReportTimeout type="long" value="5"/>
    <Timeout type="long" value="120"/>
    <Duration type="long" value="300"/>
    <Warmup type="long" value="0"/>
    <UseIterations type="bool" value="False"/>
    <Iterations type="long" value="200"/>
    <UseRateControl type="bool" value="False"/>
    <TargetRPS type="long" value="40"/>
    <PerfCollectionInterval type="long" value="10"/>
    <TestType type="long" value="1"/>
    <GenerateUsers type="bool" value="True"/>
    <UsersToGenerate type="long" value="5000"/>
    <CollectPerfCounters type="bool" value="False"/>
    <CollectRuntimeHistory type="bool" value="true"/>
    <Locale type="long" value="1033"/>
    <CollectPerPageData type="bool" value="true"/>
  </TestProperties>
  <DynamicTest >
    <Language type="string" value="VBScript"/>
    <NumberOfThreads type="long" value="5"/>
    <FollowRedirects type="bool" value="False"/>
    <RedirectDepth type="long" value="15"/>
  </DynamicTest>
</DefaultValues >
```

```
<Network >
  <Enable type="bool" value="False"/>
  <NATRange type="long" value="0"/>
  <Distribution.9.6K type="long" value="1"/>
  <Distribution.14.4K type="long" value="4"/>
  <Distribution.28.8K type="long" value="5"/>
  <Distribution.56K type="long" value="20"/>
  <Distribution.128K type="long" value="15"/>
  <Distribution.512K type="long" value="15"/>
  <Distribution.T1 type="long" value="15"/>
  <Distribution.Unlimited type="long" value="15"/>
  <PropagationDelay type="long" value="200"/>
  <NATStartAddr type="string" value="10.10.1.1"/>
  <NATStopAddr type="string" value="10.10.1.255"/>
  <ClientPortStart type="long" value="1000"/>
  <ClientPortEnd type="long" value="5000"/>
</Network>
<StaticTest >
  <DefaultServer type="string" value="localhost"/>
  <DefaultMethod type="string" value="GET"/>
  <DefaultHTTPVer type="string" value="HTTP/1.1"/>
  <DefaultPort type="long" value="80"/>
  <SessionCount type="long" value="5"/>
  <UseRandomDelay type="bool" value="True"/>
  <MinDelay type="long" value="10"/>
  <MaxDelay type="long" value="1500"/>
  <FollowRedirects type="bool" value="True"/>
  <RedirectDepth type="long" value="15"/>
</StaticTest>
</DefaultValues>
```

ACT jako narzędzie niezależne

Jak dotąd opisaliśmy sposób korzystania z narzędzia ACT z poziomu Visual Studio, jednak ta metoda nie zapewnia dostępu do wszystkich oferowanych przez nie funkcji. Dodatkowe funkcje, dostępne w wersji niezależnej, to:

- raportowanie w postaci graficznej,
- dzielenie raportów,
- dodatkowe statystyki,
- definiowanie użytkowników,
- monitorowanie licznika wydajności.

W celu zapoznania się ze wszystkimi możliwościami narzędzia ACT należy użyć go jako samodzielnej aplikacji. Zagadnieniu temu poświęcono niniejszy punkt.

Tworzenie projektu ACT

W celu użycia narzędzia ACT w zakresie testowania wydajności najpierw należy utworzyć i wdrożyć aplikację WWW w ramach jednej z omówionych wcześniej konfiguracji. Następnie można utworzyć projekt ACT, postępując według poniższej procedury:

1. Uruchamiamy polecenie *Start/Wszystkie programy/Visual Studio .NET 2003/Visual Studio .NET Enterprise Features* i wybieramy *Microsoft Application Center*.
2. Wybieramy polecenie *New/New Project (Ctrl+N)*.
3. W wyświetlonym oknie dialogowym wpisujemy nazwę projektu i określamy jego lokalizację (domyślnie jest to katalog *C:\Documents and Settings\\Moje dokumenty\ACT Projects*). Następnie klikamy przycisk *OK*.
4. Projekt można zapisać w dowolnym momencie, wybierając polecenie *File/Save Project (Ctrl+S)*.

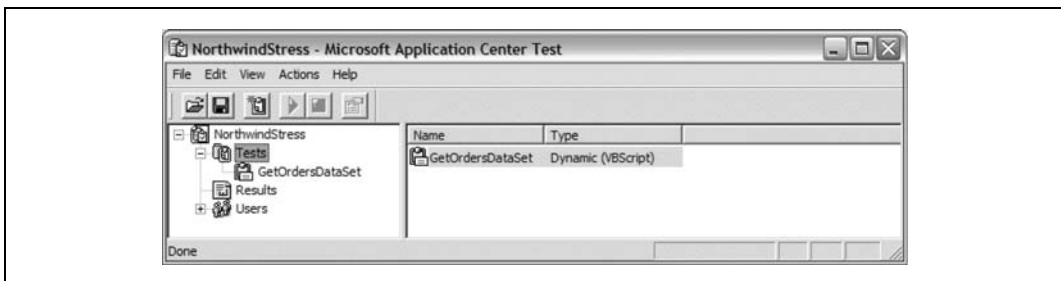
Tworzenie nowych testów

Do projektu ACT można dodawać nowe testy, wykonując opisane poniżej działania:

1. W lewej części okna klikamy prawym przyciskiem myszy wierzchołek *Tests* i wybieramy polecenie *New Test*. Zostaje wyświetlony kreator tworzenia testu.
2. Jako źródło testu określamy opcję *Record New Test*.
3. W oknie definiowania działań rejestrowanych w przeglądarce klikamy przycisk *Start Recording*. Zostaje otwarte nowe okno przeglądarki Internet Explorer (ACT używa IE, nawet jeśli nie jest domyślną przeglądarką w systemie).
4. W przeglądarce wpisujemy adres URL naszej aplikacji. Na koniec „wykonujemy” zdefiniowane przypadki testowe i zamykamy przeglądarkę.
5. W kreatorze klikamy przycisk *Stop Recording*, a następnie *Next*.
6. Podajemy nazwę testu i klikamy przycisk *Next*, a następnie *Finish*.
7. Wygenerowany test zostaje wyświetlony w prawym oknie.

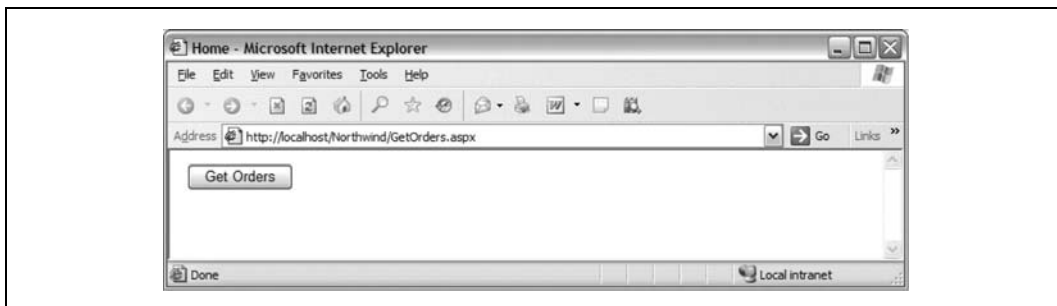
Przykładowy projekt ACT

W celu zademonstrowania użycia narzędzia ACT tworzymy prostą aplikację WWW o nazwie *Northwind*, która zawiera pojedynczą stronę WWW o nazwie *GetOrders.aspx*. Jest ona wykorzystywana w celu pobierania wszystkich zleceń z bazy danych *Northwind* na lokalnej bazie *SQL Server*. W ramach narzędzia ACT tworzymy projekt o nazwie *NorthwindStress* i rejestrujemy test *GetOrdersDataSet*, co przedstawiono na rysunku 10.10.

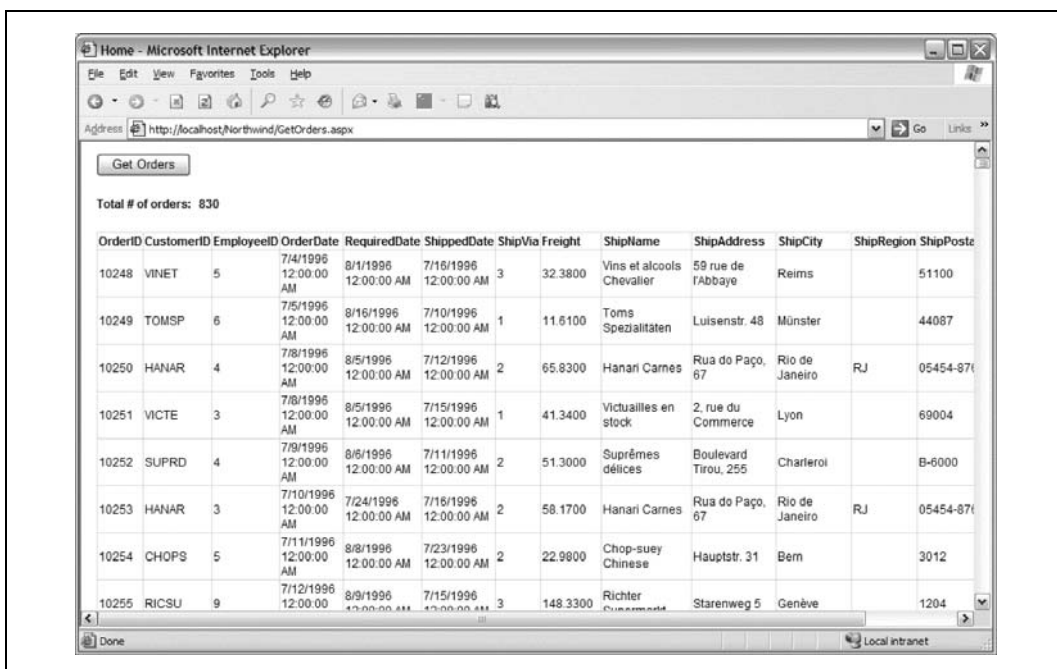


Rysunek 10.10. Project ACT o nazwie *NorthwindStress*

Test *GetOrdersDataSet* jest bardzo prosty: otwieramy stronę WWW, klikamy przycisk *Get Orders* i wyświetlamy wyniki. Na rysunku 10.11 przedstawiono stronę tuż po jej otwarciu, zaś na rysunku 10.12 — z uwzględnieniem wyników.



Rysunek 10.11. Początkowy wygląd strony *Get Orders*



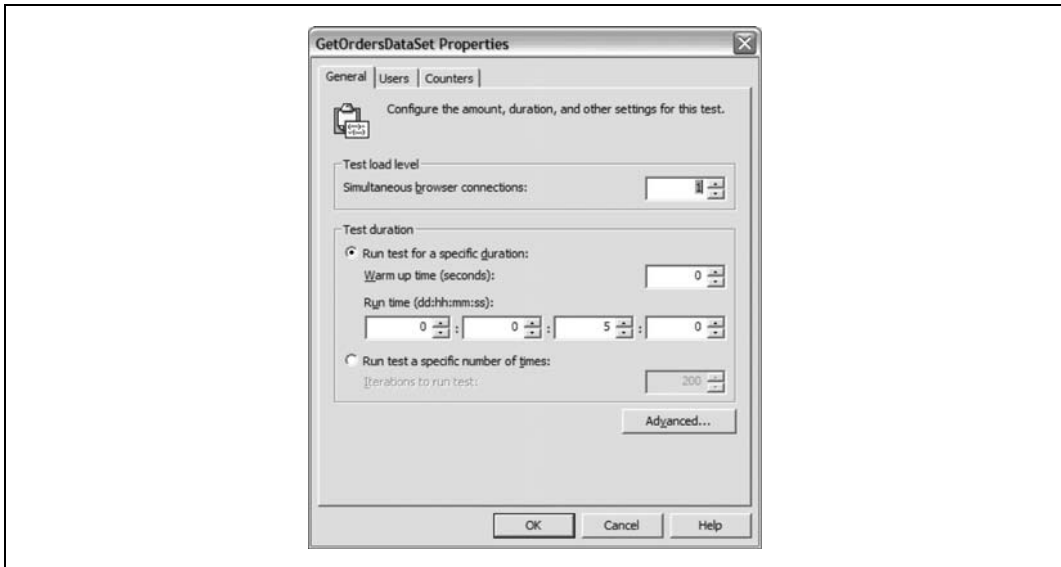
Rysunek 10.12. Strona *Get Orders* z wyświetlonymi wynikami

W rzeczywistości właśnie takie dwa widoki zarejestrowano w projekcie ACT *GetOrders* ➔ *DataSet*. Końcem przypadku testowego jest strona z wynikami.

Właściwości testu

Każdy test ACT posiada zestaw właściwości, które pozwalają określić poziomy obciążenia, czas trwania testu, liczbę jego powtórzeń, użytkowników oraz przechwytywane liczniki wydajności. W celu przejrzania tych ustawień należy kliknąć prawym przyciskiem myszy

nazwę testu i wybrać opcję *Properties*. Zostaje wówczas wyświetlone okno *Properties* z zakładką *General*, co przedstawiono na rysunku 10.13.



Rysunek 10.13. Zakładka *General* w oknie właściwości

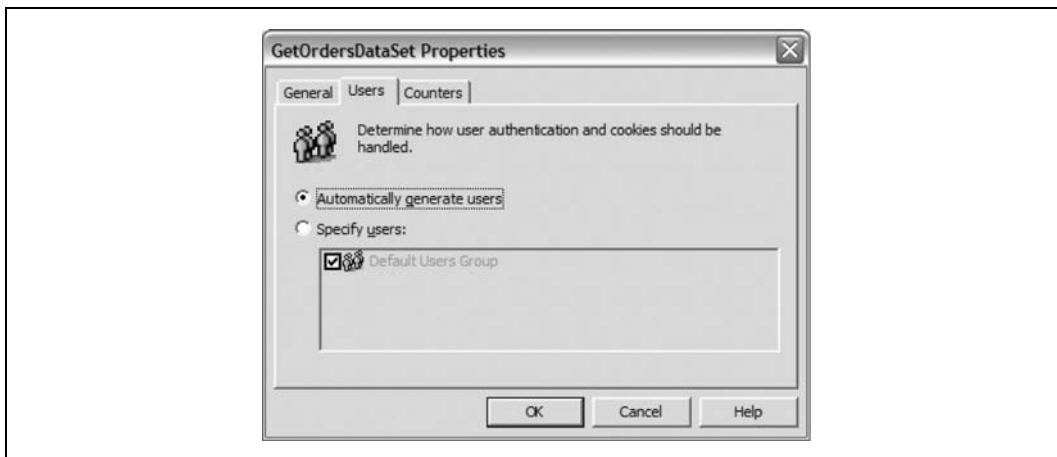
Na zakładce tej określa się poziom obciążenia i czas trwania testu. Domyślnie test otwiera jedno połączenie przeglądarki internetowej na okres pięciu minut. Taka konfiguracja domyślna jest odpowiednia dla przeprowadzenia testu wstępnego. Na zakładce tej spędza się najwięcej czasu, konfigurując ustawienia testu.

Kliknięcie przycisku *Advanced* powoduje wyświetlenie pola wyboru, które określa, czy mają zostać wygenerowane szczegółowe wyniki testu, co jest ustawieniem domyślnym. Najlepszym rozwiązaniem jest nie zmienianie tego ustawienia.

Kolejną zakładką jest zakładka *Users*, którą przedstawiono na rysunku 10.14.

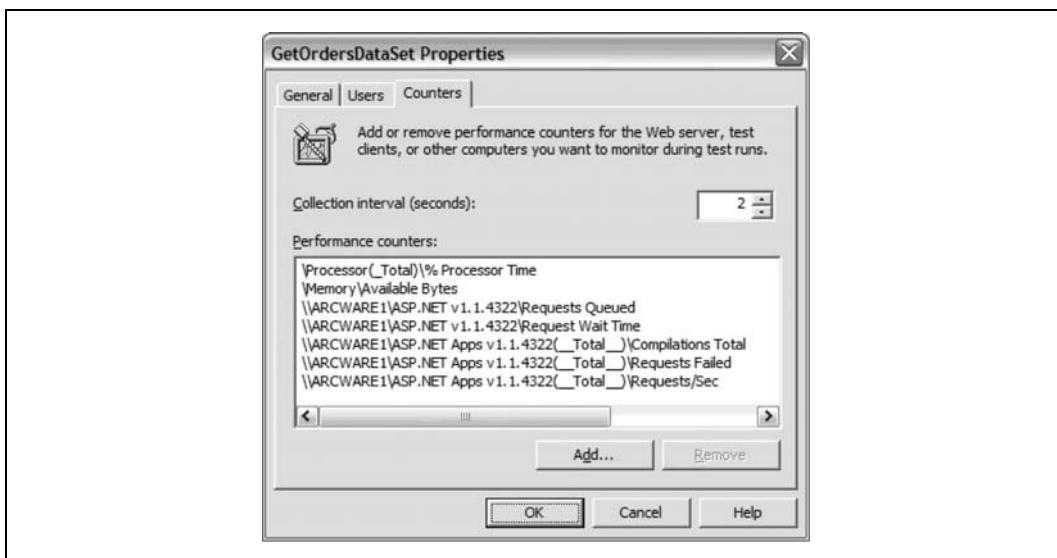
W przypadku testów, które nie muszą obsługiwać cookies tworzonych i używanych w czasie ich działania, użytkownicy mogą być automatycznie generowani przez narzędzie ACT w czasie testu zgodnie z potrzebami. Automatyczne generowanie użytkowników pomaga uniknąć potencjalnych problemów występujących wówczas, gdy liczba użytkowników symulowana w toku przeprowadzania testu jest zbyt mała w stosunku do wymaganego poziomu obciążenia.

W przypadku testów wymagających uwierzytelniania lub kiedy użytkownik chce przejrzeć albo ponownie wykorzystać tworzone cookies, należy utworzyć użytkowników i grupy użytkowników, a następnie przejść na zakładkę *Users*. Można tu określić nazwę, domenę i hasło każdego z tych użytkowników.



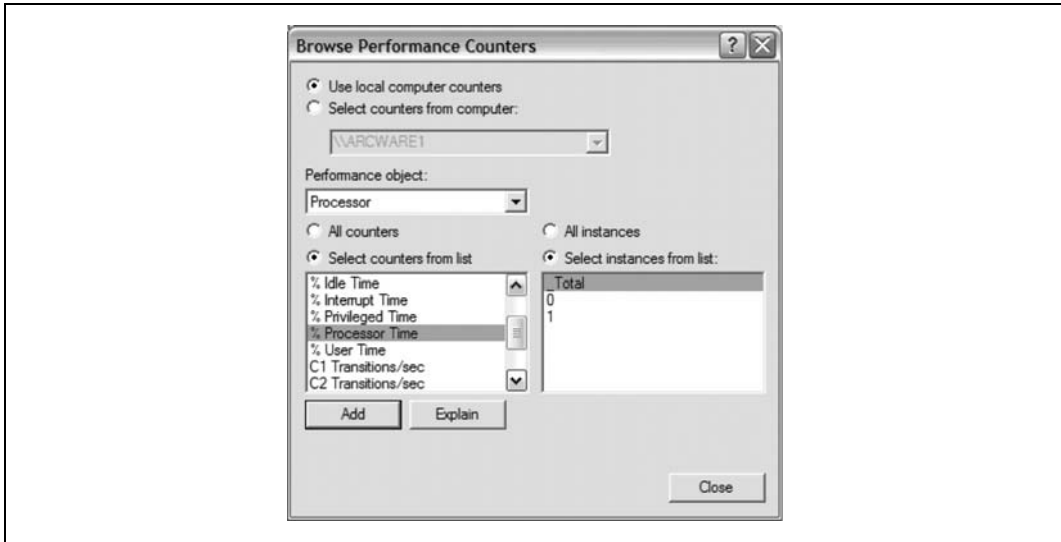
Rysunek 10.14. Zakładka General w oknie właściwości

Ostatnią zakładką jest zakładka *Counters*. Służy ona do dodawania wszelkich liczników wydajności, które chcemy przechwytywać — związanych z klientem i (lub) serwerem WWW. W prezentowanym przykładzie dodano kilka takich liczników i zmieniono przedział czasu między kolejnymi przechwytywanymi danymi z liczników na dwie sekundy, co przedstawiono na rysunku 10.15.



Rysunek 10.15. Zakładka Counters w oknie właściwości

Na zakładce tej można dodawać liczniki, klikając przycisk *Add*. Zostaje wówczas wyświetlone okno dialogowe, na którym można wybrać odpowiedni licznik. Przedstawiono je na rysunku 10.16. Jeżeli Czytelnik korzystał kiedyś z narzędzia *Performance Monitor*, okno to będzie mu znajome.

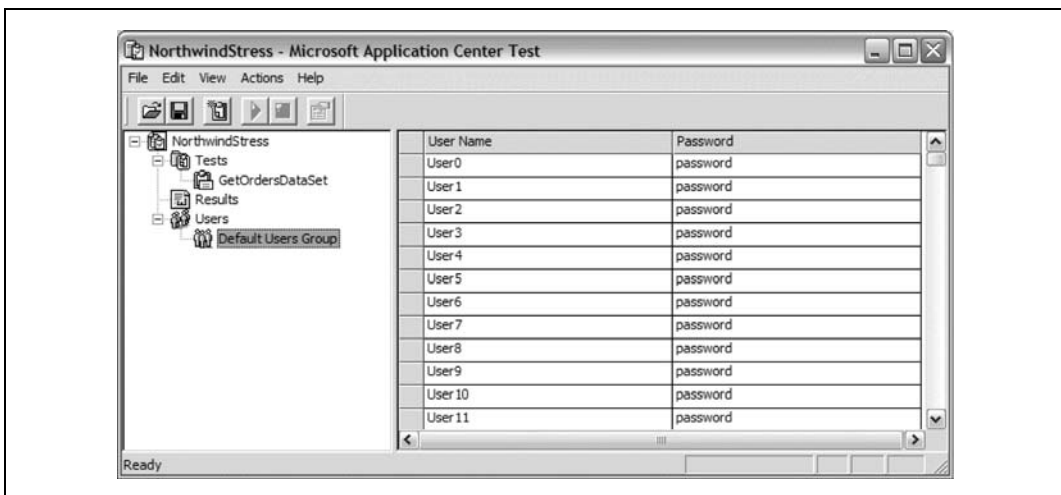


Rysunek 10.16. Okno dialogowe obsługi liczników wydajności

Jak widać, użycie narzędzia ACT w trybie niezależnym oferuje o wiele więcej opcji konfiguracyjnych.

Definiowanie użytkowników

Kolejną korzyścią wynikającą z użycia narzędzia ACT poza środowiskiem Visual Studio jest możliwość definiowania użytkowników i grup użytkowników dla prowadzonych testów. Istnieje możliwość wykorzystania przygotowanej grupy użytkowników *Default Users Group*, która zawiera 200 użytkowników (od User0 do User199) z hasłami postaci *password*, co przedstawiono na rysunku 10.17.

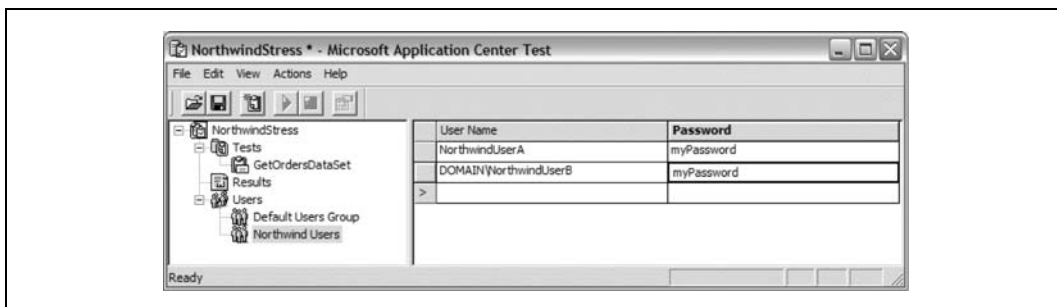


Rysunek 10.17. Domyślna grupa użytkowników

W celu utworzenia własnej grupy użytkowników można wykorzystać jedną z trzech metod: wpisać ich ręcznie, pozwolić narzędziu ACT na ich wygenerowanie lub zaimportowanie ich z listy zawierającej wpisy rozdzielone przecinkami. Należy zauważyć, że wszyscy użytkownicy muszą należeć do pewnej grupy.

W celu utworzenia własnej grupy użytkowników najpierw zaznaczamy wierzchołek *Users* w lewym oknie, klikamy prawym przyciskiem myszy i wybieramy polecenie *Add*. Powoduje to dodanie grupy o nazwie *New Users Group*. Jeżeli użytkownik zechce zmienić jej nazwę, wystarczy aby kliknął prawym przyciskiem myszy i wybrał polecenie *Rename* oraz wpisał nową nazwę.

W celu ręcznego dodania użytkowników należy zaznaczyć odpowiednią grupę użytkowników, a następnie przejść na koniec listy należących do niej użytkowników. W nowym wierszu należy wprowadzić nazwę użytkownika i jego hasło. Przykład takich działań przedstawiono na rysunku 10.18.



Rysunek 10.18. Ręczne dodanie użytkownika

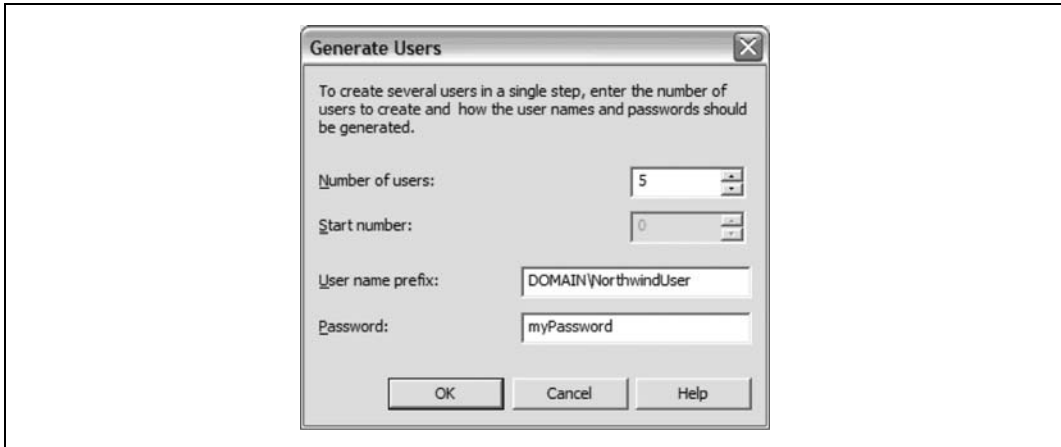


W celu dodania użytkowników należących do określonej domeny należy użyć zapisu `NAZWA_DOMENY\Nazwa_użytkownika`.

W celu automatycznego wygenerowania użytkowników przez narzędzie ACT należy wykonać następujące działania:

1. Zaznaczamy docelową grupę użytkowników. Uruchamiamy z menu polecenie *Actions/Generate Users*.
2. Zostaje wyświetlone okno dialogowe *Generate Users*. Wprowadzamy w nim liczbę użytkowników, prefiks nazwy oraz hasło (patrz rysunek 10.19). Następnie klikamy przycisk *OK*.
3. W prawej części okna zostanie wyświetlona wygenerowana lista użytkowników.

W przypadku generowania użytkowników należy zwrócić uwagę na jeden element — *Start number* (numer początkowy). Narzędzie ACT zawsze generuje użytkowników, rozpoczynając od numeru 1 (jak widać na rysunku, pole umożliwiające zmianę numeru początkowego jest nieaktywne). W prezentowanym przykładzie utworzeni zostali użytkownicy od `DOMAIN\NorthwindUser1` do `DOMAIN\NorthwindUser5`.



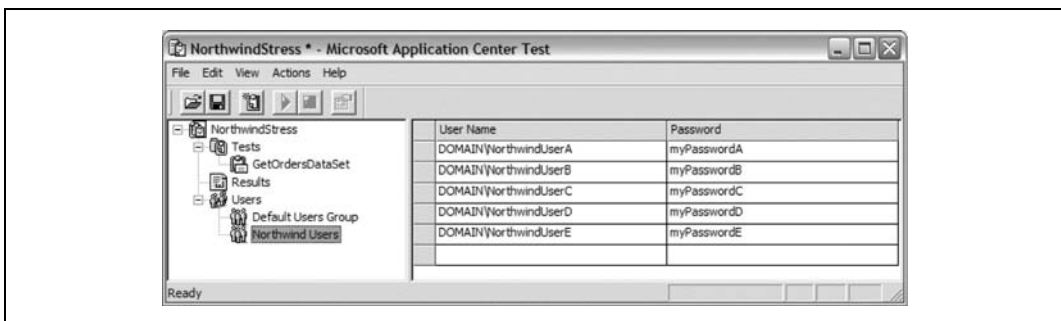
Rysunek 10.19. Okno dialogowe generowania użytkowników

Ostatnią metodą tworzenia użytkowników jest zaimportowanie ich z pliku zawierającego wpisy rozdzielone przecinkami (ACT nie zaimportuje danych z pliku o innym formacie). Plik powinien zawierać nazwy użytkowników i odpowiadające im hasła. W celu przeprowadzenia importu należy wykonać następujące działania:

1. Zaznaczamy docelową grupę użytkowników. Uruchamiamy z menu polecenie *Actions/Import Users*.
2. Znajdujemy plik z danymi i klikamy przycisk *Open*.
3. W prawej części okna zostanie wyświetlona wygenerowana lista użytkowników

Poniżej przedstawiono zawartość przykładowego pliku (na rysunku 10.20 przedstawiono listę użytkowników zaimportowanych z tego pliku):

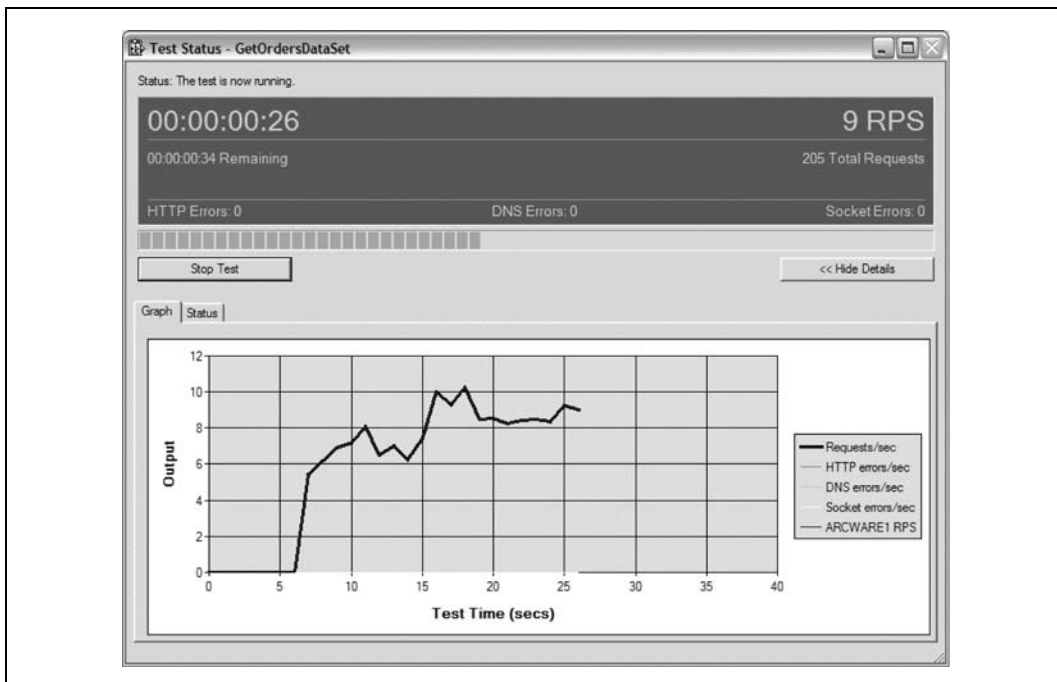
```
DOMAIN\NorthwindUserA, mypasswordA
DOMAIN\NorthwindUserB, mypasswordB
DOMAIN\NorthwindUserC, mypasswordC
DOMAIN\NorthwindUserD, mypasswordD
DOMAIN\NorthwindUserE, mypasswordE
```



Rysunek 10.20. Lista zaimportowanych użytkowników

Uruchamianie testu

Uruchomienie testu w narzędziu ACT poza środowiskiem Visual Studio oraz w jego ramach jest bardzo podobne. W tym celu należy kliknąć prawym przyciskiem myszy nazwę testu i z wyświetlonego menu kontekstowego wybrać polecenie *Start Test*. W obu przypadkach w czasie działania testu są wyświetlane statystyki, jednak użycie ACT w trybie niezależnym pozwala na obserwowanie eleganckich wykresów zamiast wierszy testu. Przykład takiej reprezentacji graficznej przedstawiono na rysunku 10.21.

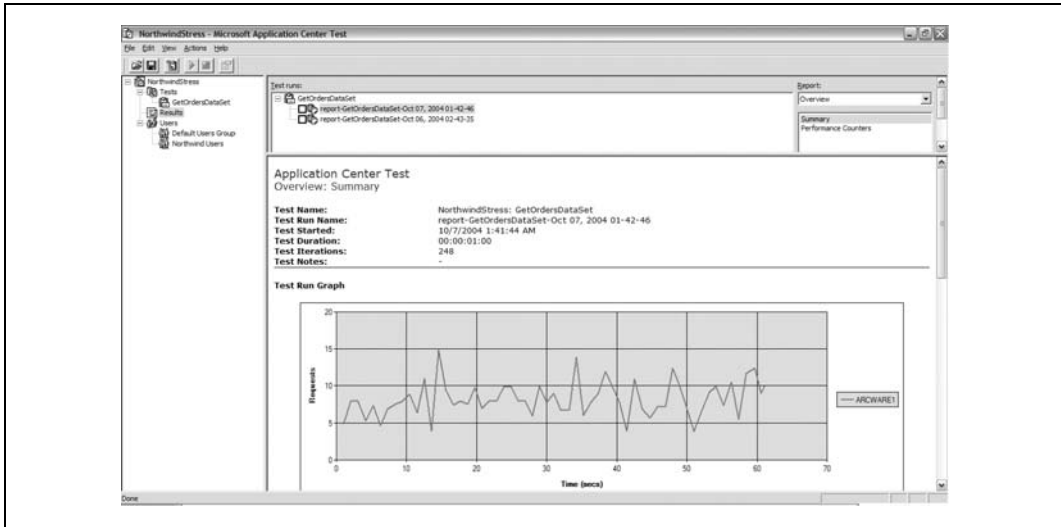


Rysunek 10.21. Wykres statystyk uruchomionego testu

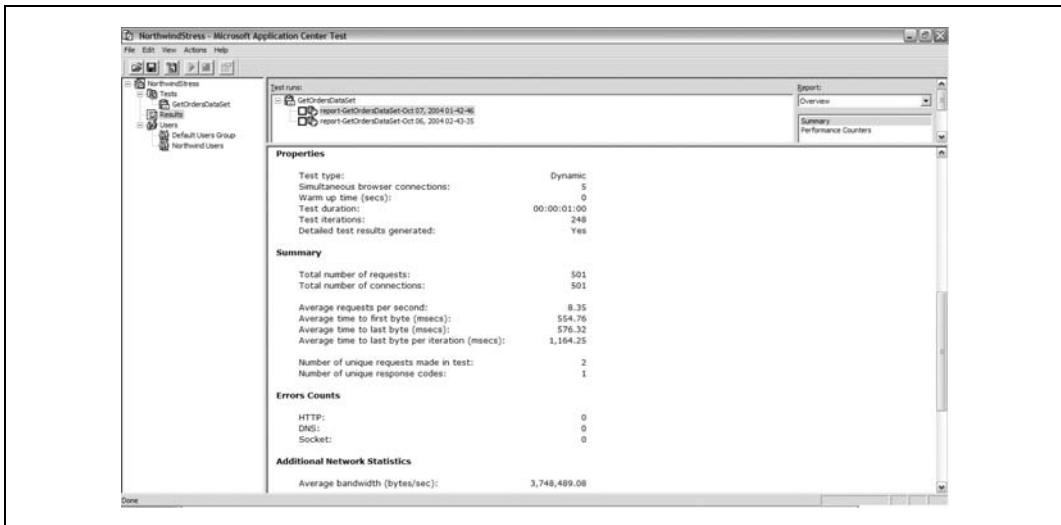
Przeglądanie wyników

Po zakończeniu testu można sprawdzić jego wyniki, zaznaczając wierzchołek *Results* znajdujący się w lewej części okna, a następnie wybierając odpowiedni raport testu z listy *TestRuns* (górne okno po lewej stronie). Na rysunku 10.22 przedstawiono migawkę, zaś na rysunku 10.23 bardziej szczegółowe statystyki.

Jak zapewne Czytelnik pamięta, w ramach testu *GetOrdersDataSet* dodaliśmy kilka liczników wydajności. Statystyki dla tych liczników można przejrzeć po wybraniu opcji *Performance Counters* w lewym górnym oknie. Na rysunku 10.24 przedstawiono statystyki liczników wydajności dla omawianego przykładu.



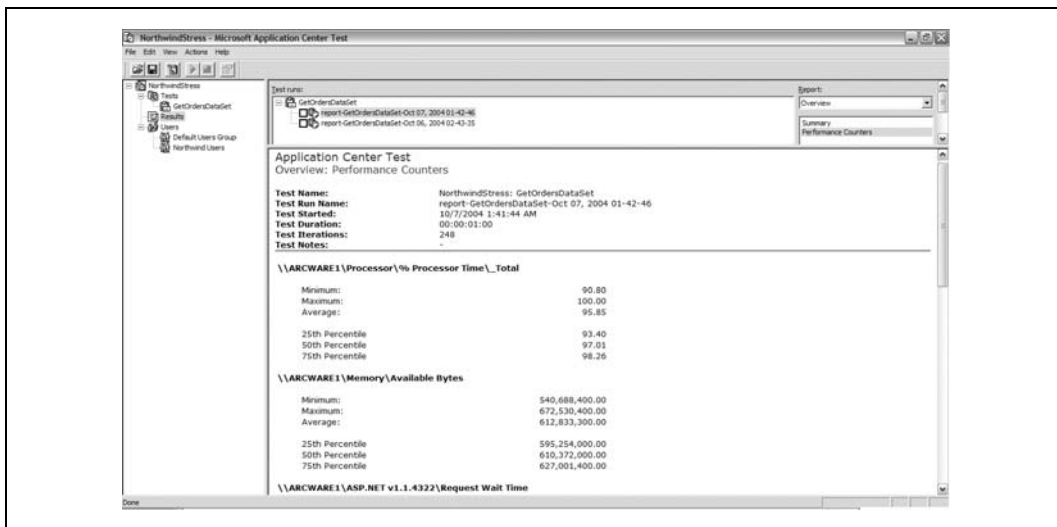
Rysunek 10.22. Migawka raportu podsumowującego



Rysunek 10.23. Statystyki raportu podsumowującego

Czytelnik poznał zatem metody tworzenia i zarządzania testami wydajnościowymi aplikacji WWW przy użyciu narzędzia ACT — zarówno z poziomu, jak i spoza środowiska Visual Studio. Głównym powodem używania ACT z poziomu tego środowiska jest jego ścisła integracja z utworzonymi projektami ASP.NET, jednak wówczas nie da się wykorzystać wszystkich funkcji oferowanych przez ACT. Jego użycie poza środowiskiem Visual Studio oferuje więcej opcji, takich jak graficzna prezentacja wyników, obsługa grup użytkowników i monitorowanie liczników wydajności.

— Dave Donaldson



Rysunek 10.24. Statystyki liczników wydajności

SPOSÓB
80.

Zaciemnianie kodu

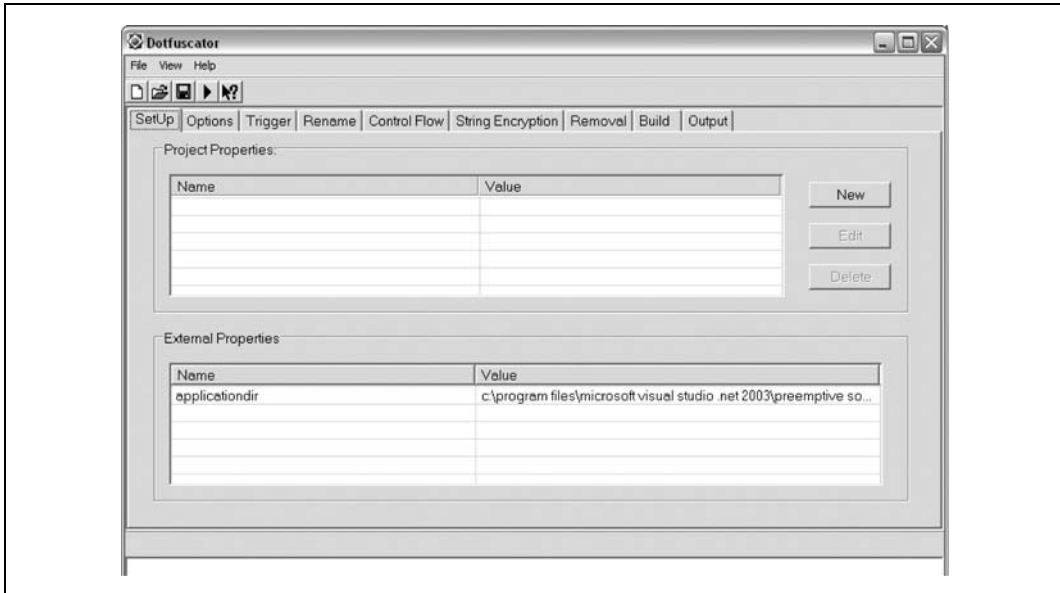
Nawet jeśli samemu się ma problemy z odczytem własnego kodu, wskazane może się okazać jeszcze większe utrudnienie innym osobom zadania dekompilacji rozproszonych podzespołów.

Podobnie jak w świecie programów pisanych w języku Java, istnieje wiele narzędzi służących do dekompilacji kodu .NET [Sposób 63.]. Zarówno wirtualna maszyna Javy (JVM), jak i wspólne środowisko uruchomieniowe języków (CLR) platformy .NET oferują mechanizmy wykorzystania metody refleksji w celu „zaglądania” do skompilowanych podzespołów. Bez wątpienia niejeden programista poczuje się tym faktem zagrożony, wszakże może chodzić o biblioteki klas zawierające tajny kod logiki biznesowej, którego wpadnięcie w niepowołane ręce mogłoby spowodować utratę dominującej pozycji na rynku. Na szczęście środowisko Visual Studio zawiera narzędzie noszące nazwę Dotfuscator, które służy do zapewnienia, aby wynik działania procesu dekompilacji w jak największym stopniu przypominał niezrozumiałą kombinację ciągów liter i cyfr.

Tworzenie projektu Dotfuscator

W celu wykorzystania narzędzia Dotfuscator najpierw należy utworzyć projekt w Visual Studio i skompilować go (w omawianym przykładzie utworzono projekt zawierający klasę o nazwie Simple-Math). Następnie należy utworzyć projekt *Dotfuscator Project* dla otrzymanego podzespołu, wykonując następujące działania:

1. Z menu środowiska Visual Studio uruchamiamy polecenie *Tools/Dotfuscator Community Edition*. Powoduje to uruchomienie narzędzia Dotfuscator.
2. W oknie dialogowym *Select Project Type* zaznaczamy opcję *Create New Project* i klikamy przycisk *OK*. Zostaje wówczas wyświetlone okno narzędzia zawierające kilka zakładek (patrz rysunek 10.25).



Rysunek 10.25. Okno narzędzia Dotfuscator

Choć narzędzie to pozwala na konfigurowanie wielu opcji, wymagane jest określenie tylko dwóch z nich w celu utworzenia zaciemnionej wersji podzespołu. Chodzi tu o plik(i) wyzwalacza oraz katalog docelowy procesu konsolidacji. Po ustawieniu tych opcji można zapisać projekt Dotfuscator w celu późniejszego wykorzystania. Projekty te są zapisywane w formacie XML.

3. Przechodzimy na zakładkę *Trigger*, gdzie brak jest jakichkolwiek wpisów. Klikamy przycisk *Browse*, znajdujemy odpowiedni podzespół i klikamy przycisk *Open*. Podzespół zostaje dodany do projektu.
4. Przechodzimy na zakładkę *Build*. W celu określenia treści pola *Destination Directory* klikamy przycisk *Browse*, znajdujemy katalog, w którym chcemy umieścić zaciemnioną wersję podzespołu, i klikamy przycisk *OK*. Należy zapewnić, aby nie był to ten sam katalog, w którym znajduje się podzespół *Debug*.
5. Uruchamiamy polecenie *File/Save Project (Ctrl+S)*. Przechodzimy do katalogu, w którym chcemy zapisać plik XML, nadajemy mu nazwę i klikamy przycisk *Save*.

Konsolidacja projektu Dotfuscator

Po utworzeniu i zapisaniu projektu Dotfuscator można go skonsolidować. W tym celu uruchamiamy polecenie *File/Build (Ctrl+B)*. Spowoduje to ponowne wykonanie konsolidacji podzespołu z uwzględnieniem opcji zaciemniania określonych w projekcie Dotfuscator.

Posiadając zaciemnioną wersję podzespołu, możemy poddać go badaniu przez dwa dobrze znane narzędzia — ILDASM oraz Reflector.

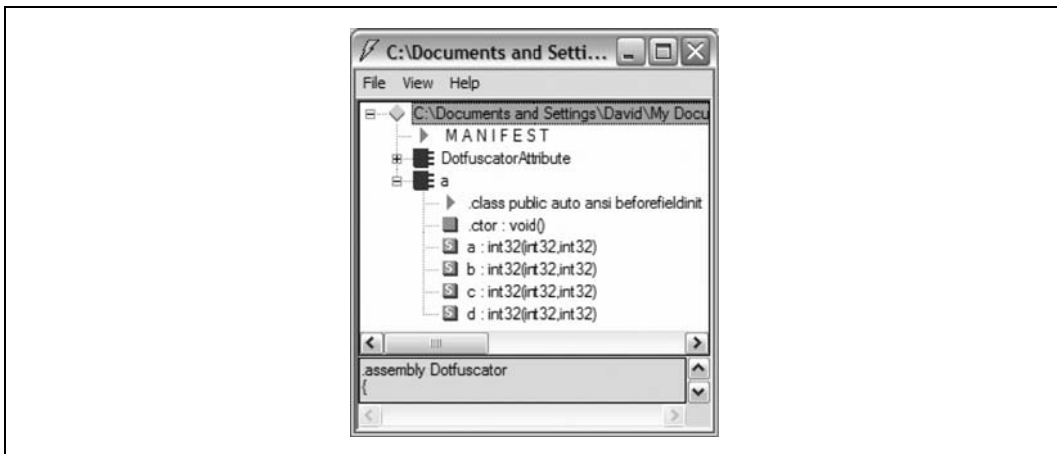
Zbadanie wyników za pomocą narzędzia ILDASM

Jak wiadomo, narzędzie ILDASM może służyć do przeglądania wykazu (manifestu) podzespołu oraz jego kodu IL [Sposób 63.]. Z narzędzia tego można skorzystać w celu zweryfikowania skuteczności działań narzędzia Dotfuscator, porównując zaciemnioną wersję podzespołu z wersją oryginalną. Jako przykładu użyjemy klasy `SimpleMath`.

Uruchamiamy więc ILDASM i otwieramy niezaciemnioną wersję podzespołu. Następnie uruchamiamy drugi egzemplarz ILDASM, otwierając w nim wersję zaciemnioną. W obu przypadkach rozwijamy listę klas w celu bliższego ich zbadania. Na rysunku 10.26 przedstawiono wersję niezaciemnioną, zaś na rysunku 10.27 — wersję zaciemnioną.



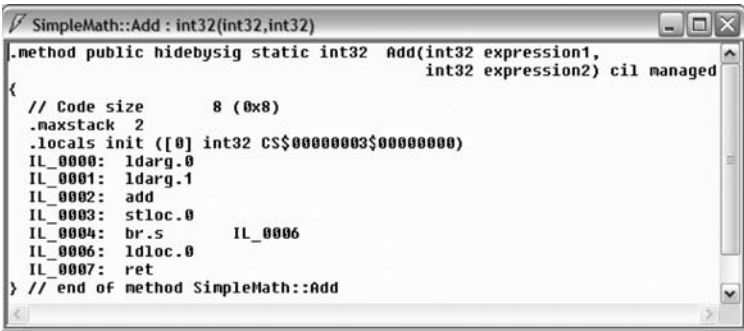
Rysunek 10.26. Podzespół w wersji niezaciemnionej



Rysunek 10.27. Podzespół w wersji zaciemnionej

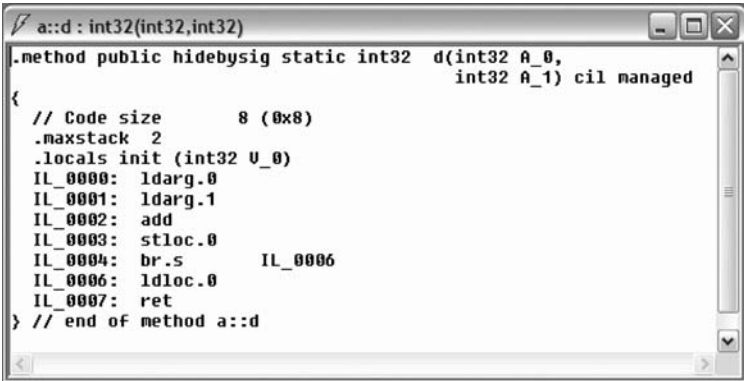
Warto zauważyć, że nazwa klasy (SimpleMath) i jej metody (Add, Divide, Multiply, Subtract) są wyraźnie widoczne w podzespole niezaciemnionym, natomiast w przypadku wersji zaciemnionej podzespół nie zawiera żadnych nazw znaczących. Gdyby ktoś miał użyć narzędzia ILDASM względem zaciemnionego kodu w celu jego bliższego poznania, nie uzyskałby wielkiego wsparcia z jego strony.

Poniżej przyjrzymy się samemu kodowi IL. Dla uproszczenia zajmiemy się tylko jedną metodą — Add. Na rysunku 10.28 przedstawiono niezaciemniony kod IL metody Add, zaś na rysunku 10.29 — jego wersję zaciemnioną.



```
SimpleMath::Add : int32(int32,int32)
.method public hidebysig static int32 Add(int32 expression1,
                                           int32 expression2) cil managed
{
    // Code size      8 (0x8)
    .maxstack 2
    .locals init ([0] int32 CS$00000003$00000000)
    IL_0000: ldarg.0
    IL_0001: ldarg.1
    IL_0002: add
    IL_0003: stloc.0
    IL_0004: br.s      IL_0006
    IL_0006: ldloc.0
    IL_0007: ret
} // end of method SimpleMath::Add
```

Rysunek 10.28. Kod IL niezaciemnionej wersji metody Add



```
a::d : int32(int32,int32)
.method public hidebysig static int32 d(int32 A_0,
                                          int32 A_1) cil managed
{
    // Code size      8 (0x8)
    .maxstack 2
    .locals init (int32 U_0)
    IL_0000: ldarg.0
    IL_0001: ldarg.1
    IL_0002: add
    IL_0003: stloc.0
    IL_0004: br.s      IL_0006
    IL_0006: ldloc.0
    IL_0007: ret
} // end of method a::d
```

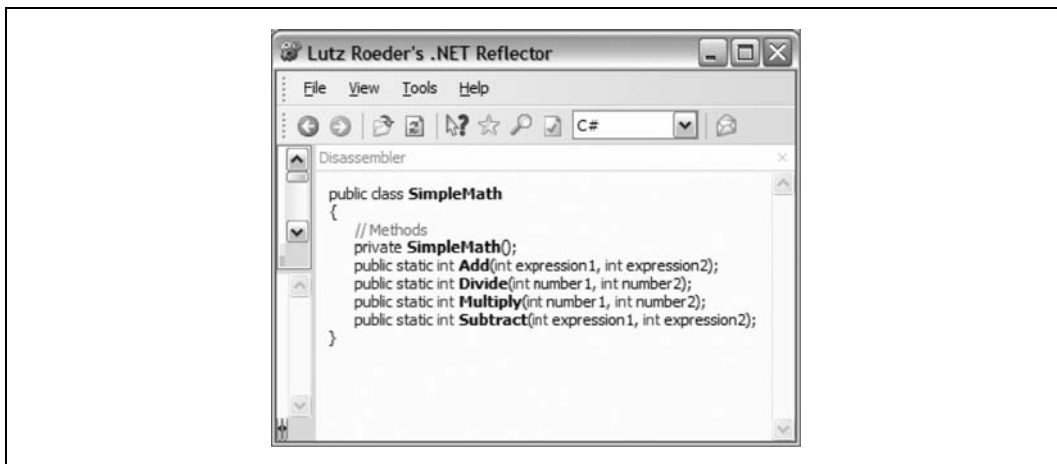
Rysunek 10.29. Kod IL zaciemnionej wersji metody Add

Warto zauważyć, że jedyną istniejącą między nimi różnicą jest sygnatura metody. Faktyczna ścieżka wykonania kodu jest identyczna, a stąd oba podzespoly będą działać dokładnie tak samo. Z zaciemnieniem kodu nie wiąże się żaden narzut obliczeniowy.

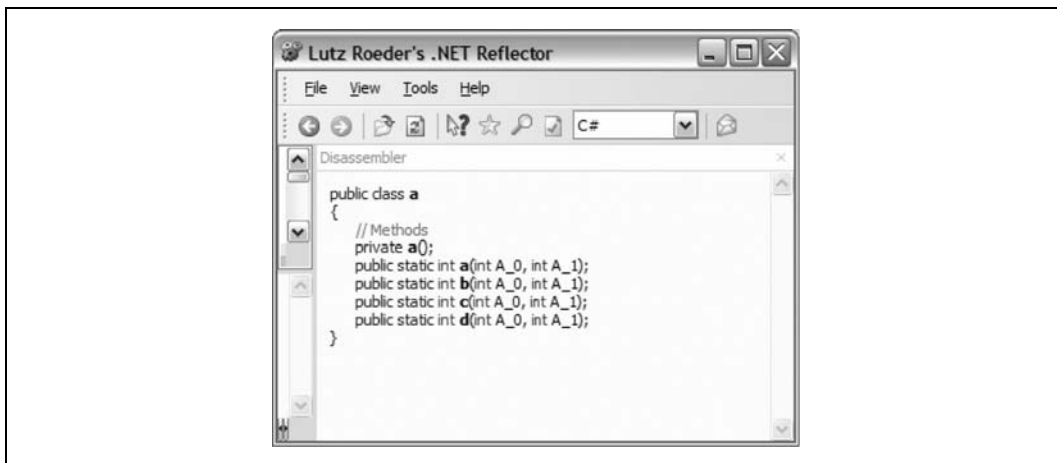
Zbadanie wyników za pomocą narzędzia Reflector

Reflector to narzędzie, którego można używać w celu „zglądania” w strukturę podzespołów .NET i przeglądania ich kodu oraz logiki [Sposób 64.]. Nie tworzy ono wiernej kopii faktycznego kodu podzespołu, ale nie ma to w tym przypadku znaczenia. Wykorzystuje się je w celu zapewnienia możliwości poznania logiki podzespołów. Dlatego też, wciąż posiłkując się klasą `SimpleMath`, zbadamy różnice, jakie pokazuje Reflector w przypadku niezaciemnionej oraz zaciemnionej wersji podzespołu.

Uruchamiamy dwa egzemplarze narzędzia Reflector: w jednym z nich otwieramy podzespół niezaciemniony, zaś w drugim podzespół zaciemniony. Na rysunkach przedstawiono zdeasemblowaną postać klasy `SimpleMath`; rysunek 10.30 przedstawia wersję niezaciemnioną, zaś rysunek 10.31 — wersję zaciemnioną.



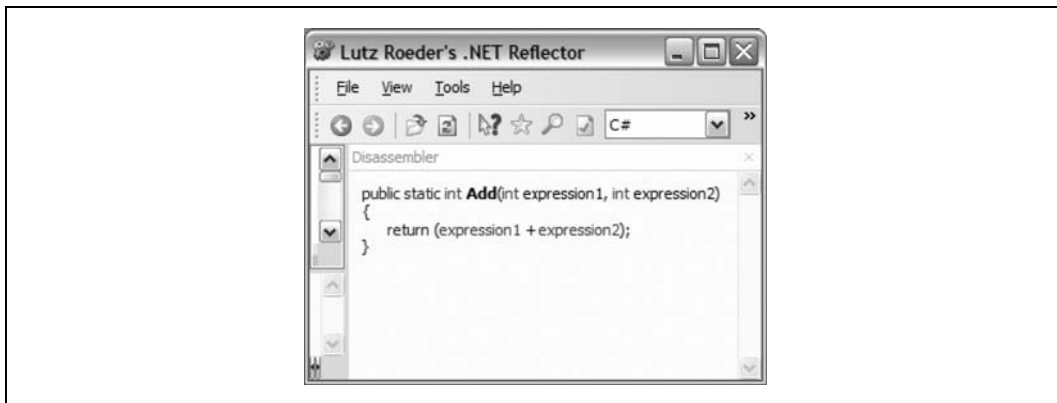
Rysunek 10.30. Niezaciemniona wersja klasy w narzędziu Reflector



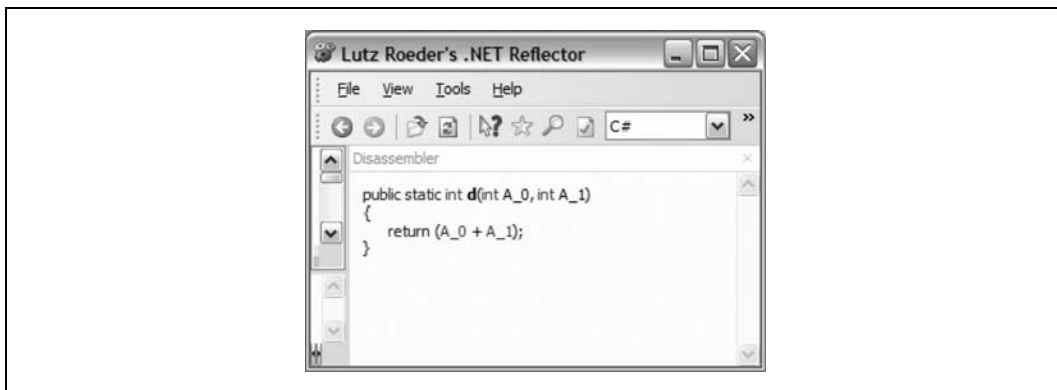
Rysunek 10.31. Zaciemniona wersja klasy w narzędziu Reflector

Podobnie jak w przypadku narzędzia ILDASM Reflector nie prezentuje faktycznych nazw klasy i metod w przypadku zaciemnionego kodu (ponieważ kod IL zaciemnionego podzespołu nie zawiera ich).

Prawdziwe możliwości Reflectora wynikają z możliwości przeglądania zdeasemblowanego kodu każdej metody, więc spróbujmy wykorzystać je właśnie w ten sposób w przypadku metody `Add`. Na rysunku 10.32 przedstawiono zdeasemblowaną wersję kodu niezaciemnionego, zaś na rysunku 10.33 — kodu zaciemnionego.



Rysunek 10.32. Zdeasemblowana postać niezaciemnionej wersji metody `Add`



Rysunek 10.33. Zdeasemblowana postać zaciemnionej wersji metody `Add`

Jak widać na tym prostym przykładzie, zdeasemblowane wersje są bardzo podobne. Jednak w celu stwierdzenia, że metoda `d` służy do dodawania dwóch liczb, należy przejrzeć zdeasemblowany kod każdej metody z zaciemnionej wersji podzespołu, a następnie stwierdzić, że metoda `d` wykonuje po prostu operację dodawania dwóch liczb. Każdy choćby nieco bardziej skomplikowany podzespół .NET skorzysta znacznie więcej na zaciemnieniu swojego kodu.

Zaciemnianie kodu to w rzeczywistości tylko utrudnienie — posiadając wystarczającą ilość czasu i odpowiednie umiejętności, nawet najbardziej zaciemniony kod można zdekompilować i przeanalizować. Jedynym pewnym sposobem zapewnienia, aby kod nie został zdekompilowany, jest nieumieszczanie go na maszynie klienta. Jedną z takich technik polega na zawarciu istotnego kodu logiki biznesowej w ramach usługi sieciowej wywoływanej z poziomu aplikacji. Fakt, że zaciemnianie nie stanowi gwarancji bezpieczeństwa kodu, nie oznacza, że jest bezwartościowe, gdyż zapobiega dekompilacji aplikacji przez większość osób.

— Dave Donaldson



SPOSÓB

81.

Generowanie kodu na podstawie diagramów UML

Istnieje możliwość usprawnienia swoich działań związanych z kodowaniem poprzez użycie narzędzia Visio w celu automatycznego generowania kodu.

Kiedy firma Microsoft w 1999 roku kupiła firmę Visio, zaczęto prace nad lepszym zintegrowaniem narzędzia Visio z pakietem oprogramowania Office i Visual Studio. Spowodowało to wprowadzenie wielu ulepszeń do samego Visio, szczególnie w zakresie diagramów UML do projektowania aplikacji. UML to standard przemysłowy pozwalający architektom i programistom na tworzenie dokumentów projektowych i diagramów, które są powszechnie zrozumiałe i używane w toku procesu tworzenia oprogramowania.

Jednym z usprawnień narzędzia Visio jest możliwość generowania kodu na podstawie dokumentów projektowych, a w szczególności diagramów klas używanych w celu generowania kodu klas w Visual Studio. Visio umożliwia generowanie kodu w języku C#, VB oraz C++ i istnieje nawet możliwość tworzenia w Visio własnych szablonów określających sposób generowania kodu. W przypadku zespołów programistycznych korzystających z dokumentów projektowych może to stanowić znaczne usprawnienie w zakresie tworzenia kodu.

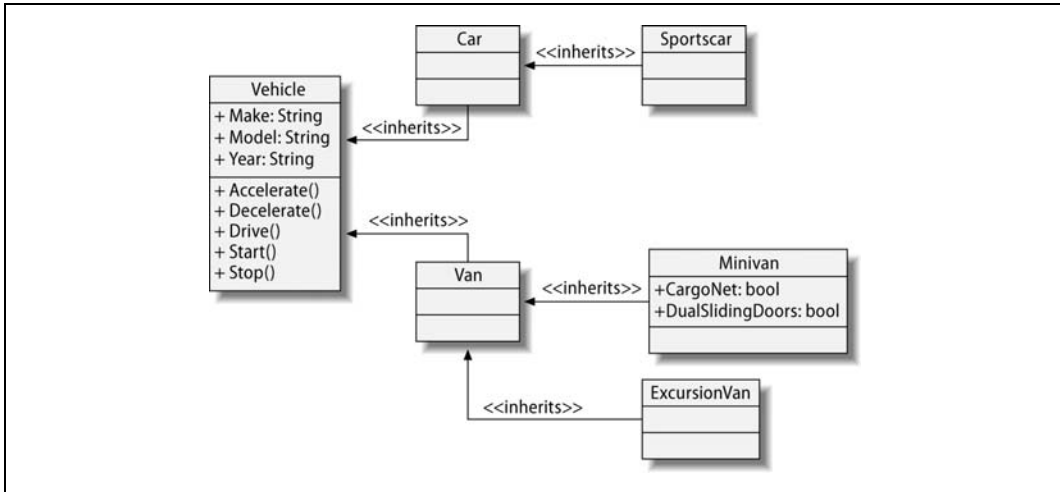


Opisywana funkcja jest dostępna tylko w wersji Visio for Enterprise Architects dostarczanej z systemem Visual Studio .NET 2003 Enterprise Architect Edition. Żadna inna wersja programu Visio nie oferuje mechanizmu generowania kodu.

Prosty diagram klas

W celu zapewnienia, aby narzędzie Visio wygenerowało kod dla zaprojektowanych klas, najpierw należy utworzyć diagram klas UML. W tym celu uruchamiamy polecenie *File/New/Software/UML Model Diagram*. Następnie tworzymy diagram klas, używając komponentów z okna *UML Static Structure*. Na rysunku 10.34 przedstawiono przykładowy diagram klas.

Rysunek 10.34 przedstawia diagram klas stanowiący dokumentację hierarchii klas *Vehicle* (pojazd). Ten dość prosty przykład pokazuje abstrakcyjną klasę bazową o nazwie *Vehicle*, która posiada kilka właściwości (*Make* (marka), *Model* (model), *Year* (rok produkcji)) oraz metod (*Accelerate* (przyspiesz), *Decelerate* (zwolnij), *Drive* (jedź), *Start* (uruchom silnik), *Stop* (wyłącz silnik)). Diagram klas przedstawia również inne klasy implementujące tę klasę bazową oraz istniejące między nimi związki.



Rysunek 10.34. Diagram klas dla hierarchii klas Vehicle

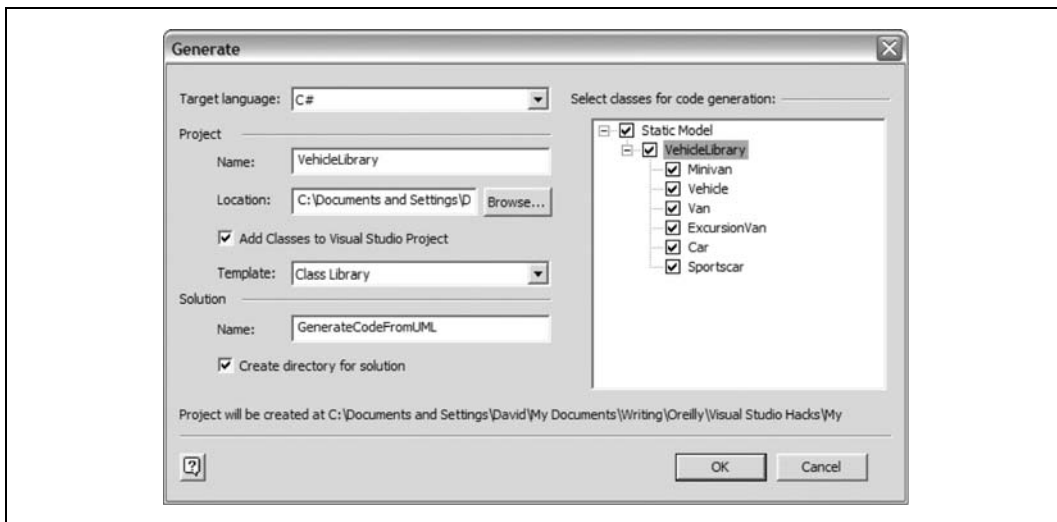
Generowanie kodu

Po utworzeniu diagramu klas w programie Visio w celu wygenerowania odpowiadającego im kodu w Visual Studio należy postąpić zgodnie z opisaną poniżej procedurą:

1. Z menu *File* wybieramy polecenie *UML/Code/Generate*. Zostaje wówczas wyświetlone okno dialogowe *Generate*.
2. Jako *Target Language* (język docelowy) wybieramy opcję C# (lub C++ albo Visual Basic).
3. Wprowadzamy nazwę projektu (*Project Name*) i wybieramy jego lokalizację (*Location*).
4. Zaznaczamy pole *Add classes to Visual Studio Project* (dodaj klasy do projektu Visual Studio)
5. Wybieramy używany szablon. Lista szablonów zawiera kilka typów projektów Visual Studio.
6. Wprowadzamy nazwę rozwiązania (*Solution Name*) i zaznaczamy pole *Create Directory for Solution* (utwórz katalog dla rozwiązania).
7. Zaznaczamy klasy, dla których ma zostać wygenerowany kod.
8. Naciskamy przycisk *OK*. Na rysunku 10.35 przedstawiono przykładowe okno dialogowe *Generate*.

Przeglądanie kodu

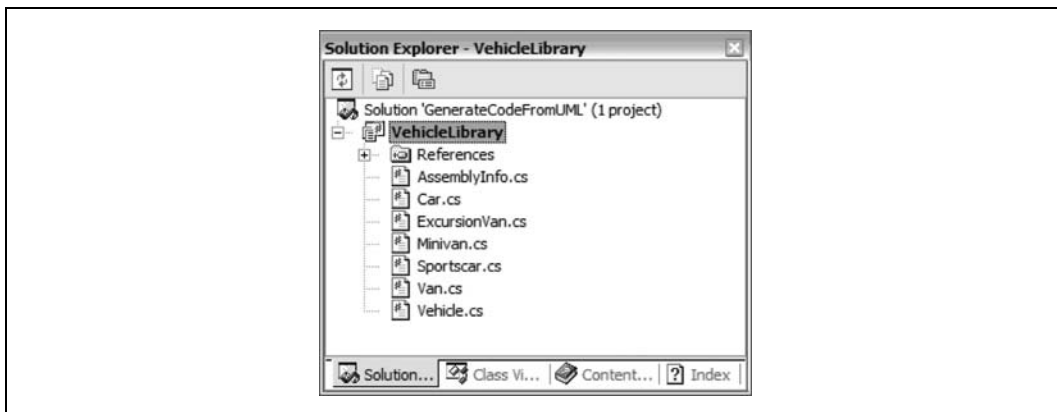
Po wygenerowaniu kodu klas z poziomu programu Visio warto przyjrzeć mu się bliżej. Przechodzimy do lokalizacji określonej w opisanych powyżej krokach. Powinien znajdować się tam folder o nazwie nadanej rozwiązaniu. Folder ten zawiera plik rozwiązania systemu Visual Studio oraz folder o nazwie nadanej projektowi. Przykładowo w opisywanym przypadku rozwiązanie nosi nazwę *GenerateCodeFromUML*, zaś projekt — *VehicleLibrary*.



Rysunek 10.35. Okno dialogowe *Generate*

Dwukrotnie klikamy wygenerowany pliku rozwiązania w celu otwarcia nowego egzemplarza Visual Studio. Najprawdopodobniej w uruchomionym środowisku zostanie wyświetlony w edytorze kod pliku *Class1.cs*. Tu Czytelnik może zadać pytanie — skąd taka nazwa, przecież nie utworzyliśmy klasy o nazwie *Class1*. Najprawdopodobniej wynika to z występowania pewnego błędu w generatorze kodu narzędzia Visio. Plik ten można po prostu zignorować lub skasować.

Istotną rzeczą jest fakt, że użytkownik będzie miał dostęp do każdej z klas w oknie *Solution Explorer*, co przedstawiono na rysunku 10.36.



Rysunek 10.36. Wygenerowane pliki klas w oknie *Solution Explorer*

W tym momencie posiadamy w pełni funkcjonalny projekt Visual Studio. Możemy od razu zacząć go modyfikować zgodnie z wymaganiami, jednak najpierw przyjrzymy się wygenerowanemu kodowi. Poniżej podano kod abstrakcyjnej klasy bazowej *Vehicle*:

```
// Static Model
public abstract class Vehicle
{
    public string Make;
    public string Model;
    public string Year;

    public virtual void Accelerate( )
    {
    }

    public virtual void Decelerate( )
    {
    }

    public virtual void Drive( )
    {
    }

    public virtual void Start( )
    {
    }

    public virtual void Stop( )
    {
    }

} // END CLASS DEFINITION Vehicle
```

Jak widać, Visio poprawnie generuje kod dla zaprojektowanych klas i może pozwolić zaoszczędzić wiele czasu, szczególnie wówczas, gdy posiada się bibliotekę klas zawierającą wiele klas o wielu właściwościach.

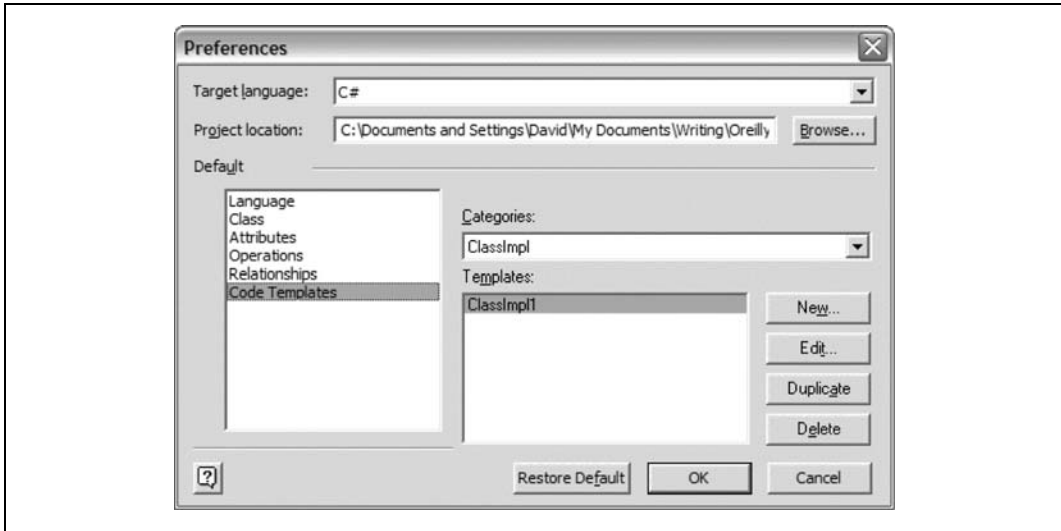
Dodatkowe możliwości

Visio dobrze sprawdza się we wstępnym generowaniu kodu klas. Jednak może się pojawić problem, kiedy użytkownikowi nie spodoba się sposób, w jaki kod ten zostanie wygenerowany. Może również być i tak, że jego format będzie nieco odstawał od stosowanych przez użytkownika konwencji. Istnieje jednak możliwość edytowania i (lub) tworzenia własnych szablonów generowania kodu, dzięki czemu Visio generuje kod w taki sposób, jaki odpowiada użytkownikowi.

W celu otworzenia edytora szablonów kodu należy wybrać polecenie *UML/Code/Preferences*. Spowoduje to wyświetlenie okna dialogowego *Preferences*. Dostępnych jest tu kilka ustawień, jednak dla celów bieżącego sposobu wystarczy, jeśli skupimy się na utworzeniu dwóch nowych szablonów — jednego dla klas i jednego dla właściwości.

Tworzenie nowego szablonu klas

W oknie *Preferences* na liście *Default* zaznaczamy pozycję *Code Templates*. Spowoduje to wyświetlenie dodatkowych ustawień, co przedstawiono na rysunku 10.37.



Rysunek 10.37. Okno dialogowe Preferences z wybraną opcją Code Templates

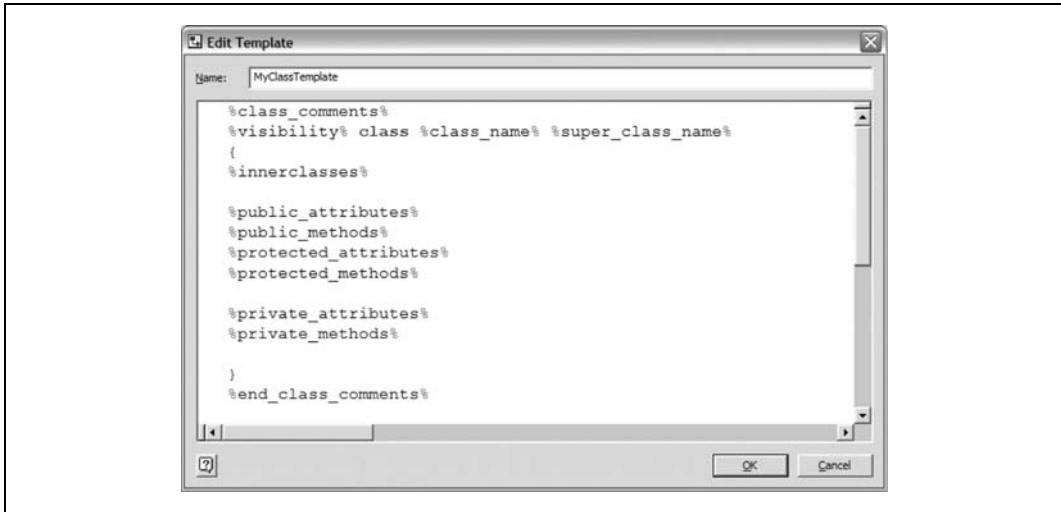
Domyślnie Visio proponuje szablon klas o nazwie `ClassImpl1`. Najlepszym sposobem utworzenia własnego szablonu jest skopiowanie szablonu `ClassImpl1`, a następnie jego zmodyfikowanie. W tym celu należy wykonać następujące działania:

1. Z listy rozwijanej *Categories* wybieramy szablon `ClassImpl`. Powoduje to pojawienie się na liście *Templates* szablonu `ClassImpl`.
2. Klikamy przycisk *Duplicate*. Powoduje to wyświetlenie okna *Edit Template* (patrz rysunek 10.38). Nie będziemy tu opisywać go szczegółowo, ale wystarczy stwierdzić, że prezentuje ono połączenie zwykłego tekstu z makrami Visio. Makra są zapisywane w formie `%nazwa_makro%`. Jest ich zbyt wiele, by je tu omawiać, ale ich listę można przejrzeć, otwierając system pomocy Visio i wyświetlając temat *Use built-in macros to speed up code formatting*.
3. Nadajemy nowemu szablónowi nazwę i modyfikujemy go zgodnie z potrzebami.
4. Następnie naciskamy przycisk *OK*. Utworzony szablon pojawia się na liście *Templates*.
5. Na liście *Default* zaznaczamy pozycję *Class*.
6. W polu *Class Template* wybieramy odpowiedni szablon klasy z listy *Implementation*. Naciskamy przycisk *OK*.
7. Od tej pory za każdym razem, gdy Visio będzie generować pliki klas, użyje nowego szablonu klas zamiast domyślnego.

Tworzenie nowego szablonu właściwości (atrybutów)

Domyślnie Visio tworzy kod dla właściwości w następującym formacie:

```
public bool CargoNet;
```



Rysunek 10.38. Okno *Edit Template*

Jednak styl kodowania używany przez użytkownika może wymagać zapisu właściwości w następujący sposób:

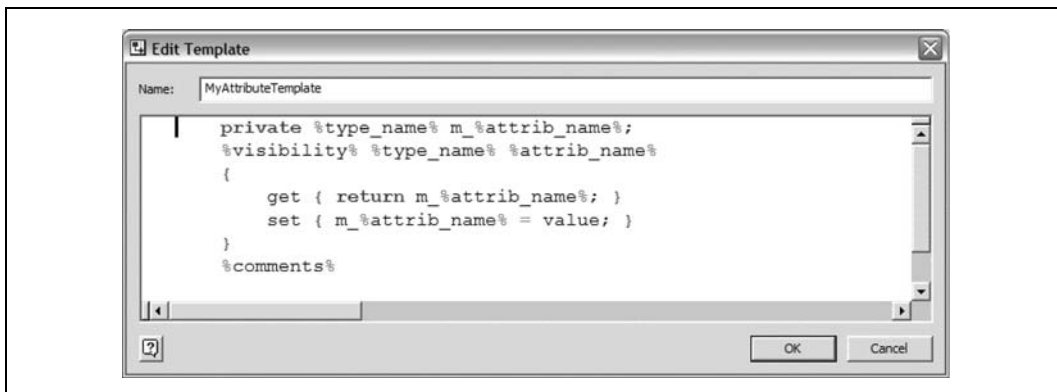
```
private bool m_CargoNet;
public bool CargoNet
{
    get { return m_CargoNet; }
    set { m_CargoNet = value; }
}
```

W celu utworzenia własnego szablonu właściwości (nazywanych w Visio atrybutami) należy wykonać następujące działania:

1. W oknie dialogowym *Preferences* na liście *Default* zaznaczamy pozycję *Code Templates*.
2. Z listy *Categories* wybieramy pozycję *Attribute*.
3. Na liście szablonów pojawi się szablon o nazwie *Attribute1*. Klikamy przycisk *Duplicate*.
4. Ponownie pojawia się okno *Edit Template*. Dla potrzeb omawianego przykładu należy zapisać szablon tak, jak przedstawiono to na rysunku 10.39.
5. Klikamy przycisk *OK*. Na liście *Templates* pojawia się nazwa naszego szablonu.
6. Na liście *Default* zaznaczamy pozycję *Attributes*.
7. Wybieramy szablon atrybutu z listy *Templates*.
8. Klikamy przycisk *OK*. Od tej pory, kiedy program Visio będzie generować kod właściwości, wykorzysta zdefiniowany przez nas szablon zamiast domyślnego.

Ponowne wygenerowanie kodu i jego przejrzanie

Po utworzeniu własnych szablonów dla klas i właściwości ponownie generujemy kod, postępując zgodnie z opisaną wcześniej procedurą. Następnie otwieramy wygenerowane



Rysunek 10.39. Nowy szablon kodu programu Visio dla właściwości

rozwiązanie oraz plik *Minivan.cs* w edytorze Visual Studio. Jak widać, format właściwości został zaktualizowany w celu odzwierciedlenia formatu utworzonego szablonu:

```
// Static Model

public class Minivan : Van
{
    private bool m_CargoNet;
    public bool CargoNet
    {
        get { return m_CargoNet; }
        set { m_CargoNet = value; }
    }

    private bool m_DualSlidingDoors;
    public bool DualSlidingDoors
    {
        get { return m_DualSlidingDoors; }
        set { m_DualSlidingDoors = value; }
    }
}

// END CLASS DEFINITION Minivan
```

Powyżej opisano prosty przykład użycia Visio w celu automatycznego generowania kodu klas dla zwiększenia wydajności tworzenia programu. Jednak szablony kodu Visio mogą być dostosowywane do szczególnych wymagań i warto poeksperymentować z tą opcją w celu znalezienia odpowiedniej postaci generowanego kodu.



Program Visio oferuje ograniczoną obsługę mechanizmu inżynierii wahadłowej (ang. *round-tripping*) — umożliwia generowanie kodu na podstawie diagramów, jednak kiedy dokona się ich zmian, nie potrafi zaktualizować kodu w oparciu o owe zmiany — umożliwia jedynie ponowne wygenerowanie kodu od początku. Visio umożliwia także generowanie diagramów na podstawie kodu, jednak nie obsługuje mechanizmu ich aktualizowania, a jedynie ponowne tworzenie. Pewne inne dostępne narzędzia, takie jak Rational XDE, obsługują mechanizm inżynierii wahadłowej w znacznie szerszym zakresie.

— Dave Donaldson

SPOSÓB
82.

Generowanie diagramów UML na podstawie kodu

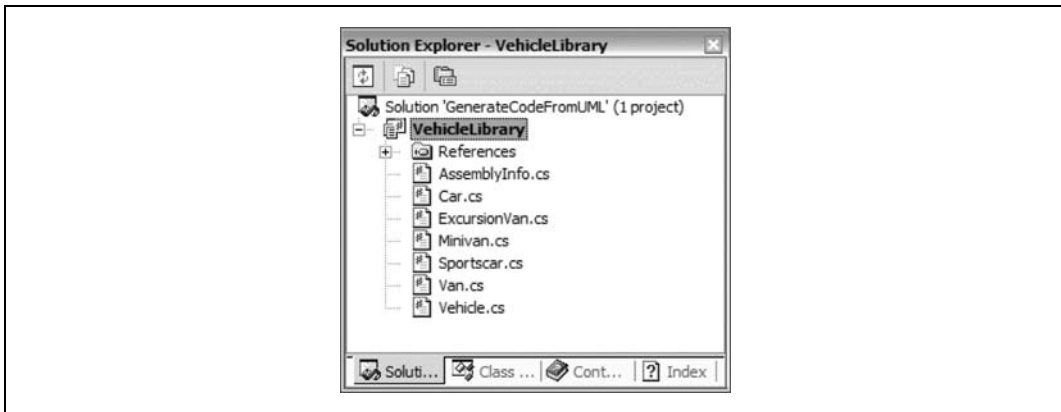
Istnieje możliwość wykorzystania Visual Studio w celu generowania dokumentów projektowych w programie Visio.

Jak wszyscy dobrzy twórcy oprogramowania staramy się o zapewnienie aktualności dokumentacji projektowej... Prawdopodobnie wcale tak nie jest. Któż bowiem chciałby stale wracać do już wykonanych działań i aktualizować dokumenty projektowe w czasie pisania kodu? Większość twórców oprogramowania zwykle nie ma na to czasu, ponieważ są zbyt zajęci usiłowaniami poprawnego zaimplementowania aplikacji.

W kwestii tej może posłużyć pomocą system Visual Studio, generując takie dokumenty projektowe, a konkretnie diagramy klas, dokonując inżynierii wstecznej zapisanego kodu. Choć nie zastąpi i nie zaktualizuje się w ten sposób wszystkich dokumentów projektowych, działania te mogą skrócić czas związany z ich samodzielnym pielęgnowaniem. Zasadniczo niniejszy sposób stanowi odwrotność sposobu [Sposób 81.].

Inżynieria wsteczna kodu

Najpierw musimy utworzyć projekt Visual Studio. Może to być projekt dowolnego typu i zapewne będzie to projekt ostatnio edytowany lub utworzony. Dla potrzeb niniejszego sposobu zostanie wykorzystana biblioteka `VehicleLibrary` zdefiniowana w sposobie [Sposób 81.]. Na rysunku 10.40 raz jeszcze przedstawiono okno Solution Explorer dla tego projektu.

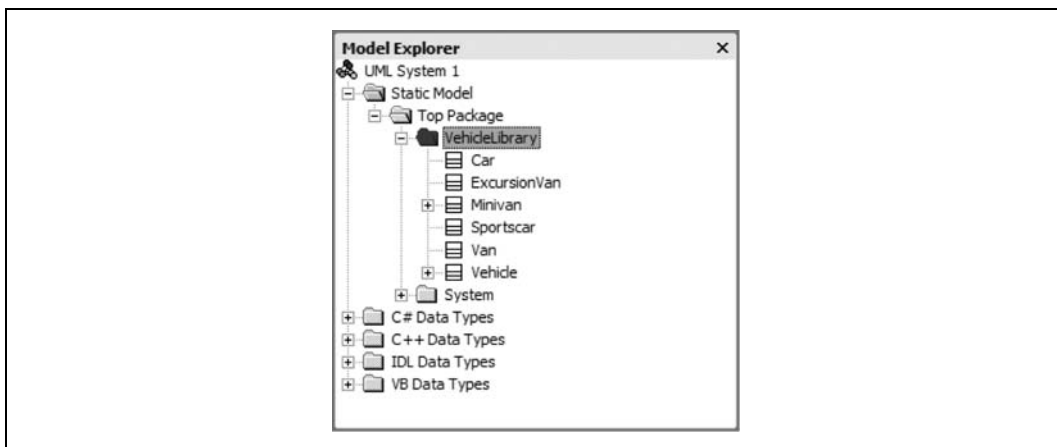


Rysunek 10.40. Projekt *Vehicle Library* w oknie *Solution Explorer*

W celu dokonania inżynierii wstecznej kodu tego projektu należy wykonać następujące działania:

1. Uruchamiamy polecenie *Project/Visio UML/Reverse Engineer*.
2. Przechodzimy do katalogu, w którym zostanie zapisany plik Visio, nadajemy mu nazwę i naciskamy przycisk *Save*.

3. W zależności od rozmiaru i złożoności projektu proces dokonywania inżynierii wstecznej może nieco potrwać.
4. Po zakończeniu tego procesu zostaje automatycznie uruchomiony egzemplarz programu Visio. W oknie *Model Explorer* zostają przedstawione automatycznie wygenerowane klasy (patrz rysunek 10.41).



Rysunek 10.41. Klasy wygenerowane automatycznie w programie Visio

5. W systemie Visual Studio wygenerowany plik Visio zostaje z kolei dodany do projektu w folderze *Solution Items*.

Tworzenie diagramu klas

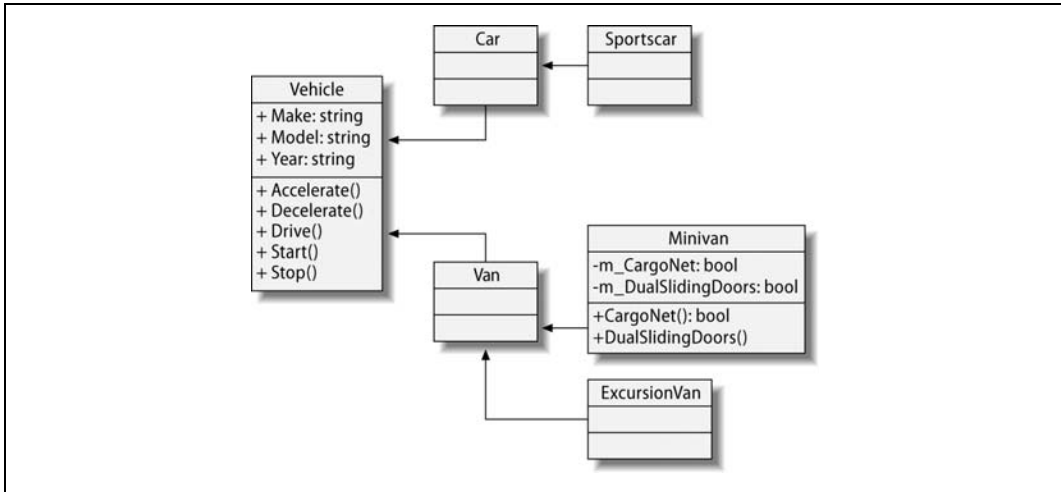
Niestety, to wszystkie możliwości oferowane przez mechanizm inżynierii wstecznej. Diagram klas użytkownik musi utworzyć samodzielnie. W zależności od rozmiaru hierarchii klas może to stanowić niewielki lub duży problem. Na szczęście Visio obsługuje istniejące związki między wygenerowanymi klasami.

Przykładowo, kiedy przeciągamy klasy na diagram, Visio automatycznie określa istniejące między nimi związki i rysuje linie generalizacji za użytkownika. Adekwatny przykład przedstawiono na rysunku 10.42.

Kiedy użytkownik przeciąga klasy z okna Model Explorer na diagram struktur statycznych w programie Visio (patrz rysunek 10.42), wszystkie linie są rysowane za użytkownika. Nie trzeba pamiętać o wszystkich związkach istniejących między klasami, które w omawianym przykładzie okazują się reprezentować dziedziczenie.

Jak widać, Visual Studio można wykorzystać w celu dokonania inżynierii wstecznej klas zapisanych w kodzie projektu i wykorzystać je na diagramach klas, diagramach sekwencji oraz innych powiązanych dokumentach projektowych w programie Visio.

— Dave Donaldson



Rysunek 10.42. Przeciąganie klas do diagramu programu Visio