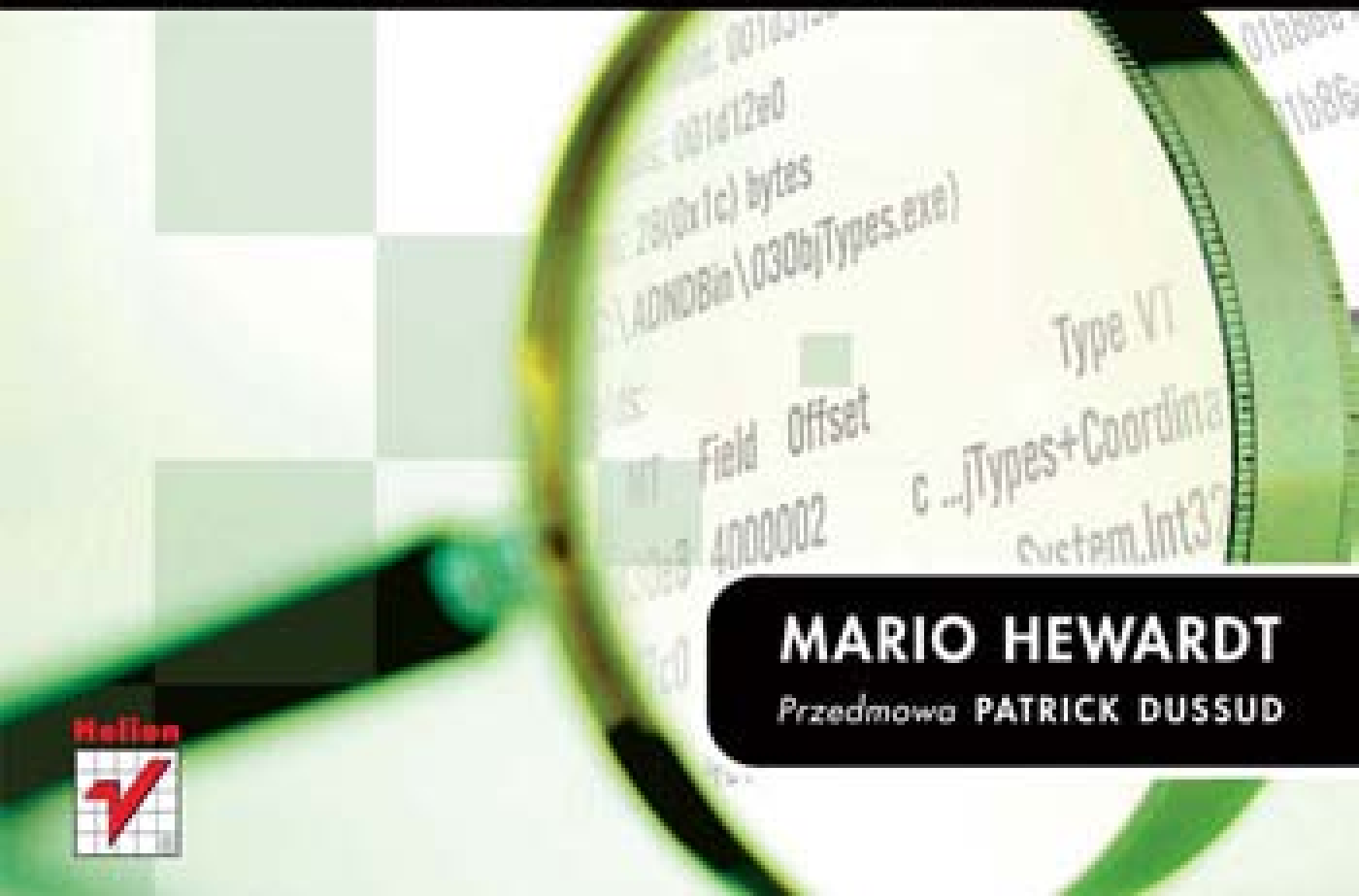
SPRAWDŹ, JAK ROZWIĄZUJĄ PROBLEMY NAJLEPSI SPECJALIŚCI!

JAK: STWORZYĆ NIEZAWODNY KOD? ROZWIĄZYWAĆ PROBLEMY Z SYNCHRONIZACJĄ WĄTKÓW? WYKORZYSTAĆ USŁUGĘ RAPORTOWANIA BŁĘDÓW?

DEBUGOWANIE .NET

ZAAWANSOWANE TECHNIKI DIAGNOSTYCZNE



MARIO HEWARDT

Przedmowa PATRICK DUSSUD

Helion

helion.pl

Idź do

- Spis treści
- Przykładowy rozdział

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

Debugowanie .NET. Zaawansowane techniki diagnostyczne



Autor: Mario Hewardt
Tłumaczenie: Łukasz Piwko
ISBN: 978-83-246-2721-9
Tytuł oryginału: Advanced .NET Debugging
Format: 168×237, stron: 528

Sprawdź, jak rozwiązują problemy najlepsi specjaliści!

- Jak stworzyć niezawodny kod?
- Jak rozwiązywać problemy z synchronizacją wątków?
- Jak wykorzystać usługę raportowania błędów?

Czy znasz programistę, który nigdy w życiu nie użył debuggera? Dzisiejsze rozbudowane i skomplikowane systemy informatyczne wymagają znakomitej znajomości zaawansowanych narzędzi diagnostycznych. Bez nich wykrycie nawet najprostszego potknięcia mogłoby zająć długie godziny, jeśli nie dni, a poświęcony czas wcale nie gwarantowałby sukcesu. Niestety, posługiwanie się zaawansowanymi narzędziami wymaga równie zaawansowanej wiedzy. Jeśli nie masz pewności, czy zgromadziłeś już wszystkie potrzebne Ci informacje, sięgnij po tę książkę – pozwoli to błyskawicznie uzupełnić brakujące dane!

W pierwszej części książki będziesz mógł zapoznać się z podstawami programowania oraz narzędziami wykorzystywanymi w codziennej pracy programisty. Poznasz między innymi metody sterowania wykonywaniem programu, techniki inspekcji obiektowej, sposoby operowania na wątkach oraz polecenia diagnostyczne platformy .NET. Część druga zawiera obszerny zakres informacji dotyczących praktycznych aspektów diagnostyki oprogramowania. Dowiesz się, jak wykrywać błędy w zarządzaniu stertą, jak rozwiązywać kłopoty z pamięcią oraz jak radzić sobie z najbardziej skomplikowanymi problemami dotyczącymi wątków. Na samym końcu zapoznasz się z zaawansowanymi zagadnieniami diagnostyki oprogramowania, takimi jak usługi raportowania błędów, pliki obrazu oraz wyczerpujące opisy najlepszych narzędzi. W książce „Debugowanie .NET. Zaawansowane techniki diagnostyczne" znajdziesz między innymi takie zagadnienia:

- Podstawowe informacje o dostępnych narzędziach
- Podstawy CLR
- Techniki diagnostyczne – sterowanie wykonaniem programu, punkty wstrzymania
- Inspekcja obiektowa oraz inspekcja kodu
- Dostępne polecenia wewnętrzne CLR oraz polecenia diagnostyczne
- Techniki odzyskiwania pamięci
- Rozwiązywanie problemów z synchronizacją wątków
- Metody diagnozowania wycieków interoperacyjności
- Wykorzystanie plików obrazu
- Generowanie zrzutów z wykorzystaniem debuggera
- Wykorzystanie usługi raportowania błędów
- Dostępne dodatkowe narzędzia, wspierające proces diagnostyczny

Twórz niezawodne oprogramowanie!

Kontakt

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 32 230 98 63
e-mail: helion@helion.pl
© Helion 1991–2010

SPIS TREŚCI

Przedmowa **17**

Wstęp **19**

Podziękowania **29**

O autorze **31**

CZĘŚĆ I PODSTAWY **33**

Rozdział 1. Podstawowe wiadomości o narzędziach **35**

Debugging Tools for Windows 36
.NET 2.0 — Redistributable 37
.NET 2.0 — SDK 38
SOS 40
SOSEX 42
CLR Profiler 43
Mierniki wydajności 46
Reflector for .NET 47
PowerDbg 48
Managed Debugging Assistants 50
Podsumowanie 53

Rozdział 2. Podstawy CLR **55**

Przegląd wysokopoziomowy 55
CLR i program ładujący Windows 59
Ładowanie obrazów rodzimych 60
Ładowanie zestawów .NET 62
Domeny aplikacji 66
Systemowa domena aplikacji 69
Wspólna domena aplikacji 70
Domyślna domena aplikacji 70

Zestawy ... 70
Manifest zestawu ... 72
Metadane typu ... 74
Tabela bloku synchronizacji ... 81
Uchwyt do typu ... 85
Deskryptory metod ... 92
Moduły ... 93
Tokeny metadanych ... 96
EEClass ... 98
Podsumowanie ... 100

Rozdział 3. Podstawowe techniki diagnostyczne ... 101

Debuger i proces docelowy debugera ... 101
Symbole ... 106
Sterowanie wykonywaniem programu ... 109
Przerywanie wykonywania ... 109
Wznawianie wykonywania ... 110
Przemierzanie kodu ... 112
Kończenie sesji diagnostycznej ... 116
Ładowanie rozszerzeń kodu zarządzanego ... 116
Ładowanie rozszerzenia SOS ... 118
Ładowanie rozszerzenia SOSEX ... 120
Kontrolowanie procesu debugowania CLR ... 121
Ustawianie punktów wstrzymania ... 121
Punkty wstrzymania na funkcjach skompilowanych przez kompilator JIT ... 124
Punkty wstrzymania na funkcjach jeszcze nieskompilowanych ... 127
Punkty wstrzymania w zestawach prekompilowanych ... 130
Punkty wstrzymania na metodach uogólnionych ... 133
Inspekcja obiektowa ... 134
Zrzucanie pamięci surowej ... 137
Zrzuty typów wartościowych ... 140
Zrzuty podstawowych typów referencyjnych ... 146
Zrzuty tablic ... 147
Zrzuty obiektów na stosie ... 153
Sprawdzanie rozmiarów obiektów ... 155
Zrzuty wyjątków ... 156
Operacje na wątkach ... 161
Polecenie ClrStack ... 162
Polecenie Threads ... 165
Polecenie DumpStack ... 169
Polecenie EEStack ... 171
COMState ... 171
Inspekcja kodu ... 172
Dezasemblacja kodu ... 172
Wydobywanie deskryptora metod z adresu kodu ... 174
Wyświetlanie instrukcji języka pośredniego ... 175

Polecenia wewnętrzne CLR 175
Sprawdzanie wersji CLR 176
Znajdowanie deskryptora metod po nazwie 176
Wykonywanie zrzutu bloku synchronizacji obiektu 177
Wykonywanie zrzutu tabeli metod obiektu 177
Wykonywanie zrzutu informacji o stercie zarządzanej i systemie odzyskiwania pamięci 178
Polecenia diagnostyczne 178
Znajdowanie domeny aplikacji obiektu 179
Informacje o procesie 179
Polecenia rozszerzenia SOSEX 180
Rozszerzona obsługa punktów wstrzymania 180
Zarządzane metadane 184
Dane stosu 185
Inspekcja obiektów 187
Automatyczne wykrywanie zakleszczeń 188
Polecenia dotyczące sterty zarządzanej i systemu odzyskiwania pamięci 190
Zapisywanie zrzutów awaryjnych w plikach 192
Podsumowanie 194

Część II Diagnostyka stosowana 195

Rozdział 4. Program ładujący zestawy 197

Informacje wstępne 197
Tożsamość zestawu 198
Globalny bufor zestawów 202
Domyślny kontekst ładowania 205
Kontekst „load-from” 206
Kontekst „load-without” 207
Prosty błąd ładowania zestawu 207
Błąd kontekstu ładowania 214
Interoperacyjność i wyjątek DllNotFoundException 222
Debugowanie LCG 224
Podsumowanie 229

Rozdział 5. Sterta zarządzana i odzyskiwanie pamięci 231

Architektura pamięci systemu Windows 232
Alokowanie pamięci 236
System odzyskiwania pamięci 241
Pokolenia 242
Korzenie 251
Finalizacja 258

Odzyskiwanie pamięci 267
Sterta obiektów dużych 269
Unieruchamianie obiektów 274
Tryby odzyskiwania pamięci 280
Diagnozowanie uszkodzonej sterty zarządzanej 281
Diagnozowanie fragmentacji sterty zarządzanej 289
Diagnozowanie wyjątków braku pamięci 298
Podsumowanie 316

Rozdział 6. Synchronizacja 317

Podstawy synchronizacji 317
Mechanizmy synchronizacji wątków 318
Zdarzenia 323
Muteksy 325
Semafory 326
Monitory 327
ReaderWriterLock(Slim) 328
Pula wątków 330
Wewnętrzne mechanizmy synchronizacji 331
Nagłówek obiektu 331
Bloki synchronizacji 333
Blokady lekkie 337
Scenariusze synchronizacji 341
Proste zakleszczenie 341
Wyjątki blokad porzuconych 349
Anulowanie wątków 354
Zawieszenie finalizatora 358
Podsumowanie 366

Rozdział 7. Interoperacyjność 369

P/Invoke 369
Interoperacyjność COM 376
Opakowanie wywoływane w czasie wykonywania 377
Diagnozowanie wywołań P/Invoke 382
Konwencje wywoływania 383
Delegaty 388
Diagnozowanie wycieków interoperacyjności 396
Diagnozowanie finalizacji interoperacyjności COM 402
Podsumowanie 411

Część III Techniki zaawansowane 413

Rozdział 8. Debugowanie poawaryjne 415

Pliki obrazu — podstawowe wiadomości 416
Generowanie zrzutów przy użyciu debugera 418
Generowanie plików zrzutu za pomocą narzędzia ADPlus 425
Diagnozowanie plików zrzutu 427
Warstwa dostępu do danych 428
Analizowanie plików zrzutu — nieobsłużone wyjątki .NET 432
Usługa raportowania błędów 433
Architektura usługi Windows Error Reporting 434
Podsumowanie 461

Rozdział 9. Narzędzia dodatkowe 463

PowerDbg 463
Instalowanie narzędzia PowerDbg 464
Polecenie Analyze-PowerDbgThreads 466
Polecenie Send-PowerDbgCommand 467
Rozszerzanie PowerDbg 469
Visual Studio 472
Integracja z SOS 472
Debugowanie platformy .NET na poziomie kodu źródłowego 476
Visual Studio 2010 479
Program profilujący CLR 484
Uruchamianie programu profilującego CLR 484
Widok podsumowania 486
Widoki histogramu 488
Widoki wykresów 489
WinDbg i polecenie cmdtree 491
Podsumowanie 493

Rozdział 10. CLR 4.0 495

Narzędzia 495
Debugging Tools for Windows 496
Pakiet .NET 4.0 Redistributable 496
SOS 496
Sterta zarządzana i odzyskiwanie pamięci 497
Rozszerzone narzędzia diagnostyczne 498
Odzyskiwanie pamięci w tle 503

Synchronizacja 504
Pula wątków i zadań 505
Monitor 505
Bariera 506
Klasa CountdownEvent 507
Klasa ManualResetEventSlim 507
Klasa SemaphoreSlim 507
Klasy SpinWait i SpinLock 507
Interoperacyjność 508
Debugowanie poawaryjne 509
Podsumowanie 510

Skorowidz 511

ROZDZIAŁ 8

DEBUGOWANIE POAWARYJNE

W poprzednich dwóch częściach książki poznaliśmy wiele znakomitych narzędzi wspomagających w pracy programistę diagnozującego aplikacje. Narzędzia te należy włączyć do procesu rozwoju oprogramowania, aby zapewnić jak najwyższy stopień jego niezawodności. Mimo iż wszystkie te narzędzia stanowią doskonałą pomoc w znajdowaniu błędów poprzez automatyzowanie procesu ich wyszukiwania, nie gwarantują, że gotowy produkt będzie absolutnie bezbłędny.

Po dostarczeniu programu do użytkowników problemy pojawią się na pewno i najprawdopodobniej stanie się to w najmniej oczekiwanym momencie — najczęściej podczas pracy użytkownika. W zależności od rodzaju błąd może całkowicie zniechęcić użytkownika do dalszego korzystania z programu lub tylko nieco uprzykrzyć mu życie. W obu tych przypadkach można się spodziewać telefonu od zdenerwowanego klienta żądającego wyjaśnień, czemu jego produkt nie działa tak, jak powinien. Jedną z możliwości w takiej sytuacji jest poproszenie go o zdalny dostęp do jego komputera. Mimo iż czasami jest to możliwe, użytkownicy najczęściej podchodzą do tego niechętnie i nie zgadzają się na takie rozwiązanie. Powodów, dla których klienci nie chcą umożliwiać dostępu do swoich komputerów, jest wiele. Poniżej przedstawiam kilka najczęstszych z nich:

- Zasady ustalone w firmie nie pozwalają na przyjmowanie połączeń przychodzących.
- Zdalne diagnozowanie wymaga podłączenia debugera do jednego lub większej liczby procesów, co oznacza przestój w pracy. Jeśli proces, który ma zostać zdiagnozowany, działa na ważnym serwerze, klient nie będzie chciał zaakceptować na nim przestoju.
- Diagnozowanie procesu poprzez tryb użytkownika lub jądra oznacza, że programiści mają pełny dostęp do stanu urządzenia, włącznie z zawartością jego pamięci. Niektórzy użytkownicy mogą obawiać się w takim przypadku naruszenia prywatności.

Jeśli klient odmówi dostępu zdalnego, a odtworzenie problemu na lokalnym komputerze jest niemożliwe, to czy w ogóle da się coś zrobić? Tak, należy wówczas zastosować techniki tzw. debugowania poawaryjnego (ang. *postmortem debugging*). Proces ten składa się z następujących etapów:

1. Wywołanie awarii.
2. Wykonanie zrzutu (obrazu) stanu systemu w chwili awarii (albo — w niektórych przypadkach — przed awarią i po niej).
3. Przesłanie zrzutu do przeanalizowania specjalistom.

W tym rozdziale zostały opisane różne sposoby wykonywania takich zrzutów — czasami nazywanych plikami obrazu — różne rodzaje tych obrazów oraz techniki ich analizowania. Ponadto poznamy znakomitą usługę gromadzenia plików obrazu o nazwie Windows Error Reporting.

Zaczniemy od podstawowych zagadnień związanych z plikami obrazu.

Pliki obrazu — podstawowe wiadomości

Jak już wiemy, plik obrazu stanowi reprezentację stanu określonego procesu. Pliki takie tworzy się przede wszystkim w celu umożliwienia diagnozowania aplikacji, gdy nie ma możliwości przeprowadzenia diagnostyki na działającym programie. Wygenerowany plik obrazu wysyła się do specjalisty, który analizuje jego zawartość, nie mając dostępu do urządzenia, na którym wystąpił problem. Praca ta polega na zapisaniu pliku na innym komputerze i analizowaniu go za pomocą narzędzi do diagnozowania poawaryjnego debugera. Jakie informacje zawiera taki plik obrazu? To zależy od sposobu jego utworzenia. Wyróżnia się dwa główne rodzaje plików obrazu:

- pełne zrzuty,
- zrzuty minimalne.

Pełny zrzut zawiera całą przestrzeń pamięci procesu, obraz pliku wykonywalnego, tabelę uchwytów i inne informacje wykorzystywane przez debugery. Przy generowaniu pełnego zrzutu nie ma możliwości wybierania, które informacje mają zostać zapisane. Plik taki można jednak przekonwertować na zrzut minimalny za pomocą debugera.

Minimalny plik zrzutu może zawierać różne informacje, których wybór jest dokonywany przez moduł generujący. Plik taki może zawierać zarówno informacje o konkretnym wątku, jak i pełny opis obrazowanego procesu.

Może to zabrzmi dziwnie, ale największy możliwy minimalny plik zrzutu może zawierać więcej informacji niż zrzut pełny. Dlatego w tej części rozdziału skoncentruję się na zrzutach minimalnych.

Narzędzia, za pomocą których można generować takie pliki, zostały zwięźle opisane w tabeli 8.1.

Tabela 8.1. Narzędzia do generowania zrzutów

Nazwa	Opis
Debugery Windows	Debugery systemu Windows umożliwiają tworzenie zrzutów o różnych rozmiarach oraz pozwalają kontrolować cały proces generowania pliku
ADPlus	ADPlus to składnik pakietu Debugging Tools for Windows. Narzędzie to może działać jako monitor systemu wykonujący zrzut w chwili wystąpienia awarii lub zawieszenia. Ponadto program ten ma opcję powiadamiania o wystąpieniu awarii
Windows Error Reporting	Windows Error Reporting to usługa firmy Microsoft umożliwiająca zarejestrowanie się w witrynie do raportowania błędów. Gdy w którejś z aplikacji danego użytkownika wystąpi awaria, z komputera, w którym ona działa, wysyłany jest raport do witryny Windows Error Reporting (WER). Następnie użytkownik może pobrać ten raport z witryny WER wraz z plikiem zrzutu w celu jego przeanalizowania

W tym podrozdziale nauczymy się generować pliki obrazu za pomocą debugerów Windows i narzędzia ADPlus. Usługa Windows Error Reporting zostanie opisana nieco dalej.

W celu zilustrowania procesu tworzenia plików obrazu wykorzystamy przykładową prostą aplikację alokującą pamięć na stercie, zapisującą w niej dane i następnie ulegającą awarii. Kod źródłowy tej aplikacji znajduje się na listingu 8.1.

Listing 8.1. Prosty przykład aplikacji, która ulega awarii

```
using System;
using System.Text;
using System.Runtime.InteropServices;

namespace Advanced.NET.Debugging.Chapter8
{
    class SimpleExc
    {
```

```
        static void Main(string[] args)
        {
            SimpleExc s = new SimpleExc();
            s.Run();
        }

        public void Run()
        {
            Console.WriteLine("Naciśnij dowolny klawisz, aby rozpocząć");
            Console.ReadKey();

            ProcessData(null);
        }

        public void ProcessData(string data)
        {
            if (data == null)
            {
                throw new ArgumentException("Argument NULL");
            }
            string s = "Witaj: " + data;
        }

    }
}
```

Kod źródłowy i plik binarny powyższego programu znajdują się w następujących lokalizacjach:

- Kod źródłowy: *C:\ADND\Chapter8\SimpleExc*
- Plik binarny: *C:\ADNDBin\08SimpleExc.exe*

Powód awarii tej aplikacji powinien być dość oczywisty. Wywołanie funkcji `ProcessData` powoduje zgłoszenie wyjątku `ArgumentException`, ponieważ została jej przekazana wartość `null`. Na początek wygenerujemy zrzut za pomocą debugera.

Generowanie zrzutów przy użyciu debugera

Uruchom aplikację, której kod źródłowy znajduje się na listingu 8.1, i pozwól jej działać do wystąpienia wyjątku.

```
…
…
…
ModLoad: 77bb0000 77bb6000   C:\Windows\system32\NSI.dll
ModLoad: 79060000 790b6000   C:\Windows\Microsoft.NET\Framework\v2.0.50727
\mscorjit.dll
(1860.958): CLR exception - code e0434f4d (first chance)
```

```
(1860.958): CLR exception - code e0434f4d (!!! second chance !!!)
eax=0020eaec ebx=e0434f4d ecx=00000001 edx=00000000 esi=0020eb74 edi=00416bd0
eip=767142eb esp=0020eaec ebp=0020eb3c iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
KERNEL32!RaiseException+0x58:
767142eb c9              leave
0:000> .loadby sos mscorwks
0:000> !ClrStack
OS Thread Id: 0x958 (0)
ESP       EIP
0020ebc4 767142eb [HelperMethodFrame: 0020ebc4]
0020ec68 00e10177 Advanced.NET.Debugging.Chapter8.SimpleExc.ProcessData
(System.String)
0020ec80 00e1010c Advanced.NET.Debugging.Chapter8.SimpleExc.Run()
0020ec88 00e100a7 Advanced.NET.Debugging.Chapter8.SimpleExc.Main
(System.String[])
0020eeac 79e7c74b [GCFrame: 0020eeac]
```

W tym momencie wygenerujemy zrzut do przeanalizowania w trybie poawaryjnym. Najważniejszą kwestią do rozwiązania w przypadku generowania zrzutu jest podjęcie decyzji, ile informacji zapisać. Ogólna zasada jest taka, że im więcej danych znajduje się w pliku obrazu, tym więcej informacji będziemy mieli do wykorzystania w pracy. Najważniejszym czynnikiem ograniczającym jest oczywiście rozmiar pliku zrzutu. W niektórych przypadkach, np. na serwerach o ostrych zasadach bezpieczeństwa, utworzenie gigantycznego zrzutu jest niemożliwe i trzeba zadowolić się okrojoną ilością informacji.

Do utworzenia pliku obrazu służy polecenie `.dump`. Opcja `/m` tego polecenia oznacza, że ma zostać utworzony zrzut minimalny. Opis wszystkich opcji tego polecenia znajduje się w tabeli 8.2.

Oprócz opcji sterujących wykonywaniem zrzutu należy podać jeszcze nazwę pliku, w którym zrzut zostanie zapisany. Jeśli nie zostanie określona pełna ścieżka katalogu, w którym ma zostać zapisany ten plik, zostanie on zapisany w katalogu uruchomieniowym debugera. Poniżej znajduje się przykładowe polecenie wykonujące pełny zrzut pamięci i zapisujące plik w wybranym katalogu.

```
.dump /mf c:\08dumpfile.dmp
```

Uruchomimy polecenie `.dump` na naszej uszkodzonej aplikacji:

```
0:000> .dump /mf 08dumpfile.dmp
Creating dumpfile.dmp - mini user dump
Dump successfully written
```

Tabela 8.2. Opcje polecenia .dump

Opcja	Opis
a	Generuje kompletny zrzut minimalny z włączonymi wszystkimi opcjami. Zrzut taki zawiera pełne dane dotyczące pamięci, uchwytów, modułu, podstawowe dane o pamięci oraz informacje na temat wątku. Równoznaczne z `/mfFhut`
f	Generuje zrzut minimalny zawierający wszystkie dostępne i zastrzeżone strony procesu
F	Generuje zrzut minimalny zawierający podstawowe informacje o pamięci potrzebne debugerowi do odtworzenia całej przestrzeni adresowej pamięci wirtualnej
h	Generuje zrzut minimalny zawierający informacje o uchwytach
u	Generuje zrzut minimalny zawierający informacje o niezaładowanych modułach. Opcja dostępna tylko w systemie Windows Server 2003
t	Generuje zrzut minimalny zawierający informacje czasowe dotyczące wątków. Informacje te to m.in. czas utworzenia oraz czas pracy w trybach użytkownika i jądra
i	Generuje zrzut minimalny zawierający informacje o pamięci pomocniczej. Pamięć pomocnicza to pamięć (oraz niewielki obszar ją otaczający) wskazywana przez wskaźnik stosu lub pamięć rezerwową
p	Generuje zrzut minimalny zawierający bloki środowisk procesu i wątku
w	Generuje zrzut minimalny zawierający wszystkie zastrzeżone prywatne strony do zapisu i odczytu
d	Generuje zrzut minimalny zawierający wszystkie segmenty danych obrazu
c	Generuje zrzut minimalny zawierający segmenty kodu obrazu
r	Generuje zrzut minimalny odpowiedni dla środowisk o zaostrzonych wymogach dotyczących prywatności. Opcja ta czyści (zastępuje zerami) wszystkie informacje niepotrzebne do odtworzenia stosu (włącznie ze zmiennymi lokalnymi)
R	Generuje zrzut minimalny odpowiedni dla środowisk o zaostrzonych wymogach dotyczących prywatności. Ta opcja usuwa pełne ścieżki modułów, uniemożliwiając w ten sposób odtworzenie struktury katalogów

Wygenerowany plik zrzutu powinien mieć rozmiar około 64 MB. Plik ten należy załadować w innej instancji debugera przy użyciu przełącznika `-z`. Aby załadować wygenerowany przez nas plik, należy napisać następujące polecenie:

```
c:\>ntsd –z 08dumpfile.dmp
```

Po wczytaniu pliku debuger wyświetli następujące dane:

```
…
…
…
Loading Dump File [c:\08dumpfile.dmp]
User Mini Dump File with Full Memory: Only application data is available
Executable search path is:
Windows Server 2008 Version 6001 (Service Pack 1) MP (2 procs) Free x86 compatible
Product: WinNt, suite: SingleUserTS
Debug session time: Mon Mar  2 06:25:10.000 2009 (GMT-8)
System Uptime: 5 days 7:44:57.406
Process Uptime: 0 days 0:02:39.000
.........................
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for
ntdll.dll -
This dump file has an exception of interest stored in it.
The stored exception information can be accessed via .ecxr.
(1860.958): CLR exception - code e0434f4d (first/second chance not available)
eax=0020eaec ebx=e0434f4d ecx=00000001 edx=00000000 esi=0020eb74 edi=00416bd0
eip=767142eb esp=0020eaec ebp=0020eb3c iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
kernel32!RaiseException+0x58:
767142eb c9              leave
```

W górnej części powyższych danych znajdują się podstawowe informacje na temat wczytanego pliku zrzutu. Wśród nich jest lokalizacja pliku, jego typ oraz opis dostępnych danych. Kolejny interesujący nas fragment znajduje się bliżej końca. Jest to opis powodu wystąpienia awarii (wyjątek CLR). Mając ten plik, można zdiagnozować problem na dowolnym komputerze — bez dostępu do urządzenia, na którym wystąpiła awaria. Szczegółowy opis technik wykonywania analizy poawaryjnej znajduje się nieco dalej.

Jedną z wad techniki wykonywania zrzutów za pomocą debugera jest konieczność podłączenia go do ulegającego awarii procesu. Może się wydawać, że to niewielki problem, ale wyobraź sobie, że błąd występuje tylko raz na jakiś czas i nie udało się trafić z podłączeniem debugera właśnie w tym czasie. Dobrze by było, gdybyśmy mieli możliwość nakazania systemowi Windows wykonania zrzutu, gdy wystąpi awaria procesu. Taka możliwość jest i technikę tę potocznie nazywa się nastawieniem debugera poawaryjnego. Domyślnym programem tego typu używanym przez system Windows jest Dr Watson (wycofywany w Windows Vista i nowszych wersjach systemu na rzecz nowszej technologii). Dr Watson generuje plik zrzutu, gdy nastąpi awaria procesu, i umożliwia wysłanie tego pliku do firmy Microsoft do analizy. Wykorzystywany debuger poawaryjny można jednak zmienić przy użyciu poleceń wiersza poleceń przedstawionych w tabeli 8.3.

Tabela 8.3. Ustawienia debugera poawaryjnego

Polecenie	Wartość rejestru Aedebug\Debugger	Opis
`WinDbg –I`	`windbg.exe -p %ld –e %ld -g`	Ustawia WinDbg jako debuger poawaryjny. Należy pamiętać, że litera `-I` musi być wielka.
`cdb –iae`	`cdb.exe -p %ld -e %ld –g`	Ustawia cdb jako debuger poawaryjny.
`ntsd –iae`	`ntsd.exe –p %ld -e %ld –g`	Ustawia ntsd jako debuger poawaryjny.
`drwtsn32 –i`	`drwtsn32 -p %ld -e %ld –g`	Ustawia program Dr Watson jako debuger poawaryjny.

Generowanie pliku zrzutu

W systemie Windows Vista wprowadzono ważną zmianę w stosunku do starszych wersji tego systemu, dotyczącą sposobu zapisywania plików zrzutu w lokalnej pamięci przez technologię raportowania błędów. W starszych wersjach systemu Windows wygenerowane pliki były domyślnie zapisywane w komputerze przez program Dr Watson. Dostęp do tych plików miał każdy, kto chciał przeprowadzić diagnozę określonego pliku. W systemie Windows Vista Dr Watson przeszedł na emeryturę, a jego miejsce zajął bardziej niezawodny mechanizm raportowania błędów. Ten nowy system przy domyślnych ustawieniach nie zapisuje plików zrzutu w pamięci lokalnej. Aby zmienić to domyślne ustawienie, należy ustawić wartość rejestru `ForceQueue` na `1`. Powoduje to kolejkowanie wszystkich plików zrzutu w pamięci lokalnej przed ich wysłaniem do firmy Microsoft. Ścieżka w rejestrze do tej wartości jest następująca:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Windows Error Reporting
```

Po ustawieniu wartości `ForceQueue` na `1` wszystkie pliki zrzutu będą zapisywane w następującej lokalizacji:

Procesy działające w kontekście systemowym i podniesionym:

```
%ALLUSERSPROFILE %\Microsoft\Windows\WER\[ReportQueue|ReportArchive]
```

Pozostałe procesy:

```
%LOCALAPPDATA%\Microsoft\Windows\WER\[ReportQueue|ReportArchive]
```

Co tak naprawdę dzieje się w czasie wykonywania poleceń z tabeli 8.3? Nic wielkiego. Polecenia te zmieniają tylko wartości niektórych kluczy rejestru, które są sprawdzane przez system Windows przy wykrywaniu awarii procesów. Poniżej znajduje się ścieżka rejestru wykorzystywana do nastawiania debugera poawaryjnego:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\AeDebug
```

Klucz `Aedebug` działa znakomicie w przypadku diagnozowania aplikacji rodzimych. Natomiast do sterowania procesem diagnozowania poawaryjnego aplikacji zarządzanych służą dwie inne wartości — `DbgManagedDebugger` i `DbgJITDebugLaunchSetting`. Znajdują się one w następującym kluczu:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework
```

Wartość DbgManagedDebugger

Jeśli debugowanie poawaryjne zostanie włączone poprzez wartość `DbgJIT ↪DebugLaunchSettings`, wartość rejestru `DbgManagedDebugger` określa, który debuger ma zostać uruchomiony w przypadku wystąpienia nieobsłużonego wyjątku. Aby np. domyślnym debugerem uruchamianym w przypadkach wystąpienia nieobsłużonych wyjątków był ntsd, wartość rejestru `DbgManagedDebugger` należy ustawić następująco:

```
c:\program files\debugging tools for windows (x86)\ntsd.exe -p %ld
```

Debuger określony w wartości `DbgManagedDebugger` nie musi zostać uruchomiony natychmiast po wystąpieniu nieobsłużonego wyjątku. Zamiast tego zostanie wyświetlone okno dialogowe, w którym można wybrać uruchomienie debugera lub zamknięcie aplikacji.

Jedno z najczęściej zadawanych pytań na temat debugowania poawaryjnego brzmi: jak wymusić automatyczne generowanie zrzutu w odpowiedzi na wystąpienie awarii? Aby to zrobić, należy ustawić wartość rejestru `DbgManagedDebugger` w następujący sposób:

```
ntsd –pv –p %ld –c ".dump /u /ma <ścieżka do pliku zrzutu>; .kill; qd
```

Powyższe ustawienie oznacza, że jeśli wystąpi awaria, ma zostać włączony debuger ntsd oraz uruchomione polecenie wykonania zrzutu. Następnie sesja ma zostać zamknięta poprzez odłączenie debugera.

Do precyzyjnego określania działań w przypadku wystąpienia nieobsłużonego wyjątku służy wartość rejestru `DbgJITDebugLaunchSetting` opisana w następnym podrozdziale.

Wartość DbgJITDebugLaunchSetting

Wartość rejestru `DbgJITDebugLaunchSetting` służy do określania działań w przypadku wystąpienia nieobsłużonego wyjątku. Jeśli wartość ta zostanie ustawiona na `0`, w przypadku awarii będzie wyświetlane okno dialogowe, w którym użytkownik będzie mógł zdecydować, co robić. Należy pamiętać, że okno to jest wyświetlane tylko dla procesów interaktywnych (np. usług), pozostałe procesy będą po prostu zamykane. Przykładowe okno dialogowe przedstawiono na rysunku 8.1.



Rysunek 8.1. Przykładowe okno dialogowe wyświetlone po awarii aplikacji zarządzanej

Powyższe okno poawaryjne informuje o wystąpieniu problemu w programie o nazwie *08SimpleExc.exe*. Użytkownik może się w nim zdecydować na debugowanie programu (przycisk *Debuguj*) lub jego zamknięcie (przycisk *Zamknij program*). Jeśli zostanie naciśnięty przycisk *Debuguj*, system sprawdzi ustawienie wartości `DbgJITDebugLaunchSetting` i uruchomi wyznaczony w niej debuger.

Jeśli wartość `DbgJITDebugLaunchSetting` zostanie ustawiona na `1`, aplikacja, która uległa awarii, zostanie zamknięta i zostanie zwrócony zrzut stosu wywołań.

Jeśli wartość ta będzie ustawiona na `2`, zostanie uruchomiony debuger określony w wartości `DbgManagedDebugger` bez wyświetlania żadnych okien dialogowych.

Natomiast ustawienie wartości `DbgJITDebugLaunchSetting` na `16` spowoduje wyświetlanie dla procesów interaktywnych okna opisanego powyżej, a dla pozostałych procesów — uruchomienie debugera określonego w wartości `DbgManagedDebugger`.

Mimo iż funkcje debugerów są wystarczające do wygenerowania odpowiedniego pliku zrzutu, dostępne jest jeszcze jedno narzędzie, które służy do generowania tych plików — ma ono nazwę ADPlus.

Generowanie plików zrzutu za pomocą narzędzia ADPlus

ADPlus to narzędzie do monitorowania awarii procesów i automatyzowania generowania plików zrzutu z możliwością powiadamiania użytkownika lub komputera o zaistniałej sytuacji. ADPlus to skrypt sterowany z wiersza poleceń. Firma Microsoft zaleca uruchamianie go przy użyciu interpretera *cscript.exe*. Oprócz opcji wiersza poleceń ADPlus można również konfigurować za pomocą plików konfiguracyjnych, które umożliwiają bardziej precyzyjne sterowanie działaniem programu.

ADPlus może działać w jednym z dwóch trybów:

- Tryb zawieszeń — służy do diagnozowania procesów wykazujących oznaki zawieszenia (np. nie robią żadnego postępu albo wykorzystują 100% mocy procesora). Aby monitorować zawieszone procesy, ADPlus musi zostać uruchomiony po ich zawieszeniu.
- Tryb awaryjny — służy do diagnozowania procesów wykazujących oznaki wystąpienia awarii. ADPlus musi zostać uruchomiony przed procesem, w którym nastąpiła awaria.

Generowanie plików zrzutu za pomocą narzędzia ADPlus zilustruję na przykładzie awarii programu *08SimpleExc.exe* (tryb awaryjny). Uruchom program *SimpleExc.exe*:

```
C:\ADNDBin\08SimpleExc.exe
```

Przed naciśnięciem klawisza w celu kontynuowania działania aplikacji wpisz następujące polecenie:

```
C:\>adplus.vbs -crash -pn 08SimpleExc.exe -y
 SRV*c:\Symbols*http://msdl.microsoft.com/download/symbols
```

Przełącznik `-crash` przestawia narzędzie ADPlus na tryb awaryjny, `-pn` służy do określania nazwy procesu, który ma być monitorowany, a `-y` — do ustawiania ścieżki symboli, która ma być wykorzystywana podczas działania ADPlus. Wielką zaletą przełącznika `-pn` jest możliwość monitorowania dowolnej liczby instancji dowolnego procesu.

Po zakończeniu działania programu ADPlus zapisuje wynik swojego działania w pliku w katalogu instalacyjnym debugerów systemu Windows. Nazwa tego katalogu ma następującą strukturę:

```
<runtype>_Mode__Date_<data uruchomienia>__Time_<czas działania>
```

Np. w tym przypadku po zakończeniu działania ADPlus utworzył następujący katalog:

```
c:\Program Files\Debugging Tools for Windows (x86)\
↪Crash_Mode__Date_03-02-2009__Time_08-31-43AM
```

Domyślną ścieżkę można zmienić za pomocą przełącznika `-o`.

W powyższym katalogu zostało zapisanych kilka plików, ale najważniejsze z nich mają rozszerzenie *.dmp*, ponieważ zawierają wszystkie informacje zrzutowe. Jak widać, w katalogu znajduje się kilka plików zrzutu. Dlaczego jest więcej niż jeden plik dla jednej awarii? Program ADPlus automatyzuje proces gromadzenia plików zrzutu i dlatego generuje kolejne pliki w przypadku wystąpienia z góry określonych sytuacji. Powód wygenerowania każdego pliku można odczytać z jego nazwy. W naszym przykładzie program ADPlus wygenerował następujące pliki zrzutu:

```
PID-4448__08SIMPLEEXC.EXE__1st_chance_Process_Shut_
Down__full_1e20_2009-03-02_08-32-17-440_1160.dmp

PID-4448__08SIMPLEEXC.EXE__2nd_chance_NET_CLR__full_
1e20_2009-03-02_08-32-08-384_1160.dmp
```

ADPlus wygenerował pełny plik zrzutu po wystąpieniu pierwszego zdarzenia zamknięcia procesu. Następnie wygenerował kolejny pełny zrzut po wystąpieniu wyjątku .NET. Czy potrzebne nam są te wszystkie pliki w naszej sytuacji? Nie. Dla nas najbardziej interesujący jest drugi plik — wygenerowany po wystąpieniu wyjątku .NET. Oczywiście w niektórych sytuacjach takie okresowe generowanie plików zrzutu jest bardzo pomocne, ponieważ można odtworzyć przebieg systematycznego psucia się procesu. ADPlus umożliwia również ustawianie częstotliwości zapisywania informacji oraz warunków, w jakich ma to być robione, co w istocie oznacza, że pozwala na sterowanie działaniem debugera za pomocą skryptów. Szczegółowe informacje na temat pisania skryptów w ADPlus znajdują się w dokumentacji programu. Należy jednak koniecznie pamiętać, że ADPlus nie wykonuje za pomocą swoich skryptów żadnych czarodziejskich sztuczek. Jest to tylko wygodny sposób na wprowadzanie dyrektyw debugera, które są następnie tłumaczone na zwykłe zautomatyzowane polecenia. Aby zobaczyć, jak ta wygodna konfiguracja przekłada się na polecenia debugera, można zajrzeć do folderu *CDBScripts* znajdującego się w tym samym katalogu co pliki zrzutu. W naszym przykładzie w folderze tym znajduje się plik o nazwie *PID-4448__08SimpleExc.exe.cfg* zawierający wszystkie polecenia debugera użyte w sesji diagnostycznej.

Ostatnią ważną rzeczą, którą trzeba wiedzieć na temat programu ADPlus, jest sposób określania typu pliku zrzutu generowanego w odpowiedzi na wystąpienie awarii. Służą do tego cztery przełączniki wiersza poleceń:

- `-FullOnFirst` — generuje pełny zrzut przy pierwszym wystąpieniu wyjątku.
- `-MiniOnSecond` — generuje zrzut minimalny przy drugim wystąpieniu wyjątku.
- `-NoDumpOnFirst` — blokuje generowanie zrzutu minimalnego przy pierwszym wystąpieniu wyjątku. Opcja ta jest przydatna w przypadku aplikacji, które dobrze obsługują pierwsze wystąpienie wyjątku.
- `-NoDumpOnSecond` — blokuje generowanie zrzutu minimalnego przy drugim wystąpieniu wyjątku.

ADPlus to wygodne, elastyczne i bardzo pomocne narzędzie do monitorowania procesów i gromadzenia informacji o występujących w nich awariach. W tym podrozdziale zostały opisane podstawowe wiadomości o tym programie. Warto jednak poświęcić trochę czasu na zapoznanie się z innymi jego możliwościami, jak np. skrypty i techniki definiowania niestandardowych procedur obsługi wyjątków, które umożliwiają generowanie zrzutów w odpowiedzi na wystąpienie niestandardowych wyjątków.

Znając dwa najczęściej stosowane sposoby tworzenia plików zrzutu, możemy przystąpić do ich praktycznego wykorzystania i zapoznać się z procesem rozwiązywania problemów przy ich użyciu.

Diagnozowanie plików zrzutu

Mamy już pliki zrzutu i powierzono nam zadanie znalezienia przyczyny problemów z procesem. Co konkretnie możemy zrobić z dostarczonymi nam plikami? Czy możemy wykonać z nich zrzut pamięci, obejrzeć uchwyty albo wykonać kod krok po kroku? Przypominam, że plik zrzutu jest tylko statycznym obrazem stanu procesu w określonym momencie. Dlatego ustawianie punktów wstrzymania i wykonywanie kodu krok po kroku jest niemożliwe. Korzystanie z plików zrzutu najłatwiej porównać z ręcznym debugowaniem. Mam na myśli to, że możemy tylko oglądać stan aplikacji i na tej podstawie musimy opracować teorię na temat tego, jaki kod doprowadził do zaistniałej sytuacji. Nietrudno się domyślić, że odtwarzanie sekwencji wykonywania kodu na podstawie statycznej analizy jest znacznie trudniejsze niż analizowanie programu w czasie działania. Niemniej jednak w statycznym trybie nadal można korzystać z wielu poleceń, które przetwarzają dane

na bardziej zrozumiałą postać, i w większości przypadków, jeśli tylko okaże się wystarczająco dużo cierpliwości, udaje się odnaleźć przyczynę problemów.

Przed przejściem do analizowania wygenerowanych plików zrzutu musimy poznać dwa bardzo ważne pojęcia — pliki symboli i warstwa dostępu do danych. Ponieważ pliki zrzutu nie zawierają symboli, symbole te muszą być dostępne z jakiegoś innego źródła w czasie sesji diagnostycznej. We wcześniejszych częściach tej książki poznaliśmy najważniejsze polecenia związane z symbolami. Drugą ważną rzeczą jest tzw. warstwa dostępu do danych CLR (ang. *data access layer* — DAC), z której bardzo często korzysta rozszerzenie SOS debugera w celu dostarczenia wszystkich potrzebnych w sesji informacji.

Warstwa dostępu do danych

Podczas diagnozowania aplikacji rodzimych wiele informacji można zdobyć poprzez przeglądanie surowej pamięci. Natomiast w kodzie zarządzanym rozszerzenie SOS zwracane przez siebie dane tworzy na podstawie informacji uzyskanych w dużym stopniu od CLR. Rozszerzenie to, odbierając surowe dane, musi je poprawnie przetworzyć, korzystając z pomocy w postaci wywołań do CLR (tzn. wykonywania kodu CLR). Za związane z tym funkcje w CLR odpowiada moduł o nazwie warstwa dostępu do danych (DAC), którego implementacja znajduje się w pliku *mscordacwks.dll*. Ponieważ system CLR jest cały czas ulepszany, jego warstwa DAC również zmienia się z każdą wersją (włącznie z poprawkami). Można to łatwo sprawdzić, zaglądając do folderu instalacyjnego każdej wersji .NET zainstalowanej w komputerze. Np. w moim komputerze plik *mscordacwks.dll* znajduje się w następującym folderze:

```
c:\Windows\Microsoft.NET\Framework\v2.0.50727
```

Natomiast w komputerach z zainstalowanym programem Visual Studio 2010 CTP plik ten znajduje się w następujących dwóch folderach (co oznacza, że z CLR 4.0 został dostarczony nowy plik *mscordacwks.dll*):

```
c:\Windows\Microsoft.NET\Framework\v2.0.50727
c:\Windows\Microsoft.NET\Framework\v4.0.11001
```

Znajomość lokalizacji tego pliku jest niezbędna, ponieważ debuger zażąda podania mu tej informacji. W czasie debugowania na żywo ten problem nie występuje, ponieważ rozszerzenie SOS znajduje ten plik, szukając go

w tym samym katalogu, w którym znajduje się wersja środowiska wykonawczego, w jakim jest przeprowadzana sesja diagnostyczna. Natomiast w czasie debugowania poawaryjnego wersja CLR użyta w aplikacji może być inna niż wersja dostępna na komputerze, na którym jest wczytywany i diagnozowany plik zrzutu. Przypominam zatem, że rozszerzenie SOS debugera odwołuje się do pliku *mscordacwks.dll*, który wykonuje kod CLR, oraz że załadowanie odpowiedniej wersji tego pliku ma kluczowe znaczenie. Ponieważ posiadanie odpowiedniej wersji tego pliku jest tak ważne, firma Microsoft udostępnia wszystkie istniejące jego wersje na swoim serwerze symboli publicznych. Jeśli debuger zostanie skierowany do tego serwera (za pomocą polecenia `symfix` lub innego podobnego), sam znajdzie potrzebny mu plik. Zdarzają się jednak sytuacje, w których trzeba własnoręcznie określić lokalizację tego pliku. Może się to np. zdarzyć, gdy określonej wersji pliku nie ma na serwerze symboli (rzadko) lub gdy plik ten nie został zapisany w tym samym miejscu, w którym znajdował się na komputerze, na którym został wykonany zrzut. W takich sytuacjach należy skorzystać z polecenia `cordll` służącego do określania sposobu ładowania pliku *mscordacwks.dll*. Polecenie to jest bezcenne, gdy wystąpią problemy z dopasowaniem wersji tego pliku. W tabeli 8.4 znajduje się opis przełączników polecenia `cordll`.

Skąd wiadomo, czy należy się zainteresować poleceniem `cordll`? Zazwyczaj jeśli wystąpi niezgodność wersji pliku *mscordacwks.dll*, rozszerzenie SOS zwraca następującą informację o błędzie (lub coś podobnego), gdy nie może wykonać jakiegoś polecenia:

```
Failed to load data access DLL, 0x80004005
Verify that 1) you have a recent build of the debugger (6.2.14 or newer)
2) the file mscordacwks.dll that matches your version of mscorwks.dll is
in the version directory
3) or, if you are debugging a dump file, verify that the file
mscordacwks___.dll is on your symbol path.
4) you are debugging on the same architecture as the dump file.
For example, an IA64 dump file must be debugged on an IA64
machine.

You can also run the debugger command .cordll to control the debugger's
load of mscordacwks.dll. .cordll -ve -u -l will do a verbose reload.
If that succeeds, the SOS command should work on retry.

If you are debugging a minidump, you need to make sure that your
executable path is pointing to mscorwks.dll as well.
```

Przyjrzymy się wszystkim propozycjom zgłoszonym powyżej przez debuger. Pierwsza z nich jest prosta, ponieważ debuger prosi nas, abyśmy się upewnili, czy korzystamy z najnowszej wersji debugerów (6.2.14 lub nowszej).

Tabela 8.4. Przełączniki polecenia cordll

Przełącznik	Opis
`-l`	Ładuje moduły debugujące, szukając pliku DLL na domyślnych ścieżkach
`-u`	Usuwa moduły debugujące z pamięci
`-e`	Włącza debugowanie CLR
`-d`	Wyłącza debugowanie CLR
`-D`	Wyłącza debugowanie CLR i usuwa moduły debugujące
`-N`	Ponownie ładuje moduły debugujące
`-lp`	Określa ścieżkę do katalogu zawierającego moduły debugujące
`-Se`	Umożliwia używanie wersji o krótkiej nazwie modułu debugującego — *mscordacwks.dll*
`-sd`	Wyłącza możliwość używania wersji o krótkiej nazwie modułu debugującego — *mscordacwks.dll*. Jeśli ten przełącznik zostanie użyty, debuger spodziewa się, że nazwa modułu debugującego będzie miała następujący format: *mscordacwks_<spec>.dll*, gdzie *<spec>* to *<architektura>_<architektura>_<wersja pliku>*
`-ve`	Włącza tryb wzbogacony. Tryb ten jest przydatny przy pracy z nieprawidłowymi dopasowaniami, ponieważ dostarcza informacji na temat tego, jak debuger próbuje załadować moduły debugujące
`-vd`	Wyłącza tryb wzbogacony

Druga podpowiedź jest równie prosta. Debuger podpowiada nam, abyśmy się upewnili, że nasza wersja pliku *mscordacwks.dll* jest taka sama jak wersja, która jest ładowana. Jak pamiętamy, plik *mscordacwks.dll* powinien znajdować się w tym samym folderze co plik *mscorwks.dll*. Trzecia sugestia jest najciekawsza ze wszystkich. W tym przypadku debuger informuje, że jeśli jest wykonywane diagnozowanie pliku zrzutu, należy sprawdzić, czy plik *mscordacwks__.dll* znajduje się na ścieżce symboli. Co to za plik? W tabeli 8.4 znajduje się polecenie `-sd` włączające długą nazwę pliku *mscordacwks.dll*. Długa nazwa to nazwa z dodaną informacją o architekturze i numerem kompilacji pliku DLL. Dzięki tym informacjom można zmodyfikować ścieżkę symboli, aby wskazywała właściwy plik DLL, oraz ponownie wykonać polecenie `cordll`, aby ponownie załadować plik *mscordacwks.dll*. Jeśli np. do wykonania pliku zrzutu wykorzystano wersję 1.1.1.0 pliku *mscordacwks.dll* oraz dokonano tego w komputerze x86, można

nazwę pliku *mscordacwks.dll* zmienić na *mscordacwks_x86_x86_1.1.1.0.dll*, ustawić ścieżkę symboli debugerów na jego lokalizację i ponownie załadować moduły debugujące za pomocą polecenia `cordll`.

```
0:008> .sympath+ <path to renamed module>
0:008> .cordll -ve -u -l
```

Czwarta sugestia dotyczy upewnienia się, że architektura komputera, na którym przeprowadzana jest diagnostyka, zgadza się z architekturą komputera, na którym został wykonany zrzut. Ponieważ debuger wykonuje swoją pracę poprzez wykonywanie kodu w DAC, zalecane jest, aby debugować pliki zrzutu w architekturze o takiej samej liczbie bitów jak ta, w której utworzono plik. Jeśli np. użyto by 64-bitowego debugera do wygenerowania pliku zrzutu procesu 32-bitowego działającego w systemie 64-bitowym przy użyciu nakładki WOW64, nie udałoby się przeprowadzić diagnostyki tego pliku.

W ostatnim wierszu powyższych danych znajduje się pytanie, czy na ścieżce plików wykonywalnych znajduje się plik *mscorwks.dll*. Zawartość ścieżki plików wykonywalnych można zmienić w debugerze za pomocą polecenia `exepath` (lub `exepath+` w przypadku dodawania ścieżek). Np. w przypadku diagnozowania pliku zrzutu, przy wykonywaniu którego plik *mscorwks.dll* znajdował się w katalogu *c:\windows\microsoft.net\framework\v2.0.50727*, można zastosować poniższe polecenie ustawiające poprawnie ścieżkę plików wykonywalnych (następnie należy użyć polecenia `.reload`, aby nowa ścieżka została wczytana przez debuger):

```
0:008> .exepath+ c:\windows\microsoft.net\framework\v2.0.50727
Executable image search path is: c:\windows\microsoft.net\framework\v2.0.50727
0:008> .reload
```

We wszystkich dotychczas opisanych sytuacjach przyjęto założenie, że plik *mscordacwks.dll* jest dostępny w jednym lub innym miejscu (na serwerze symboli publicznych lub na komputerze lokalnym). Jeśli nie można znaleźć wersji pliku DLL, która została użyta przy generowaniu pliku zrzutu, najlepiej jest poprosić osobę, która to zrobiła, o przesłanie odpowiedniej wersji pliku *mscordacwks.dll*. Po otrzymaniu pliku można go załadować, stosując techniki opisane powyżej.

Określenie właściwej wersji pliku *mscordacwks.dll* bywa czasami trudne i uzyskanie biegłości w jej znajdowaniu wymaga trochę czasu i przeprowadzenia kilku prób. Po załadowaniu tego pliku rozszerzenie SOS będzie w pełni funkcjonalne i będzie można ponowić diagnozowanie.

Analizowanie plików zrzutu — nieobsłużone wyjątki .NET

W poprzednim podrozdziale wygenerowaliśmy plik zrzutu awaryjnej aplikacji i naszym zadaniem jest znalezienie przyczyny problemów, mając do dyspozycji tylko ten plik.

Aby móc wykorzystać ten plik, musimy o tym poinformować debuger. Do tego służy przełącznik `-z`:

```
C:> ntsd –z C:\08dumpfile.dmp
```

Pierwszą ważną informacją zwróconą przez debuger jest wyjątek CLR:

```
This dump file has an exception of interest stored in it.
The stored exception information can be accessed via .ecxr.
(2dc4.2a08): CLR exception - code e0434f4d (first/second chance not available)
eax=0024ef20 ebx=e0434f4d ecx=00000001 edx=00000000 esi=0024efa8 edi=002c43e8
eip=767142eb esp=0024ef20 ebp=0024ef70 iopl=0         nv up ei pl nz ac po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000212
kernel32!RaiseException+0x58:
767142eb c9              leave
```

Z powyższych danych wynika, że badany przez nas plik został wygenerowany z powodu wystąpienia wyjątku CLR. Kolejnym krokiem będzie dokładne przyjrzenie się temu wyjątkowi.

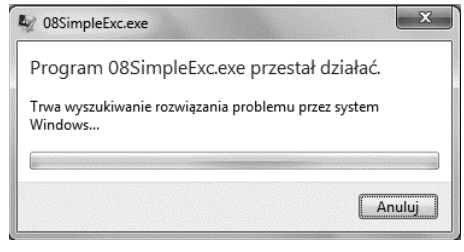
```
0:000> kb
ChildEBP RetAddr  Args to Child
0024ef70 79f071ac e0434f4d 00000001 00000001 kernel32!RaiseException+0x58
0024efd0 79f0a629 01b66c20 00000000 00000000
mscorwks!RaiseTheExceptionInternalOnly+0x2a8
0024f094 01630197 01b658d0 0024f0e0 0024f0fc mscorwks!JIT_Throw+0xfc
WARNING: Frame IP not in any known module. Following frames may be wrong.
00000000 00000000 00000000 00000000 00000000 0x1630197
0:000> !pe 01b66c20
Exception object: 01b66c20
Exception type: System.ArgumentException
Message: Argument NULL
InnerException: <none>
StackTrace (generated):
    SP       IP       Function
    0024F09C 01630197
08SimpleExc!Advanced.NET.Debugging.Chapter8.SimpleExc.ProcessData
(System.String)+0x57
    0024F0B4 01630124
08SimpleExc!Advanced.NET.Debugging.Chapter8.SimpleExc.Run()+0x34
    0024F0C8 016300A7 08SimpleExc!Advanced.NET.Debugging.Chapter8.SimpleExc.Main
(System.String[])+0x37

StackTraceString: <none>
HResult: 80070057
```

Sądząc po tym wyjątku, możemy wywnioskować, że aplikacja uległa awarii z powodu nieprawidłowej wartości (NULL) argumentu przekazanego w wywołaniu metody. Na podstawie dostarczonego zrzutu stosu wywołań możemy przeprowadzić prosty przegląd kodu, dzięki czemu uda się znaleźć problem. Mimo iż powyższy przykład jest bardzo prosty, stanowi on dowód, że diagnozowanie aplikacji zarządzanych po wystąpieniu awarii jest jak najbardziej możliwe. W procesie tym można korzystać ze wszystkich bardzo pomocnych poleceń rozszerzeń debugera SOS i SOSEX.

Usługa raportowania błędów

Każdy użytkownik systemu Windows przynajmniej raz w życiu widział okno dialogowe widoczne na rysunku 8.2.

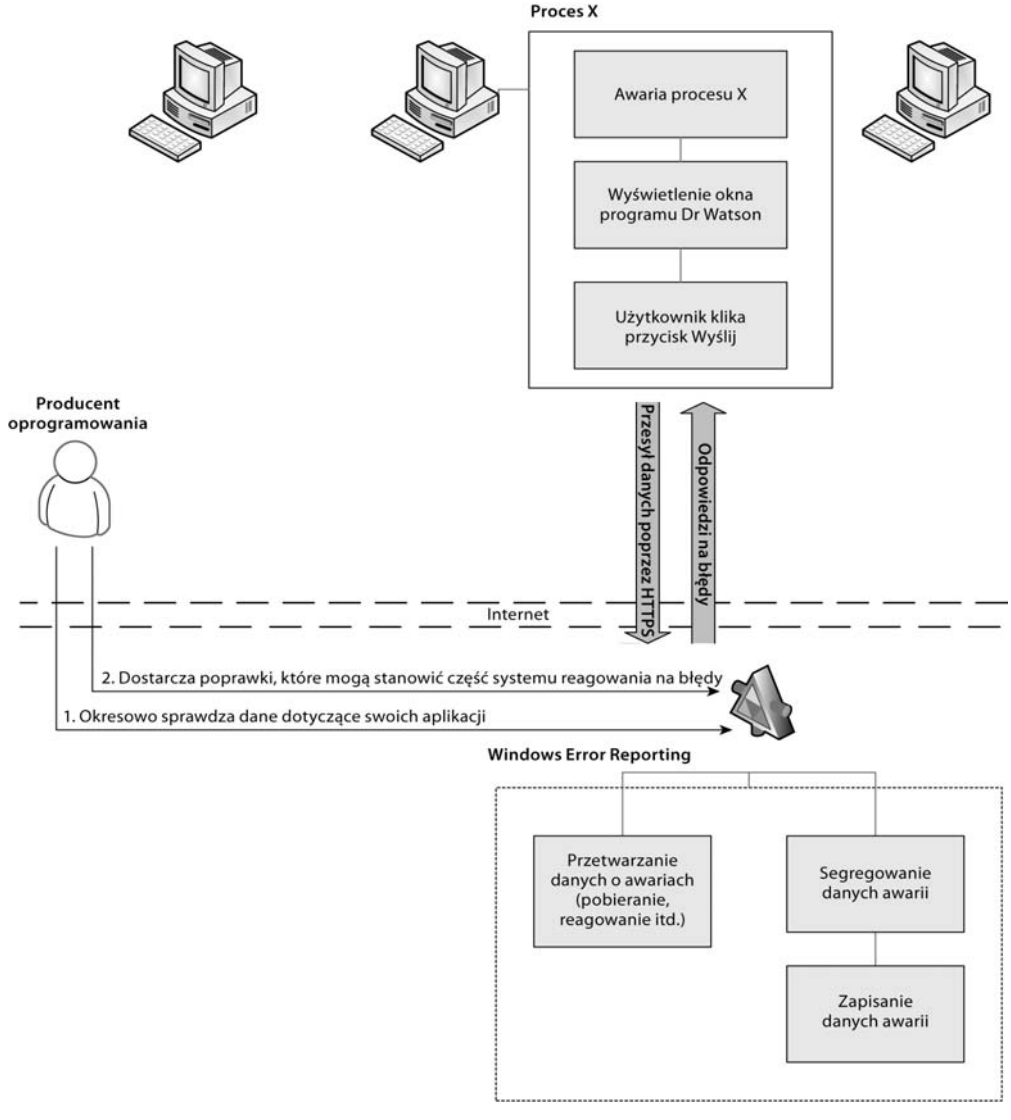


Rysunek 8.2. Okno dialogowe programu Dr Watson

Okno to jest graficznym elementem technologii o nazwie Usługa raportowania błędów (ang. *Windows Error Reporting* — WER). Można się w nim zdecydować na wysłanie raportu o błędzie firmie Microsoft. Jeśli użytkownik wyśle taki raport, zostanie on przesłany zabezpieczonym kanałem (HTTPS) do bazy danych Microsoftu, w której zostanie przydzielony do odpowiedniej kategorii i zapisany w celu późniejszego przeanalizowania. Jak nietrudno się domyślić, w raporcie tym znajduje się również plik zrzutu, na podstawie którego programiści mogą spróbować znaleźć przyczynę zaistniałego problemu. Z Usługi raportowania błędów mogą korzystać aplikacje różnych producentów, nie tylko firmy Microsoft. Raportowaniu podlega każdy proces systemu Windows wywołujący jakąś awarię. Aby jednak uzyskać dostęp do raportów zgłaszanych przez nasze aplikacje, musimy najpierw zarejestrować się w Usłudze raportowania błędów. W tym podrozdziale dowiesz się, jak działa usługa WER, co jest w jej ramach wysyłane, jak zarejestrować się w tej usłudze oraz jak pobierać z niej raporty.

Architektura usługi Windows Error Reporting

Windows Error Reporting to usługa gromadzenia danych dotyczących awarii umożliwiająca firmie Microsoft i innym producentom oprogramowania łatwe uzyskanie dostępu do informacji związanych z ich programami. Ogólny schemat działania usługi WER został przedstawiony na rysunku 8.3.



Rysunek 8.3. Schemat usługi WER

Dwa najważniejsze elementy powyższego schematu to:

- Komputery, na których działają aplikacje sprawiające problemy i które wysyłają raporty do WER.
- Producent oprogramowania monitorujący raporty na temat jego aplikacji przysyłane do WER.

Wyobraźmy sobie, że gdzieś na świecie znajduje się komputer, na którym działa jakaś aplikacja (proces X na rysunku 8.3) wyprodukowana przez firmę ADND. Program ten ulega awarii, co powoduje wyświetlenie okna dialogowego modułu Dr Watson, w którym użytkownik zgadza się na wysłanie raportu o błędzie poprzez bezpieczny kanał HTTPS do usługi WER. Usługa ta segreguje przychodzące raporty i zapisuje informacje o błędach w odpowiednich kategoriach. Użytkownik z firmy ADND wysyła zapytania do bazy danych WER na temat aplikacji swojej firmy i w odpowiedzi uzyskuje informacje. Teraz firma ADND może rozwiązać zaistniały problem i zdefiniować odpowiedź, która zostanie przedstawiona użytkownikowi aplikacji przez moduł Dr Watson przy następnej awarii. Reakcja ta może być przekazana w formie poprawki do programu lub jakiejś pomocnej informacji.

Jak widać, usługa WER to bardzo pomocny system umożliwiający bezpieczne gromadzenie informacji o błędach i wykorzystanie ich przez producentów oprogramowania do badania jakości ich aplikacji. Ponadto producenci mogą publikować odpowiedzi na poznane problemy i dodawać je do systemu reagowania WER, dzięki czemu klienci bez problemu uzyskują poprawki, gdy są one dostępne.

Po co wysyłać informacje o błędach

Gdy Dr Watson obudzi się, aby poinformować Cię o wystąpieniu awarii, możesz się zastanawiać, po co wysyłać te raporty. Czy ktoś coś z nimi robi? Prawda jest taka, że firma Microsoft bardzo poważnie traktuje raporty o błędach. Dlatego właśnie w ogóle powstał cały ten system raportowania. Dane z raportów są aktywnie analizowane i rozsyłane po różnych działach, których dotyczą. Gdy źródło problemu zostanie odkryte i programiści opracują łatkę, zostaje ona opublikowana (zwykle poprzez usługę Microsoft Update) do zainstalowania na komputerach użytkowników. Innymi słowy, każdy użytkownik powinien wysyłać wszystkie raporty o błędach, aby firma Microsoft lub inni producenci mogli przeanalizować problem i opracować jego rozwiązanie.

W dalszej części tego podrozdziału do zilustrowania procesu korzystania z usługi WER będzie wykorzystywana aplikacja *08SimpleExc.exe* z wcześniejszego podrozdziału.

Pierwszą czynnością, jaką należy wykonać, aby móc korzystać z usługi WER, jest rejestracja w tej usłudze.

Rejestrowanie się w usłudze WER

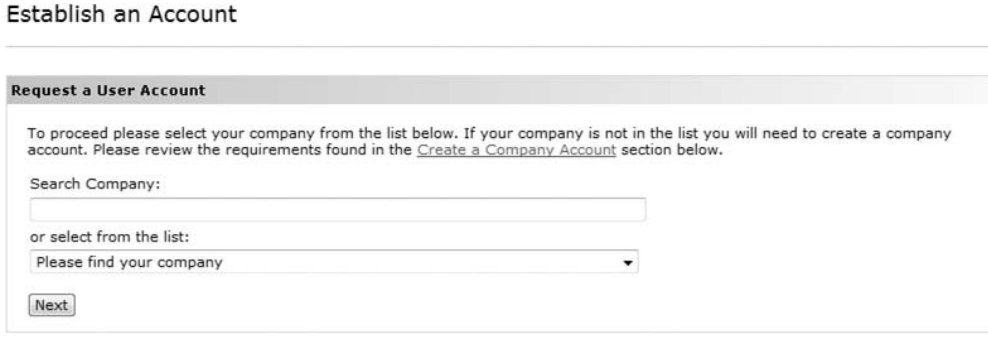
Aby stać się użytkownikiem usługi WER (tzn. móc pobierać z niej raporty), należy się zarejestrować. Proces rejestracji dzieli się na dwa etapy:

- Utworzenie konta użytkownika.
- Utworzenie konta firmowego.

Proces rejestracji można rozpocząć na poniższej stronie:

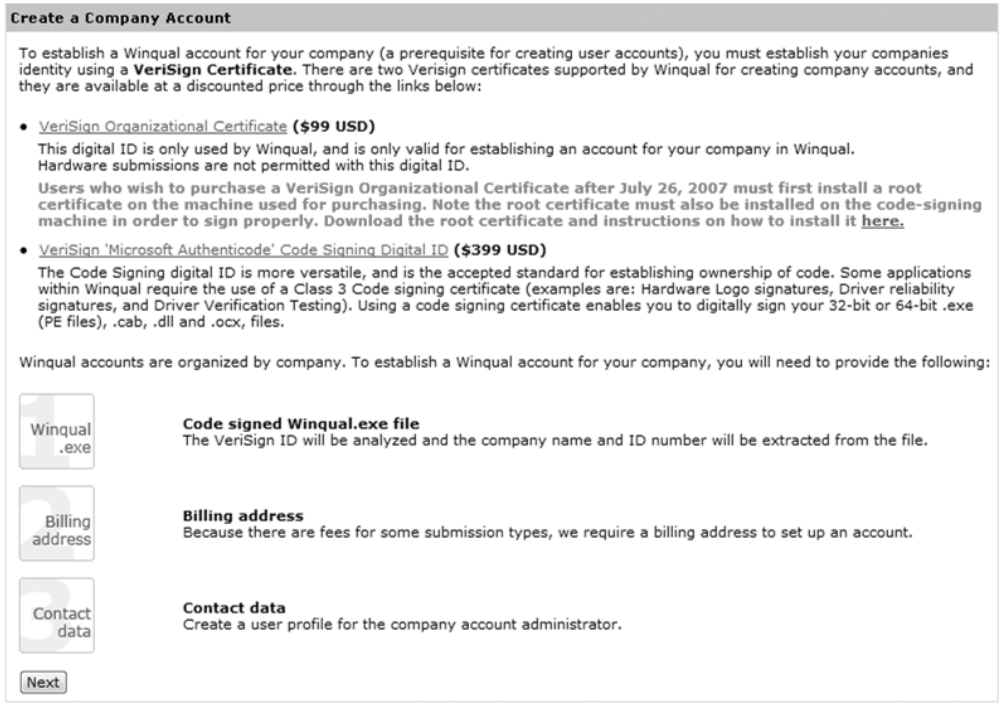
- *https://winqual.microsoft.com/SignUp/*

Po wpisaniu powyższego adresu w oknie przeglądarki zostanie wyświetlona strona widoczna na rysunku 8.4.



Rysunek 8.4. Pierwsza strona procesu rejestracji w usłudze WER

Aby utworzyć konto użytkownika, należy mieć konto firmowe. Jeśli masz już konto firmowe, możesz go poszukać lub wybrać jego nazwę z listy rozwijanej. Gdy już wybierzesz swoje konto firmowe, kliknij przycisk *Next*, aby przejść do kolejnego etapu rejestracji. Ponieważ nie utworzyliśmy jeszcze konta firmowego, przechodzimy do sekcji *Create a Company Account* (rysunek 8.5).



Rysunek 8.5. Tworzenie konta firmowego

Tworzenie konta firmowego składa się z trzech etapów:

1. Utworzenie podpisanego pliku *Winqual.exe*. Każda firma, która chce założyć konto w usłudze WER, musi dokonać autoidentyfikacji. Do tego celu jest wykorzystywany cyfrowy certyfikat podpisywania kodu Class 3 lub certyfikat instytucji certyfikującej, który można kupić w firmie VeriSign (*www.verisign.com/code-signing/content-signing-certificates/winqual-partners/index.html*). Po otrzymaniu certyfikatu należy nim podpisać plik *Winqual.exe* i wysłać do firmy Microsoft do weryfikacji.
2. Dostarczenie informacji rozliczeniowych. Większość funkcji usługi WER jest bezpłatna, ale za niektóre nowe udoskonalenia trzeba zapłacić. Dlatego firma Microsoft wymaga podania informacji rozliczeniowych.
3. Podanie informacji kontaktowych poprzez utworzenie konta użytkownika, za pomocą którego można uzyskać dostęp do konta firmowego.

Zaczniemy od punktu 1 (podpisywania pliku *Winqual.exe*). Jak wspominałem, firma Microsoft ze względów bezpieczeństwa wymaga, aby każda firma rejestrująca się w usłudze WER dokonała autoidentyfikacji za pomocą podpisu kodu źródłowego Class 3 lub certyfikatu wydanego przez instytucję certyfikującą. W dalszej części tego rozdziału przyjąłem, że mamy już certyfikat VeriSign. Najpierw musimy pobrać plik do podpisania z serwera Microsoftu pod następującym adresem:

```
https://winqual.microsoft.com/signup/winqual.exe
```

Zapisz ten plik na dysku twardym w folderze *C:\Sign*. Teraz potrzebne będą narzędzia do podpisywania kodu. Narzędzia te znajdują się pod następującym adresem:

```
https://winqual.microsoft.com/signup/signcode.zip
```

Zapisz pobrany plik na dysku i wypakuj jego zawartość w folderze *C:\Sign*. Po wypakowaniu powinny pojawić się dwa pliki:

- *readme.rtf* — plik zawierający instrukcje podpisywania kodu pliku binarnego za pomocą narzędzi do podpisywania kodu. Ponadto plik ten zawiera hasło, którego należy użyć przy wypakowywaniu pliku *signcode.exe*, również znajdującego się w tym archiwum ZIP.
- *signcode.exe* — aplikacja, za pomocą której podpiszemy plik *Winqual.exe*.

Wypakuj plik *signcode.exe* (pamiętaj o podaniu hasła z pliku *readme.rtf*) w tym samym folderze, w którym znajduje się plik *Winqual.exe* (*C:\Sign*). Ponadto skopiuj do tego folderu plik certyfikatu podpisywania kodu (z rozszerzeniem *.spc*) i klucz prywatny (z rozszerzeniem *.pvk*). Do podpisania pliku *Winqual.exe* zastosuj następujące polecenie:

```
C:\Sign>signcode.exe /spc myCert.spc /v myKey.pvk –t
 http://timestamp.verisign.com/scripts/timstamp.dll winqual.exe
Succeeded
```

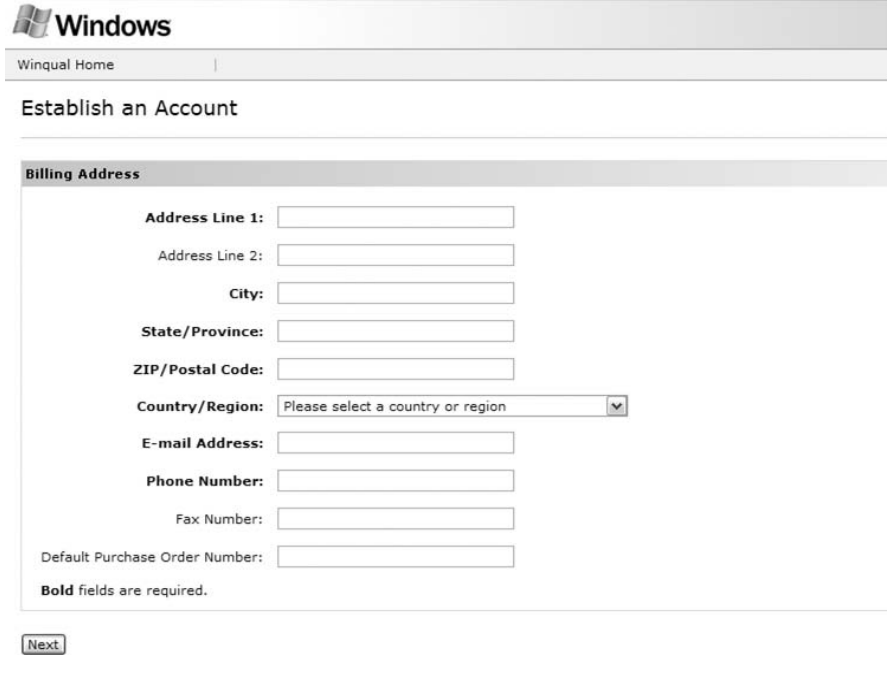
Nazwy plików *myCert.spc* i *myKey.pvk* zastąp nazwami własnych plików certyfikatu i klucza prywatnego. Podczas procesu podpisywania trzeba wprowadzić hasło klucza prywatnego. Podaj hasło dostarczone przez VeriSign podczas procesu nabywania certyfikatu. Jeśli podpisywanie powiedzie się, zostanie wyświetlony stosowny komunikat. Jeśli wystąpią jakieś błędy, sprawdź, czy poprawnie zostały wpisane nazwy plików certyfikatu i klucza prywatnego oraz czy znajdują się one w tym samym folderze co plik *signcode.exe*.

Następnym etapem procesu rejestracji jest wysłanie podpisanego pliku *Winqual.exe* do firmy Microsoft w celu weryfikacji. Na stronie widocznej na rysunku 8.5 kliknij przycisk *Next*. Na następnej stronie (rysunek 8.6) znajduje się formularz, za pomocą którego można wysłać plik do firmy Microsoft.



Rysunek 8.6. Wysyłanie podpisanego pliku Winqual.exe

Aby wysłać plik, należy wpisać w ścieżkę do niego i kliknąć przycisk *Next*. Po tym zostanie wyświetlona kolejna strona, na której należy podać informacje rozliczeniowe (rysunek 8.7).



Rysunek 8.7. Strona informacji rozliczeniowych

Jak wspominałem, większość funkcji usługi WER jest bezpłatna, ale za niektóre firma Microsoft pobiera opłaty. Informacje rozliczeniowe zostaną wykorzystane, jeśli użytkownik skorzysta z którejś z płatnych funkcji.

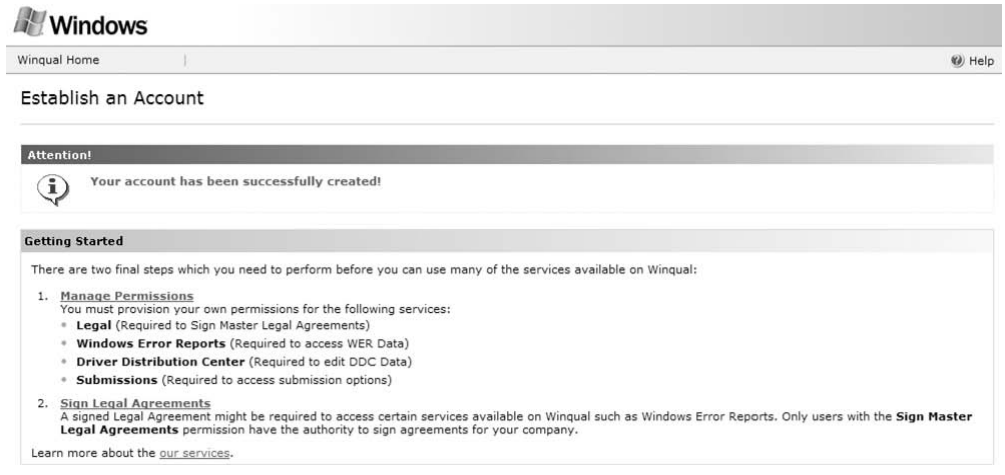
Wprowadź informacje rozliczeniowe swojej firmy (pola oznaczone drukiem pogrubionym należy obowiązkowo wypełnić) i kliknij przycisk *Next*. Przejdziesz do strony tworzenia profilu konta (rysunek 8.8).



Rysunek 8.8. Strona tworzenia profilu konta

Dane wpisane w polach *User Name* i *Password* będą służyły później do logowania do konta w usłudze. Wypełnij wszystkie pola formularza, w szczególności zwróć uwagę na wymagania dotyczące haseł opisane na dole strony. Utworzenie silnego hasła jest konieczne, aby informacje dotyczące błędów firm były bezpieczne.

Po wypełnieniu formularza kliknij przycisk *Next*, aby przejść do strony z informacją o pomyślnym zakończeniu procesu tworzenia nowego konta (rysunek 8.9).



Rysunek 8.9. Pomyślne zakończenie procesu tworzenia nowego konta

Ostatnie czynności, jakie należy wykonać przed uzyskaniem dostępu do WER, to ustalenie praw dostępu i podpisanie umów. Zaczniemy od praw dostępu. Kliknij łącze *Manage Permissions*, aby przejść na stronę zarządzania prawami dostępu widoczną na rysunku 8.10.



Rysunek 8.10. Zarządzanie prawami dostępu

W razie potrzeby zaznacz pola wyboru *Sign Master Legal Agreements*, *View WER Data* oraz *Download WER Data* i kliknij przycisk *Update*. Warto zauważyć, że z jednym kontem firmowym WER może być powiązanych wiele kont użytkownika. Jest to przydatne, gdy potrzebne są konta o różnych poziomach dostępu do danych (których określanie przedstawiono na rysunku 8.10). Np. jeden użytkownik może mieć dostęp do raportów o błędach, a inny może mieć możliwość podpisywania umów.

Teraz wrócimy do strony widocznej na rysunku 8.9, aby zakończyć proces poprzez podpisanie umowy, która jest wymagana przez firmę Microsoft. Kliknij łącze *Sign Legal Agreements*, co spowoduje przejście na stronę umowy dotyczącej korzystania z usługi Windows Error Reporting. Przeczytaj uważnie wszystkie informacje i jeśli akceptujesz warunki, wprowadź informacje na dole ostatniej strony, aby podpisać umowę. Jeśli chcesz zachować dla siebie kopię umowy, możesz wprowadzić w formularzu dane swojej firmy i ją wydrukować.

Na tym kończy się proces rejestracji. Od tej pory możesz korzystać z wszystkich funkcji usługi WER, logując się do swojego konta pod adresem *https://winqual.microsoft.com/default.aspx*.

Poruszanie się po witrynie usługi WER

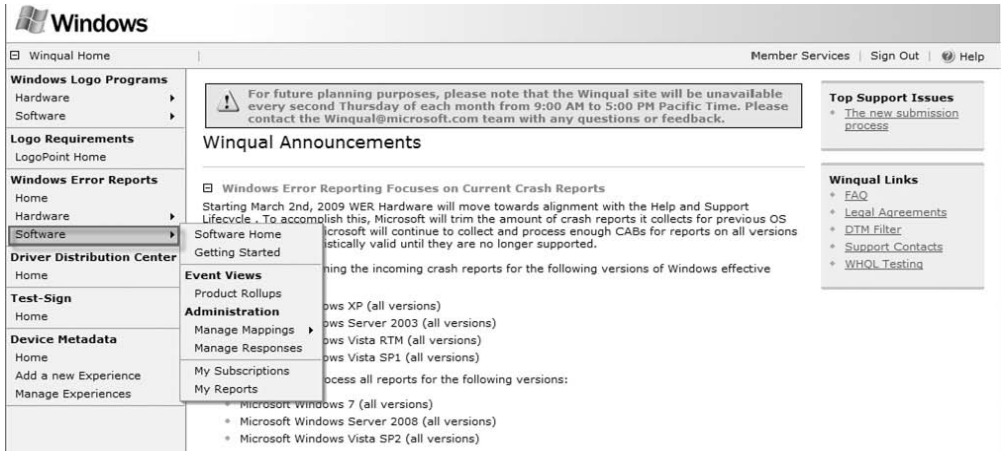
Po zalogowaniu się w usłudze WER na stronie wyświetlane są najnowsze wiadomości dotyczące Winqual. Po lewej stronie znajduje się kolumna zawierająca łącza do różnych części witryny. Trzy najważniejsze części tej kolumny to:

- *Windows Logo Programs*.
- *Windows Error Reports*.
- *Driver Distribution Center*.

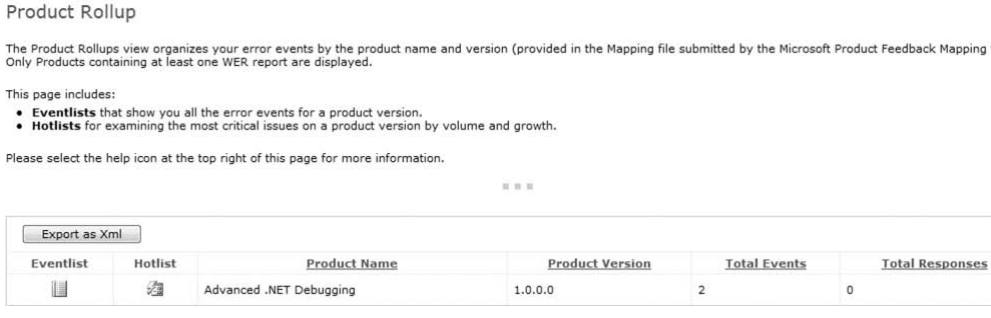
My zajmiemy się tylko sekcją *Windows Error Reports*, a mówiąc dokładniej, częścią *Software* usługi WER. Opcje dostępne w menu *Software* przedstawiono na rysunku 8.11.

Opcja *Product Rollups* w kategorii *Event Views* służy do wyświetlania raportów o błędach, posegregowanych według nazwy i wersji produktu. Taka przykładowa strona jest przedstawiona na rysunku 8.12.

Na powyższym rysunku widać, że zarejestrowano tylko jeden produkt — *Advanced .NET Debugging*. Produkt ten ma dwie kolumny, dzięki którym można uzyskać bardziej szczegółowe informacje na temat zdarzeń (awarii i innych), które mogły zostać zgłoszone:



Rysunek 8.11. Opcje menu Software usługi WER



Rysunek 8.12. Przykładowa strona Product Rollup

- *Eventlist* — kliknięcie tej ikony powoduje przejście na stronę zawierającą listę wszystkich zdarzeń, które miały miejsce w aplikacji.
- *Hotlist* — kliknięcie tej ikony powoduje przejście na stronę z informacjami na temat najczęstszych problemów w aplikacji z ostatnich 90 dni.

Kolejnym elementem menu jest kategoria *Administration*. Znajdują się w niej następujące opcje:

- *Manage Mappings* — umożliwia powiązanie plików binarnych z produktami, dzięki czemu usługa WER rozpoznaje, które pliki należą do poszczególnych produktów. Sposób tworzenia takiego powiązania pokażę nieco później.

- *Manage Responses* — umożliwia zdefiniowanie odpowiedzi na często zgłaszane przez użytkowników problemy. W istocie oznacza to utworzenie odpowiedzi zwrotnej, która może zawierać wszystko: od zwykłych informacji po łatki. Sposób definiowania odpowiedzi również pokażę nieco dalej.

Znamy już ogólny rozkład witryny WER. Teraz przejdziemy do wiązania plików binarnych z produktami, aby poinformować usługę, które pliki należą do poszczególnych produktów.

Wiązanie plików binarnych z produktami

Po zalogowaniu się na koncie należy tak ustawić konfigurację, aby wszystkie informacje o błędach naszych aplikacji były przesyłane na nasze firmowe konto. Gdy do usługi WER dotrze raport o błędzie, musi ona przyporządkować go właściwej firmie. Najważniejszym elementem tego procesu jest nazwa aplikacji. Dlatego firmy rejestrujące się w usłudze WER muszą podać w niej nazwy wszystkich swoich produktów (włącznie z wszystkimi plikami binarnymi). Informacje te są następnie przekazywane do witryny WER w odpowiednio sformatowanym pliku XML. Plików tych nie kompiluje się samodzielnie, tylko używa się do tego narzędzia WER o nazwie Microsoft Product Feedback Mapping Tool. Narzędzie to można pobrać pod następującym adresem:

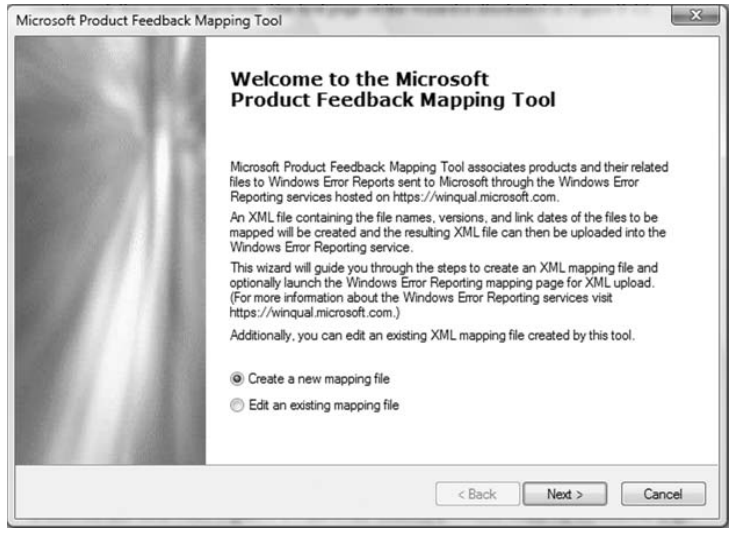
```
www.microsoft.com/downloads/details.aspx?Family
↪Id=4333E2A2-5EA6-4878-BBE5-60C3DBABC170&displaylang=en
```

Pobierzemy ten program i wypróbujemy go. Po jego zainstalowaniu kliknij *Start/Programy/Microsoft Product Feedback Mapping Tool*. Zostanie wyświetlona pierwsza strona kreatora, który poprowadzi Cię przez proces wiązania (rysunek 8.13).

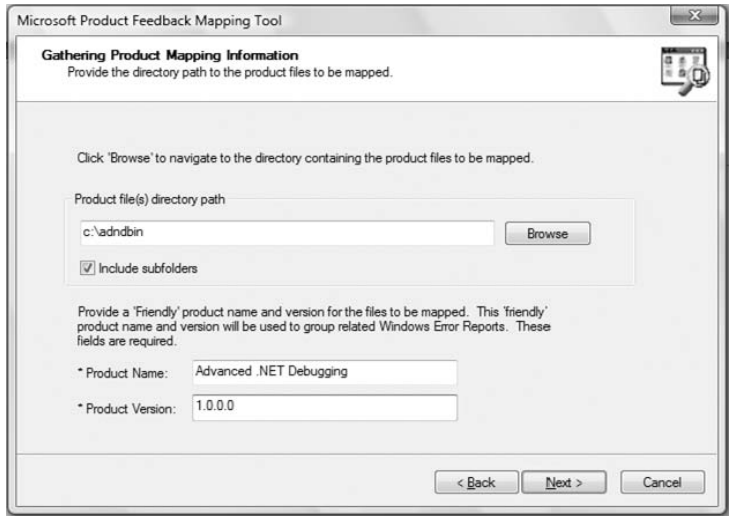
Proces tworzenia powiązania plików w usłudze WER zilustruję na przykładzie znanej nam już aplikacji *08SimpleExc.exe*. Zaznacz pole wyboru *Create a new mapping file* i kliknij przycisk *Next*. Na rysunku 8.14 pokazano następną stronę kreatora — o nazwie *Gathering Product Mapping Information*.

Poniżej znajduje się opis opcji dostępnych na tej stronie. Należy pamiętać, aby przed przejściem dalej wprowadzić takie same informacje jak na rysunku.

- *Product file(s) directory path* — określa ścieżkę do katalogu z plikami binarnymi aplikacji.



Rysunek 8.13. Narzędzie Microsoft Product Feedback Mapping Tool



Rysunek 8.14. Strona Gathering Product Mapping Information

- *Product Name* — określa nazwę produktu, z którym mają zostać związane pliki binarne. Nazwa ta powinna być łatwa do rozpoznania dla użytkowników konta WER, aby mogli oni wygodnie grupować informacje do przeszukiwania.

- *Product Version* — określa wersję produktu, z którym mają zostać związane pliki binarne. Wartość ta powinna być łatwa do rozpoznania dla użytkowników konta WER, aby mogli oni wygodnie grupować informacje do przeszukiwania.

Po wprowadzeniu wszystkich informacji w tym i kolejnym oknie kliknij przycisk *Next*. Następnie kreator zażąda podania nazwy pliku wiązania, który zostanie niebawem wygenerowany. Wprowadź następującą ścieżkę i kliknij przycisk *Next*:

```
C:\testmap.xml
```

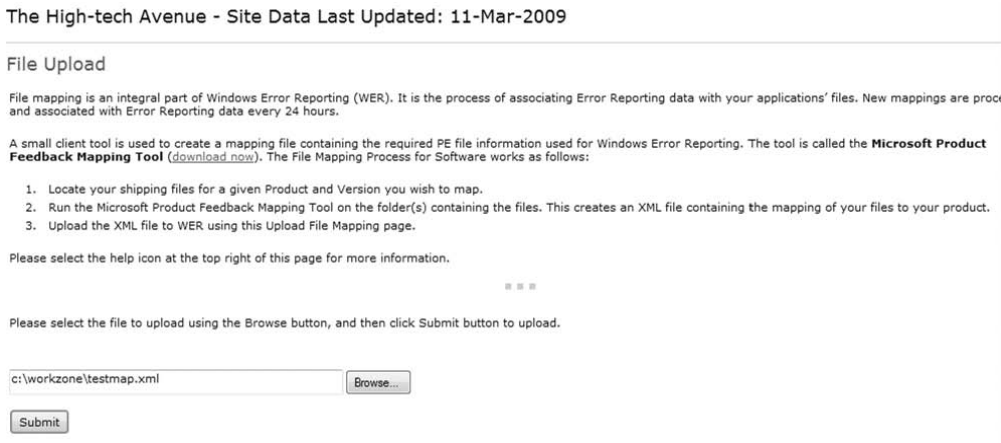
Ostatni etap procesu, polegający na wysłaniu pliku wiązań do witryny WER, jest przedstawiony na rysunku 8.15.



Rysunek 8.15. Wysyłanie pliku wiązań do usługi WER

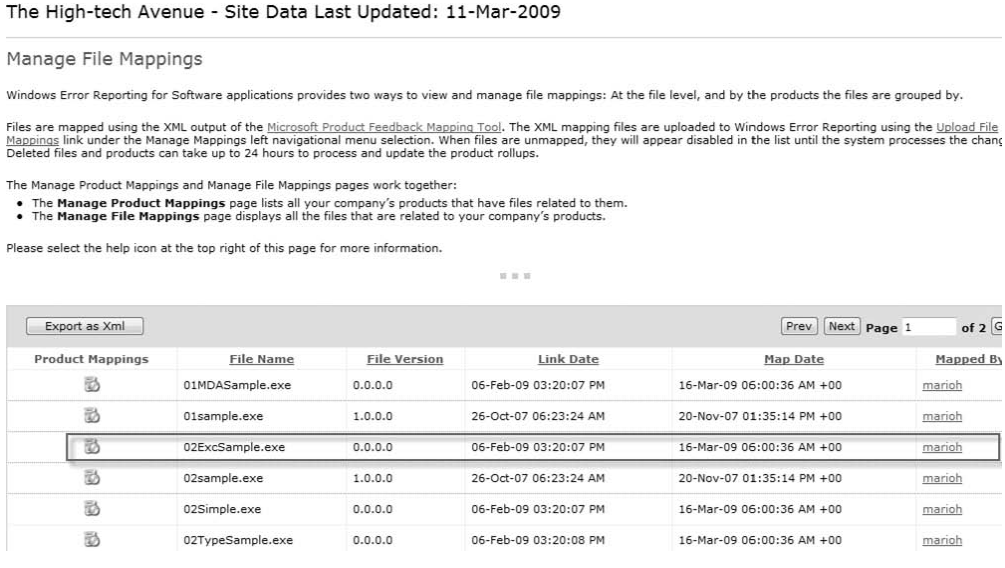
Zaznacz znajdujące się na tej stronie kreatora pole wyboru i kliknij przycisk *Finish*. Zostanie uruchomiona przeglądarka internetowa z wyświetloną stroną wysyłania pliku (rysunek 8.16).

Na tej stronie wprowadź ścieżkę do utworzonego przed chwilą pliku wiązań i kliknij przycisk *Submit*. Po pomyślnym wysłaniu pliku proces wiązania plików z produktem jest zakończony. Jeśli masz więcej produktów, musisz powtórzyć wszystkie opisane czynności dla każdego z nich osobno.



Rysunek 8.16. Strona wysyłania pliku wiązań

Wracamy do strony głównej usługi WER. Aby rozporządzać swoimi powiązaniami plików z produktami, należy po lewej stronie wybrać opcję *Manage Mappings* z menu *Software*. Można zarówno wybrać *Product and File Mappings*, jak i wysłać plik wiązań. Np. efekt wybrania łącza *File Mapping* po wysłaniu pliku wiązań przedstawia rysunek 8.17.



Rysunek 8.17. Wiązania plików w usłudze WER

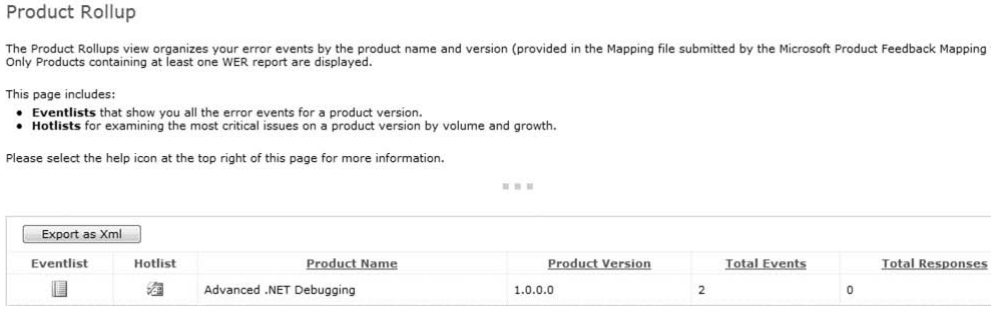
Na powyższym rysunku widać, że mamy kilka wiązań plików, z których każde ma własny zestaw atrybutów (np. data łącza i powiązania). Dodatkowo wyświetlone są informacje administracyjne, np. identyfikator i adres e-mail osoby, która utworzyła dane wiązanie.

Mając wiązania plików z produktem, możemy przejść do funkcji WER związanych z generowaniem raportów. Nauczymy się generować raporty z informacji otrzymanych od użytkowników oraz szczegółowo analizować te raporty (np. zrzuty awaryjne).

Wysyłanie zapytań do usługi WER

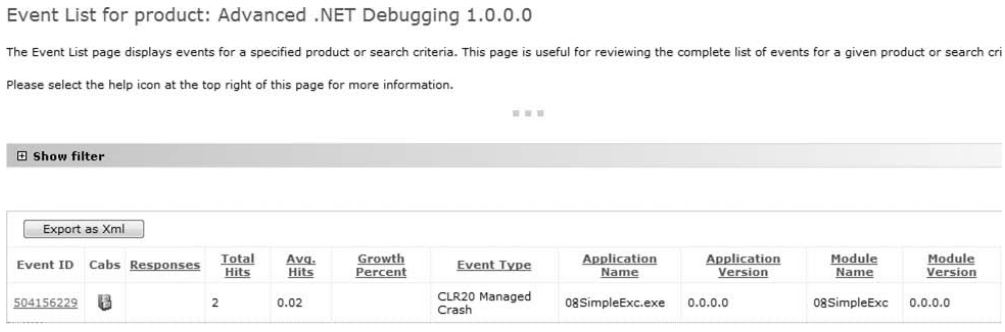
Mamy już konto w usłudze WER oraz powiązaliśmy plik binarny *08Simple Exc.exe* z produktem. Czas zatem na zapoznanie się z mechanizmem pobierania z WER informacji na temat zgłoszonych błędów w aplikacji. Uruchomimy nasz program klika razy i nakażemy modułowi Dr Watson wysłanie do witryny WER informacji o awariach. Należy pamiętać, że od wysłania raportu do jego pojawienia się w WER musi upłynąć nieco czasu.

Gdy wysłane raporty będą już dostępne, na stronie *Product Rollup* zostanie wyświetlona tabela produktów, jak na rysunku 8.18.



Rysunek 8.18. Strona Product Rollup z wyszczególnionymi błędami

Na rysunku 8.18 znajduje się nasz produkt (*Advanced .NET Debugging*) oraz ogólna liczba zaraportowanych zdarzeń. Dodatkowo w kolumnach *Eventlist* i *Hotlist* znajdują się ikony wyświetlające wszystkie zdarzenia, które wystąpiły w konkretnym produkcie, oraz błędy, które powtarzały się najczęściej w ciągu ostatnich 90 dni. Lista *Hotlist* jest znakomitym sposobem na zidentyfikowanie najważniejszych problemów z aplikacją. Na rysunku 8.19 znajduje się przykładowa strona listy zdarzeń.


Event List for product: Advanced .NET Debugging 1.0.0.0

The Event List page displays events for a specified product or search criteria. This page is useful for reviewing the complete list of events for a given product or search cri

Please select the help icon at the top right of this page for more information.

Show filter

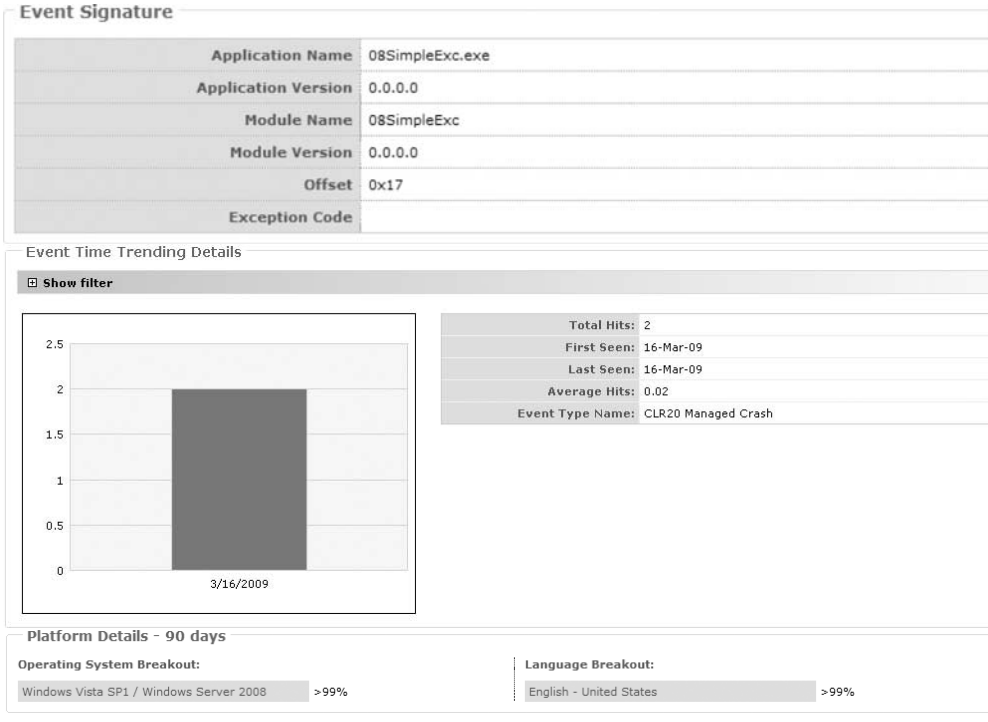
Export as Xml

Event ID	Cabs	Responses	Total Hits	Avg. Hits	Growth Percent	Event Type	Application Name	Application Version	Module Name	Module Version
504156229			2	0.02		CLR20 Managed Crash	08SimpleExc.exe	0.0.0.0	08SimpleExc	0.0.0.0

Rysunek 8.19. Lista zdarzeń produktu Advanced .NET Debugging

Na stronie tej znajduje się tabela, w której wierszach zapisane są informacje o poszczególnych zdarzeniach. Na rysunku 8.19 widać, że zostało zgłoszone tylko jedno zdarzenie, ale za to dwukrotnie. W tabeli znajduje się również informacja o tym, jakiego rodzaju zdarzenie spowodowało wygenerowanie danego raportu. W naszym przypadku jest to typ *CLR20 Managed Crash*. Oznacza to, że zdarzenie miało miejsce z powodu awarii w aplikacji zarządzanej wykorzystującej CLR 2.0. Klikając identyfikator zdarzenia, można wyświetlić szczegółowe informacje na jego temat. Strona szczegółów zdarzenia jest podzielona na trzy części:

- *Event Signature* — ponieważ z każdym produktem może być związanych wiele zdarzeń, każde zdarzenie musi mieć niepowtarzalny identyfikator. Składniki takiego identyfikatora, zapewniające jego niepowtarzalność, są następujące: nazwa i wersja aplikacji, nazwa i wersja modułu, miejsce (przesunięcie) w module, który spowodował wystąpienie zdarzenia. Jak widać na rysunku 8.20, w module *08SimpleExc.exe* awarię wywołało miejsce o przesunięciu *4734*.
- *Event Time Trending Details* — wykres znajdujący się w tej sekcji przedstawia czasowy przebieg występowania zdarzenia. Na rysunku 8.20 widać, że nasze zdarzenie wystąpiło 16 marca, a później częstotliwość jego występowania malała.
- *Platform Details* — zawiera szczegółowe informacje dotyczące konkretnego zdarzenia, np. system operacyjny i język. Informacje z tej sekcji są bezcenne przy identyfikowaniu problemów występujących tylko w określonych konfiguracjach i umożliwiają wysnucie wniosków, że np. problem zdarza się tylko w angielskich wersjach językowych programu.



Rysunek 8.20. Szczegóły zdarzenia modułu 08SimpleExc.exe

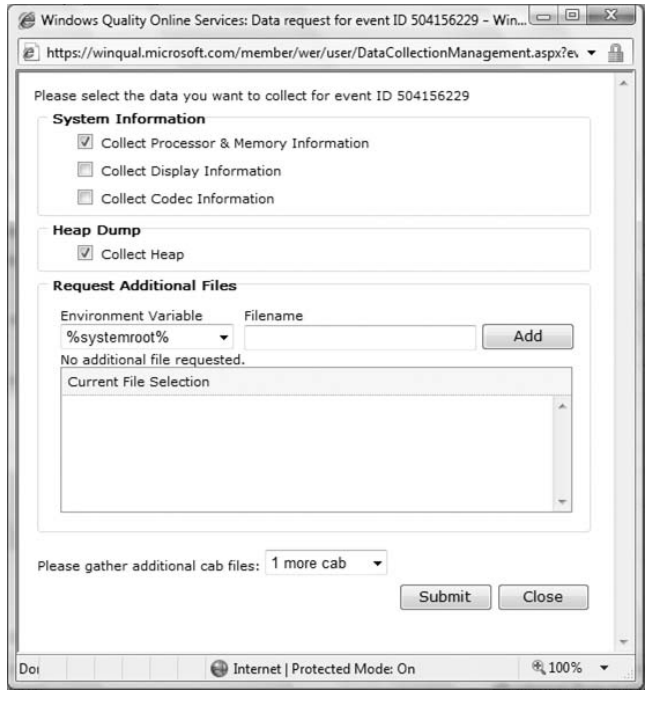
Na rysunku 8.20 widoczne są tylko dane dotyczące aplikacji *08Simple Exc.exe*.

Strona szczegółów zdarzeń zawiera również sekcję o nazwie *Data Collection*, którą przedstawiono na rysunku 8.21.



Rysunek 8.21. Sekcja gromadzenia danych

W sekcji *Data Collection* można przejść do listy dostępnych plików *.cab* określonego zdarzenia poprzez kliknięcie ikony *Cab Status* albo zmienić zasadę gromadzenia danych dla wybranego zdarzenia, klikając ikonę *Data Request*. Na rysunku 8.22 przedstawiono okno zasady gromadzenia danych.



Rysunek 8.22. Zasada gromadzenia danych

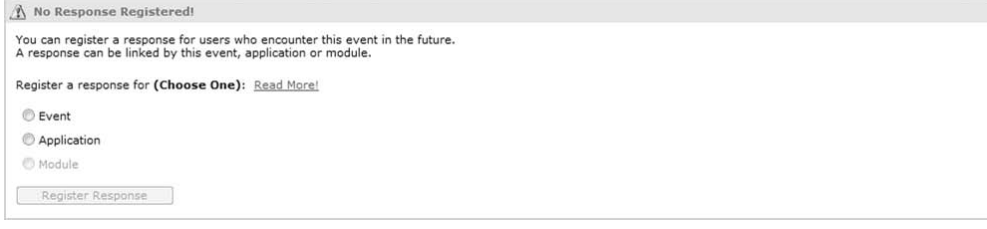
Pamiętaj, gdy w aplikacji wystąpi awaria, komputer, na którym to się stało, łączy się z usługą WER w celu sprawdzenia ustawionej zasady gromadzenia danych. Jeśli zasada została zmieniona i wymaga ponownego wysłania, komputer klienta tworzy nowy plik *.cab* zgodnie z zasadą i wysyła go do WER. W oknie zasady gromadzenia danych można określić, jakie dodatkowe informacje mają być zbierane w tym procesie. Oprócz informacji systemowych można pobrać dane sterty oraz dodatkowe pliki określone za pomocą zestawu predefiniowanych zmiennych środowiskowych. Ponadto można określić liczbę dodatkowych plików *.cab*, które mają zostać zebrane.

Ostatnia ważna kolumna tabeli przedstawionej na rysunku 8.19 ma nazwę *Cabs*. Kliknięcie ikony powoduje wyświetlenie listy dostępnych dla zdarzenia plików *.cab*. Plik *.cab* to zbiór plików reprezentujących informacje o zdarzeniu (jeden plik *.cab* przypada na jedno wysłanie) wysyłane przez użytkowników, którzy zdecydują się wysłać raport do firmy Microsoft. Jednym z najważniejszych plików takiego zbioru jest zrzut wykonany w chwili wystąpienia awarii. Plik ten można wykorzystać do zdiagnozowania problemu poawaryjnie za pomocą technik opisanych wcześniej.

Wiemy już, jakie informacje są udostępniane poprzez usługę WER — wszystko od ogólnego przeglądu zdarzeń po szczegółowe dane oparte na informacjach przesłanych przez użytkowników. Teraz zwrócimy naszą uwagę na ostatni etap procesu — udzielanie odpowiedzi użytkownikom po rozwiązaniu ich problemu.

Odpowiadanie na problemy

Aby odpowiedzieć użytkownikom na określone zdarzenie, należy przejść na jego stronę *Event Details*. Jeśli jeszcze nie została zarejestrowana dla niego żadna odpowiedź, na górze strony będą znajdowały się opcje służące do rejestrowania odpowiedzi (rysunek 8.23).



Rysunek 8.23. Opcje odpowiedzi na zdarzenie

Są trzy różne poziomy rejestrowania odpowiedzi:

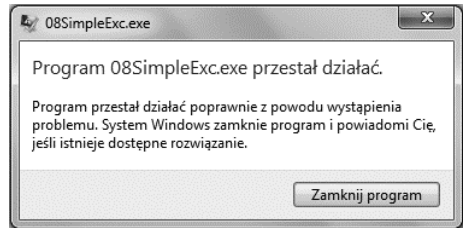
- *Event* — wykorzystywany do publikowania poprawek pojedynczych zdarzeń, które nie zostaną wcielone do aktualizacji produktu.
- *Application* — poziom aplikacji umożliwia tworzenie odpowiedzi, które zostaną przedstawione wszystkim użytkownikom mającym określoną wersję programu. Taka odpowiedź może mieć postać aktualizacji (np. nowej wersji).
- *Module* — poziom modułu umożliwia tworzenie odpowiedzi przeznaczonych dla użytkowników używających określonej wersji naszego modułu. Taka odpowiedź może mieć postać aktualizacji (np. nowej wersji).

My wybierzemy opcję rejestracji odpowiedzi na pojedyncze zdarzenie. Kliknij pole wyboru *Event*, a następnie kliknij przycisk *Register Response*. Teraz trzeba podać szczegóły odpowiedzi na zdarzenie. Do zarejestrowania odpowiedzi potrzebne są następujące informacje:

- *Products* — nazwa produktu.
- *URL of Solution/Info* — adres URL odpowiedzi. Adres ten powinien prowadzić na stronę zawierającą wszystkie informacje wymagane do udzielenia odpowiedzi.
- *Response Template* — szablon odpowiedzi. Można wykorzystać gotowy szablon albo zdefiniować własny. Przykładowe gotowe szablony to: *System Does Not Meet Minimum Requirements* (system nie spełnia podstawowych wymagań), *Product Upgrade* (uaktualnienie produktu) czy *Upgrade to New Version* (uaktualnienie do nowej wersji). Zależnie od wybranej z tej listy opcji wygląd pola podglądu może się zmieniać.
- *Response Template Preview* — podgląd informacji, które zostaną opublikowane w odpowiedzi.
- *Additional Information* — dodatkowe informacje, które chcielibyśmy dodać do odpowiedzi.

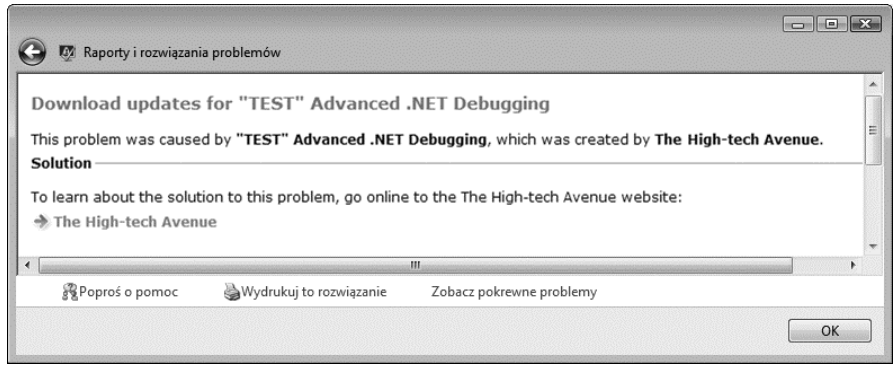
Po wprowadzeniu wszystkich informacji kontynuujemy rejestrację odpowiedzi, przechodząc na stronę *Response Management* (zarządzanie odpowiedziami), na której znajduje się lista wszystkich odpowiedzi, jakie zarejestrowaliśmy do tej pory. Należy zauważyć, że nowo zarejestrowana odpowiedź nie jest publikowana natychmiast, lecz najpierw przechodzi przez proces zatwierdzania, który trwa kilka dni. Na stronie *Response Management* można również zarządzać wszystkimi wcześniej utworzonymi odpowiedziami. Można obejrzeć ich szczegóły, dokonać w nich zmian oraz usunąć te, które są już nieaktualne.

Jak nasza odpowiedź zostanie przedstawiona użytkownikowi? Następnym razem, gdy nastąpi awaria programu, dla której zdefiniowano odpowiedź, zostanie wyświetlone okno dialogowe widoczne na rysunku 8.24.



Rysunek 8.24. Okno dialogowe wyświetlane w przypadku awarii aplikacji, dla której jest opublikowane rozwiązanie

Jeśli użytkownik kliknie przycisk *Zamknij program*, zostanie wyświetlone powiadomienie, że dla problemu, który wystąpił, jest dostępne rozwiązanie. Kliknięcie komunikatu spowoduje wyświetlenie okna dialogowego zawierającego tekst rozwiązania, jak na rysunku 8.25.



Rysunek 8.25. Okno dialogowe zawierające treść rozwiązania problemu z aplikacją

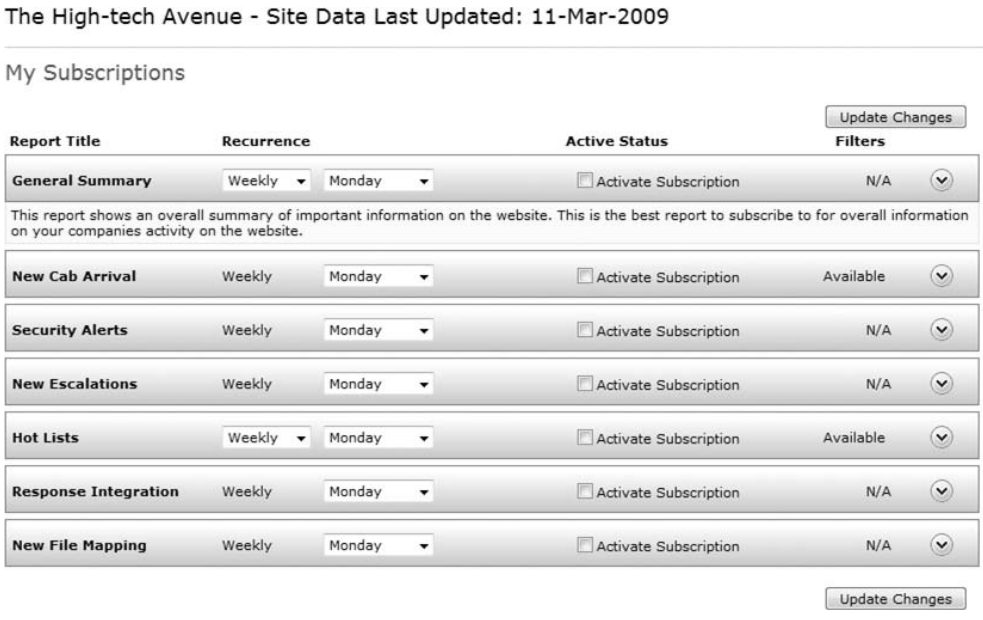
W tym przypadku użytkownik może kliknąć łącze *The High-tech Avenue*, które przeniesie go na stronę zawierającą szczegółowy opis rozwiązania problemu.

Raportowanie i subskrypcje

Dwie nowe funkcje dodane w usłudze WER to raportowanie i subskrypcje. Subskrypcje umożliwiają otrzymywanie powiadomień dotyczących określonych zdarzeń. Aby uzyskać dostęp do tej funkcji, należy w menu *Software* wybrać element *My Subscriptions*. Na rysunku 8.26 przedstawione zostały ustawienia dostępne w ramach tej funkcji.

Aby aktywować wybraną subskrypcję, należy zaznaczyć jej pole wyboru. Dodatkowo można określić datę (a w niektórych przypadkach także częstotliwość) wysyłania powiadomienia pocztą e-mail.

Funkcja raportowania to zestaw standardowych raportów, które można generować dla danych przechowywanych w WER. W czasie pisania tej książki dostępny był tylko jeden raport — o nazwie *Response Satisfaction*, który przedstawiał ogólną jakość odpowiedzi oraz liczbę wyświetleń i wysłanych przez użytkowników wyników ankiety. Funkcja raportowania jest stosunkowo nowa i w przyszłości należy się spodziewać powiększenia liczby raportów.



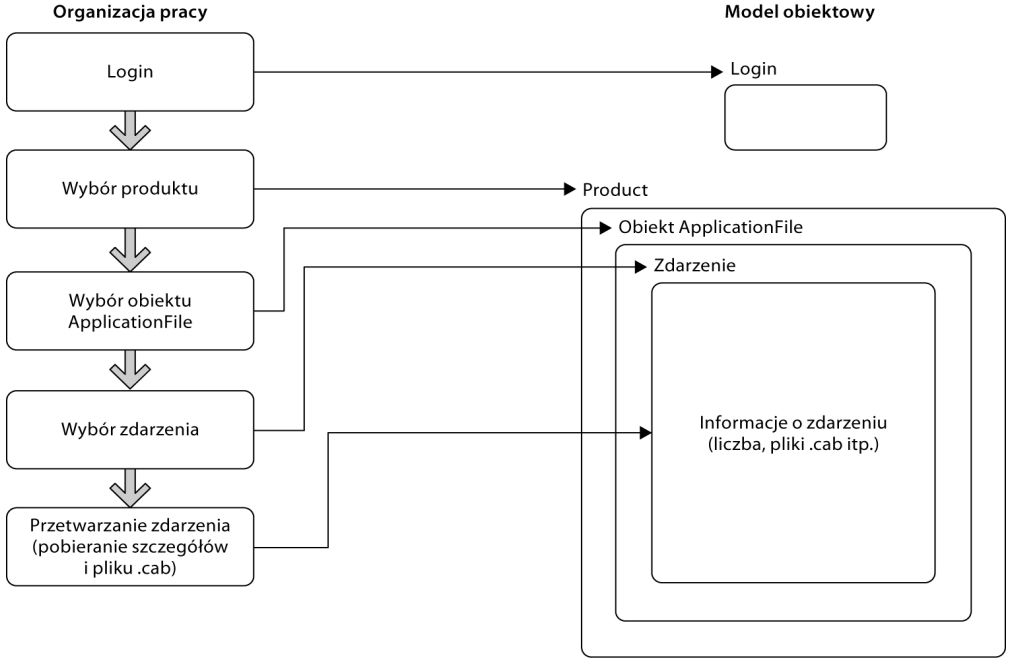
Rysunek 8.26. Opcje subskrypcji

Jak widać, WER to bardzo pomocna usługa umożliwiająca monitorowanie działania aplikacji na komputerach rzeczywistych użytkowników. Możliwość wysyłania przez użytkowników informacji o awariach programu to niezwykle cenna technologia, która znacząco przyczynia się do zmniejszenia problemów powodowanych przez błędy w programach.

Dostęp do usługi WER z poziomu programu

Usługa Windows Error Reporting jest dla firm znakomitym narzędziem do monitorowania ich aplikacji na komputerach użytkowników, umożliwiającym podejmowanie środków zaradczych w razie wystąpienia problemów. Wystarczy utworzyć konto i zarejestrować swoje produkty, wiążąc z nimi pliki binarne, aby uzyskać dostęp do obszernych baz danych na temat błędów występujących w aplikacjach działających w realnych warunkach. Jedną z wad tej usługi jest konieczność regularnego logowania się w witrynie w celu sprawdzenia, czy pojawiły się nowe informacje o zdarzeniach. Jeśli coś jest, można pobrać odpowiadający wybranemu zdarzeniu plik *.cab*, aby dokładniej zbadać problem. Znacznie lepszym sposobem na monitorowanie aplikacji poprzez WER jest jednak skorzystanie z udostępnianego przez tę usługę API. API usług sieciowych umożliwiają firmom tworzenie własnych

systemów monitorujących, które automatycznie pobierają dane o zdarzeniach z serwera. Warstwę dostępu do swoich usług sieciowych zespół WER opublikował w serwisie CodePlex (*http://www.codeplex.com/*). Można tam dodatkowo znaleźć kilka przykładowych projektów (m.in. gadżet dla systemu Windows Vista pobierający informacje z WER). Ponadto opublikowany został również zestaw klienta (*http://wer.codeplex.com/Release/ProjectReleases.aspx?ReleaseId=12825*) udostępniający model obiektowy WER, co znacznie ułatwia pracę nad własnymi aplikacjami monitorującymi. Rysunek 8.27 przedstawia ogólny schemat działania usługi WER i modelu klient-obiekt.



Rysunek 8.27. Schemat pracy WER i modelu klient-obiekt

Korzystanie z WER zawsze zaczyna się od zalogowania się klienta w systemie za pomocą klasy `Login`. Dane, których używa się do tego logowania, to dane użytkownika z uprawnieniami dostępu do konta (np. głównego administratora). Jeśli logowanie się powiedzie, obiekt logowania najczęściej jest zapisywany w pamięci podręcznej aplikacji klienckiej i wykorzystywany we wszystkich innych działaniach WER. Następnie aplikacja może przeliczyć produkty dostępne na koncie (wcześniej poddane wiązaniu za pomocą na-

rzędzia *Product Feedback Mapping Tool*) i wybrać te, które nas interesują. Do reprezentowania produktów służy klasa `Product`. Każdy egzemplarz klasy `Product` zawiera zbiór aplikacji odpowiadających każdemu plikowi binarnemu powiązanemu w procesie wiązania. Aplikację reprezentuje klasa `ApplicationFile`, która umożliwia uzyskanie szczegółowych informacji o aplikacji. Podobnie jak klasa `Product` gromadzi zestaw aplikacji, każda aplikacja gromadzi zestaw zdarzeń (reprezentowanych przez klasę `Event`). Każdy taki obiekt zdarzenia zawiera szczegółowe informacje, jak np. typ zdarzenia i związane z nim pliki *.cab*. Klient może wykorzystać te informacje na różne sposoby, np. zapisać je w bazie danych albo dodać do systemu automatycznego śledzenia błędów.

Technikę integrowania własnego klienta z usługą WER przedstawię na przykładowej aplikacji konsolowej, która będzie pobierała szczegóły określonego zdarzenia. Kod źródłowy tego programu znajduje się na listingu 8.2.

Listing 8.2. Przykładowa aplikacja pobierająca dane z usługi WER

```
using System;
using System.IO;
using System.Text;
using System.Runtime.InteropServices;
using Microsoft.WindowsErrorReporting.Services.Data.API;

namespace Advanced.NET.Debugging.Chapter8
{
    class WerConsole
    {
        static void Main(string[] args)
        {
            WerConsole s = new WerConsole();
            s.Run();
        }

        public void Run()
        {
            int eventId;
            string product, file, cabLoc, userName, password;
            Login login;

            Console.Write("Podaj nazwę użytkownika: ");
            userName = Console.ReadLine();

            Console.Write("Podaj hasło: ");
            password = Console.ReadLine();

            Console.WriteLine("Logowanie do WER...");
            login=WerLogin(userName, password);
            Console.WriteLine("Logowanie powiodło się");
```

```
    Console.Write("Podaj produkt: ");
    product=Console.ReadLine();

    Console.Write("Podaj plik: ");
    file=Console.ReadLine();

    Console.Write("Podaj identyfikator zdarzenia: ");
    eventId=Int32.Parse(Console.ReadLine());

    Console.Write("Podaj miejsce do zapisywania plików .cab: ");
    cabLoc = Console.ReadLine();
    if (Directory.Exists(cabLoc) == false)
    {
        Directory.CreateDirectory(cabLoc);
    }

    Event e=GetEvent(product, file, eventId, ref login);
    Console.WriteLine("Zdarzenie pobrane");
    Console.WriteLine("Identyfikator zdarzenia: " + e.ID);
    Console.WriteLine("Liczba wystąpień zdarzenia: " + e.TotalHits.ToString());
    Console.WriteLine("Zapisywanie plików .cab...");
    foreach (Cab c in e.GetCabs(ref login))
    {
        try
        {
            c.SaveCab(cabLoc, true, ref login);
        }
        catch (Exception)
        {
        }
    }
    Console.WriteLine("Miejsce zapisania plików .cab: " + cabLoc);
}

public Login WerLogin(string userName, string password)
{
    Login login = new Login(userName, password);
    login.Validate();
    return login;
}

public Event GetEvent(string pr,
                      string fi,
                      int eventId,
                      ref Login login)
{
    foreach (Product p in Product.GetProducts(ref login))
    {
        if (p.Name == pr)
        {
            ApplicationFileCollection ac =
                      p.GetApplicationFiles(ref login);
            foreach (ApplicationFile file in ac)
            {
                if (file.Name == fi)
                {
```

```
                        EventPageReader epr=file.GetEvents();
                        while (epr.Read(ref login) == true)
                        {
                            EventReader er = epr.Events;
                            while (er.Read() == true)
                            {
                                Event e = er.Event;
                                return e;
                            }
                        }
                    }
                }
            }
        }
        throw new Exception("Nie znaleziono zdarzenia");
    }
  }
}
```

Kod źródłowy i plik binarny tego programu znajdują się w następujących lokalizacjach:

- Kod źródłowy: *C:\ADND\Chapter8\WerConsole*
- Plik binarny: *C:\ADNDBin\08WerConsole.exe*

Na potrzeby tego przykładu zestaw klienta WER (*Microsoft.Windows ErrorReporting.Services.Data.API.dll*) został zapisany w folderze *C:\ADND\Chapter8\WerConsole* i jest automatycznie zapisywany w folderze *C:\ADNDBin* w procesie kompilacji projektu. Najnowszą wersję zestawu klienta WER można znaleźć w witrynie CodePlex pod następującym adresem:

```
http://wer.codeplex.com/Release/ProjectReleases.aspx?ReleaseId=12825
```

Jak widać na listingu 8.2, jest to prosta aplikacja wiersza poleceń prosząca na początku użytkownika o podanie następujących informacji:

- Nazwa użytkownika — nazwa użytkownika potrzebna do zalogowania się w usłudze WER.
- Hasło — hasło użytkownika określonego wcześniej. Należy zwrócić uwagę, że wprowadzone hasło będzie widoczne w konsoli.

Po otrzymaniu nazwy użytkownika i hasła aplikacja przekazuje te informacje jako parametry za pomocą metody `Login`. W przypadku powodzenia metoda ta zwraca egzemplarz klasy `Login` reprezentujący nową sesję WER. Egzemplarz ten będzie wykorzystywany w dalszej pracy. Po połączeniu się z usługą WER aplikacja prosi o podanie dodatkowych informacji:

- Produkt — nazwa interesującego nas produktu.
- Plik — interesujący nas plik aplikacji.
- Identyfikator zdarzenia — identyfikator interesującego nas zdarzenia.
- Miejsce do zapisywania plików *.cab* — miejsce, w którym mają zostać zapisane pliki *.cab* związane z określonym identyfikatorem zdarzenia.

Po uzyskaniu powyższych informacji klient łączy się z usługą WER i szuka zdarzenia. Robi to w następujący sposób:

1. Szuka określonego produktu, przeszukując wszystkie produkty zarejestrowane przez firmę (związane z użytkownikiem o określonej nazwie).
2. Szuka określonego pliku aplikacji, przeszukując wszystkie pliki aplikacji związane z produktem znalezionym w punkcie 1.
3. Szuka zdarzenia, przeszukując wszystkie zdarzenia związane z plikiem aplikacji znalezionym w punkcie 2.
4. Jeśli znajdzie zdarzenie, pobiera wszystkie związane z nim pliki *.cab*.

Należy zauważyć, że we wszystkich wymienionych wyżej operacjach, w których potrzebne jest odwołanie do WER, potrzebny jest również egzemplarz logowania (tzn. ustalonej sesji) przekazywany jako parametr.

Zobaczmy przykładową sesję działania naszego klienta:

```
C:\ADNDBin>08WerConsole.exe
Enter user name: MarioH
Enter password: <password>
Login into WER...
Login succeeded
Enter Product: Advanced .NET Debugging
Enter File: 08SimpleExc.exe
Enter Event ID: 504156229
Enter Location to store CABs: c:\zone\CAB
Event successfully retrieved
Event ID: 504156229
Event Total Hits: 2
Storing CABs...
CABs stored to: c:\zone\CAB
```

Czas pobierania plików może być nieco długi, jeśli do pobrania jest dużo plików zawierających duże ilości informacji. W folderze na pliki *.cab* znajdują się teraz następujące pliki:

```
C:\ADNDBin>dir /B c:\Zone\cab
504156229-CLR20ManagedCrash-0605004230.cab
504156229-CLR20ManagedCrash-0605004408.cab
504156229-CLR20ManagedCrash-0605004551.cab
```

```
504156229-CLR20ManagedCrash-0605004647.cab
504156229-CLR20ManagedCrash-0605004808.cab
504156229-CLR20ManagedCrash-0605004930.cab
504156229-CLR20ManagedCrash-0605005030.cab
504156229-CLR20ManagedCrash-0605005112.cab
504156229-CLR20ManagedCrash-0606022813.cab
504156229-CLR20ManagedCrash-0606025125.cab
```

Możemy teraz wypakować pobrane pliki *.cab*, załadować do debugera odpowiedni plik zrzutu i spróbować znaleźć przyczynę problemu za pomocą technik debugowania poawaryjnego.

Programowy dostęp do usługi WER to niezwykle przydatna technologia umożliwiająca firmom tworzenie własnych systemów automatycznego monitorowania aplikacji, które mogą bez problemu zintegrować ze swoimi systemami śledzenia błędów. Bez trudu można sobie wyobrazić usługę, która okresowo łączy się z WER w celu pobrania do bazy danych nowych informacji o błędach i która jest zintegrowana z firmowym systemem śledzenia błędów. Dzięki temu programiści są na bieżąco informowani o błędach i mogą szybko na nie reagować.

Podsumowanie

Debugowanie poawaryjne to jedna z najważniejszych technik w pracy inżyniera oprogramowania. Po dostarczeniu programu do klientów bardzo trudno jest rozwiązywać jakiekolwiek problemy, dlatego posiadanie wiedzy i możliwość szybkiego oraz precyzyjnego reagowania na błędy są warunkiem zminimalizowania kłopotów po stronie klienta.

W tym rozdziale dowiedzieliśmy się, dlaczego czasami konieczne jest zastosowanie technik debugowania poawaryjnego. Wiemy już, jakie są potrzebne do tego informacje oraz jakie narzędzia służą do ich zbierania. Wiemy również, jak wykorzystać dane zgromadzone w debugerze, aby rozwiązać zaistniały problem.

Szczegółowo została opisana usługa Windows Error Reporting, która umożliwia monitorowanie aplikacji w realnych zastosowaniach oraz udostępnia informacje o błędach (np. zrzuty awaryjne) związanych z poszczególnymi zdarzeniami i umożliwia zdefiniowanie reakcji na te błędy. Ponadto najnowsze udoskonalenia usługi WER pozwalają na dostęp do niej z poziomu programów, dzięki czemu można jeszcze lepiej zorganizować proces konserwacji programu.