15A

Odizolowany magazyn danych

Każdy program platformy .NET Framework ma dostęp do lokalnego magazynu danych unikalnego dla tego programu i określanego mianem **odizolowanego magazynu danych**. Tego rodzaju magazyn danych jest użyteczny, gdy program nie może uzyskać dostępu do standardowego systemu plików i nie ma możliwości zapisu danych w katalogach wskazywanych przez `ApplicationData`, `Local ↪ApplicationData`, `CommonApplicationData`, `MyDocuments` itd. (zob. sekcję „Katalogi specjalne" we wcześniejszej części rozdziału). Tak jest w przypadku aplikacji Silverlight oraz aplikacji w technologii ClickOnce wdrażanych za pomocą restrykcyjnych uprawnień *Internet*.

Odizolowany magazyn danych charakteryzuje się następującymi wadami:

- Niewygodne w użyciu API.
- Możliwość przeprowadzania operacji odczytu i zapisu jedynie za pomocą `IsolatedStorageStream`. Nie można pobrać ścieżki dostępu do pliku lub katalogu, a następnie wykorzystać zwykłe plikowe operacje wejścia-wyjścia.
- Magazyny danych komputera (odpowiednik `CommonApplicationData`) nie pozwalają użytkownikom o ograniczonych uprawnieniach systemu operacyjnego na usuwanie lub nadpisywanie plików, jeśli zostały utworzone przez innego użytkownika (choć można je modyfikować). W efekcie mamy do czynienia z błędem.

Pod względem zapewnienia bezpieczeństwa odizolowany magazyn danych został zaprojektowany w taki sposób, aby raczej nie pozwolić nam wychodzić na zewnątrz, niż innym wchodzić do środka. Dane w odizolowanym magazynie danych są silnie chronione przed innymi aplikacjami platformy .NET Framework działającymi w ramach bardziej restrykcyjnych uprawnień (np. strefa *Internet*). W pozostałych przypadkach nie ma silnych zabezpieczeń uniemożliwiających innej aplikacji uzyskanie dostępu do odizolowanego magazynu danych, jeśli naprawdę będzie taka potrzeba. Zaletą odizolowanego magazynu danych jest to, że aplikacje muszą się stosować do pewnych reguł podczas wzajemnej współpracy, która nie może zachodzić przypadkowo lub beztrosko.

Aplikacje działające w piaskownicy zwykle mają nałożone ograniczenie dotyczące wielkości odizolowanego magazynu danych. Wielkość domyślna wynosi 1 MB dla aplikacji Silverlight i wdrażanych za pomocą restrykcyjnych uprawnień *Internet*.

Hostowane aplikacje oparte na interfejsie użytkownika (np. Silverlight) mogą poprosić użytkownika o zgodę na zwiększenie odizolowanego magazynu danych, co wymaga wywołania metody `IncreaseQuotaTo()` w egzemplarzu `IsolatedStorage`. Wymieniona metoda musi być wywołana z poziomu zdarzenia zainicjowanego przez użytkownika, takiego jak kliknięcie przycisku. Jeżeli użytkownik wyrazi zgodę na zwiększenie limitu, wartością zwrotną metody będzie `true`.

Wielkość bieżącego limitu można sprawdzić za pomocą właściwości `Quota`.

Typy izolacji

Odizolowany magazyn danych może stosować separację na podstawie zarówno programu, jak i użytkownika. Mamy więc trzy podstawowe rodzaje separacji:

Lokalny magazyn danych użytkownika
: Jeden dla użytkownika, programu i komputera.

Podlegający roamingowi magazyn danych użytkownika
: Jeden dla użytkownika i programu.

Magazyn danych komputera
: Jeden dla programu i komputera (współdzielony przez wszystkich użytkowników programu).

Dane w podlegającym roamingowi magazynie użytkownika podążają za nim w sieci, przy odpowiedniej obsłudze ze strony systemu operacyjnego i domeny. Jeżeli tego rodzaju obsługa jest niedostępna, zachowanie magazynu danych odpowiada lokalnemu magazynowi danych.

Jak dotąd omówiliśmy separację odizolowanych magazynów danych pod względem „programu". Z perspektywy odizolowanego magazynu danych w zależności od wybranego trybu programem może być:

- zestaw;
- zestaw działający w kontekście określonej aplikacji.

Druga z wymienionych możliwości jest określana mianem **izolacji domeny** i stosowana znacznie częściej niż **izolacja podzespołu**. Segregacja izolacji domeny odbywa się na podstawie dwóch czynników: aktualnie wykonywany zestaw oraz plik wykonywalny (lub aplikacja sieciowa, która pierwotnie go uruchomiła). Z kolei izolacja podzespołu odbywa się jedynie na podstawie aktualnie wykonywanego podzespołu. Dlatego też różne aplikacje wywołujące ten sam zestaw będą współdzieliły ten sam magazyn danych.

Podzespoły i aplikacje są identyfikowane na podstawie ich silnych nazw. Jeżeli nie ma silnej nazwy, pod uwagę brana jest pełna ścieżka dostępu do pliku podzespołu (lub adres URI). Oznacza to, że po przeniesieniu lub zmianie nazwy podzespołu odizolowany magazyn danych będzie wyzerowany.

Ogólnie rzecz ujmując, mamy sześć rodzajów odizolowanych magazynów danych. Ich porównanie znajduje się w tabeli 15a.1.

Nie istnieje coś takiego jak izolacja na podstawie jedynie domeny. Jeżeli odizolowany magazyn danych chcemy współdzielić między wszystkimi podzespołami aplikacji, wówczas możemy skorzystać z pewnego prostego rozwiązania. Wystarczy w jednym z podzespołów udostępnić metodę publiczną odpowiedzialną za utworzenie i zwrot egzemplarza obiektu `IsolatedStorageFileStream`. W takim przypadku każdy zestaw będzie mógł uzyskać dostęp do odizolowanego magazynu danych po otrzymaniu obiektu `IsolatedStorageFile` — ograniczenia dotyczące izolacji są nakładane podczas konstrukcji obiektu, a nie w trakcie jego późniejszego użycia.

Podobnie nie istnieje coś takiego jak izolacja na podstawie jedynie komputera. Jeżeli odizolowany magazyn danych chcemy współdzielić między różnymi aplikacjami, wówczas rozwiązaniem jest przygotowanie wspólnego podzespołu, do którego będą się odwoływać wszystkie aplikacje. Ten zestaw powinien udostępniać metodę odpowiedzialną za przygotowanie i zwrot egzemplarza obiektu `IsolatedStorageFileStream` (izolacja na poziomie podzespołu). Aby takie rozwiązanie działało, współdzielony zestaw musi mieć silną nazwę.

Tabela 15a.1. Kontenery odizolowanych magazynów danych

Typ	Komputer?	Aplikacja?	Podzespół?	Użytkownik?	Metoda otrzymania magazynu danych
Domena użytkownika (domyślnie)	✓	✓	✓	✓	`GetUserStoreForDomain()`
Domena roamingu		✓	✓	✓	
Domena komputera	✓	✓	✓		`GetMachineStoreForDomain()`
Podzespół użytkownika	✓		✓	✓	`GetUserStoreForAssembly()`
Podzespół roamingu			✓	✓	
Podzespół komputera	✓		✓		`GetMachineStoreForAssembly()`

Odczyt i zapis w odizolowanym magazynie danych

Odizolowany magazyn danych używa strumieni, które działają niezwykle podobnie do zwykłych strumieni plików. W celu pobrania odizolowanego magazynu danych dla strumienia najpierw trzeba podać rodzaj żądanej izolacji, co odbywa się za pomocą jednej z metod statycznych w egzemplarzu `IsolatedStorageFile`, jak przedstawiliśmy wcześniej w tabeli 15a.1. Następnie wykorzystujemy go do przygotowania egzemplarza `IsolatedStorageFileStream` wraz z nazwą pliku oraz odpowiednim trybem pliku (`FileMode`):

```
// klasy IsolatedStorage znajdują się w przestrzeni nazw System.IO.IsolatedStorage

using (IsolatedStorageFile f =
       IsolatedStorageFile.GetMachineStoreForDomain())
using (var s = new IsolatedStorageFileStream ("hi.txt",FileMode.Create,f))
```

```
using (var writer = new StreamWriter (s))
  writer.WriteLine ("Witaj, świecie!");

// odczyt danych z magazynu

using (IsolatedStorageFile f =
       IsolatedStorageFile.GetMachineStoreForDomain())
using (var s = new IsolatedStorageFileStream ("hi.txt", FileMode.Open, f))
using (var reader = new StreamReader (s))
  Console.WriteLine (reader.ReadToEnd());         // Witaj, świecie!
```

Egzemplarz `IsolatedStorageFile` ma kiepską nazwę, ponieważ nie przedstawia pliku, ale raczej *kontener* dla plików (czyli w zasadzie katalog).

Lepszym (choć znacznie bardziej rozwlekłym) sposobem otrzymania egzemplarza `IsolatedStorageFile` jest wywołanie `IsolatedStorageFile.GetStore()` i przekazanie odpowiedniego połączenia opcji `IsolatedStorageScope` (jak pokazano na rysunku 15.6 nieco dalej w rozdziale):

```
var flags = IsolatedStorageScope.Machine
          | IsolatedStorageScope.Application
          | IsolatedStorageScope.Assembly;

using (IsolatedStorageFile f = IsolatedStorageFile.GetStore (flags,
  typeof (StrongName), typeof (StrongName)))
{
  ...
```

Zaletą zastosowania przedstawionego rozwiązania jest możliwość wskazania wywołaniu `GetStore()` rodzaju *cechy* do rozważenia podczas identyfikacji programu zamiast dokonywania automatycznego wyboru. Najczęściej w podzespołach programu będziemy stosować silne nazwy (jak to zostało zrobione w tym przykładzie), ponieważ są one unikatowe i łatwo zapewnić ich spójność w różnych wersjach.

Niebezpieczeństwo związane z umożliwieniem środowisku uruchomieniowemu CLR automatycznego wyboru cechy wynika z tego, że uwzględnia ono również podpisy w technologii Authenticode (zob. rozdział 18.). Takie podejście jest zwykle oczekiwane, ponieważ oznacza, że zmiana związana z Authenticode spowoduje wywołanie zmiany tożsamości. W szczególności, jeśli rozpoczniesz pracę bez użycia technologii Authenticode, a następnie zdecydujesz o jej zastosowaniu, wtedy środowisko uruchomieniowe CLR uzna aplikację za inną z perspektywy odizolowanego magazynu danych. Skutkiem może być utrata danych przez użytkowników między poszczególnymi wersjami programu.

Egzemplarz `IsolatedStorageScope` to typ wyliczeniowy opcji, którego elementy składowe muszą się łączyć w odpowiedni sposób, aby otrzymać poprawny magazyn danych. Na rysunku 15a.1 pokazano wszystkie poprawne połączenia. Zwróć uwagę na możliwość uzyskania dostępu do podlegających roamingowi magazynów danych (są one podobne do lokalnych magazynów danych, ale mogą „podróżować” za pomocą profili roamingu w rejestrze Windows).

	Podzespół	Podzespół i domena
Użytkownik lokalny	Podzespół | Użytkownik	Podzespół | Domena | Użytkownik
Użytkownik podlegający roamingowi	Podzespół | Użytkownik | Roaming	Podzespół | Domena | Użytkownik | Roaming
Komputer	Podzespół | Komputer	Podzespół | Domena | Komputer

Rysunek 15a.1. Poprawne połączenia opcji IsolatedStorageScope

W poniższym fragmencie kodu pokazano, jak utworzyć odizolowany magazyn danych. W kodzie wykorzystaliśmy izolację podzespołu i użytkownika podlegającego roamingowi:

```
var flags = IsolatedStorageScope.Assembly
          | IsolatedStorageScope.User
          | IsolatedStorageScope.Roaming;

using (IsolatedStorageFile f = IsolatedStorageFile.GetStore (flags,
                                                             null, null))
using (var s = new IsolatedStorageFileStream ("a.txt", FileMode.Create, f))
using (var writer = new StreamWriter (s))
  writer.WriteLine ("Witaj, świecie!");
```

Położenie magazynu danych

Poniżej wymieniono lokalizacje, w których platforma .NET Framework umieszcza pliki odizolowanych magazynów danych:

Zakres	Lokalizacja
Użytkownik lokalny	`[LocalApplicationData]\IsolatedStorage`
Użytkownik podlegający roamingowi	`[ApplicationData]\IsolatedStorage`
Komputer	`[CommonApplicationData]\IsolatedStorage`

Położenie każdego katalogu wskazywanego przez właściwość w nawiasach kwadratowych można otrzymać za pomocą wywołania `Environment.GetFolderPath()`. Poniżej przedstawiono wartości domyślne dla systemów Windows Vista i nowszych:

Zakres	Lokalizacja
Użytkownik lokalny	`\Users\<`*użytkownik*`>\AppData\Local\IsolatedStorage`
Użytkownik podlegający roamingowi	`\Users\<`*użytkownik*`>\AppData\Roaming\IsolatedStorage`
Komputer	`\ProgramData\IsolatedStorage`

Z kolei poniżej przedstawiono wartości domyślne dla systemu Windows XP:

Zakres	Lokalizacja
Użytkownik lokalny	`\Documents and Settings\<użytkownik>\Ustawienia lokalne\Dane aplikacji\IsolatedStorage`
Użytkownik podlegający roamingowi	`\Documents and Settings\<użytkownik>\Dane aplikacji\IsolatedStorage`
Komputer	`\Documents and Settings\All Users\Dane aplikacji\IsolatedStorage`

Są to jedynie katalogi bazowe, a właściwe dane pozostają umieszczone głęboko w labiryncie podkatalogów o nazwach wygenerowanych na podstawie wartości hash dla podzespołów. Istnieją powody do zarówno używania, jak i nieużywania odizolowanych magazynów danych. Z jednej strony, izolacja stała się możliwa — jeżeli aplikacja stosująca ograniczenia oparte na uprawnieniach będzie chciała ingerować w inną, wówczas może nie być w stanie sprawdzić zawartości katalogu pomimo takich samych uprawnień do systemu plików. Z drugiej strony, przeprowadzenie administrowania administracji z zewnątrz jest niepraktyczne. Czasami jest przydatne (lub wręcz niezbędne) przeprowadzenie edycji pliku konfiguracyjnego XML w *Notatniku*, aby zapewnić prawidłowe uruchomienie aplikacji. W przypadku odizolowanego magazynu danych staje się to niepraktyczne.

Sprawdzenie dostępnych odizolowanych magazynów danych

Obiekt `IsolatedStorageFile` oferuje metody przeznaczone do wyświetlenia plików znajdujących się w magazynie danych:

```
using (IsolatedStorageFile f = IsolatedStorageFile.GetUserStoreForDomain())
{
  using (var s = new IsolatedStorageFileStream ("f1.x",FileMode.Create,f))
    s.WriteByte (123);

  using (var s = new IsolatedStorageFileStream ("f2.x",FileMode.Create,f))
    s.WriteByte (123);

  foreach (string s in f.GetFileNames ("*.*"))
    Console.Write (s + " ");                    // f1.x f2.x
}
```

Istnieje także możliwość tworzenia i usuwania podkatalogów, podobnie jak plików:

```
using (IsolatedStorageFile f = IsolatedStorageFile.GetUserStoreForDomain())
{
  f.CreateDirectory ("podkatalog");

  foreach (string s in f.GetDirectoryNames ("*.*"))
    Console.WriteLine (s);                              // podkatalog

  using (var s = new IsolatedStorageFileStream (@"podkatalog\sub1.txt",
                                                 FileMode.Create, f))
  s.WriteByte (100);

  f.DeleteFile (@"podkatalog\sub1.txt");
  f.DeleteDirectory ("podkatalog");
}
```

W przypadku wystarczających uprawnień można również sprawdzić wszystkie odizolowane magazyny danych utworzone przez bieżącego użytkownika, a także wszystkie magazyny danych komputera. Ta funkcja może złamać prywatność programu, ale na pewno nie prywatność użytkownika. Spójrz na poniższy fragment kodu:

```
System.Collections.IEnumerator rator =
  IsolatedStorageFile.GetEnumerator (IsolatedStorageScope.User);

while (rator.MoveNext())
{
  var isf = (IsolatedStorageFile) rator.Current;

  Console.WriteLine (isf.AssemblyIdentity);       // silna nazwa lub adres URI
  Console.WriteLine (isf.CurrentSize);
  Console.WriteLine (isf.Scope);                  // użytkownik + ...
}
```

Metoda `GetEnumerator()` jest niecodzienna w zakresie akceptacji argumentu (dlatego zawierająca ją klasa jest niezbyt przyjazna pętli `foreach`). Metoda `GetEnumerator()` akceptuje jedną z trzech wymienionych poniżej wartości:

`IsolatedStorageScope.User`
: Wyświetla wszystkie lokalne magazyny danych należące do bieżącego użytkownika.

`IsolatedStorageScope.User | IsolatedStorageScope.Roaming`
: Wyświetla wszystkie podlegające roamingowi magazyny danych należące do bieżącego użytkownika.

`IsolatedStorageScope.Machine`
: Wyświetla wszystkie magazyny danych w komputerze.

Po otrzymaniu egzemplarza obiektu `IsolatedStorageFile` jego zawartość można wyświetlić za pomocą metod `GetFiles()` lub `GetDirectories()`.