Содержание

ГЛАВА 4

Данные пакета и их перемещение

Практически в каждом приложении требуется возможность сохранять состояние от своего имени или же от имени пользователя. Раньше для этой цели на платформе .NET Framework предлагалась технология, известная как *изолированное хранилище*. Но теперь эта технология встроена в ОС Windows. Когда пакет регистрируется (устанавливается) пользователем, Windows подготавливает хранилище, в которое приложение из данного пакета может записывать или читать из него данные от имени пользователя. Это хранилище тесно связано с пакетом. Когда пакет снимается с регистрации (удаляется), Windows автоматически уничтожает данное хранилище. Оно общедоступно для всех приложений, выполняющихся как составные части пакета (в том же самом контейнере приложения, как поясняется в самом конце книги), включая любые фоновые задачи, рассматриваемые в главе 9.

Пользоваться данными пакета очень просто. Для этого достаточно обратиться к классу `ApplicationData`, определение которого приведено ниже[1].

```
public sealed class ApplicationData {
   // вызвать этот метод первый раз, чтобы получить доступ к
   // объекту типа ApplicationData из данного пакета
   public static ApplicationData Current { get; }

   // Возвращает папку для хранения временных файлов из пакета
   // ПРИМЕЧАНИЕ: файлы могут быть в любой момент удалены
   // из этой папки системой
   public StorageFolder TemporaryFolder { get; }

   // Члены этого класса для доступа к данным, которые всегда
   // находятся на ПК пользователя
   public ApplicationDataContainer LocalSettings { get; }
   public StorageFolder LocalFolder { get; }

   // Члены этого класса служат для доступа к данным, которые
   // могут быть перемещены среди всех ПК пользователя
   public ApplicationDataContainer RoamingSettings { get; }
   public StorageFolder RoamingFolder { get; }
   public UInt64 RoamingStorageQuota { get; }
   public void SignalDataChanged();
   public event TypedEventHandler<ApplicationData, Object> DataChanged;
```

[1] Этот класс напрасно не назван `PackageData`, чтобы точнее отражать связь данных с пакетом, а не с отдельным приложением.

```
    // Члены этого класса, связанные с контролем версий данных пакета.
    // Они, как правило, используются в том случае, когда пользователь
    // переходит на новую версию пакета.
    public UInt32 Version { get; }
    public IAsyncAction SetVersionAsync(
       UInt32 desiredVersion, ApplicationDataSetVersionHandler handler);
    // Члены этого класса, очищающие все (или некоторые) данные пакета,
    // приходящиеся на каждого пользователя
    public IAsyncAction ClearAsync();
    public IAsyncAction ClearAsync(ApplicationDataLocality locality);
}
```

Для доступа к данным пакета, приходящимся на каждого пользователя, приложению нужно сначала получить доступ к его собственным хранилищам данных пакета, запросив свойство `Current` из класса `ApplicationData` следующим образом:

```
ApplicationData appData = ApplicationData.Current;
```

Имея ссылку на объект класса `ApplicationData` из пакета, можно получить доступ ко всем членам экземпляра этого класса. Далее нужно определить *местоположение* данных. Ниже вкратце описываются три вида местоположения данных.

- **Временное местоположение.** Используется подобно кешу; здесь, как правило, хранятся данные, получение которых требует затрат времени и средств. Полученные в итоге данные сохраняются в файле, размещаемом в папке `TemporaryFolder`. Следует иметь в виду, что любые файлы, сохраняемые в этой папке, могут быть удалены системой в любой момент — даже тогда, когда приложение все еще выполняется. Система обычно планирует задание на еженедельную очистку этих данных. А пользователь может изменить частоту выполнения этого задания с помощью планировщика заданий в Windows. Запланированная задача находится по пути `Microsoft\Windows\ApplicationData\CleanupTemporaryState`. Кроме того, пользователь может всегда выполнить утилиту очистки диска в Windows (Disk Cleanup — **`CleanMgr.exe`**).
- **Локальное местоположение.** Используется для данных, которые требуется хранить постоянно. Система не удаляет данные в этом местоположении, если только пользователь не удалит из нее сам пакет.
- **Перемещаемое местоположение.** Используется для любых данных, которые требуется автоматически копировать системными средствами из одного ПК пользователя в другой. Пользователи приобретают лицензии на приложения для Магазина Windows, позволяющие устанавливать одно и то же приложение на многих ПК. Благодаря такому местоположению данные пакета остаются одинаковыми на всех ПК пользователя, для чего используется модель конечной согласованности. Более подробно перемещаемое местоположение данных рассматривается ниже, в разделе “Перемещение данных пакета”.

Как только будет выбран конкретный вид местоположения данных, следует решить, каким образом их хранить и получать доступ к ним. Данные можно хранить двумя способами.

- **В настройках.** Настройки организованы в словарь, состоящий из пар “ключ–значение”. Каждый ключ является строкой до 255 символов, представляющих

наименование настройки и значение одного из простых типов WinRT или же одномерный массив значений этих типов, но размером не больше 8 Кбайт. Настройки являются коллекцией, состоящей не из пар "ключ–значение", а из иерархии с глубиной до 32 уровней. Это означает, что для хранения коллекций, состоящих из пар "ключ–значение", можно создать древовидную иерархию контейнеров, чтобы организовать данные пакета нужным образом.

- **В папке хранения.** Папки хранения предназначены для хранения элементов настроек размером больше 8 Кбайт, трактуемых как файлы (изображений, видео- и аудиозаписей). В этих папках можно создать любые файлы, а также подпапки, чтобы организовать иерархию хранения данных. В самих файлах можно хранить любые данные. На самом деле в них содержатся массивы байтов, но можно воспользоваться расширенными типами данных на платформе .NET, если сделать их сериализуемыми, а затем сериализовать объекты в массивы байтов. Подробнее об управлении файлами и их содержимым речь пойдет в главах 5 и 6.

Безусловно, основное назначение данных пакета — постоянное их хранение в промежутках между последовательными вызовами приложения. Так, если приложение прекращает свою работу, данные пакета продолжают существовать. Они будут существовать даже в том случае, если пользователь перейдет на новую версию пакета. Но когда это произойдет, возможно, придется изменить схему данных пакета. Для этой цели в классе `ApplicationData` предоставляются члены, позволяющие осуществлять контроль версий данных, как поясняется далее, в разделе "Контроль версий данных пакета".

Настройки данных пакета

Ниже приведен пример кода, в котором демонстрируется запись и чтение настройки из локальных данных пакета.

```
// получить доступ к хранилищам данных пакета
ApplicationData appData = ApplicationData.Current;

// сохранить элемент в коллекции:
// Ключ = "DataUpdatedAt", Значение = DateTimeOffset.Now
appData.LocalSettings.Values["DataUpdatedAt"] = DateTimeOffset.Now;

// затем прочитать элемент из коллекции:
DateTimeOffset dataUpdatedAt = (DateTimeOffset) appData.LocalSettings.
Values["DataUpdatedAt"];
```

В настройках можно сохранять значения только простых типов данных WinRT (но не .NET). К действительным типам данных WinRT относятся следующие: `Boolean`, `UInt8`, `Int16`, `UInt16`, `Int32`, `UInt32`, `Int64`, `UInt64`, `Single`, `Double`, `Char`, `String`, `DateTimeOffset`, `TimeSpan`, `Guid`, `Point`, `Size` и `Rect`. Значения этих типов данных можно также хранить в одномерных массивах, но результирующее значение не может быть больше 8 Кбайт. Тем не менее не существует ограничения на количество пар "ключ–значение", которые можно ввести в коллекцию. Свойство `Values` возвращает объект класса `ApplicationDataContainerSettings`, реализующего интерфейс `IObservableMap<String, Object>`, предоставляющий событие, которое наступает всякий раз, когда в коллекции изменяется элемент.

Помимо простых типов данных, имеется еще один специальный тип данных, которым можно воспользоваться, — ApplicationDataCompositeValue. Этот тип данных позволяет сохранять данные фрагментарно на уровне атомарных операций. Допустим, имеется следующий фрагмент кода:

```
String videoName = ...;
appData.LocalSettings.Values["LastVideoWatched"] = videoName;

// Допустим, что здесь возникает необрабатываемое исключение!

Int32 videoPosition = ...;
appData.LocalSettings.Values["LastVideoPosition"] = videoPosition;
```

В таком случае вполне возможно, что приложение может сохранить последнюю видеозапись, которую просматривал пользователь, а затем прекратить свою работу, прежде чем будет сохранено место, на котором пользователь остановился, просматривая фильм. Если в дальнейшем пользователь снова запустит приложение на выполнение, оно загрузит последнюю видеозапись, которую просматривал пользователь, но перейдет к последнему месту в предыдущей просмотренной им видеозаписи, что вряд ли ему понравится. Поэтому приведенный выше фрагмент кода можно переписать следующим образом, устранив ошибку с помощью типа данных ApplicationDataCompositeValue:

```
ApplicationDataCompositeValue compositeValue =
      new ApplicationDataCompositeValue {
   { "LastVideoWatched", videoName },
   { "LastVideoPosition", videoPosition }
};
appData.LocalSettings.Values["LastVideoInfo"] = compositeValue;

// здесь показано, как организуется чтение данных:
compositeValue = (ApplicationDataCompositeValue) appData.LocalSettings.
Values["LastVideoInfo"];
videoName = (String) compositeValue["LastVideoWatched"];
videoPosition = (Int32) compositeValue["LastVideoPosition"];
```

Теперь присваивание объекта типа ApplicationDataCompositeValue в настройках осуществляется по принципу “все или ничего”, т.е. необрабатываемое исключение означает, что при последующем запуске на выполнение приложение будет иметь в своем распоряжении данные о предыдущей или последней видеозаписи. Эти данные не могут быть отчасти старыми и отчасти новыми. В одном объекте типа ApplicationDataCompositeValue допускает хранить данные объемом до 64 Кбайт.

Для построения и отладки приложения полезно знать, каким образом настройки данных пакета постоянно сохраняются в Windows. Эти настройки сохраняются в файле Settings.dat куста системного реестра. Этот файл находится в каталоге %LocalAppData%\Packages\ИмяСемействаПакетов\Settings. Для просмотра и видоизменения настроек данных пакета этот файл можно загрузить с помощью утилиты RegEdit.exe. Для этого запустите утилиту **RegEdit.exe** на выполнение, выберите сначала узел HKEY_LOCAL_MACHINE, а затем команду File⇨Load Hive (Файл⇨Загрузить куст). Откройте файл Settings.dat в появившемся диалоговом окне Load Hive и присвойте кусту имя ключа. Затем разверните узел HKEY_LOCAL_MACHINE, выберите имя только что созданного ключа и исследуйте его значение. И хотя таким способом нетрудно

просматривать значения ключей системного реестра, все данные в них, к сожалению, отображаются в виде массивов байтов, что несколько затрудняет их чтение и редактирование.

С другой стороны, можно воспользоваться настольным приложением Package Explorer, упоминавшимся в главе 2, чтобы просмотреть все настройки данных пакета. В этом приложении для доступа к данным пакета используется метод `CreateForPackageFamily()` из общедоступного класса `Windows.Management.Core.ApplicationDataManager`.

Настольные приложения действуют довольно эффективно при доступе к данным пакета приложений для Магазина Windows. На их основе можно, например, создать инструментальное средство, экспортирующее и импортирующее данные пакета, что было бы удобно для неоднократного тестирования прикладного кода по заранее известному набору исходных данных. Кроме того, некоторые производители выпускают настольные диагностические средства, которыми потребители могут пользоваться для фиксации информации о своей среде, включая и данные пакета. Эту информацию потребители могут пересылать производителям, чтобы те помогли им решить возникающие у них затруднения. И как только данные пакета будут экспортированы, их можно импортировать на многие ПК, чтобы настроить все эти ПК одинаковым образом. Так, настольное приложение или сценарий может использоваться на предприятии для предварительной установки настроек конфигурации, символьных строк подключения в базе данных и прочих параметров приложения для Магазина Windows.

Папки хранения данных пакета

Ниже приведен пример кода, демонстрирующий запись и чтение данных в файл из папки локального хранения данных пакета.

```
// получить доступ к хранилищам данных пакета
ApplicationData appData = ApplicationData.Current;

// сохранить некоторых текст в файле:
StorageFile file =
   await appData.LocalFolder.CreateFileAsync("SomeAppData");
await FileIO.WriteTextAsync(file, "Here is some data to persist");

// затем прочитать этот текст из файла:
file = await appData.LocalFolder.GetFileAsync("SomeAppData");
String persistedData = await FileIO.ReadTextAsync(file);
```

В файлах сохраняются только массивы байтов, но имеется немало способов преобразовать расширенные типы данных в массивы байтов. Например, методы `WriteTextAsync()` и `ReadTextAsync()` из класса `FileIO`, вызываемые в приведенном выше примере кода, выполняют взаимное преобразование символьных строк и массивов байтов с использованием кодировки UTF-8. Подробнее о манипулировании файлами речь пойдет в главах 5 и 6.

В Windows не накладывается никаких ограничений ни на сами файлы данных пакета, ни на их количество. Эти файлы могут быть очень крупными (размером до 2^{64} байтов), занимая почти все дисковое пространство на пользовательском ПК. Для того чтобы выяснить, сколько дискового пространства занимают отдельные пакеты, выберите команду Settings⇨Change PC Settings⇨Search And Apps⇨App Sizes (Параметры⇨Изменить параметры ПК⇨Поиск и Приложения⇨Размеры приложений). Указываемый размер обычно составляет сумму из самого пакета и его папок.

Для целей отладки полезно знать, что все файлы хранения данных пакета находятся в следующих каталогах.

- Временные файлы **%LocalAppData%\Packages\ИмяСемействаПакетов\TempState**
- Локальные файлы **%LocalAppData%\Packages\ИмяСемействаПакетов\LocalState**
- Перемещаемые файлы **%LocalAppData%\Packages\ИмяСемействаПакетов\RoamingState**

Контроль версий данных пакета

Со временем, пересматривая свое приложение с целью создания новых версий его пакета, вы, скорее всего, обнаружите, что для данных пакета требуется изменить вид или формат пакета данных. Следовательно, для того чтобы пользователь мог перейти на новую версию пакета, вам придется заранее обновить данные пакета. А для того чтобы упростить манипулирование данными пакета, система связывает с ними номер версии (32-разрядное целое число без знака). По умолчанию данным пакета присваивается нулевой номер версии. Номер версии данных пакета запрашивается следующим образом:

```
var appDataVersion = ApplicationData.Current.Version;
```

Следует иметь в виду, что между контролем версий пакета и его данных не обязательно должна быть какая-то связь. Например, во всех трех первоначальных версиях пакета может использоваться нулевая версия данных пакета. А начиная с версии 4 пакета может быть решено перейти к первой версии данных пакета. В таком случае данные пакета придется обновить, если пользователь собирается сразу установить версию 4 пакета, минуя предыдущие.

Если вы решите обновить версию данных пакета, вам придется написать специальный метод, переводящий предыдущую версию данных пакета на новую версию. Этот метод должен выглядеть следующим образом:

```
private async void AppDataSetVersion(
      SetVersionRequest setVersionRequest) {
   // Для обновления файлов оставить этот метод объявляемым
   // как async, воспользоваться откладыванием и подождать
   // завершения методов файлового ввода-вывода.
   // Если же обновляются только настройки, удалить модификатор
   // async из объявления метода и код откладывания из его тела.
   var deferral = setVersionRequest.GetDeferral();
   switch (setVersionRequest.CurrentVersion) {
      case 0:
         // ЧТО СДЕЛАТЬ: ввести здесь код для перехода от нулевой
         // к последней версии данных пакета
         break;
      case 1:
         // ЧТО СДЕЛАТЬ: ввести здесь код для перехода от первой
         // к последней версии данных пакета
         break;
   }
   deferral.Complete();
}
```

Этот метод вызывается следующим образом:

```
const UInt32 appDataLatestVersion = 2;
if (ApplicationData.Current.Version < appDataLatestVersion)
   ApplicationData.Current.SetVersionAsync(
      appDataLatestVersion, AppDataSetVersion);
      .AsTask().GetAwaiter().GetResult();
```

Следует иметь в виду, что когда происходит возврат из метода обработки версии `AppDataSetVersion()`, система связывает номер последней версии с данными пакета, чтобы при последующих обращениях к свойству `Version` из класса `ApplicationData` возвращалась последняя версия. Кроме того, если метод обработки версии `AppDataSetVersion()` генерирует необрабатываемое исключение, система не связывает номер последней версии с данными пакета. Тем не менее некоторые данные пакета могут быть успешно обновлены до новой версии. Если же при обновлении данных пакета возникает неожиданное исключение, то, скорее всего, придется удалить все эти данные, вызвав метод `ClearAsync()` из класса `ApplicationData`, а затем перестроить все данные пакета сызнова.

Теперь возникает следующий вопрос: где разместить этот код? Наиболее подходящим для него местом служит фоновая задача, запускаемая системным триггером, завершающим обслуживание (подробнее об этом — в главе 9). Таким образом, когда пользователь устанавливает последнюю версию пакета, начинает выполняться фоновая задача, обновляющая данные пакета. Но если нет желания создавать фоновую задачу, то рассматриваемый здесь код можно разместить в самом приложении. В таком случае метод `SetVersionAsync()`, скорее всего, придется вызвать синхронно, как показано в предыдущем примере. Это приведет к блокировке дальнейшего исполнения потока, чтобы исключить доступ к данным пакета из остальных частей кода приложения до тех пор, пока обновление не завершится полностью.

А если данный код выполняется асинхронно, то его можно вызвать в приложении из метода `Main()` или из конструктора класса `App`, обеспечив обновление данных пакета при активизации приложения, независимо от того, каким образом она осуществляется[2]. Единственная трудность такого обновления версии данных пакета состоит в том, что оно выполняется при отображении начального экрана приложения, что несколько задерживает взаимодействие пользователя с приложением. Следовательно, если обновление данных пакета может отнять много времени, то придется выбрать вариант синхронного обновления данных. Но в этом случае следует принять меры, чтобы исключить доступ к данным из других потоков исполнения до тех пор, пока обновление не завершится полностью.

Для тестирования метода обработки версий `AppDataSetVersion()` можно очистить данные пакета и установить исходный номер их версии, перейдя к папке `Properties` текущего проекта в Visual Studio, выбрав панель Debugging (Отладка) и установив флажок Uninstall And Then Re-Install My Package (Удалить, а затем переустановить мой пакет).

[2] В C# не разрешается объявлять метод `Main()` или конструктор с ключевым словом `async`, поэтому использовать оператор await в них нельзя. Но это препятствие можно обойти, если вызвать метод `SetVersionAsync()` синхронно.

Перемещение данных пакета

Пользователи все чаще стали регулярно работать на нескольких ПК. Например, один ПК может быть у них дома, другой — на работе, а третий (планшетный) — в пути. Таким пользователям было вы крайне неудобно настраивать все свои ПК одинаково под каждое приложение, с которым они работают. Именно поэтому в Windows теперь предоставляется возможность перемещения любых настроек пакетов и файлов хранения для приложений из Магазина Windows. Пользователю достаточно настроить приложение лишь один раз, чтобы эта настройка распространилась на все ПК, где он установил пакет данного приложения. Кроме того, приложение может предложить пользователю непрерывное взаимодействие, когда он начинает операцию с приложением на одном ПК и продолжает ее на другом ПК. Допустим, пользователь запускает приложение для просмотра видеозаписей на своем настольном ПК, а затем берет свой планшетный компьютер и открывает на нем то же самое приложение. В таком случае воспроизведение видеозаписи должно быть продолжено с того места, где оно было остановлено на ПК. На самом деле основной способ синхронизировать текущие настройки приложения на нескольких ПК пользователя состоит в том, чтобы заблокировать его ПК.

Многие настройки в самой ОС Windows также перемещаются автоматически среди пользовательских ПК, например, фоновые изображения рабочего стола, избранные интернет-ресурсы, языковые и региональные настройки. Для перемещения настроек пользователи должны войти в систему на своих ПК под Windows, воспользовавшись своей учетной записью в корпорации Microsoft или связав с ней локальную или доменную учетную запись. Для связывания своей учетной записи в корпорации Microsoft с ОС Windows пользователям достаточно выбрать команду Settings⇨Change PC Settings⇨Settings⇨Your Account⇨Connect To A Microsoft Account (Параметры⇨Изменить параметры ПК⇨Ваша учетная запись⇨Связать с учетной записью в корпорации Microsoft).

Как и во всем остальном в Windows, пользователь сохраняет постоянный контроль над своими данными и взаимодействием с системой. В этой связи пользователь сам решает, какие именно данные следует перемещать, перейдя к панели Sync Settings по команде Settings⇨Change PC Settings⇨SkyDrive⇨Sync Settings (Параметры⇨Изменить параметры ПК⇨Служба SkyDrive⇨Настройки синхронизации), как показано на рис. 4.1. Бо*л*ьшая часть этих настроек управляет настройками операционной системы, но перемещением данных пакета управляют настройки приложения из раздела App Settings. Пользователи могут также указать, желают ли они воспользоваться потенциально дорогими сетевыми соединениями для перемещения данных (см. главу 7). Следует также иметь в виду, что на предприятиях администраторы могут настраивать или блокировать перемещение данных среди подключенных к домену машин с помощью групповой политики.

При наличии бесплатной поддержки реализовать перемещение настроек приложения проще простого. Для этого достаточно ввести настройки в свойство `RoamingSettings` из класса `ApplicationData` или соответствующие файлы в свойство `RoamingFolder` того же самого класса, а обо всем остальном позаботится ОС Windows. В частности, Windows автоматически переместит данные, как только будет установлено соединение с Интернетом, и надежно сохранит их на серверах, предоставляемых корпорацией

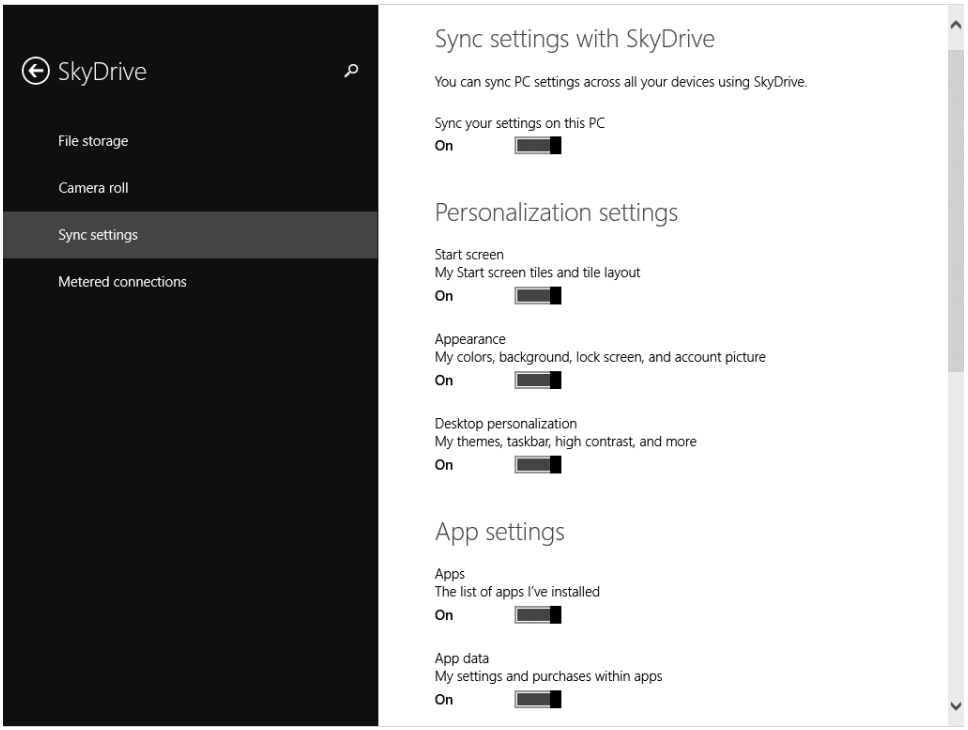

Рис. 4.1. *Пользователи управляют перемещением своих данных на панели* Sync Settings

Microsoft для постоянного хранения пользовательских данных[3]. Однако серверы корпорации Microsoft накладывают ограничение на объем сохраняемых данных пакета, которые можно перемещать. Чтобы выяснить этот объем, следует запросить свойство `RoamingStorageQuota` из класса `ApplicationData`. В настоящее время это свойство возвращает объем в 100 Кбайт. Если же данные пакета превышают эту предельную норму, Windows прекращает всякое их перемещение[4]. Поэтому на вас как на разработчика приложения возлагается обязанность заранее спланировать порядок использования хранилища для перемещаемых данных пакета. Это означает, что вы должны перемещать данные небольшими порциями. Например, вместо содержимого перемещайте ссылки на него. Если же требуется перемещать объем данных свыше разрешаемой предельной нормы, вам придется построить свою инфраструктуру перемещения (серверы, механизмы

[3] Данные пакета фактически сохраняются по учетной записи пользователя в службе SkyDrive, а пользователю разрешается не только хранить, но и получать непосредственный доступ к данным пакета, которые обслуживает служба SkyDrive.

[4] Не следует забывать, что Windows не ограничивает объем данных, сохраняемых в настройках или папках для перемещения, но накладывает ограничение на объем данных, которые могут быть синхронизированы с облачным хранилищем.

аутентификации и передачи данных и т.д.)[5]. Назначение функции перемещения данных, предоставляемой в Windows, — разрешить любому разработчику, студенту или просто любителю перемещать данные, не вдаваясь в подробности организации такой инфраструктуры.

Файлы некоторых типов не подлежат перемещению. Так, если разместить в папке для перемещения архивный файл с расширением **`.zip`** или **`.cab`**, он не будет перемещен. Это особенно хлопотно, поскольку многие разработчики стремятся уплотнить перемещаемые данные, чтобы вместить как можно больше данных в предельную норму в 100 Кбайт. Корпорация Microsoft не документирует список типов файлов, которые не подлежат перемещению. Если же файл, размещаемый в папке для перемещения, на самом деле не перемещается, попробуйте изменить его расширение.

Не храните пароли в данных пакетах. Вместо этого создайте объект типа `Windows.Security.Credentials.PasswordCredential` с нужным паролем, а затем введите этот объект в объект типа `Windows.Security.Credentials.PasswordVault`. Благодаря этому объект типа `PasswordCredential` безопасно перемещается среди пользовательских ПК. Однако, если объект типа `PasswordCredential` вводится в объект типа `PasswordVault` на ПК, подключенном к домену, этот объект не подлежит перемещению, чтобы учетные данные домена не вышли за его пределы в Интернет. Для просмотра учетных данных, постоянно сохраняемых для всех приложений, воспользуйтесь компонентом Credential Manager (Диспетчер учетных данных) в Windows.

Как правило, перемещаемые настройки пакета и соответствующие файлы синхронизируются в течение нескольких минут, но удачное завершение этого процесса не гарантируется из-за возможных сбоев в сети. Поэтому функция перемещения не распространяется на те случаи, когда два или более ПК используются параллельно или же когда данные передаются с одного ПК на другой. И если пользователь сначала изменяет перемещаемые данные на двух или более ПК, а затем восстанавливается соединение с Интернетом, то данные, записанные в последний раз, синхронизируются на всех ПК этого пользователя, причем старые данные перезаписываются. Как упоминалось ранее в соответствующем разделе, контроль версий данных пакета выполняется независимо от самого пакета. ОС Windows не перемещает новую версию данных пакета на ПК с прежней их версией, поскольку это может привести к тому, что прежняя версия приложения, скорее всего, перестанет нормально работать при последующей активизации.

ОС Windows не синхронизирует данные, когда пользователь выходит из системы, а ПК переходит в спящий режим, поскольку это может отрицательно сказаться на производительности. Но если на ПК поддерживается режим ожидания с подключением, то синхронизация в конечном итоге происходит, даже если ПК якобы отключен. А если пакет установлен только на одном ПК пользователя, то и в этом случае Windows синхронизирует данные приблизительно один раз в сутки. Это означает, что пользователь может

[5] С этой целью в приложении можно, в частности, воспользоваться прикладными программными интерфейсами API службы SkyDrive, чтобы явным образом синхронизировать данные пакета с хранилищем этой службы по учетной записи пользователя в ней. Но для это потребуется разрешение пользователя.

испортить или потерять такой ПК, приобрести вместо него новый, установить на нем пакет, после чего синхронизированные данные появятся на новом ПК. Если же пакет удаляется со всех ПК пользователя, перемещаемые данные этого пакета остаются в облачном хранилище в течение месяца, после чего удаляются. И если в течение этого месяца пользователь установит пакет на ПК, то на него будут перемещены данные пакета, постоянно находящиеся в облачном хранилище.

Если пользователю требуется очистить все данные (возможно, из соображений конфиденциальности) после удаления пакета из системы, он может перейти на веб-страницу по адресу `https://SkyDrive.Live.com/Win8PersonalSettingsPrivacy/` и щелкнуть на кнопке Remove (удалить). В итоге все данные, перемещенные пользователем, будут удалены из облачного хранилища, но перемещаемые данные тех пакетов, которые все еще установлены, будут просто скопированы ОС Windows обратно в облачное хранилище. Удаление данных из облачного хранилища может оказаться удобным на стадии разработки приложения, поскольку эта операция позволяет очистить любые неправильные данные, созданные во время разработки. Перемещаемые данные можно очистить и программно, вызвав метод `ClearAsync()` из класса `ApplicationData` и передав ему признак `ApplicationDataLocality.Roaming` в качестве параметра.

Иногда требуется очень быстро переместить какую-нибудь настройку с одного ПК на все остальные. Например, пользователь может перейти со своего настольного ПК на планшетный компьютер, чтобы сразу же продолжить просмотр видеозаписи. В таком случае синхронизация видеоданных в течение нескольких минут покажется пользователю слишком медленной. В качестве выхода из этого положения в Windows разрешается, чтобы какой-нибудь одной настройке в пакете был присвоен высокий приоритет для быстрой ее синхронизации (как правило, в течение одной минуты). Настройка перемещается очень быстро, если присвоить ей ключ с именем `"HighPriority"` (Высокий приоритет) и сохранить этот ключ в корневом объекте класса `ApplicationDataContainer`, как показано ниже.

```
// ПРИМЕЧАНИЕ: использовать объект типа ApplicationDataCompositeValue
// для быстрого перемещения значений нескольких настроек
ApplicationDataCompositeValue compositeValue =
      new ApplicationDataCompositeValue {
   { "LastVideoWatched", videoName },
   { "LastVideoPosition", videoPosition }
};
appData.LocalSettings.Values["HighPriority"] = compositeValue;
```

Следует избегать непрерывного обновления настройки `"HighPriority"`, а вместо этого обновлять ее в определенные моменты времени, когда, например, пользователь приостанавливает или прекращает воспроизведение видеозаписи; когда приложение приостанавливает свою работу; а возможно, и с промежутками один раз в минуту. Аналогично значение настройки иногда требуется прочитать при запуске приложения на выполнение, при возобновлении его работы или при наступлении события `DataChanged` из класса `ApplicationData` (подробнее об этом — далее, в разделе “Уведомления об изменениях в данных пакета”). Если значение настройки `"HighPriority"` равно `ApplicationDataCompositeValue`, как в приведенном выше примере, то оно не должно содержать данные объемом больше 8 Кбайт для быстрого перемещения. А если

оно содержит данные объемом больше 8 Кбайт, то эта настройка будет перемещена с той же быстротой, как и обычная настройка.

Для того чтобы помочь разработчикам приложений проверить данные, перемещаемые из их пакетов, корпорация Microsoft предлагает инструментальное средство контроля над перемещением данных Roaming Monitor, интегрированное в ИСР Visual Studio. Это инструментальное средство можно загрузить по адресу `http://VisualStudioGallery.MSDN.Microsoft.com/3ccf8c24-5e72-4ba0-b3e9-d822ca345fd0`. С его помощью можно контролировать, просматривать и манипулировать перемещаемыми настройками пакета, а также организовать перемещение данных по требованию. Аналогично можно принудить систему синхронизировать перемещаемые данные пакета с облачным хранилищем, выполнив следующую команду:

```
SchTasks.exe /run /i /tn Microsoft\Windows\SettingSync\
BackgroundUploadTask
```

После того как ПК выгрузит данные пакета в службу SkyDrive, ОС Windows отправит извещающее уведомление (подробнее об этом — в главе 8) на остальные ПК данного пользователя, принудив их загрузить последние данные пакета из службы SkyDrive. И наконец, для того чтобы упростить поиск неполадок, система протоколирует события, связанные с перемещением данных, в следующих системных журналах регистрации событий:

- Applications And Services Logs⇨Microsoft⇨Windows⇨SettingSync.
- Applications And Services Logs⇨Microsoft⇨Windows⇨PackageStateRoaming.

Уведомления об изменениях в данных пакета

Когда ОС Windows копирует перемещаемые данные пакета из облачного хранилища на все ПК пользователя, инициирует событие `DataChanged` из класса `ApplicationData`. И если это событие регистрируется в приложении, то последнее обновляется новыми перемещаемыми настройками, оперативно изменяя свое поведение по ходу взаимодействия с ним пользователя. Следует, однако, иметь в виду, что Windows инициирует это событие в потоке исполнения из пула потоков. Следовательно, если требуется обновить пользовательский интерфейс приложения, для этого придется воспользоваться классом `CoreDispatcher`.

Событие `DataChanged` автоматически инициируется в Windows всякий раз, когда на ПК загружаются новые перемещаемые данные пакета. Но это событие может быть инициировано и в самом приложении, если вызвать метод `SignalDataChanged()` из класса `ApplicationData`. Этот метод полезно вызывать в том случае, если настройка изменяется в одной части приложения, а другую его часть требуется уведомить, что настройки изменились, и поэтому в ней могут быть запрошены новые их значения. Это полезно сделать и в том случае, если приложению требуется уведомить об изменениях в настройках одну из фоновых задач, и наоборот, как поясняется в главе 9.