CLR INSIDE OUT

Isolated Storage In Silverlight 2

Justin Van Patten

Contents

In the August 2008 *MSDN Magazine* CLR Inside Out column, Andrew Pardoe gave a brief overview of isolated storage in Silverlight 2 ("Program Silverlight with the CoreCLR"). In this column, I'll provide more details on isolated storage in Silverlight along with some best practices for using it in your own apps.

Before diving into isolated storage, let me first explain what I/O functionality is available in Silverlight. Silverlight applications are partial-trust applications that run in a sandboxed environment inside a Web browser. As such, arbitrary access to the file system is not permitted. And for good reason—you wouldn't want random, potentially malicious Silverlight applications you come across while browsing the Web to be accessing your personal files or wreaking havoc with your system.

But there are valid scenarios where it is useful to allow Silverlight applications some amount of limited access to the file system, whether it's for reading files (with the end user's consent) or storing data locally on the client. Both scenarios are supported in Silverlight 2 in a limited and safe way: the former is enabled with the OpenFileDialog and the latter with isolated storage.

Like cookies, isolated storage provides the ability to store data on the client between application invocations. But unlike cookies, isolated storage is a full-fledged virtual file system, providing applications with the ability to create, read, write, delete, and enumerate files and directories inside the virtual file system. Isolated storage can be used in the same way as cookies, to maintain state and simple application settings, but it can also be used to save large amounts of data locally on the client.

Isolated storage isn't new to Silverlight; it has been part of the .NET Framework since v1.0. Today Silverlight ships with a simplified subset of the .NET Framework's isolated storage APIs, but it also includes new APIs that provide additional functionality along with some new concepts.

Isolation

As you may expect, one of the key facets of isolated storage is isolation. Isolated storage is composed of many different unique stores, each of which can be thought of as its own virtual file system. Paths cannot escape the bounds of the virtual file system, which effectively isolates the store from the rest of the file system. This prevents a malicious application from being able to access arbitrary files and directories on the disk—such as ..\..\..\..\Windows\System32—where it could cause damage or access sensitive information.

Silverlight applications have access to two different stores: a user + application store (or application store) that is isolated by user identity and application identity and a user + site store (or site store) that is isolated by user identity and site identity. Isolating stores based on the identity of the user, application, and site means that an application can only access the stores to which it is permitted access; it cannot access the stores of other applications.

The user identity that Silverlight employs is the same user identity of the underlying operating system. Silverlight isolates per user by storing all isolated storage data in the current user's local application data directory. **Figure 1** has more details on where isolated storage is located for each operating system.

Figure 1 Location of Isolated Storage in the File System

Operating System	Location in File System
Windows Vista	<SYSTEMDRIVE>\Users\<user>\AppData\LocalLow\Microsoft\Silverlight\is
Windows XP	<SYSTEMDRIVE>\Documents and Settings\<user>\Local Settings\Application Data\Microsoft\Silverlight\is
Mac OS X	/Users/<user>/Library/Application Support/Microsoft/Silverlight/is

The application store is a unique store per application. A Silverlight application can only access its application store; it cannot access any other application's application store. The application store is based on the identity of the user and the identity of the application. The identity of a Silverlight application is the full URL to the Silverlight application's XAP file. For example, http://microsoft.com/app.xap is the application identity of a Silverlight application hosted at http://microsoft.com/app.xap. The application identity is case insensitive, so both http://MICROSOFT.COM/app.XAP and http://microsoft.com/app.xap have the same identity. The application store isn't new to Silverlight; it has existed in the .NET Framework since v2.0. The following code shows how to create a file in the application's application store:

Copy Code
```
try {
    using (var store = IsolatedStorageFile.GetUserStoreForApplication())
    using (var stream = store.CreateFile("hello.txt"))
    using (var writer = new StreamWriter(stream)) {
        writer.Write("Hello World");
    }
}
catch (IsolatedStorageException) {
    // Isolated storage not enabled or an error occurred
}
```

Silverlight Isolated Storage Best Practices

When you use isolated storage, following these guidelines will help you avoid problems and make the most of the protection isolated storage provides.

- Wrap all calls to isolated storage within try/catch blocks to be resilient to potential IsolatedStorageExceptions, which can be thrown if isolated storage is disabled or if the store has been deleted.
- If your Silverlight application needs to store a lot of data in isolated storage, consider hosting it on its own site so that it won't affect other applications on the site and other applications won't affect it.
- If you have a group of Silverlight applications that need to share data on the client, host them on the same site.
- Keep isolated storage paths as small as possible to prevent the internal full path from reaching the 260-character limit.
- Encrypt sensitive data stored in isolated storage.
- Use IsolatedStorageSettings to store objects and simple settings in isolated storage.
- Use IsolatedStorageFile if you want to use file and stream-based APIs, are storing large amounts of data, or need fine-grained control over the contents of isolated storage.

The isolated storage APIs used here are located in the System.IO.IsolatedStorage namespace. You can get an instance of the application store by calling IsolatedStorageFile.GetUserStoreForApplication. If isolated storage is enabled (it is by default when Silverlight is installed), this will return an IsolatedStorageFile instance that can be used to work with the store. If the end user has disabled isolated storage, an IsolatedStorageException is thrown.

End users can enable/disable isolated storage, so it is important to always wrap the call to IsolatedStorageFile.GetUserStoreForApplication inside a try/catch block. In fact, IsolatedStorageException could be thrown from any isolated storage operation. This could happen if the end user deletes isolated storage while the application is using it (the next operation will result in an IsolatedStorageException). Plus, if multiple instances of the application are running (in multiple browser windows or tabs, for example) and one of the instances deletes the store, then the next isolated storage operation performed by one of the other application instances will throw an IsolatedStorageException.

The best practice is to wrap all calls to isolated storage inside try/catch blocks to guard against IsolatedStorageExceptions. I've wrapped this first code sample inside a try/catch block to illustrate this, but to keep things simple, the rest of the code samples will not show the try/catch block. Please consider all remaining code samples as being implicitly wrapped in a try/catch block. (For a rundown of best practices, see the sidebar "Silverlight Isolated Storage Best Practices.")

Now that you've seen the application store, let's look at the site store. The site store is a shared store to which all Silverlight applications from the same site have access. This enables applications from the same site to share data between them.

The same way Microsoft Office 2007 has common preferences across its suite of applications, you could imagine developing a suite of Silverlight applications that are all hosted on the same site and share a common set of preferences stored on the client. The site store is based on the identity of the user and the identity of the site. The site identity of a Silverlight application is similar to the application identity, but it only includes the scheme, host, and port of the URL, without the path to the XAP file. Like the application identity, the site identity is also case insensitive. The shared site store is new in Silverlight and is not available in the .NET Framework.

IsolatedStorageFile.GetUserStoreForSite can be used to get an instance of the site store. The following code shows this in action:

Copy Code

```
using (var store = IsolatedStorageFile.GetUserStoreForSite())
using (var stream = store.CreateFile("sharedhello.txt"))
using (var writer = new StreamWriter(stream)) {
    writer.Write ("Hello Shared World");
}
```

Another application with the same site identity could read the shared file with the following code:

Copy Code

```
using (var store = IsolatedStorageFile.GetUserStoreForSite()) {
    if (store.FileExists("sharedhello.txt")) {
        using (var stream = store.OpenFile("sharedhello.txt", FileMode.Open))
        using (var reader = new StreamReader(stream)) {
            string contents = reader.ReadToEnd(); // "Hello Shared World"
        }
    }
}
```

To better understand how a given URL to a Silverlight application maps to an application identity and site identity, refer to **Figure 2**. As you can see, even though the first four URLs are different, they all map to the same application identity and site identity. This is due to the fact that identities are case insensitive, query strings are not included, and port 80 is the default port for the HTTP scheme. However, if another port is specified, such as 8080, then the identity of the application is considered different. If the leading "www" from the host is omitted, the application identity is considered different because the host is different.

Figure 2 Silverlight Application URL and Associated Application and Site Identities

Application URL	Application Identity	Site Identity
http://www.microsoft.com/app.xap	HTTP://WWW.MICROSOFT.COM/APP.XAP	HTTP://WWW.MICROSOFT.COM
http://www.MICROSOFT.COM/app.XAP	HTTP://WWW.MICROSOFT.COM/APP.XAP	HTTP://WWW.MICROSOFT.COM
http://www.microsoft.com/app.xap?foo=bar	HTTP://WWW.MICROSOFT.COM/APP.XAP	HTTP://WWW.MICROSOFT.COM
http://www.microsoft.com:80/app.xap	HTTP://WWW.MICROSOFT.COM/APP.XAP	HTTP://WWW.MICROSOFT.COM
http://www.microsoft.com:8080/app.xap	HTTP://WWW.MICROSOFT.COM:8080/APP.XAP	HTTP://WWW.MICROSOFT.COM:8080
http://microsoft.com/app.xap	HTTP://MICROSOFT.COM/APP.XAP	HTTP://MICROSOFT.COM
https://www.microsoft.com/app.xap	HTTPS://WWW.MICROSOFT.COM/APP.XAP	HTTPS://WWW.MICROSOFT.COM
http://www.microsoft.com/demo.xap	HTTP://WWW.MICROSOFT.COM/DEMO.XAP	HTTP://WWW.MICROSOFT.COM

The secure HTTPS URL will also yield a different identity because the scheme is different (HTTPS instead of HTTP). The last URL listed in **Figure 2** will have a different application identity than the first four URLs in the table, but it will have the same site identity, which means these two apps could choose to share data with each other on the client by storing data in their shared site store.

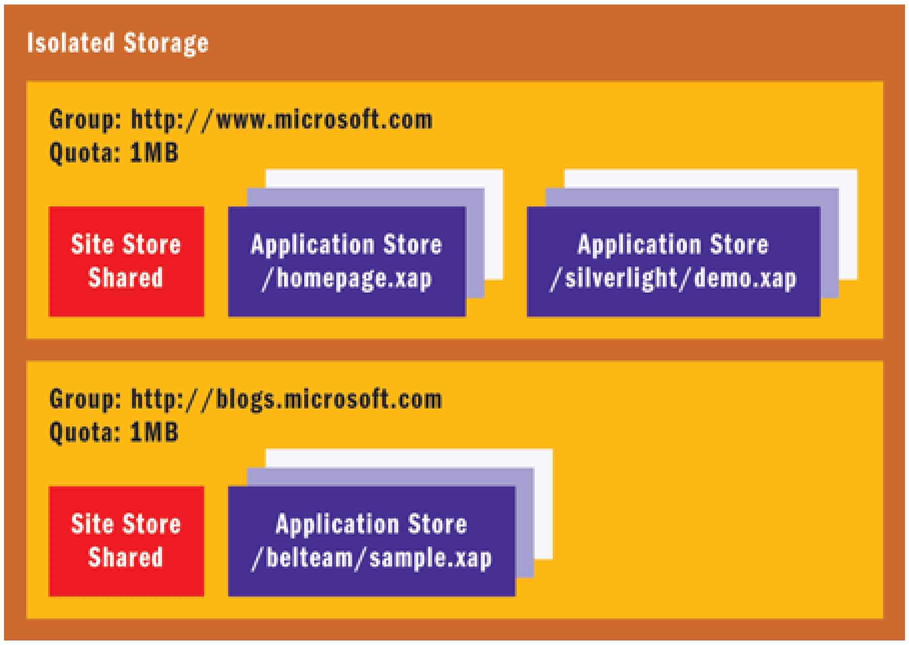
Isolated Storage
Group: http://www.microsoft.com
Quota: 1MB
Site Store Shared
Application Store /homepage.xap
Application Store /silverlight/demo.xap
Group: http://blogs.microsoft.com
Quota: 1MB
Site Store Shared
Application Store /belteam/sample.xap

Figure 3 **Isolated Storage Quota Groups**

Quota

A quota is a limit on the amount of isolated storage space that can be used by an application. This includes the data stored in isolated storage as well as some additional overhead for each directory and new file created (1KB each).

Unlike the .NET Framework, Silverlight uses a concept called quota groups for managing quotas in isolated storage. A quota group is a group of isolated stores that share the same quota. In Silverlight, stores are grouped by site, so all applications that have the same site identity share the same isolated storage quota. For example, all applications hosted on microsoft.com share the same quota.

Isolated storage can have zero or more quota groups. **Figure 3** shows two hypothetical quota groups: one for microsoft.com and another for blogs.microsoft.com. Each quota group has at most one shared site store and zero or more application stores.

By default, each quota group is assigned a default quota of 1MB. This should be enough space for applications to store simple settings and text files in isolated storage. However, since the quota is shared across all applications from the same site, the quota can easily be reached if just one of the applications uses a lot of space. If a site hosts several Silverlight applications and each application stores data in isolated storage, the amount of free space can diminish quickly. In addition, if an application needs to save a lot of data on the client, 1MB of free space won't always be enough.

To address this, if an application needs more space in isolated storage, it can request a larger quota. When an application requests a larger quota, Silverlight prompts the end user to allow or deny the request. The following code shows how to request a larger quota:

Copy Code

```
using (var store = IsolatedStorageFile.GetUserStoreForApplication()) {
    // 5 MB of isolated storage space is needed
    int spaceNeeded = 1024 * 1024 * 5;

    if (store.AvailableFreeSpace < spaceNeeded) {
        if (store.IncreaseQuotaTo(store.Quota + spaceNeeded)) {
            // The user accepted the request
        }
    }
}
```

In this code, the AvailableFreeSpace property is used to determine how much free space is available in isolated storage. The code knows it needs 5MB of free space. And if 5MB is not available, it requests a larger quota by calling IncreaseQuotaTo passing it the requested quota size in bytes (in this case the current quota plus the additional space that's needed).

The call to IncreaseQuota must be made from a user-initiated event, such as within a handler for a button-click event. If it is not called from a user-initiated event, Silverlight will not prompt the user and the request will be denied (the method will return false). If called from a user-initiated event and the end user allows the request, IncreaseQuotaTo returns true and the quota for the group is increased to the requested size.

Because quotas are per site in Silverlight, you may want to host your Silverlight application differently depending upon the needs of your site and application. For example, imagine you're creating a Silverlight video player that saves the user's top 10 favorite videos in isolated storage. Each video can be up to 10MB, which means the application will need to increase the quota to something more than 100MB. You could host the application on your main site (for example, contoso.com/player.xap), which may work fine if player.xap were the only Silverlight application hosted on the site. But if there are many other Silverlight applications on contoso.com, they'll all be competing for space in isolated storage.

To solve this, you may want to consider hosting the video player application on its own sub domain (such as player.contoso.com/player.xap). Similarly, if you plan to create a suite of Silverlight applications that need to share data, you'll want to host all of them on the same site and make sure they work well together.

Silverlight uses quota groups to make it easier for end users to manage the use of isolated storage on their computers. Instead of being presented with a laundry list of URLs to different Silverlight XAP files, end users are presented with a list of Web sites that are using isolated storage. This way, end users can quickly decide which Web sites they trust and which Web sites they do not trust to store data on their machines.

To end users, isolated storage is known as Application Storage. An end user can manage isolated storage using the Silverlight Configuration dialog. To open the dialog, right-click on any Silverlight application that's running in a Web browser and click Silverlight Configuration in the menu that pops up. Then click on the Application Storage tab in the Silverlight Configuration dialog. This tab lists the Web sites that are currently using isolated storage along with the current used size and quota for each. This list is essentially the list of quota groups. From here, end users can delete groups or enable and disable isolated storage.

Files and Directories

Creating directories and files in isolated storage is easy. Take a look at **Figure 4**. Here you'll see CreateDirectory, which is used to create directories. You can create top-level directories as well as subdirectories. To get a listing of the directories in a store, use GetDirectoryNames.

Figure 4 Creating Directories and Files

Copy Code

```
using (var store = IsolatedStorageFile.GetUserStoreForApplication()) {
    store.CreateDirectory(@"dir1");
    store.CreateDirectory(@"dir1\subdir1");
    store.CreateDirectory(@"dir1\subdir2");
    store.CreateDirectory(@"dir2");

    string[] topLevelDirs = store.GetDirectoryNames(); // { "dir1", "dir2" }
    string[] dir1SubDirs = store.GetDirectoryNames(@"dir1\*");
    // { "subdir1", "subdir2" }

    store.CreateFile(@"toplevel.txt");
    store.CreateFile(@"dir1\subdir1\subfile.txt");

    string[] files = store.GetFileNames(); // { "toplevel.txt" }
    string[] subfiles = store.GetFileNames(@"dir1\subdir1\*");
    // { "subfile.txt" }
}
```

There is also an overload of GetDirectoryNames that accepts a searchPattern string that supports single-character ("?") and multi-character ("*") wild cards. Notice how a multi-character ("*") wild card is used to get the directories contained inside "dir1". Creating files is similar to creating directories: files can be created in the root of the store or inside directories.

When creating files and directories in isolated storage, keep in mind that the relative isolated storage paths are expanded to full system paths internally (Silverlight applications never see these full paths). The full paths are limited to 260 characters on both Windows and Mac. So although your relative path inside isolated storage may only be 10 characters, those 10 characters will translate into a much longer path when appended to the root system path to where isolated storage is located on the file system. If the resulting system path ends up being more than 260 characters, an exception is thrown.

The location of isolated storage depends on the current user and operating system. Back in **Figure 1** you saw where isolated storage is located on Windows Vista, Windows XP, and Mac OS X.

The Microsoft\Silverlight\is directory contains the internal isolated storage structure. This is where information on quota groups, stores, bookkeeping data, and so on is stored, along with the actual data, by Silverlight. The specifics of the files and directories inside this directory are not important (these are implementation details that we reserve the right to change in future releases), but it's important to note that the directory structure does add additional overhead to the length of the full path.

For example, say a Silverlight application has created a file named foo.txt in the root of its application store. On Windows Vista, the full obfuscated system path to the file will look something like this:

Copy Code

```
C:\Users\<user>\AppData\LocalLow\Microsoft\Silverlight\is\plnvdo2y.zmg\
01ftptzg.h3o\1\s\4onbrsocvqffhjl0kn0sfxcidggvvqzoyl1to4ulrpif1vkwyaaahaa\f\foo.txt
```

On Windows XP it will look something like this:

Copy Code

```
C:\Documents and Settings\<user>\Local Settings\Application
Data\Microsoft\Silverlight\is\plnvdo2y.zmg\01ftptzg.h3o\1\s\
4onbrsodcvqffhjl0kn0sfxcidggvvqzoylto4ulrpif1vkwyaaahaa\f\foo.txt
```

And on Mac OS X it will look something like this:

Copy Code

```
/Users/<user>/Library/Application
Support/Microsoft/Silverlight/is/plnvdo2y.zmg/01ftptzg.h3o/1/s/
4onbrsodcvqffhjl0kn0sfxcidggvvqzyl1to4ulrpif1vkwyaaahaa/f/foo.txt
```

Assuming <user> is 10 characters, the full path length to foo.txt is 158 characters on Windows Vista, 190 characters on Windows XP, and 167 characters on Mac OS X. Thinking about it more generally, isolated storage paths are limited to 109 characters on Windows Vista, 77 characters on Windows XP, and 100 characters on Mac OS X assuming the current user's user name is 10 characters in length.

To be safe, the best practice is to keep isolated storage paths as small as possible to prevent the full path from reaching the 260-character limit.

One interesting thing to note about the locations listed in **Figure 1** is the fact that isolated storage is not part of the browser's temporary Internet files. This means that isolated storage is shared across all supported browsers, and when you (or your users, to be precise) clear out the temporary Internet files, the contents of isolated storage are not deleted.

This behavior is by design, because applications can store user-created data (such as documents and photos) in isolated storage along with other preferences and settings that a user might not want to be deleted along with the temporary Internet files. To delete the contents of isolated storage, use the Silverlight Configuration dialog as described previously.

One other thing to note is that although the location of isolated storage is obfuscated and stores are isolated from each other, the data stored in isolated storage is not protected (that is, it is not encrypted). This means that other (non-Silverlight) applications running on the machine may be able to access the data. If you need to store sensitive data in isolated storage, encrypt the data before storing it (there are encryption APIs available in Silverlight; see System.Security.Cryptography on the MSDN library.)

IsolatedStorageSettings

Silverlight includes a new convenience class called IsolatedStorageSettings that can be used to store objects in isolated storage quickly and easily. IsolatedStorageSettings is a dictionary collection that supports saving key-value pairs in either the application store or site store. It automatically takes care of serializing the objects and saving them in isolated storage. The serialized objects are saved to a file in isolated storage called __LocalSettings.

The code in **Figure 5** shows how to use IsolatedStorageSettings to save a simple object in the application store. Here I define a simple class called MySettings with three properties: Width, Height, and Color. Then I create an instance of MySettings and populate it with some values. Next, I add it to IsolatedStorageSettings.ApplicationSettings under the settings key.

Finally I call Save, which serializes the object and saves the serialized data to the application store. Retrieving the object happens to be just as easy:

Copy Code

```
MySettings settings;
if (!IsolatedStorageSettings.ApplicationSettings.TryGetValue("settings",
    out settings)) {
    // Settings was not previously stored in isolated storage
    // create a new object initialized with default values
    settings = new MySettings {
        Width = 100,
        Height = 100,
        Color = Colors.Orange
    };
}
```

This code uses the TryGetValue method to avoid an exception if the key is not found (meaning the data had not been previously saved). The code in **Figure 5** saves the object to the application store. To save objects to the site store, use IsolatedStorageSettings.SiteSettings instead of IsolatedStorageSettings.ApplicationSettings.

Figure 5 Saving an Object to the Application Store

Copy Code

```
public class MySettings {
    public int Width { get; set; }
    public int Height { get; set; }
    public Color Color { get; set; }
}
var settings = new MySettings {
    Width = 200,
    Height = 300,
    Color = Colors.Blue
};

IsolatedStorageSettings.ApplicationSettings["settings"] = settings;
IsolatedStorageSettings.ApplicationSettings.Save();
```

If you only need to store simple objects or settings in isolated storage, using IsolatedStorageSettings makes a lot of sense. But if you need more granular control over the format of data stored in isolated storage or if you are storing large amounts of data, you'll likely want to use the file and stream-based isolated storage APIs.

For more on isolated storage in Silverlight, see Cutting Edge in February 2009.

Send your questions and comments to clrinout@microsoft.com.

Justin Van Patten is a Program Manager on the CLR team at Microsoft where he works on the Base Class Libraries. He can be reached via the BCL team blog at blogs.msdn.com/bclteam.