Silverlight 4

پژوهش و نگارش: وحید نصیری
Vahid_Nasiri@Yahoo.Com
بهار ۱۳۸۹

فهرست مطالب

فصل ۱۴ – آشنایی با مفاهیم مرتبط با شیء Application و مدیریت آن

مقدمه

همانند برنامه‌های WPF ، در Silverlight نیز فایل‌های App.xaml / App.xaml.cs نقطه‌ی آغازین برنامه محسوب می‌شوند. در WPF فضای نام System.Windows.Application در فایل PresentationFramework.dll قرار دارد اما در Silverlight این فضای نام در اسمبلی System.Windows.dll مخصوص آن قرار گرفته است. در ادامه به بررسی بیشتر جزئیات شیء Application خواهیم پرداخت.

مقابله با خطاهای مدیریت نشده

امکان مقابله با خطاها در کدهای Silverlight به کمک عبارات try/catch میسر است. اما ممکن است قسمتی از کدهای ما بین عبارات try و catch محصور نشده باشند و در زمان اجرا سبب بروز استثنایی گردند. در این حالت Silverlight از کلاس استاندارد App.xaml.cs برای مدیریت این نوع خطاهای مدیریت نشده استفاده خواهد نمود. کدهای قسمتی از این فایل را که به این مورد مرتبط است، در ادامه مشاهده می‌نمائید :

App.xaml.cs

```
using System;
using System.Windows;

namespace SilverlightApplication61
{
    public partial class App
    {
        public App()
        {
           this.UnhandledException += Application_UnhandledException;
           InitializeComponent();
        }

        private void Application_UnhandledException(object sender,
```

```
                        ApplicationUnhandledExceptionEventArgs e)
        {
            if (!System.Diagnostics.Debugger.IsAttached)
            {
                e.Handled = true;
                Deployment.Current.Dispatcher.BeginInvoke(
                            () => ReportErrorToDOM(e));
            }
        }

        private void ReportErrorToDOM(
                     ApplicationUnhandledExceptionEventArgs e)
        {
            try
            {
                string errorMsg = e.ExceptionObject.Message +
                          e.ExceptionObject.StackTrace;
                errorMsg =
                  errorMsg.Replace('"', '\'').Replace("\r\n", @"\n");

                System.Windows.Browser.HtmlPage.Window.Eval(
    @"throw new Error(\"Unhandled Error in Silverlight Application " +
    errorMsg + "\");");
            }
            catch (Exception)
            {
            }
        }
    }
}
```

در این کلاس، سطر بعد در حین آغاز برنامه، متد رویدادگردان خطاهای مدیریت نشده را معرفی می‌نماید:

```
this.UnhandledException += Application_UnhandledException;
```

در این متد به صورت پیش فرض اگر حضور Debugger تشخیص داده شود، با تنظیم e.Handled به true سبب ادامه‌ی کار برنامه شده و سپس جزئیات خطای حاصل به صورت یک اسکریپت، به مرورگر مورد استفاده گزارش خواهد شد (شکل ۱).

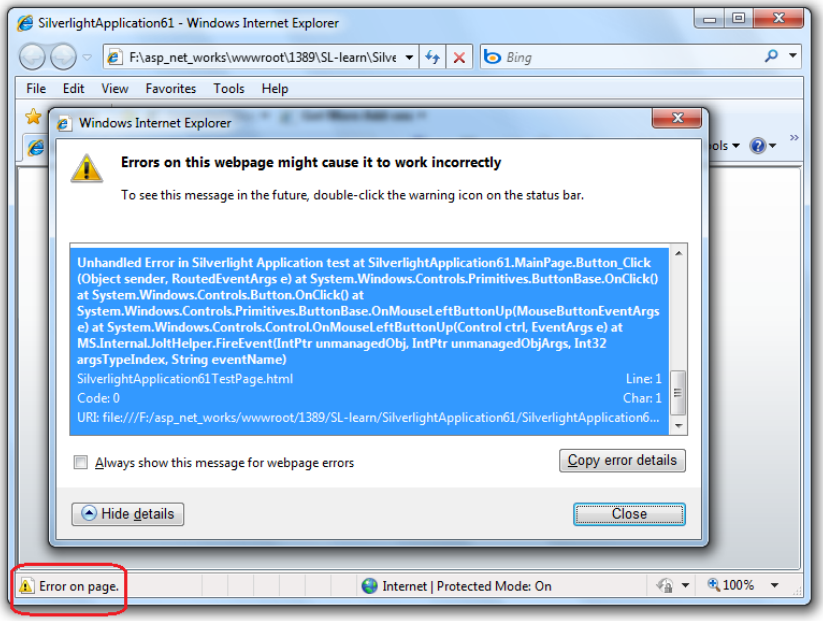
بدیهی است در اینجا اگر e.Handled به true تنظیم نشود، برنامه در همین لحظه خاتمه یافته و از کار خواهد افتاد. در این حالت کاربر برای ادامه‌ی کار با برنامه، مجبور خواهد شد بر روی دکمه‌ی Refresh صفحه‌ی جاری کلیک نماید تا برنامه از نو بارگذاری و نمایش داده شود.

اگر از فصل‌های قبل به خاطر داشته باشید، نحوه‌ی ارائه این خطا به کاربر در حین تعریف شیء Object نمایش افزونه‌ی Silverlight ، به کمک پارامتر onError و متد جاوا اسکریپتی منتسب به آن، مدیریت می‌گردد :

HTML

```
<object data="data:application/x-silverlight-2,"
   type="application/x-silverlight-2" width="100%" height="100%">
      ...
      <param name="onError" value="onSilverlightError" />
      ...
```

از آنجائیکه عموما برنامه‌های کاربردی و تجاری Silverlight به همراه یک WCF Service نیز ارائه می‌شوند، بهتر است متدی را در سمت سرور جهت ثبت خطاهای سمت کاربر برنامه‌ی Silverlight در یک بانک اطلاعاتی تعریف نمود و سپس از این امکانات در متد Application_UnhandledException استفاده کرد. همواره بررسی خطاهای ثبت شده در حین کار، در بالابردن کیفیت نهایی محصول بسیار مؤثر هستند.



شکل ۱– نمایش خطاهای مدیریت نشده به کاربر از طریق امکانات مرورگر

آشنایی با سایر خواص و رخدادهای شیء Application

دو روال رویدادگردان مهم دیگر نیز به صورت پیش فرض در فایل App.xaml.cs هر برنامه‌ی جدید Silverlight که توسط VS.NET ایجاد شده است، وجود دارد:

App.xaml.cs

```
using System;
using System.Windows;

namespace SilverlightApplication61
{
    public partial class App
    {
        public App()
        {
            this.Startup += this.Application_Startup;
            this.Exit += this.Application_Exit;
            InitializeComponent();
        }

        private void Application_Startup(object sender,
                        StartupEventArgs e)
        {
            this.RootVisual = new MainPage();
        }

        private void Application_Exit(object sender, EventArgs e)
        {

        }
    }
}
```

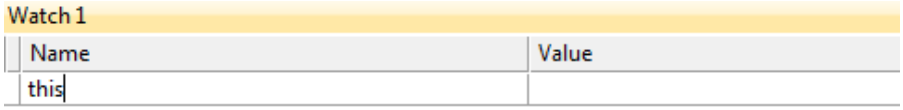
در روال رویدادگردان آغازین برنامه مهمترین عملیاتی که عموما انجام می‌شود تعیین صفحه‌ای است که آغاز کننده‌ی برنامه بوده و بدون ذکر آن، کاربران با یک صفحه‌ی خالی نمایش دهنده‌ی صفحه‌ی بارگذاری افزونه‌ی Silverlight روبرو گردیده و این صفحه محو نخواهد شد (شکل ۴).

روال Application_Exit در زمان خاتمه‌ی برنامه (بستن مرورگر و یا حتی مراجعه‌ی کاربر به یک صفحه‌ی دیگر) رخ داده و اجرا خواهد شد. از این متد می‌توان برای ذخیره سازی تنظیمات کاربر استفاده نمود.

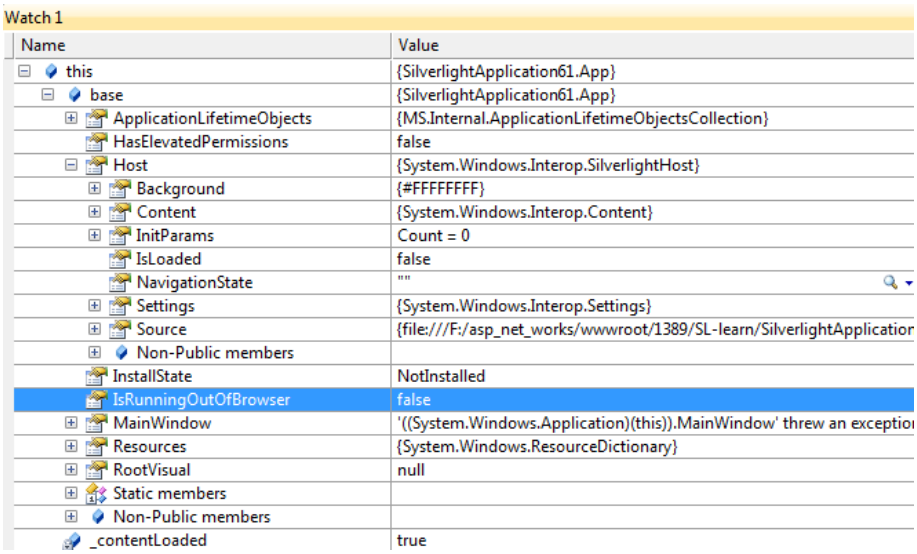
برای آشنایی بیشتر با خواص شیء this تعریف شده در فایل App.xaml.cs یا همان شیء Application ، بر روی سطر زیر در VS.NET یک Break point قرار داده و سپس در watch window ، عبارت this را وارد نموده و کلید Enter را فشار دهید (شکل‌های ۲ و ۳).

```
            this.RootVisual = new MainPage();
```

در اینجا خواصی مانند اینکه آیا برنامه در خارج از مرورگر در حال اجرا است (از ویژگی‌های جدید Silverlight 3 به بعد)، رنگ زمینه، Host جاری و سایر تنظیمات برنامه را می‌توان مشاهده نمود.



شکل ۲– مشاهده‌ی خواص شیء this توسط Watch window در VS.NET



شکل ۳– مرور کلیه خواص جاری منتسب به شیء Application در برنامه

در جدول بعد مروری خواهیم داشت بر مهم‌ترین خواص شیء Application در Silverlight :

Application Property	توضیحات
Current	این خاصیت Static ، دسترسی عمومی در سطح برنامه را به خواص شیء Application مهیا می‌سازد.
Host	اطلاعاتی را از میزبان افزونه‌ی Silverlight ارائه می‌دهد.
Resources	با کمک آن می‌توان به اطلاعات منابع تعریف شده در سطح برنامه دسترسی داشت. برای نمونه در فصل تعریف قالب برای برنامه‌های Silverlight از آن برای تغییر پویای قالب برنامه استفاده کردیم.

سفارشی سازی صفحه‌ی آغازین بارگذاری یک برنامه‌ی Silverlight

به صفحه‌ی آغازین بارگذاری یک برنامه‌ی Silverlight ، Splash screen و همچنین Loading screen نیز گفته می‌شود. در این قسمت قصد داریم این صفحه‌ی آغازین پیش فرض را (شکل ۴) مطابق سلیقه‌ی خود طراحی و پیاده سازی نمائیم.

شکل ۴– صفحه‌ی آغازین پیش فرض بارگذاری یک برنامه‌ی Silverlight

برای این منظور یک پروژه‌ی جدید Silverlight را به همراه Web Site میزبان آن آغاز نمائید. وجود این میزبان جهت تعریف صفحه‌ی آغازین سفارشی خود، نیاز خواهد بود.

برای اینکه این صفحه‌ی آغازین بارگذاری را بر روی کامپیوتر توسعه‌ی خود بهتر بتوانیم مشاهده کنیم، یا سطر مربوط به this.RootVisual را در فایل App.xaml.cs تبدیل به توضیحات و Comment نمائید یا یک فایل حجیم را به پروژه افزوده (یک فایل چند صد مگابایتی) و سپس در VS.NET به خواص فایل مراجعه کرده و Build Action آن‌را مساوی Content قرار دهید. پس از Compile و اجرای برنامه، صفحه‌ی آغازین بارگذاری را جهت بررسی بهتر مثال قسمت جاری می‌توان مشاهده نمود (بدیهی است این فایل حجیم را در زمان ارائه‌ی نهایی برنامه حذف خواهید نمود!).

اکنون جهت تعریف صفحه‌ی آغازین سفارشی، مطابق مراحل بعد عمل نمائید:

۱. یک فایل از نوع Text file به نام OrangeSplashScreen.XAML را به پروژه‌ی ASP.NET اضافه کنید. دقت داشته باشید که المان ریشه‌ی این فایل را نباید از نوع Use control تعریف نمود:

OrangeSplashScreen.XAML

```
<Grid
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Background="Orange">
    <!-- do not use UserControl for root element-->
    <TextBlock x:Name="progressTextBlock"
               HorizontalAlignment="Center">Loading...</TextBlock>
</Grid>
```

۲. به خواص این فایل در VS.NET مراجعه نموده و Build Action آن را به Content تغییر دهید.

۳. سپس فایل SilverlightApplicationTestPage.html را گشوده و دو پارامتر splashScreenSource و onSourceDownloadProgressChanged را مطابق کدهای زیر اضافه نمائید:

SilverlightApplication62TestPage.html

```
       ...
        function onSourceDownloadProgressChanged(sender, eventArgs) {
              sender.findName("progressTextBlock").Text =
                      "Loading: " +
                       Math.round((eventArgs.progress * 100))
                      + "%";
```

```
        }
       </script>

    </script>
</head>
<body>
    <form id="form1" runat="server" style="height:100%">
    <div id="silverlightControlHost">
      <object data="data:application/x-silverlight-2,"
        type="application/x-silverlight-2" width="100%" height="100%">
       ...
          <param name="splashScreenSource"
                 value="OrangeSplashScreen.XAML" />
          <param name="onSourceDownloadProgressChanged"
                 value="onSourceDownloadProgressChanged" />
       ...
```

در اینجا مسیر فایل سفارشی XAML تعریف شده و همچنین نحوه‌ی اعمال درصد تغییرات به TextBlock قرارگرفته در فایل XAML را ملاحظه می‌نمائید. به این ترتیب می‌توان صفحه‌ای را با طرح بندی کاملا دلخواه و سفارشی، ایجاد و همچنین از درصد بارگذاری رخ داده نیز آگاه شد.

شاید علاقمند باشید که در این صفحه یک نوار پیشرفت را نیز نمایش دهید. برای این منظور فایل OrangeSplashScreen.XAML را به صورت بعد ویرایش نمائید:

OrangeSplashScreen.XAML

```
<StackPanel
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Background="Orange">
    <!-- do not use UserControl for root element-->
    <TextBlock HorizontalAlignment="Center"
               x:Name="progressTextBlock"
               >Loading...</TextBlock>
    <Grid HorizontalAlignment="Center">
        <Rectangle x:Name="rectBorder" StrokeThickness="1"
              Stroke="Yellow"
              Height="7" Width="200" HorizontalAlignment="Left"/>
        <Rectangle x:Name="rectBar" Fill="Brown"
              Height="7" Width="0" HorizontalAlignment="Left" />
    </Grid>
</StackPanel>
```

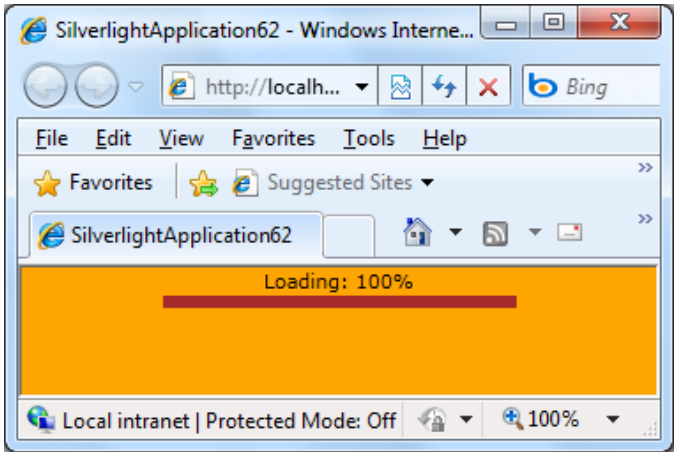
در اینجا از دو مستطیل جهت نمایش یک ProgressBar استفاده خواهیم نمود. سپس کدهای جاوا اسکریپتی مربوط به تغییر عرض این مستطیل‌ها بر اساس درصد پیشرفت بارگذاری به شرح بعد خواهند بود:

SilverlightApplication62TestPage.html

```
...
function onSourceDownloadProgressChanged(sender, eventArgs) {
      sender.findName("progressTextBlock").Text =
               "Loading: " + Math.round((eventArgs.progress * 100))
               + "%";

      var rectBar = sender.findName("rectBar");
      var rectBorder = sender.findName("rectBorder");
      if (eventArgs.progress)
        rectBar.Width = eventArgs.progress * rectBorder.Width;
      else
         rectBar.Width = eventArgs.get_progress() * rectBorder.Width;
}
...
```

برای مشاهده‌ی نتیجه‌ی کار، بر روی فایل تغییر یافته‌ی فوق یا همان SilverlightApplication62TestPage.html در VS.NET کلیک راست کرده و گزینه‌ی View in browser را انتخاب کنید. نتیجه‌ی آغازین حاصل باید همانند شکل ۵ باشد (بدیهی است اگر صفحه‌ی آغازین برنامه فایل aspx. متناظر است، این تغییرات را باید به آن فایل نیز اعمال نمود و یا از یک فایل جاوا اسکریپتی به عنوان قالب قابل بکارگیری مجدد در فایل‌های مختلف فوق استفاده نمائید).



شکل ۵- صفحه‌ی نمایش درصد بارگذاری اولیه سفارشی

ذخیره سازی اطلاعات برنامه بر روی کامپیوتر سمت مشتری

برنامه‌های Silverlight مجوز نوشتن در مکان‌های دلخواهی بر روی Hard disk کاربر یا خواندن اطلاعات از آن‌ها را ندارند. بدیهی است اگر این قابلیت فراهم می‌شد مدل امنیتی محدود مرورگرها زیر سؤال می‌رفت. اما خبر

خوش آن است که دو مدل متفاوت جهت خواندن و یا نوشتن اطلاعات در کامپیوتر سمت مشتری توسط برنامه‌های Silverlight مهیا است:

- استفاده از امکانی به نام Isolated storage بدون نیاز به کسب مجوز از کاربر
- استفاده از FileDialogs و کسب مجوز صریح از کاربر جهت خواندن و نوشتن فایل‌ها

با کمک Isolated storage تنها به قسمت بسیار کوچکی از hard disk سیستم می‌توان دسترسی داشت. محل این مکان ایزوله شده در اختیار برنامه‌ی Silverlight قرار نخواهد گرفت. همچنین به مکان ایزوله‌ی سایر برنامه‌های Silverlight که توسط URI آن‌ها مجزا می‌گردند دسترسی نداشته و علاوه بر آن به اطلاعات کاربران دیگری در همان فضای ذخیره سازی ایزوله شده نیز دسترسی نخواهیم داشت (تنها Windows administrators از این محدودیت اعمالی مستثنا هستند). ایجاد هر فضای ذخیره سازی ایزوله شده‌ی جدید به ازای URI فایل XAP برنامه و کاربر جاری سیستم صورت می‌گیرد. فضای مهیای در دسترس آن نیز یک مگابایت است که امکان افزایش آن تنها با مجوز کاربر تا ۱۰۰ مگابایت میسر می‌باشد (این فضای پیش فرض برای حالت اجرای در خارج از مرورگر، ۲۵ مگابایت درنظر گرفته شده است). Isolated storage را می‌توان جایگزینی برای Cookie های متداول برنامه‌های وب درنظر گرفت.

همانطور که ملاحظه می‌نمائید طراحی امنیتی این مکان ذخیره سازی ویژه با دقت فراوان صورت گرفته است و بهترین استفاده از آن ذخیره سازی تنظیمات برنامه به ازای هر کاربر، فعالیت‌های اخیر آن‌ها و یا ذخیره سازی اطلاعات فرم‌های کاربران و یا خطاهای حاصل و سپس ارسال آن‌ها به Web Service در زمانی که امکان اتصال مهیا است، می‌باشد. لازم به ذکر است که Isolated storage برخلاف Cookie ها تاریخ انقضاء نداشته و حتی با حذف فایل‌های موقتی مرورگرها نیز حذف نخواهند شد.

مکان مخفی ذخیره سازی Isolated Storage در سیستم عامل‌های مختلف را در جدول زیر می‌توانید ملاحظه نمائید :

سیستم عامل	محل قرارگیری
Windows Vista & 7	<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

برای دسترسی به فضایی خارج از این مکان محدود شده، کلاس‌های SaveFileDialog و OpenFileDialog ارائه شده‌اند که در هر دو مورد تحت نظارت و مجوز کاربر، امکان استفاده از آن‌ها وجود دارد.

نحوه‌ی استفاده از Isolated storage

در جدول زیر به صورت خلاصه و کاربردی نحوه‌ی استفاده از کلاس‌های مرتبط با Isolated storage بیان شده‌اند:

عملیات	نحوه‌ی انجام

ایجاد یک پوشه جدید.
از همین دست می‌توان به متدهای DeleteDirectory برای حذف یک پوشه، CreateFile و DeleteFile برای ایجاد و حذف یک فایل، Remove جهت حذف کامل isolated store ، OpenFile جهت گشودن یک فایل، FileExists و DirectoryExists جهت بررسی وجود یک فایل و پوشه، GetFileNames و GetDirectoryNames جهت دریافت اطلاعات نام فایل‌ها و پوشه‌ها نیز اشاره کرده که به همین ترتیب توسط شیء isf ایجاد شده در این مثال قابل دسترسی خواهند بود.
	`using System.IO;`
	`using System.IO.IsolatedStorage;`
	
	`public class IsolatedStorageHelper`
	`{`
	`  public static void CreateDirectory(string dir)`
	`  {`
	`  using (var isf =`
	
	`IsolatedStorageFile.GetUserStoreForApplication())`
	`  {`
	`  isf.CreateDirectory(dir);`
	`  }`
	`  }`
	`}`

مثالی تکمیلی در مورد ایجاد پوشه‌ها و بازیابی نام آن‌ها	`using (var store =`
	
	`IsolatedStorageFile.GetUserStoreForApplication())`
	`{`
	`  store.CreateDirectory(@"dir1");`
	`  store.CreateDirectory(@"dir1\subdir1");`
	`  store.CreateDirectory(@"dir1\subdir2");`
	`  store.CreateDirectory(@"dir2");`
	
	`  // { "dir1", "dir2" }`
	`  var topLevelDirs = store.GetDirectoryNames();`
	`  // { "subdir1", "subdir2" }`
	`  var dir1SubDirs =`
	`  store.GetDirectoryNames(@"dir1\*");`
	
	`  store.CreateFile(@"toplevel.txt");`
	`  store.CreateFile(@"dir1\subdir1\subfile.txt");`
	`  // { "toplevel.txt" }`
	`  var files = store.GetFileNames();`
	`  // { "subfile.txt" }`
	`  var subfiles =`
	`  store.GetFileNames(@"dir1\subdir1\*");`
	`}`

ذخیره سازی اطلاعات در فایل	`using System.IO;`
	`using System.IO.IsolatedStorage;`
	
	`public class IsolatedStorageHelper`
	`{`
	`  public static void SaveData(string data,`
	`  string fileName)`
	`  {`
	`  using (var isf =`
	
	`IsolatedStorageFile.GetUserStoreForApplication())`
	`  {`

``
        using (var isfs =
                new IsolatedStorageFileStream(
                   fileName, FileMode.Create, isf))
          {
            using (var sw = new StreamWriter(isfs))
            {
                 sw.Write(data);
            }
          }
      }
  }
}
``	

``
using System.IO;
using System.IO.IsolatedStorage;
 	
public class IsolatedStorageHelper
{
  public static string LoadData(string fileName)
  {
   using (var isf =
 	
IsolatedStorageFile.GetUserStoreForApplication())
      {
        using (var isfs =
               new IsolatedStorageFileStream(
                    fileName, FileMode.Open, isf))
        {
          using (var sr = new StreamReader(isfs))
          {
              return sr.ReadToEnd();
          }
        }
      }
  }
}
``
	بازیابی اطلاعات از یک فایل

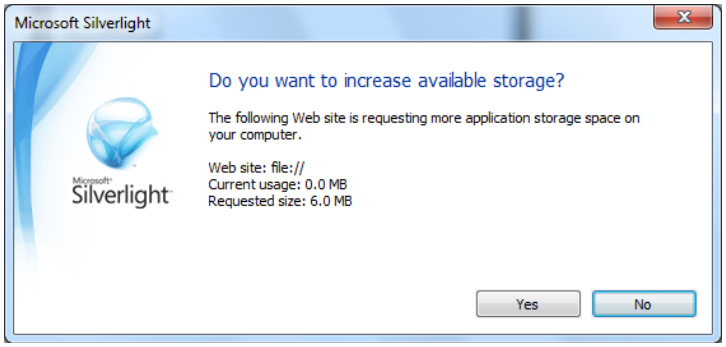
``
using (var store =
 	
IsolatedStorageFile.GetUserStoreForApplication())
{
    if (store.AvailableFreeSpace < 5000*1024)
    {
      if (store.IncreaseQuotaTo(
          store.Quota + 5000*1024 –
                 store.AvailableFreeSpace))
       {
           //The request succeeded.
           // User Has approved the quota Increase.
       }
       else
       {
           //The request failed.
           //User Has Not Approved the quota
Increase.
       }
    }
}
``
	درخواست فضای بیشتر از کاربر.
	در این مثال اگر فضای مهیا کمتر از 5000KB باشد، بر اساس میزان جیره بندی مشخص شده و فضای باقیمانده، فضای مورد نیاز جدیدی درخواست خواهد شد که کاربر باید صفحه‌ی خودکار درخواست افزایش میزان جیره بندی را تائید کند (شکل ۶). در غیر اینصورت امکان افزایش این فضا نخواهد بود.

``
IsolatedStorageSettings.ApplicationSettings["LastRunDate"] =
DateTime.Now;
IsolatedStorageSettings.ApplicationSettings["CurrentUser"] =
new Person(...);
IsolatedStorageSettings.ApplicationSettings["Key"] = Value;
IsolatedStorageSettings.ApplicationSettings.Save();
``
	ذخیره سازی تنظیمات برنامه.
	اگر کلید مورد استفاده وجود نداشته باشد به صورت خودکار ایجاد خواهد شد.

``
// Create an instance of IsolatedStorageSettings.
var userSettings =
``
	روش دیگر افزودن تنظیمات برنامه به کمک کلیدها و مقادیر

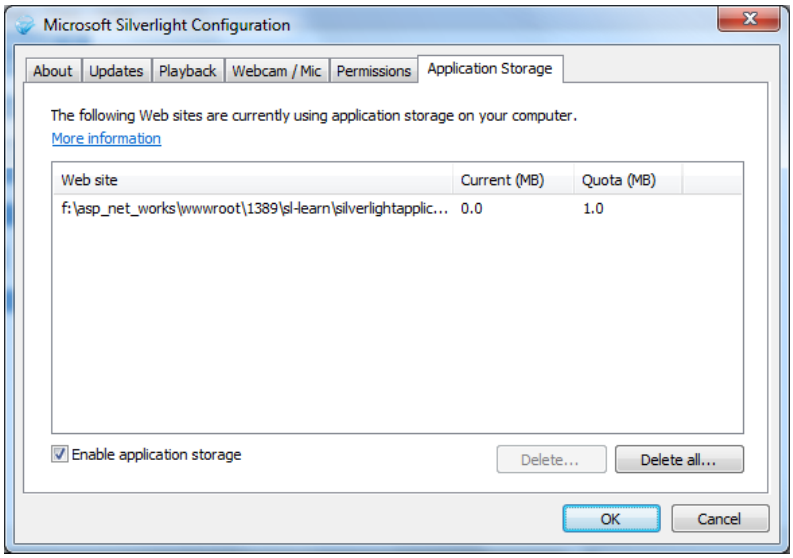
آن‌ها	`IsolatedStorageSettings.ApplicationSettings;`
	`// Add a key and its value.`
	`userSettings.Add("userImage", "BlueHills.jpg");`

بازیابی تنظیمات برنامه	`DateTime date = (DateTime)` `IsolatedStorageSettings.ApplicationSettings["LastRunDate"];`



شکل ۶– صفحه‌ی خودکار درخواست تائید و مجوز از کاربر جهت افزایش میزان جیره بندی Isolated storage .

اگر در زمان اجرای یک برنامه‌ی Silverlight ، بر روی صفحه کلیک راست کرده و از منوی ظاهر شده گزینه‌ی Silverlight را انتخاب نمائید، برگه‌ی آخر صفحه‌ی نمایان شده (شکل ۷)، بیانگر جزئیات بیشتری از میزان مصرف و حد مجاز ذخیره سازی هر کدام از برنامه‌های مرور شده توسط کاربر جاری، می‌باشد.



شکل ۷– مشاهده‌ی تنظیمات Application Storage افزونه‌ی Silverlight .

لازم به ذکر است که حین استفاده از ویژگی Isolated storage در کدهای خود، بکارگیری عبارات try/catch را فراموش نکنید؛ زیرا احتمال غیرفعال بودن آن و بسیاری حالات دیگر نیز میسر است.

بانک‌های اطلاعاتی مدفون شده مخصوص Silverlight

با کمک توانایی Isolated storage ، تعدادی بانک اطلاعاتی مدفون شده (embedded database engine) مخصوص Silverlight نیز توسط برنامه نویس‌ها جهت بکارگیری در سمت کاربر تهیه شده‌اند؛ که لیست آن‌ها را در جدول بعد ملاحظه خواهید نمود.

جدول ۱- بانک‌های اطلاعاتی مدفون شده قابل استفاده در Silverlight

بانک اطلاعاتی	صفحه‌ی خانگی
Siaqodb	http://siaqodb.com/
Silverlight Database	http://silverdb.codeplex.com/
EffiProz Database for Silverlight	http://effiprozsl.codeplex.com/
Ported SQLite to C#	http://code.google.com/p/csharp-sqlite/

نحوه‌ی استفاده از File Dialogs

زمانیکه از ویژگی Isolated storage استفاده می‌شود، جهت نوشتن و یا خواندن اطلاعات فایل‌ها، نیازی به کسب مجوز از کاربر نیست. اما اگر در برنامه نیاز به Upload یک فایل وجود داشت یا قرار است تا محتوایی بر روی Hard-Disk کامپیوتر کاربر در مکانی مشخص ذخیره گردد، چه باید کرد؟ در این موارد File Dialogs مانند OpenFileDialog و SaveFileDialog به Silverlight اضافه شده‌اند که در ابتدا مجوز لازم را از طرف کاربر دریافت کرده و سپس نسبت به خواندن و یا نوشتن فایل‌ها اقدام خواهند کرد. در هر دو حالت حاصلی که در اختیار برنامه نویس قرار می‌گیرد یک شیء Stream می‌باشد. در حالت استفاده از OpenFileDialog یک Stream فقط خواندنی جهت کار با فایل مورد نظر بازگشت داده شده و در هنگام بکارگیری SaveFileDialog، این Stream قابلیت نوشتن نیز خواهد داشت. علت ارائه‌ی Stream هم صرفا به دلایل امنیتی است. به این صورت دیگر نیازی به ارائه‌ی مسیر کامل فایل و یا امکان لیست کردن فایل‌های موجود در یک پوشه و مواردی از این دست، نخواهد بود.

لازم به ذکر است که امکان دسترسی به این صفحات ویژه در حالت تمام صفحه‌ی نمایش برنامه وجود نخواهد داشت. بنابراین بهتر است ابتدا ویژگی زیر را برای تشخیص وضعیت جاری برنامه بررسی نمود:

Application.Current.Host.Content.IsFullScreen

خواندن اطلاعات فایل‌ها با استفاده از کلاس OpenFileDialog

کلاس OpenFileDialog امکان نمایش صفحه استاندارد انتخاب و گشودن فایل‌را میسر می‌سازد. دو ویژگی آن نسبت به نمونه‌ی مشابه موجود در دنیای HTML حائز اهمیت است (تگ استاندارد <input type="file"/>):

- امکان مشخص سازی پسوندهای فایل‌های مجاز قابل انتخاب، وجود دارد (معرفی Filter).
- امکان انتخاب چندین فایل با هم نیز میسر است.

در جدول بعد یک سری از اعمال متداول با کلاس OpenFileDialog بیان شده‌اند :

عملیات	نحوه‌ی انجام

خواندن اطلاعات یک فایل متنی	//using System.IO;
	
	var dialog = new OpenFileDialog
	{
	    //مشخص سازی فیلتر مورد نظر
	    Filter = "Text Files (*.txt)|*.txt"
	};
	
	//اگر فایل انتخاب شده بود
	if (dialog.ShowDialog() == true)
	{
	    //خواندن اطلاعات استریم آن
	    using (var reader = dialog.File.OpenText())
	    {
	       string data = reader.ReadToEnd();
	    }
	}

خواندن اطلاعات چندین فایل متنی.
در اینجا خاصیت Multiselect به true تنظیم شده است و توسط شیء dialog.Files می‌توان به تک تک فایل‌ها دسترسی داشت.
	//using System.IO;
	
	var dialog = new OpenFileDialog
	{
	  Filter = "Text Files (*.txt)|*.txt",
	  Multiselect = true
	};
	// Show the dialog box.
	if (dialog.ShowDialog() == true)
	{
	  foreach (FileInfo file in dialog.Files)
	  {
	    using (var reader = file.OpenText())
	    {
	      string data = reader.ReadToEnd();
	    }
	  }
	}

خواندن اطلاعات فایل‌هایی با محتوای Binary (مانند تصاویر و امثال آن)
در اینجا بجای OpenText از OpenRead استفاده شده است.
	//using System.IO;
	
	var dialog = new OpenFileDialog
	{

```
                Filter = "All files
(*.*)|*.*",
                Multiselect = true
             };
// Show the dialog box.
if (dialog.ShowDialog() == true)
{
  foreach (FileInfo file in
dialog.Files)
   {
     using (Stream fileStream =
file.OpenRead())
     {
         //1 KB block at a time.
         byte[] buffer = new
byte[1024];
         int count;
         do
         {
             count =
fileStream.Read(buffer, 0,

buffer.Length);
         } while (count > 0);
      }
   }
}
```

برای نمایش OpenFileDialog توسط الگوی MVVM می‌توان از روش مشابه عنوان شده در هنگام معرفی کلاس Messenger مربوط به MVVM Light toolkit استفاده کرد. بدیهی است در این الگو، ViewModel برنامه نباید ارجاع مستقیمی را به عناصر UI داشته باشد.

ایجاد فایل‌ها با استفاده از کلاس SaveFileDialog

پس از آشنایی با کلاس OpenFileDialog، استفاده از کلاس SaveFileDialog ساده می‌باشد. در اینجا نیز به دلایل امنیتی صرفا یک Stream جهت نوشتن اطلاعات مورد نظر در اختیار شما قرار می‌گیرد. همچنین مجاز به تعیین نام پیش فرض در SaveFileDialog نخواهیم بود. مثالی در مورد نحوه‌ی استفاده از این کلاس در ادامه ذکر شده است:

C#

```
//using System.IO;

var saveDialog = new SaveFileDialog
                 {
                    Filter = "Text Files (*.txt)|*.txt"
                 };
if (saveDialog.ShowDialog() == true)
{
```

```
        using (var stream = saveDialog.OpenFile())
  {
     using (var writer = new StreamWriter(stream))
     {
        writer.Write("some data...");
     }
  }
}
```

در اینجا پس از پایان کار با استفاده از خاصیت SafeFileName شیء saveDialog، نام انتخابی وارد شده توسط کاربر را می‌توان تشخیص داد (بدون دسترسی به اطلاعات مسیر نهایی آن به دلایل امنیتی).