Introduction and best practices for IsolatedStorageSettings

This article explains how to use IsolatedStorageSettings and illustrates a helper class which enables using saving & retrieving (many) settings parameters easily.

Introduction

IsolatedStorageSettings is a dictionary for permanent storing values in Isolated Storage in the form of *key-value pair*. All that you have to do to store a value - is to provide a key for this value and a value itself.

You can store anything you want that is serializable, for example: user settings, layout information or application state. All settings from `IsolatedStorageSettings` are accessible after application shutdown and run again.

In this tutorial we will create a simple application that keeps one boolean value and then we are going to extend it for more convenient use.

Storing one setting with IsolatedStorageSettings

1. Run Visual Studio 2012
2. Create new Windows Phone project (7.1 or 8.0)
3. Navigate to **MainPage.xaml** and replace all control content with one checkbox:

```
<phone:PhoneApplicationPage x:Class="IsolatedStorageSample.MainPage"
                            xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                            xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                            xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
                            FontFamily="{StaticResource PhoneFontFamilyNormal}"
                            FontSize="{StaticResource PhoneFontSizeNormal}"
                            Foreground="{StaticResource PhoneForegroundBrush}">
    <CheckBox Content="Switch option"
              Name="chck"
              Checked="chck_Checked"
              Unchecked="chck_Unchecked"
              VerticalAlignment="Top" />
</phone:PhoneApplicationPage>
```

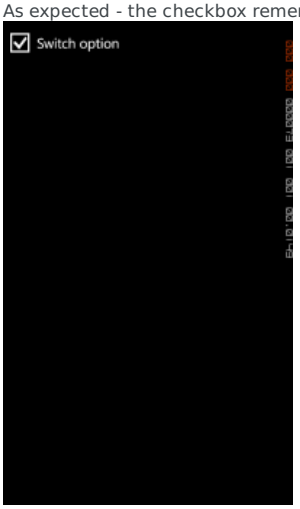
4. Navigate to **MainPage.xaml.cs** and insert code for storing and retrieving the value:

```
using System.IO.IsolatedStorage;
using System.Windows;

namespace IsolatedStorageSample
{
    public partial class MainPage
    {
        /// <summary>
        /// Object for sync access to IsolatedStorage
        /// </summary>
        private readonly object _sync = new object();

        private const string SwitchKeyName = "switch";

        // Constructor
        public MainPage()
        {
            InitializeComponent();
            //checking if property exists, if no - create it
            if (!IsolatedStorageSettings.ApplicationSettings.Contains(SwitchKeyName))
            {
                IsolatedStorageSettings.ApplicationSettings[SwitchKeyName] = true;
                SaveSettings();
            }

            //the checkbox state initializing
            chck.IsChecked = (bool) IsolatedStorageSettings.ApplicationSettings[SwitchKeyName];
        }

        private void chck_Checked(object sender, RoutedEventArgs e)
        {
            IsolatedStorageSettings.ApplicationSettings[SwitchKeyName] = true;
            SaveSettings();
        }

        private void chck_Unchecked(object sender, RoutedEventArgs e)
        {
            IsolatedStorageSettings.ApplicationSettings[SwitchKeyName] = false;
            SaveSettings();
        }

        /// <summary>
        /// Saving settings to isolated storage
```


(C) Copyright Nokia 2013. All rights reserved.

```
        /// </summary>
        private void SaveSettings()
        {
            lock (_sync)
            {
                IsolatedStorageSettings.ApplicationSettings.Save();
            }
        }
    }
}
```

5. Run the application
   As expected - the checkbox remembers last state before application shutdown.



Why do I use thread synchronization? Because I do not know where else I will decide to call `SaveSettings` in the future, it could be `ThreadPool`, Timer callback or anything else. `IsolatedStorageSettings` is easy to use.

But if you are going to have more than just one setting-value - syncing, keeping key name, checking if setting exists and saving each time it has been updated may become a headache. Better way is to implement all this functionality in a helper class only once.

Helper class for easy storage of more setting values

Now we are going to create new class and implement logic for storing and retrieving setting values. With this, it should save the setting values every time we update it (in a thread safe manner), it should automatically set the default value if setting does not exist and finally it should keep the key name - we have to provide it only once at property declaration.

We create two new classes: `IsolatedStoragePropertyHelper` and `IsolatedStorageProperty` as illustrated below.

```
using System.IO.IsolatedStorage;

namespace IsolatedStorageSample
{
    /// <summary>
    /// Helper class is needed because IsolatedStorageProperty is generic and
    /// can not provide singleton model for static content
    /// </summary>
    internal static class IsolatedStoragePropertyHelper
    {
        /// <summary>
        /// We must use this object to lock saving settings
        /// </summary>
        public static readonly object ThreadLocker = new object();

        public static readonly IsolatedStorageSettings Store = IsolatedStorageSettings.ApplicationSettings;
    }

    /// <summary>
    /// This is wrapper class for storing one setting
    /// Object of this type must be single
    /// </summary>
    /// <typeparam name="T">Any serializable type</typeparam>
    public class IsolatedStorageProperty<T>
    {
        private readonly object _defaultValue;
        private readonly string _name;
        private readonly object _syncObject = new object();

        public IsolatedStorageProperty(string name, T defaultValue = default(T))
        {
            _name = name;
            _defaultValue = defaultValue;
        }

        /// <summary>
        /Determines if setting exists in the storage
        /// </summary>
        public bool Exists
        {
            get { return IsolatedStoragePropertyHelper.Store.Contains(_name); }
        }

        /// <summary>
        /Use this property to access the actual setting value
        /// </summary>
        public T Value
        {
            get
            {
```


(C) Copyright Nokia 2013. All rights reserved.

```
            //If property does not exist - initializing it using default value
            if (!Exists)
            {
                //Initializing only once
                lock (_syncObject)
                {
                    if (!Exists) SetDefault();
                }
            }

            return (T) IsolatedStoragePropertyHelper.Store[_name];
        }
        set
        {
            IsolatedStoragePropertyHelper.Store[_name] = value;
            Save();
        }
    }

    private static void Save()
    {
        lock (IsolatedStoragePropertyHelper.ThreadLocker)
        {
            IsolatedStoragePropertyHelper.Store.Save();
        }
    }

    public void SetDefault()
    {
        Value = (T) _defaultValue;
    }
  }
}
```

You can extend this classes as you want (may be you want to save values only when application is shutting down?) or you can just copy this classes to your project and start working right now. Using `IsolatedStorageProperty` class is easier then `IsolatedStorageSettings`:

```
namespace IsolatedStorageSample
{
    using System.Windows;

    public partial class MainPage
    {
        //Setting declaration
        static readonly IsolatedStorageProperty<bool> SwitchProperty = new IsolatedStorageProperty<bool>("switch",true);

        public MainPage()
        {
            InitializeComponent();
            chck.IsChecked = SwitchProperty.Value;
        }
        private void chck_Checked(object sender, RoutedEventArgs e)
        {
            SwitchProperty.Value = true;
        }

        private void chck_Unchecked(object sender, RoutedEventArgs e)
        {
            SwitchProperty.Value = false;
        }
    }
}
```

We still have only one setting but a lot of lines are gone! - With ten or more property-settings this class become indispensable.

Lets create another setting - `FirstRunTimeProperty`:

```
using System;

namespace IsolatedStorageSample
{
    using System.Windows;

    public partial class MainPage
    {
        static readonly IsolatedStorageProperty<bool> SwitchProperty = new IsolatedStorageProperty<bool>("switch",true);

        /// <summary>
        /// Keeps first run time
        /// </summary>
        static readonly IsolatedStorageProperty<DateTime> FirstRunTimeProperty = new IsolatedStorageProperty<DateTime>("firstruntime

        public MainPage()
        {
            InitializeComponent();
            chck.IsChecked = SwitchProperty.Value;
            Loaded+=OnLoaded;
        }

        private void OnLoaded(object sender, RoutedEventArgs routedEventArgs)
        {
            //MessageBox.Show - could freeze main thread for a long time
            //Dont let it freeze the constructor
            MessageBox.Show(FirstRunTimeProperty.Value.ToString());
            //This MessageBox always shows time of first run
        }

        private void chck_Checked(object sender, RoutedEventArgs e)
        {
            SwitchProperty.Value = true;
        }

        private void chck_Unchecked(object sender, RoutedEventArgs e)
        {
            SwitchProperty.Value = false;
        }
    }
}
```

`FirstRunTimeProperty` let us know when was first application run.

(C) Copyright Nokia 2013. All rights reserved.

You can run the application two or more times to test it.

A good practice is to create a special static class, that would contain all your settings in it, so you can access any property in a program at any time:

```
namespace IsolatedStorageSample
{
    public static class SettingsContainer
    {
        public static readonly IsolatedStorageProperty<bool> TestProperty1
            = new IsolatedStorageProperty<bool>("TestProperty1");
        public static readonly IsolatedStorageProperty<string> TestProperty2
            = new IsolatedStorageProperty<string>("TestProperty2", "This is a test property");
        //...
    }
}
```


(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.