Application- and User Settings in .Net

Agenda

- Settings Overview
- .Net's BCL and VS support for settings
- WPF's Application Object's settings service
- System Settings

What Is a Setting?

- The .NET Framework allows you to create and access values that are persisted between application execution sessions
- These values are called *settings*
  - Settings can represent user preferences
  - Or other valuable information the application needs to use
    - E.g. the connection string that specifies a database
- Settings Files
  - One or more settings are stored together in a settings file, and this file is stored in a special folder
    - E.g. **`C:\Users\per\AppData\Local\`**

User name

A Setting Has Four Properties

- **Name:**
  - is used to access the value of the setting at run time
- **Type:**
  - A setting can be of any type
    - E.g. int, string, Color, Size or a user defined type
- **Scope:**
  - There are two possible values for the **Scope** property:
    - **Application**
    - **User**
- **Value:**
  - the value returned when the setting is accessed
  - The value will be of the type represented by the **Type** property

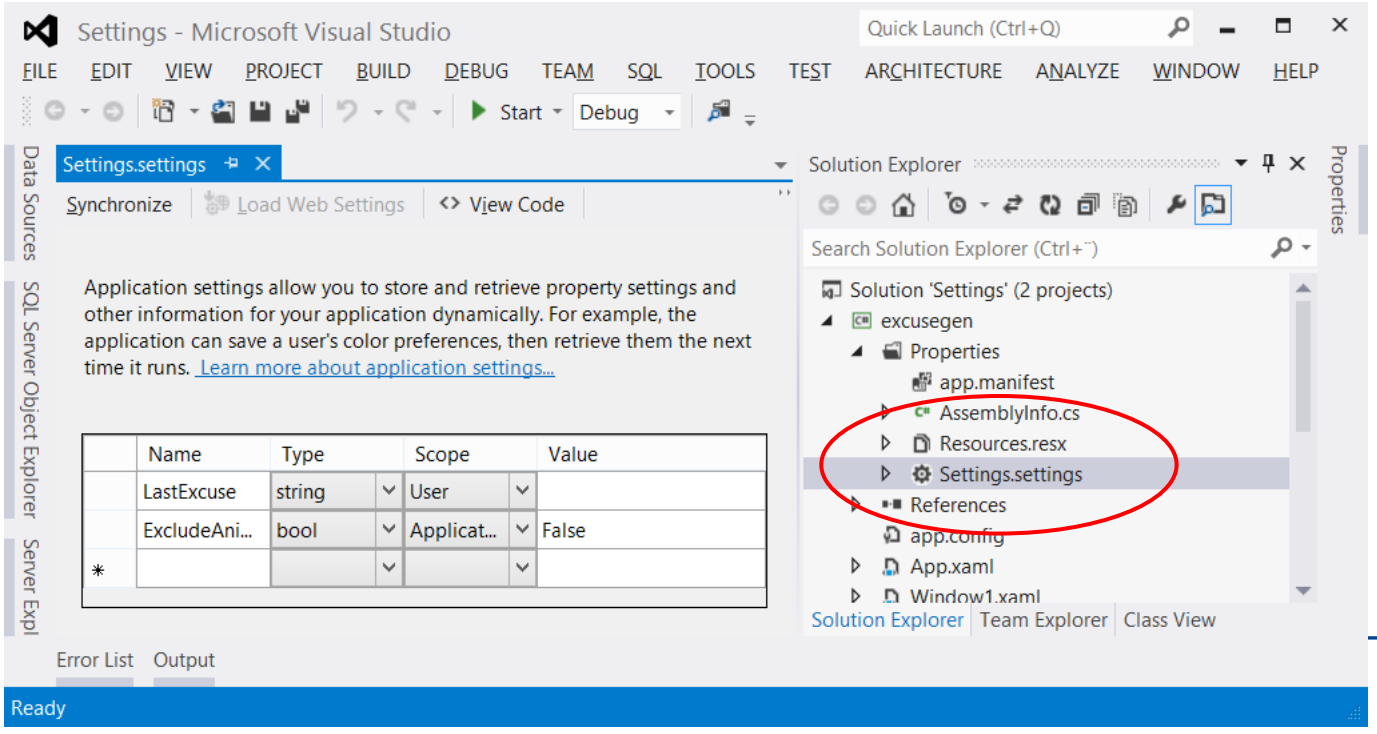
Scope

- **Application**
  - Settings with application scope represent settings that are used by the application regardless of who the user is
  - Are read-only from code at run time
  - Can only be changed at design time, or by altering the settings file manually
- **User**
  - Settings with user scope are generally less important to the actual application and are more likely to be associated with user preferences or other non-critical values
  - Are read/write from code at run time

.NET'S BCL AND VS SUPPORT FOR SETTINGS

Settings

- The preferred settings mechanism for .Net applications is the one provided by .NET BCL and Visual Studio:
  - The BCL has the ApplicationSettingsBase class from the System.Configuration namespace
  - And Visual Studio has a built-in tool for creating new settings
  - To access the settings for your application, click on the Settings tab in your project properties, or double-click the Settings.settings file



Using Settings

- To read a setting:
```
excuseTextBlock.Text = Properties.Settings.Default.LastExcuse;
```
- To change a setting:
```
Properties.Settings.Default.LastExcuse = excuses[i++];
```
  - Only user settings can be changed at runtime!
- To Save user settings between sessions:
```
Properties.Settings.Default.Save();
```

How does it Work?

- The Settings Designer manages a xml settings file and generates a class that allows you to program against the settings
- This file is named:

  <*AppName*>.exe.config and/or **user.config**

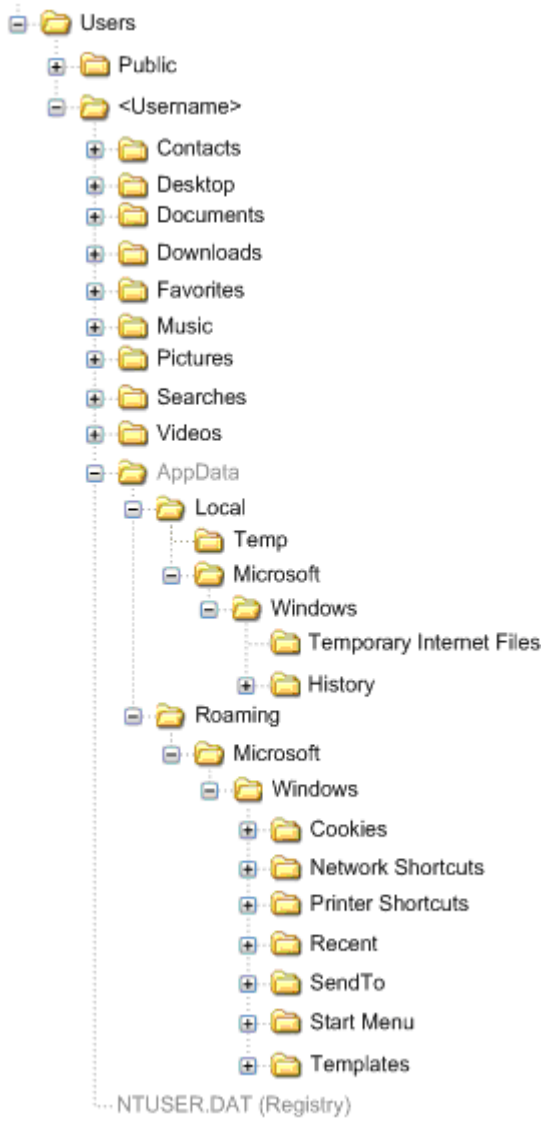
```
<userSettings>
        <excusegen.Properties.Settings>
            <setting name="LastExcuse" serializeAs="String">
                <value />
            </setting>
        </excusegen.Properties.Settings>
    </userSettings>
    <applicationSettings>
        <excusegen.Properties.Settings>
            <setting name="ExcludeAnimalExcuses"
serializeAs="String">
                <value>False</value>
            </setting>
        </excusegen.Properties.Settings>
    </applicationSettings>
```

Where are the settings kept?

- All application settings and default values for user settings are stored in the file **<*AppName*>.exe.config** located together with the application
  - typical in a sub folder to Program Files

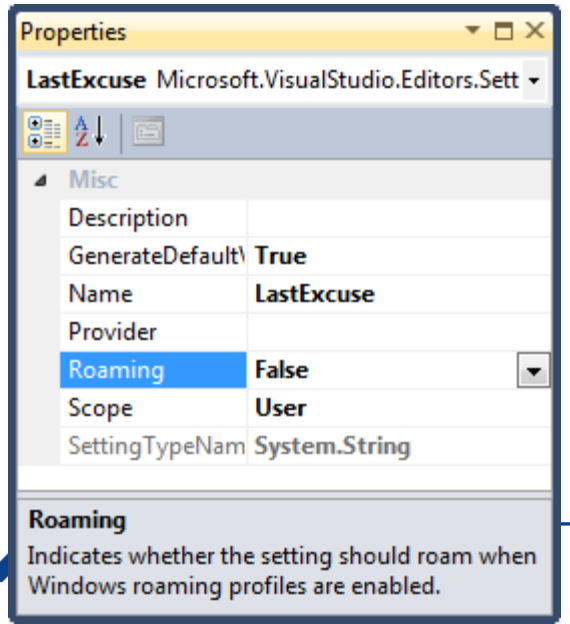
- User Settings are stored in a file named **user.config**
  and this file is stored in the user data path
  - E.g. **`C:\Users\per\AppData\Local\?`**
  - You seldom need to know exactly where it is stored, but if you do se the next slides

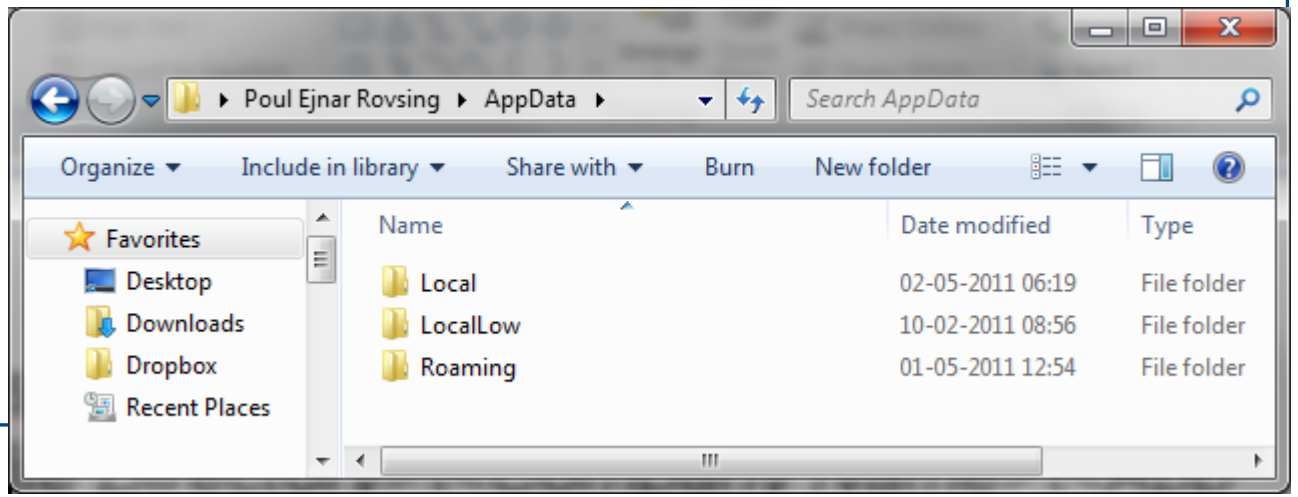
Conceptual View of the Windows User Profile



Folder for User Settings

- User settings are stored in the folder:
  *<Profile Directory>**<Company Name>**<App Name>*_*<Evidence Type>*_*<Evidence Hash>**<Version>*\user.config
- *<Profile Directory>*
  - is either the roaming profile directory or the local one
  - Settings are stored by default in the local user.config file
  - To store a setting in the roaming user.config file, you need to mark the setting with the SettingsManageabilityAttribute

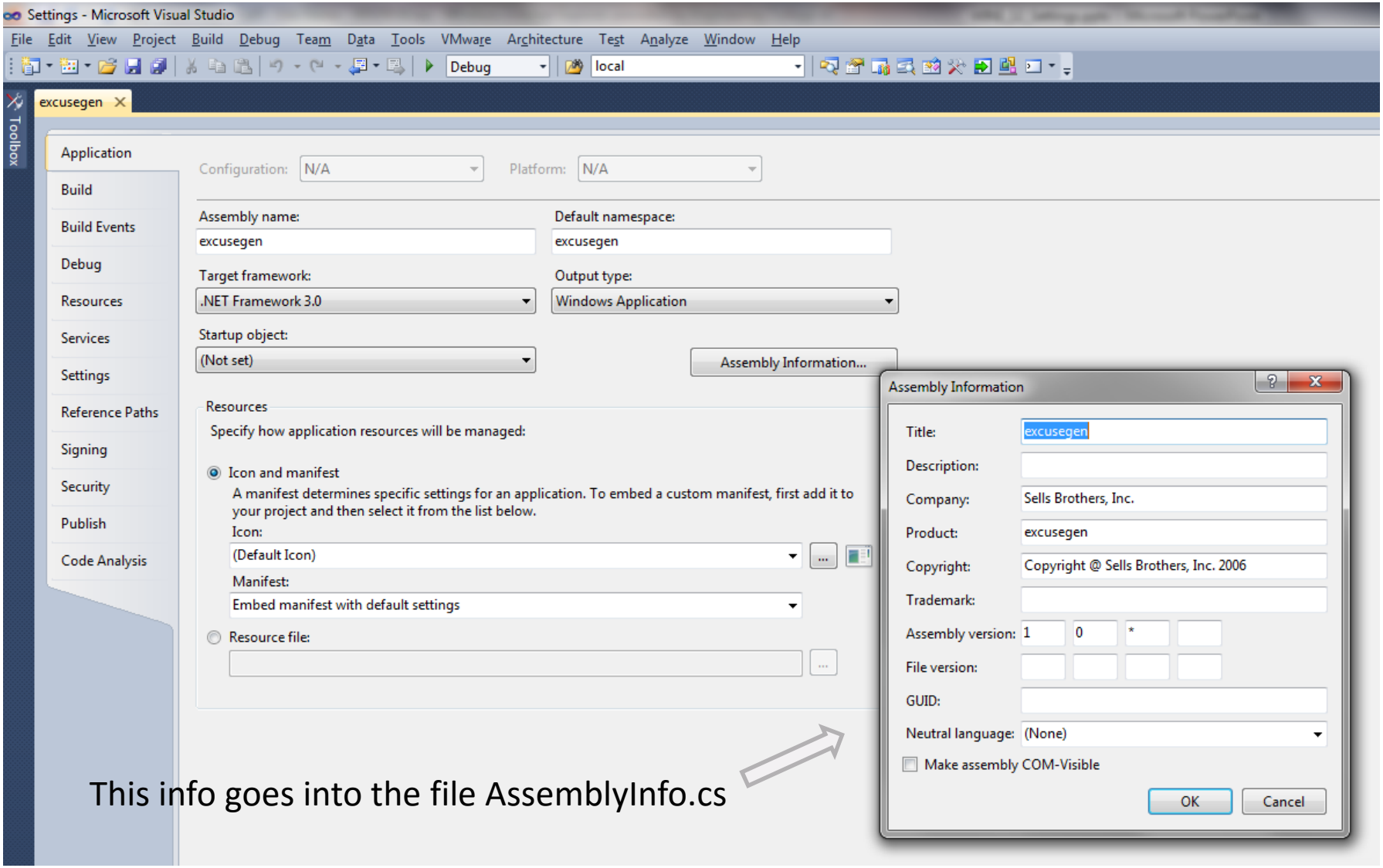




Folder for User Settings Continued

- *<Company Name>*
  - is typically the string specified by the AssemblyCompanyAttribute (with the caveat that the string is escaped and truncated as necessary)
- *<App Name>*
  - is typically the string specified by the AssemblyProductAttribute
- *<Evidence Type>* and *<Evidence Hash>*
  - information derived from the app domain evidence to provide proper app domain and assembly isolation.
- *<Version>*
  - typically the version specified in the AssemblyVersionAttribute. This is required to isolate different versions of the app deployed side by side

Application Specific Attributes



This info goes into the file AssemblyInfo.cs

Get the Path Programmatically

```
// You need to add a reference to System.configuration.dll
// and include a using System.Configuration

Configuration config =
 ConfigurationManager.OpenExeConfiguration(
                    ConfigurationUserLevel.PerUserRoamingAndLocal);
Console.WriteLine("Local user config path: {0}", config.FilePath);
```

Resetting User Settings

- Sometimes users regret the changes they apply to user settings and want to roll back to the previously stored values
  - This can easily be done programmatically with:

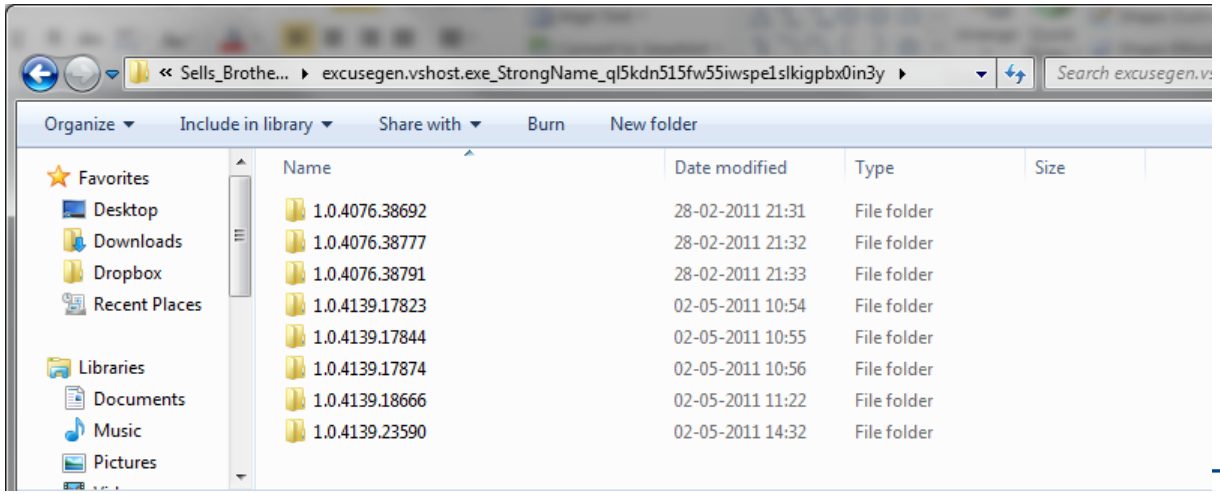
```
Properties.Settings.Default.Reload();
```

- If the values stored in the user.config files also are obscure, then you can reset them to their default values programmatically:

```
Properties.Settings.Default.Reset();
```

Version Upgrade

- Why is there a version number in the user.config path?
  - If I deploy a new version of my application, won't the user lose all the settings saved by the previous version?
    - **YES!** – unless you call Upgrade on first launch of new version
- There are couple of reasons why the user.config path is version sensitive:
  1. To support side-by-side deployment of different versions
  2. When you upgrade an application, the settings class may have been altered and may not be compatible with what's saved out
- To upgrade settings from a previous version:
  - Simply call **ApplicationSettingsBase.Upgrade()** and it will retrieve settings from the previous version that match the current version of the class and store them out in the current version's user.config file.



When To Call Upgrade?

- Okay, but how do I know when to call Upgrade?

- Have a boolean setting called CallUpgrade and give it a default value of true
  When your app starts up, you can do something like:

```
if (Properties.Settings.Default.CallUpgrade) {
    Properties.Settings.Default.Upgrade();
    Properties.Settings.Default.CallUpgrade = false;
 }
```

- This will ensure that Upgrade() is called only the first time the application runs after a new version is deployed

Additional Setting Files

- Usually one settings file (the default file) is enough, but you can add as many settings files as you wish

- To add an additional settings file:
  - right-click your project in the Solution Explorer and click
    Add – New Item – Settings File

Binding to Settings

- You can easily bind to settings
  - You can define the settings class as a resource in App.xaml:

```
<Application x:Class="Lab1.App"
     xmlns=http://schemas.microsoft.com/winfx/...
     xmlns:properties="clr-namespace:Lab1.Properties"
     …
    <Application.Resources>
        <properties:Settings x:Key="Settings" />
    </Application.Resources>
</Application>
```

  - And then bind to the specific settings using this syntax:

```
<TextBox Name="tbxAge"
         Text="{Binding
             Source={StaticResource Settings},
             Path=Default.Age}"
```

WPF's Application Object's settings service

This may be used as an simple alternative to BCL's Setting services.

The Application Object

- The Application Object is responsible for:
  - Managing the lifetime of the application
  - Tracing the visible windows
  - Dispensing resources
  - **Managing the global state of the application**

Shared application-scope properties

- Application class provides the **Properties** property to expose state that can be shared across the application.

```
// Set an application-scope property with a custom type CustomType
CustomType customType = new CustomType();
Application.Current.Properties["myKey"] = customType;

...

// Get an application-scope property
// NOTE: Need to convert since Application.Properties
// is a dictionary of System.Object
CustomType customType = (CustomType)
                        Application.Current.Properties["myKey"];
```

How To Persist Application-Scope Properties

```
protected override void OnStartup(StartupEventArgs e)
{
  using (FileStream stream = new FileStream(filePath, FileMode.Open))
  using (StreamReader reader = new StreamReader(stream))
  {
    // Restore each application-scope property individually
    while (!reader.EndOfStream)
    {
       string[] keyValue = reader.ReadLine().Split(new char[] { ';' });
       this.Properties[keyValue[0]] = keyValue[1];
    }
```

Supports only strings

```
protected override void OnExit(ExitEventArgs e)
{
  using (FileStream stream = new FileStream(filePath))
  using (StreamWriter writer = new StreamWriter(stream))
  {
    // Persist each application-scope property individually
    foreach (string key in this.Properties.Keys)
    {
      writer.WriteLine("{0};{1}", key, this.Properties[key]);
    }
  }
```

System Settings

Where to Find System Settings

- At run-time there are several classes that provide info about different system settings:
  - System.Environment
  - System.Windows.SystemFonts
  - System.Windows.SystemColors
  - System.Windows.SystemParameters

  - System.Windows.Forms.SystemInformation
    (need to add a reference to System.Windows.Forms)

References & Links

- Application Settings Overview
  http://msdn.microsoft.com/en-us/library/k4s6c3a0.aspx
- Application Settings Architecture
  http://msdn.microsoft.com/en-us/library/8eyb2ct1.aspx
- Settings Page, Project Designer
  http://msdn.microsoft.com/query/dev11.query?appId=Dev11IDEF1&l=EN-US&k=k(ApplicationSettingsOverview);k(TargetFrameworkMoniker-.NETFramework,Version%3Dv4.0)&rd=true
- User Settings in WPF
  http://blogs.msdn.com/b/patrickdanino/archive/2008/07/23/user-settings-in-wpf.aspx
- Client Settings FAQ
  http://blogs.msdn.com/b/rprabhu/archive/2005/06/29/433979.aspx
- User Settings Applied (by Jani Giannoudis)
  http://www.codeproject.com/KB/dotnet/user_settings.aspx
- Shared application-scope properties
  http://msdn.microsoft.com/en-us/library/ms743714.aspx#Other_Application_Services