Storing User Profile in .NET

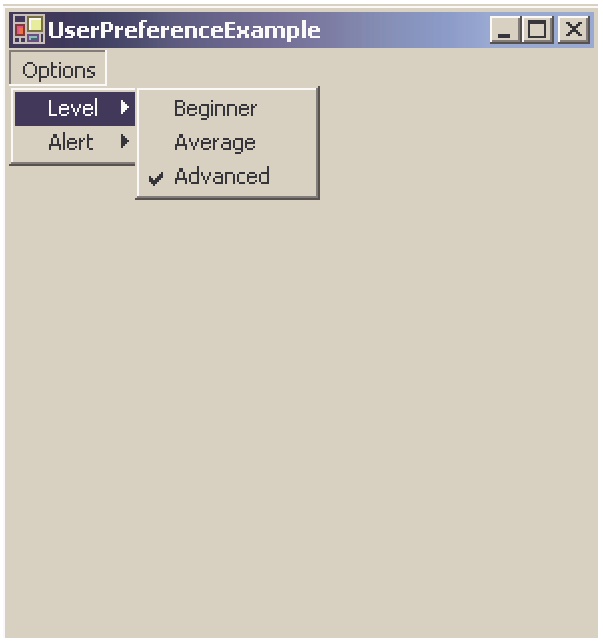
Venkat Subramaniam
venkats@agiledeveloper.com
http://www.agiledeveloper.com/download.aspx

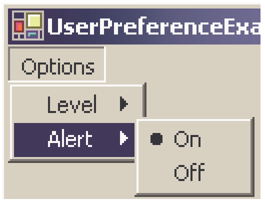
Abstract

The .NET Security does not permit a .NET application running from a shared network drive or internet to access the local hard drive. How do we then store user specific information like user preferences? The .NET class library provides classes that allow us to access the so called Isolated Storage. Isolated storage is located under the "Documents and Settings" for each user. Different storage may be specified based on the user, assembly and the domain. Isolated storage may be used to store whatever information you please. In this article we look at how to use the Isolated Storage using a simple example.

User Preference

We will explore the concepts in this article by way of an example. So, let's first write a simple application that will allow users to modify some preferences. Let's take a look at the application shown below:





The application has two menus. The Level menu allows the user to set the level to Beginner, Average or Advanced. The Alert menu allows the user to turn on or off the alert. Once a user selects the level and alert options, you would want the application to remember this. One way to perform this is to store the information in the registry (the good old way☹). Another option is to store the preferences in a file. The specific format of the file is not of concern here. One may choose to store in a regular text file or in an XML document.

Storing and Retrieving the Preferences in an XML File

Here is the code for storing and restoring the information in an XML file:

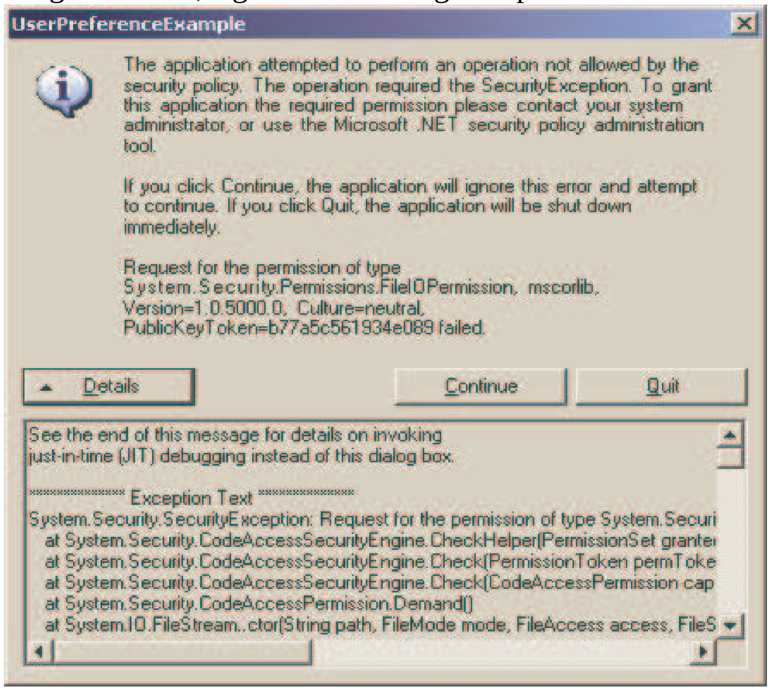
```
private void SaveSettings()
{
      string level = "Beginner";
      if (AverageMenuItem.Checked)
            level = "Average";
      if (AdvancedMenuItem.Checked)
            level = "Advanced";

      string alert = "On";
      if (AlertOffMenuItem.Checked)
            alert = "Off";

      XmlWriter writer = new XmlTextWriter("profile.xml",
                              System.Text.Encoding.UTF8);

      writer.WriteStartElement("Settings");
      writer.WriteStartElement("Level");
      writer.WriteString(level);
      writer.WriteEndElement();
      writer.WriteStartElement("Alert");
      writer.WriteString(alert);
      writer.WriteEndElement();
      writer.WriteEndElement();
      writer.Close();
}

private void RestoreSettings()
{
      if (System.IO.File.Exists("profile.xml"))
      {
            XmlDocument document = new XmlDocument();
            document.Load("profile.xml");

            string level =
                  document.GetElementsByTagName(
                        "Level")[0].FirstChild.Value;

            string alert =
                  document.GetElementsByTagName(
                        "Alert")[0].FirstChild.Value;

            switch(level)
            {
                  case "Beginner":
                        ClearLevels();
                        BeginnerMenuItem.Checked = true;
                        break;
                  case "Average":
                        ClearLevels();
                        AverageMenuItem.Checked = true;
                        break;
                  case "Advanced":
                        ClearLevels();
                        AdvancedMenuItem.Checked = true;
                        break;
                }

                AlertOffMenuItem.Checked = false;
                AlertOnMenuItem.Checked = false;
                switch(alert)
                {
                    case "On":
                        AlertOnMenuItem.Checked = true;
                        break;
                    case "Off":
                        AlertOffMenuItem.Checked = true;
                        break;
                }
            }
        }
```

The TextWriter class comes from the System.Xml namespace.

One may consider other ways to store the preferences in a file. Do not worry about that; we are storing the preferences in a file named “profile.xml” which is located in the application directory.

File Access Error

If the application is located on my local drive, the program works fine. I modify the preference for level and/or alert and exit the program. When I start the application again, it remembers the values from the previous run. The profile.xml holds the following information: <Settings><Level>Advanced</Level><Alert>On</Alert></Settings>. Now, I place the executable in a network shared drive and double click on it to execute it. When I try to change the level, I get the following exception:


UserPreferenceExample

The application attempted to perform an operation not allowed by the security policy. The operation required the SecurityException. To grant this application the required permission please contact your system administrator, or use the Microsoft .NET security policy administration tool.

If you click Continue, the application will ignore this error and attempt to continue. If you click Quit, the application will be shut down immediately.

Request for the permission of type System.Security.Permissions.FileIOPermission, mscorlib, Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b77a5c561934e089 failed.

```
See the end of this message for details on invoking
just-in-time (JIT) debugging instead of this dialog box.

************** Exception Text **************
System.Security.SecurityException: Request for the permission of type System.Securi
   at System.Security.CodeAccessSecurityEngine.CheckHelper(PermissionSet grante
   at System.Security.CodeAccessSecurityEngine.Check(PermissionToken permToke
   at System.Security.CodeAccessSecurityEngine.Check(CodeAccessPermission cap
   at System.Security.CodeAccessPermission.Demand()
   at System.IO.FileStream..ctor(String path, FileMode mode, FileAccess access, FileS
```

A .NET application that is not located on my local drive does not have access to it.

Using the IsolatedStorage

Let us modify the SaveSettings and RestoreSettings methods as shown below:

```
private void SaveSettings()
{
      string level = "Beginner";
      if (AverageMenuItem.Checked)
            level = "Average";
      if (AdvancedMenuItem.Checked)
            level = "Advanced";

      string alert = "On";
      if (AlertOffMenuItem.Checked)
            alert = "Off";

      IsolatedStorageFileStream stream =
             GetIsolatedStorage(true);

      if (stream != null)
      {
            XmlWriter writer = new XmlTextWriter(
                  stream, System.Text.Encoding.UTF8);


            writer.WriteStartElement("Settings");
            writer.WriteStartElement("Level");
            writer.WriteString(level);
            writer.WriteEndElement();
            writer.WriteStartElement("Alert");
            writer.WriteString(alert);
            writer.WriteEndElement();
            writer.WriteEndElement();
            writer.Close();
      }
}

private void RestoreSettings()
{
      IsolatedStorageFileStream stream =
                   GetIsolatedStorage(false);

      if (stream != null)
      {
            XmlDocument document = new XmlDocument();
            document.Load(stream);

            string level =
                  document.GetElementsByTagName(
                        "Level")[0].FirstChild.Value;

            string alert =
                  document.GetElementsByTagName(
                        "Alert")[0].FirstChild.Value;
            switch(level)
            {
                  case "Beginner":
                        ClearLevels();
                        BeginnerMenuItem.Checked = true;
                        break;
                  case "Average":
                        ClearLevels();
                        AverageMenuItem.Checked = true;
                        break;
                  case "Advanced":
                        ClearLevels();
                        AdvancedMenuItem.Checked = true;
                        break;
            }

            AlertOffMenuItem.Checked = false;
            AlertOnMenuItem.Checked = false;
            switch(alert)
            {
                  case "On":
                        AlertOnMenuItem.Checked = true;
                        break;
                  case "Off":
                        AlertOffMenuItem.Checked = true;
                        break;
            }
      }
}
```

The method GetIsolatedStorage is implemented as follows:

```
    private IsolatedStorageFileStream GetIsolatedStorage(
                                                    bool write)
    {
        IsolatedStorageFileStream stream = null;

        try
        {
            IsolatedStorageFile theIsolatedStorageFile =
             IsolatedStorageFile.GetUserStoreForAssembly();

            FileMode mode;
            FileAccess access;

            if (write)
            {
                mode = FileMode.Create;
                access = FileAccess.Write;
            }
            else
            {
                mode = FileMode.Open;
                access = FileAccess.Read;
            }
```

```
                        stream = new
                              IsolatedStorageFileStream("profile.xml",
                                 mode, access, theIsolatedStorageFile);
                  }
                  catch(FileNotFoundException)
                  {// We will simply return a null for the stream
                  }

                  return stream;
            }
```

In order for the code to compile, we add the following to the top of the .cs file:

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

Now I copy the application to the network drive and execute it. The application is able to retain the changes to the level and alert options and is storing the user preferences.

Let's revisit the relevant code again:

```
IsolatedStorageFile theIsolatedStorageFile =
                  IsolatedStorageFile.GetUserStoreForAssembly();

...
stream = new IsolatedStorageFileStream("profile.xml",
                  mode, access, theIsolatedStorageFile);
```

The IsolatedStorageFile is first obtained by calling the GetUserStoreFromAssembly. This provides access to the isolated storage based on the current user and the assembly requesting for the storage. If the same user were to execute two different programs (assemblies) each requesting for the storage, then the storage returned to the first one will be different from the one returned to the second one. Then we create an Isolated Storage File Stream in that Isolated Storage.

Location of the Isolated Storage

Upon executing the above program, we traversed to the following directory: "C:\Documents and Settings\<<USERNAME>>\Local Settings\Application Data**IsolatedStorage**\54dlhn35.0ij\ejpsyge2.ayo\Url.wam2czkmfhgpetnb1uqoqbyav0vo x4fe\AssemFiles," where <<USERNAME>> is the current userid. The profile.xml is stored in this directory. Depending on the options chosen to create the isolated storage, the storage may in the location specified above or under C:\Documents and Settings\<<USERNAME>>\ Application Data**IsolatedStorage**\....

It works, but what is IsolatedStorage?

Isolated Storage allows applications to save data in their own area of file system without having to specify the path in the file system. The isolation is at the assembly level, that is, the isolated storage for two applications is different if the requesting assemblies are different. Some interesting methods of IsolatedStorageFile class:

GetUserStoreForAssembly: The isolated storage is based on the User and Assembly identity. That is the same assembly within different applications will use the same storage. This is equivalent to calling:
GetStore(IsolatedStorageScope.Assembly | IsolatedStorageScope.User, null, null);

GetUserStoreForDomain: The isolated storage is based on the User, Assembly identity and application domain identity. In this case, the same assembly within different applications will use different storage. Further, different assemblies within the same application will also use different storage. This is equivalent to calling:
GetStore (IsolatedStorageScope.Assembly | IsolatedStorageScope.Domain |
IsolatedStorageScope.User, null, null);

GetStore: Allows you to get the isolated storage based on whatever scope you wish to provide.

The possible values for the IsolatedStorageScope enumeration are:
Assembly – the isolated storage is scoped to the identity of the assembly
Domain – the isolated storage is scoped to the identity of the application domain
None – No isolated storage used
User – the isolated storage is scoped to the identity of the user
Roaming – the isolated storage is placed in the location on the file system that might roam in operating systems on which roaming is enabled.

Conclusion
IsolatedStorage provides a means to store application specific data and user preferences where the security restrictions may not allow to access local hard drive. It may be created with isolation based on a combination of the user, assembly, domain, etc. Applications that need to store user profiles may benefit from this facility in .NET.

References

1. MSDN online documentation.