

Features

- Manage 8.3 format filenames
 - Convert long filename to 8.3 filename and vice-versa
- Compare paths and identify parent paths
 - Copes with long/8.3 filename differences
- Use standard Windows icons for Files
 - Functions to get icons and icon information
 - Automated *ImageList* for use with Windows Forms controls
- Store objects in XML files
 - Fully type safe
 - Use Isolated Storage or the Filesystem with the same API
 - Simplify your code
 - Manage concurrent access
 - Detect changes in files

Installation

SalientFileUtility can be downloaded as either a ZIP or MSI (Windows Installer) file. In either case, you will be able to choose where the files are installed on your system. To use classes from SalientFileUtility in your project, you must add a DLL reference to SalientFileUtility.dll (from bin folder under the installation directory) to your project.

To install the *FileIconImageList* class as a component, right-click in the “Components” or “Windows Forms” toolbar in Visual Studio and select “Add/Remove Items”. Browse for “SalientFileUtility.dll” and select it – this will add the component to the component list, already checked. The component will now appear on your toolbar and you can drag it onto the design surface of forms and components in your applications, and the necessary references are added to your project automatically.

Licensing

The default trial license for SalientFileUtility will cause a licensing exception to be thrown from time to time when using the library. When a license is obtained these exceptions will stop appearing. Licensing uses SalientProtect and works by adding an attribute to your application assembly or assemblies, enabling SalientFileUtility for assemblies signed with your strong name key. Licenses can be obtained through our website: <http://www.SalientPoint.co.uk/>.

More Information

Details of the classes in SalientFileUtility can be found in the supplied help file. Further information on all SalientPoint products including SalientFileUtility can be found on our website: <http://www.SalientPoint.co.uk/>

FileUtility

The *GetShortPath* and *GetLongPath* functions take a valid path to a directory or file, convert its short or long form respectively:

```
FileUtility.GetLongPath(@"C:\PROGRA~1") // Returns "C:\Program Files"  
FileUtility.GetShortPath(@"C:\Program Files"); // Returns "C:\PROGRA~1"
```

The *GetShellIcon* function gets the icon that the explorer shell would use for the specified file, in large or small format and with the file open or closed:

```
FileIconInfo fii = FileUtility.GetShellIcon(  
    new DirectoryInfo( @"C:\" ), false , false );
```

// fii.Icon contains the Icon

PathComparer

The *PathComparer* file contains static functions for comparing files and directories, and for checking for parent/child relationships:

```
// Returns 0 - paths are equivalent  
PathComparer.ComparePaths( @"C:\PROGRA~1" , @"C:\Program Files" );  
  
// Returns 0 - file name is the same  
PathComparer.CompareNames( @"C:\a.txt" , @"C:\Windows\a.txt" );  
  
// Returns true - there is a parent relationship  
PathComparer.IsParentOf( @"C:\" , @"C:\Windows" );  
  
// Returns true - there is a direct parent relationship  
PathComparer.IsDirectParentOf( @"C:\" , @"C:\Windows" );
```

It also provides an *IComparer* interface for use with framework functions such as *Array.Sort*:

```
FileSystemInfo [] winFiles =  
    new DirectoryInfo( @"C:\Windows" ).GetFileSystemInfos();  
  
Array.Sort( winFiles , new PathComparer() );
```

FileIconImageList

The *FileIconImageList* class can be used to provide icons to Windows Forms controls such as *TreeView* and *ListView*. Icon indexes are obtained using the *GetIconIndex* function, specifying the name of the file for which the icon should be retrieved.

```
FileIconImageList imageList = new FileIconImageList();

listView1.SmallImageList = imageList.ImageList;
listView1.View = View.SmallIcon;

FileSystemInfo [] winFiles =
    new DirectoryInfo( @"C:\" ).GetFileSystemInfos();

foreach ( FileSystemInfo fsi in winFiles )
{
    listView1.Items.Add(
        fsi.Name ,
        imageList.GetIconIndex( fsi , false , false ) );
}
```

XmlObjectStore

The *XmlObjectStore* class enables you to store and retrieve application data objects in a file. This example uses isolated storage to store and retrieve a string object – usually however the object would contain application settings or data.

```
string s1 = "Hello There";

XmlObjectStore<string> xos = new XmlObjectStore<string>( "test.xml" , true );

xos.Write( s1 );

string s2 = xos.Read();

Debug.Assert( s1 == s2 );

xos.Delete();
```

XmlObjectStoreLocked

XMLObjectStoreLocked works in exactly the same way as *XmlObjectStore* is safe for use in an environment where there could be concurrent access to the file from different threads or processes. Setting the “Locked” property will prevent any other access to the file, for example while displaying a settings dialog:

```
XmlObjectStoreLocked<SettingsObject> xos =
    new XmlObjectStoreLocked<SettingsObject>( "test.xml" );

xos.Locked = true;

SettingsObject s = xos.Read();

ShowSettingsDialog( s );

xos.Write( s );

xos.Locked = false;
```

XmlObjectStoreChecked

XMLObjectStoreChecked takes an alternative approach to dealing with concurrent access to the file. Instead of preventing changes from another thread/process it will alert your application (via an event) if changes have taken place so your application can decide how to deal with the situation.

```
void Function()
{
    XmlObjectStoreChecked<SettingsObject> xos =
        new XmlObjectStoreChecked<SettingsObject>( "test.xml" , true );

    xos.FileChangedSinceLastRead +=
        new EventHandler<XmlObjectStoreCheckedEventArgs<string>>( Handler);

    SettingsObject s = xos.Read();

    ShowSettingsDialog( s );

    xos.Write( s );
}

void Handler( object sender , XmlObjectStoreCheckedEventArgs<SettingsObject> e)
{
    DialogResult dr = MessageBox.Show(
        "Another program changed the file - overwrite its changes?" ,
        "Conflict" , MessageBoxButtons.YesNo );

    e.Cancel = ( dr == DialogResult.No );
}
```