

### Features

- *Watcher* watches for filesystem changes from a known initial state
  - Allows you to build consistent user interfaces
  - Simplifies application logic
  - Catches moves and renames missed by *FileSystemWatcher*
- *WatcherView* provides a ready-made *TreeView* control
  - Displays a hierarchy of files and directories
  - Updates to reflect changes in real-time
  - Displays standard windows icons

### Installation

SalientWatcher 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 SalientWatcher in your project, you must add a DLL reference to SalientWatcher.dll (from bin folder under the installation directory) to your project.

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

### Licensing

The default trial license for SalientWatcher 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 SalientWatcher 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 SalientWatcher can be found in the supplied help file. Further information on all SalientPoint products including SalientWatcher can be found on our website: <http://www.SalientPoint.co.uk/>

### **Watcher**

In order to use *Watcher*, the following steps are needed:

- Create a *Watcher* object
- Register for events
- Set the *EnableRaisingEvents* property to true to enable change notifications relative to the initial state.
- Use the *Root* property of the *Watcher* object to traverse the *WatcherNode* objects representing the initial state and build the UI
- Use the events to update your application to reflect filesystem changes.

The following example shows the use of the *Watcher* class:

```
static void Main(string[] args)
{
    Watcher w = new Watcher( "C:\\SalientPoint" );

    w.Created += new EventHandler<SimpleFileEventArgs> (w_Created);

    w.EnableRaisingEvents = true;

    System.Threading.Thread.Sleep( 10000 );
}

private static void w_Created( object sender,
                               SimpleFileEventArgs e)
{
    Console.WriteLine( "File Created: " + e.Node.FullName );
}
```

### **WatcherView**

*WatcherView* is a class that inherits from *TreeView* but that displays a watched section of the filesystem. It can be instantiated programmatically or as a component by adding dragging it on to your form.

The following example shows the use of the *WatcherView* class:

```
public void ShowTreeView()
{
    Form f = new Form();
    WatcherView wv = new WatcherView();

    wv.AddRoot( @"C:\Program Files\Adobe" );
    wv.AddRoot( @"D:\" );
    wv.Dock = DockStyle.Fill;

    f.Controls.Add( wv );

    f.ShowDialog();
}
```