C#
Windows (Win2003, Win2K, WinXP), .NET (.NET 1.0, .NET 1.1)
Win32, VS (VS.NET2002, VS.NET2003)
Dev

Posted 7 Sep 2004 0:00

Updated 16 Nov 2004 8:21

Top 10 tips and tricks to jazz up your C# business solutions/server products

By Salil Khedkar

I have tried to compile a list of things which we commonly need, when we sit down to code a 'business solution' or 'application server' kind of software for the first time using .NET. This list of tricks gives ideas on how to jazz things up, to give you the extra edge to win customers hearts.

Introduction

After the initial lot of tutorials and exercises, when finally I sat down to write my first professional quality C# application, I had to spend some time finding out how routine server related tasks like reading the registry, sending emails, writing to the Windows Event log, getting system information, watching folders etc. are done in .NET.

Fortunately, the rewards of the time spent were fantastic. It was wonderful to see the simplicity and elegance of the C# .NET code doing pretty complicated tasks, against the daunting C++/MFC code I used to write earlier for doing the same things.

So, I decided to compile this list of ready-to-refer tricks.

Notes:

a. Sure, you can wade through the MSDN and know all this. But this stuff is for those who are short on time and want ready-to-use simple solutions in a jiffy. You know, the ones whose boss has just yelled "Get me the results! Now!".
b. I have chucked out the `try-catch` blocks and comments from the sample code to reduce the size. Please add these life savers wherever necessary.
c. When you will run the samples in production environment, please remember that the user-context under which your program is running will require the necessary access right to do certain things. For example, to be able to write to Windows Event Log, you will require the corresponding access rights.
d. Use this article as a starter to do impressive things with your application, it will get you productive in no time. Once you have some free time, read up more from the MSDN and delve deeper into things.

Trick 1: Writing to Windows Event Log

You have written a cool application server... But as is the case with most servers, it runs in background, either as a Windows Service or may be with a hidden window. Now, how are you going to notify the System Administrator, in case some failures or important events occur? All big-shot server products use the Windows Event log for this purpose.

And so should you, with the simple function given below: (Use the relevant `EventLogEntryType` for your application)

```
using System.Diagnostics;
public void WriteEventLog(string sCallerName, string sLogLine)
{
  if (!System.Diagnostics.EventLog.SourceExists(sCallerName))
    System.Diagnostics.EventLog.CreateEventSource(sCallerName, "Application");

  EventLog EventLog1 = new EventLog();
  EventLog1.Source = sCallerName;
  EventLog1.WriteEntry (sLogLine, EventLogEntryType.Warning);
}
```

Trick 2: Ensure a single instance of your app on a machine

Sometimes, users (or QA people) can get really nasty and try to run multiple instances of your beloved little application on a single machine. Now, in case that makes you uncomfortable, just pop in the following code in your `Main()`.

Yes, it is utilizing a `Mutex` object to make sure the users don't get too smart.

```
using System.Threading;
static void Main()
{
  bool bAppFirstInstance;
  oMutex = new Mutex(true, "Global\\" + "YOUR_APP_NAME", out bAppFirstInstance);
  if(bAppFirstInstance)
    Application.Run(new formYOURAPP() or classYOURAPP());
  else
    MessageBox.Show("The threatening message you want to go for…",
        "Startup warning",
        MessageBoxButtons.OK, MessageBoxIcon.Exclamation,
        MessageBoxDefaultButton.Button1);
}
```

A techie note from Adam: The `'\\Global'` thingie ensures that the app should be a single instance on the machine, not just for this user's session.

Trick 3: Notify users with an e-mail (!)

Well, what can impress users more than a neat little mail from your cute server? The task is pretty easy with .NET, do have a look at the following function. Please don't forget to provide the correct sender account for `Message.From` and a valid SMTP server for `SmtpMail.SmtpServer`. (Then prepare to receive hugs and flying kisses from elated users.)

If you want to get one step ahead of others, here are some more tricks:

a. All cellular phones accept text emails these days. And you can use the same below-mentioned function to send a text mail to the user's mobile device. Just find out the email ID assigned by his cell provider for his cell, and you are up and running in no time. For example, 'Idea Cellular' in India routes incoming email for yourcellnumber@ideacellular.net to your mobile. Just remember to keep the mail text very short. Tip: The boss characters *love* these sweet email alerts!
b. Text mails get boring after a while. So you can use the `MailFormat.Html` instead of `MailFormat.Text` and send neatly formatted messages too. See MSDN for more details.

```
using System.Web.Mail;
private bool SendEmail(string sFrom, string sTo, string sCC,
    string sBCC, string sSubject, string sMessage, int iMailType)
{
  try
  {
    MailMessage Message = new MailMessage();

    // If sFrom is blank, system mail id is assumed as the sender.
    if(sFrom=="")
      Message.From = "default@myserver.com";
    else
      Message.From = sFrom;

    // If sTo is blank, return false
    if(sTo=="")
      return false;
    else
      Message.To = sTo;

    Message.Cc = sCC;
    Message.Bcc = sBCC;
    Message.Subject = sSubject;
    Message.Body = sMessage;
    Message.BodyFormat = MailFormat.Text;
    SmtpMail.SmtpServer = "Put a valid smtp server IP";
    SmtpMail.Send(Message);

    return true;
  }
  catch(System.Web.HttpException ehttp)
  {
    // Your exception handling code here...
    return false;
  }
  catch(Exception e)
  {
    // Your exception handling code here...
    return false;
  }
  catch
  {
    // Your exception handling code here...
    return false;
  }
}
```

Trick 4: Getting IP address given the host name

You have developed the next big socket application. But your users are not sure about the IP address of the server they want to connect to. They input host name most of the time and you require IP addresses for your code to work well. No problemo… Use the following function to get the IP address of a host, given its name:

```
using System.Net;
public string GetIPAddress(string sHostName)
{
  IPHostEntry ipEntry = Dns.GetHostByName(sHostName);
  IPAddress [] addr = ipEntry.AddressList;
  string sIPAddress = addr[0].ToString();
  return sIPAddress;
}
```

To get the IP of the local machine, use the following method before invoking `GetIPAddress()`:

```
string sHostName = Dns.GetHostName();
```

Trick 5: Writing professional log files

Well, you can always dump a few strings into a file and call it a log file. What's the big deal? The thing is - it does not look, ahem, techie. Here is a sample which creates a log file (one per day & instance) and puts in time-stamped log entries into it. The file names are such that sorting them in Windows File Explorer is very easy for the SysAd.

Also note that logging functions will require multithreading support, so I am using the lock thingie here; you can either get rid of it (for ST apps) or use something that suits you better. Kudos to Adam for help with this trick.

```
using System.IO;
public void WriteLogLine(string sCallerName, string sLogFolder,
```

```
                        long lCallerInstance, string sLogLine)
{
  lock(this)
  {
    string sFileName;
    sFileName = String.Format("{0}_{1:yyyy.MM.dd}_{2:00}.log",
                  sCallerName, DateTime.Now, lCallerInstance);
    StreamWriter swServerLog =
           new StreamWriter(sLogFolder + sFileName, true);
    swServerLog.WriteLine(
           String.Format("[{0:T}] {1}", DateTime.Now, sLogLine));
    swServerLog.Close();
  }
}
```

Trick 6: Windows Registry and you

Windows has given you a great provision to store configuration tidbits and stuff like that… The registry. Why not use it, instead of using custom configuration files?

Below are the functions to create a registry key, and set and get a string value. You can extend the get/set couple for long values and so on. Please note that for defensive programming purpose, I am putting in the supplied default value, in case a registry value is missing. I like this approach better than crying out with an exception. If you are of the other opinion, please feel free to do what you like.

Also note that, many people prefer storing the configuration information in easily modifiable and self-maintainable XML files these days. Be sure about your and your clients' preference and then choose the best suitable option.

```
using Microsoft.Win32;
using System.IO;
using System.Security;
/// <summary>
/// Tries to read the registry key, if is does not exist,
/// it gets created automatically
/// </summary>
public void CheckRegistryKey(string sKeyname)
{
  RegistryKey oRegistryKey =
    Registry.LocalMachine.OpenSubKey("SOFTWARE\\" + sKeyname, true);
  if(oRegistryKey==null)
  {
    oRegistryKey =
      Microsoft.Win32.Registry.LocalMachine.OpenSubKey("SOFTWARE", true);
    oRegistryKey.CreateSubKey(sKeyname);
  }
}
/// <summary>
/// Checks for a string value in given path,
/// if not found adds provided default value
/// </summary>
public string GetStringValue(string sKeyname,
               string sValueName, string sDefaultValue)
{
  RegistryKey oRegistryKey =
    Registry.LocalMachine.OpenSubKey("SOFTWARE\\" + sKeyname, true);
  if(oRegistryKey==null)
  {
    oRegistryKey =
      Microsoft.Win32.Registry.LocalMachine.OpenSubKey("SOFTWARE", true);
    oRegistryKey.CreateSubKey(sKeyname);
  }

  if(oRegistryKey.GetValue(sValueName)==null)
  {
    oRegistryKey.SetValue(sValueName, sDefaultValue);
    return sDefaultValue;
  }
  else
    return (string)oRegistryKey.GetValue(sValueName);
}
/// <summary>
/// Sets a string value for given key
/// </summary>
public void SetStringValue(string sKeyname, string sValueName,
                                              string sValue)
{
  RegistryKey oRegistryKey =
    Registry.LocalMachine.OpenSubKey("SOFTWARE\\" + sKeyname, true);
  if(oRegistryKey==null)
  {
    oRegistryKey =
      Microsoft.Win32.Registry.LocalMachine.OpenSubKey("SOFTWARE", true);
    oRegistryKey.CreateSubKey(sKeyname);
  }

  oRegistryKey.SetValue(sValueName, sValue);
}
```

Trick 7: Get environment conscious

If you are not yet environment conscious, it's time to be. First, thank God. This is something we always need to look up. Now with .NET, we have a single class which lets us access all this wealth of information. What are you looking for? The OS version, CLR version, user name, values of system variables such as the temp folder, physical memory mapped to your app?

All this valuable information can be extracted with the `Environment` class as illustrated below:

```
using System;
public static void GetEnvironmentInfo()
{
    // Fully qualified path of the current directory
    Console.WriteLine("CurrentDirectory: {0}", Environment.CurrentDirectory);
    // Gets the NetBIOS name of this local computer
    Console.WriteLine("MachineName: {0}", Environment.MachineName);
    // Version number of the OS
    Console.WriteLine("OSVersion: {0}", Environment.OSVersion.ToString());
    // Fully qualified path of the system directory
    Console.WriteLine("SystemDirectory: {0}", Environment.SystemDirectory);
    // Network domain name associated with the current user
    Console.WriteLine("UserDomainName: {0}", Environment.UserDomainName);
    // Whether the current process is running in user interactive mode
    Console.WriteLine("UserInteractive: {0}", Environment.UserInteractive);
    // User name of the person who started the current thread
    Console.WriteLine("UserName: {0}", Environment.UserName);
    // Major, minor, build, and revision numbers of the CLR
    Console.WriteLine("CLRVersion: {0}", Environment.Version.ToString());
    // Amount of physical memory mapped to the process context
    Console.WriteLine("WorkingSet: {0}", Environment.WorkingSet);
    // Returns values of Environment variables enclosed in %%
    Console.WriteLine("ExpandEnvironmentVariables: {0}",
        Environment.ExpandEnvironmentVariables("System drive: " +
        "%SystemDrive% System root: %SystemRoot%"));
    // Array of string containing the names of the logical drives
    Console.WriteLine("GetLogicalDrives: {0}", String.Join(", ",
                      Environment.GetLogicalDrives()));
}
```

The above function will work with console applications only (as it has `Console.Writeline`s). For your use, modify it the way you need.

Trick 8: Setup a hidden camera for watching a folder

Watching user activities inside a folder is often a task for application servers. For example, I have a server which needs to wait until user uploads an XML file into a special folder. Once user drops the file there, the server does some processing according to the file contents.

Once you know how to do this thing, I am sure you will find thousands of applications to utilize this trick. And it is not just limited to file creation events, you can watch on virtually all the activities users can do in there... ;)

```
using System.IO;
// Watches the C:\Temp folder and notifies creation of new text files
public static void WatchTempFolder()
{
    // Create the FileSystemoFileSystemWatcher object and set its properties
    FileSystemWatcher oFileSystemWatcher = new FileSystemWatcher();
    oFileSystemWatcher.Path = "C:\\Temp";
    oFileSystemWatcher.NotifyFilter = NotifyFilters.LastAccess
          | NotifyFilters.LastWrite | NotifyFilters.FileName |
            NotifyFilters.DirectoryName;
    oFileSystemWatcher.Filter = "*.txt";

    // Add event handlers.
    oFileSystemWatcher.Created += new FileSystemEventHandler(OnCreated);

    // Begin watching.
    oFileSystemWatcher.EnableRaisingEvents = true;

    // Wait for the user to quit the program.
    Console.WriteLine("Press \'q\' to quit the sample.");
    while(Console.Read()!='q');
}

// The event handler
private static void OnCreated(object source, FileSystemEventArgs e)
{
    Console.WriteLine("File: " +  e.FullPath + " " + e.ChangeType);
}
```

Please note: The `Created` event does not guarantee that the other app or user has finished writing the file. For example, many times a server needs to wait for a user to upload a file in a specific folder. The `Created` event is raised as soon as the user starts uploading the file, but the server needs to wait until the copy/upload process completes. There is no ideal method I am aware of to ensure the transfer completion. A work-around is to wait until an open operation on that file can succeed. While the copy/upload is in progress, open will fail.

The above function will work with console applications only (as it has `Console.Writeline`s). For your use, modify it the way you need.

Trick 9: Give a human face to your app

I don't know about you, but I had always starved to give my app users an extra helping hand if they need one. Microsoft provides a really neat COM component called MS Agent for this purpose. It can show a friendly cartoonish character with our app - to read text, do animations and add some life to your otherwise boring app!

(1) Add the MS Agent COM component in your VS.NET toolbox by clicking Visual Studio menu Tools > Add/remove toolbox items > COM Components. Check ‘Microsoft Agent Control 2.0’ and click ok.

(2) The agent control will now be shown under the ‘components’ tab in the VS toolbox. In your Windows Form based application, drag and drop it on your form. The component will get added to the form and you will see the following member in your form

```
public class Form1 : System.Windows.Forms.Form
{
    private AxAgentObjects.AxAgent axAgent1;
```

Now manually add the following member just after the above code.

```
private AgentObjects.IAgentCtlCharacter speaker;
```

(3) Now you have the agent infrastructure in place and you also have the speaker object. On a suitable event like form_load or button_click Add the following code:

```
try
{
    this.axAgent1.Characters.Load("merlin" , "merlin.acs");
    this.speaker = this.axAgent1.Characters["merlin"];
    this.speaker.Show(null);
    this.speaker.MoveTo(800, 600, null);
    this.speaker.Play("Explain");
    this.speaker.Speak("Give some text here for me to talk...", null);
}
catch
{
    MessageBox.Show("Invalid charater");
}
```

Note that it is better to load the agent once, typically when your form loads. And then use it on various events as you want.

(4) That's it, you are done. Execute the program and see the agent in action. In case you do not have the agent files installed, visit the following link to download agent files and if required TTS engines for desired languages:
http://www.microsoft.com/products/msagent/downloads[^]
Please note that using the Agents makes sense only in the right apps and if utilized correctly. If used in a wrong app or in a wrong way, it can be very irritating to users.

Trick 10: Invoke utilities from your program

Sometimes executing a batch file or a utility program with command line arguments is so handy than to write your own code. For example suppose your program generates an report in MS Excel format, you can use the code given below to launch the freshly generated file in Excel…

```
System.Diagnostics.Process.Start(
@"c:\program files\microsoft office\excel.exe", @"c:\myFile.xls");
```

Using the code

I have deliberately not attached code samples and project files. Because the idea is to keep the code as concise and task-specific as possible. All you need to do is, copy-paste the example functions directly into your code and start using them.

If I receive reasonable responses from users asking for downloadable code files, I will plan to upload a ToolBox class DLL which will encapsulate all this functionality so that it can be used as a component in your projects.

Last word

I have compiled this list while working on a C# .NET server product for the first time, so minor inconsistencies ought to exist. Please correct me if you have better solutions.

I hope you enjoyed this article and will be using some ideas from this in your projects... Best of luck and thanks a lot! :)

History

- Tuesday, September 07, 2004: First draft.
- Tuesday, September 09, 2004: Enhancements suggested by Adam.
- Tuesday, September 14, 2004: Remaining tricks, and some polishing up.
- Monday, November 08, 2004: Some final touches.

About Salil Khedkar

Salil got hooked to computers when he had his first intro with the 8080 in the teens. He used to do some old fashioned DOS/BASIC programming in the school labs then. Out of the increased interest in programming, he chose Computer Science later, for his graduation and post graduation with Pune University. In college he spent more time with coding cool programs and projects, than with the books… Those DOS and Windows based C++ programs gave a lot of thrill !

Apart from computers, Salil is interested in photography and loves to experiment with his Nikon 3700 digicam and Canon EOS 300v SLR. Reading and music are his other hobbies.

After 4 years as a Microsoft Professional with VC++/MFC/SQL/IIS/ASP/ATL COM experience in marketing and financial domain, Salil recently got introduced to the world of C#.NET and is now seen busy exploring the beautiful world !

Click here to view Salil Khedkar's online profile.

Discussions and Feedback

Updated: 16 Nov 2004 8:21