



Articles » Web Development » Trace and Logs » Debug and Tracing

Console Enhancements



Wesner Moise

3 Jul 2003 CPOL

This article enhances console support in .NET such as clearing, colored text, and more; enables console support in Windows apps as well as DOS apps; and also eases testing and debugging of .NET applications.

[Download source files - 17 Kb](#)

Introduction

System.Console is a class provided by the framework to handle console I/O and redirection. The Win32 API also supports a set of console APIs. Win32 supports a single console for each application (process), even a Windows Forms Application. Consoles support a number of features such as buffering, full-screen mode, colors, and so on. You can call another process and capture its output into your own console. With console applications, the console is inherited from the calling process; however, Windows application are detached from any console.

Unfortunately, the **Console** class does not take advantage of most of the features supported in the console APIs. So, I introduced a new class called **WinConsole**, that provides more of the functionality offered by the Win32 class. It provides some of the more popular console functions, but not yet all of them. It can be used in both Console and Windows Applications and probably Class Libraries as well. I plan to add the full range of functionality sometime in the future.

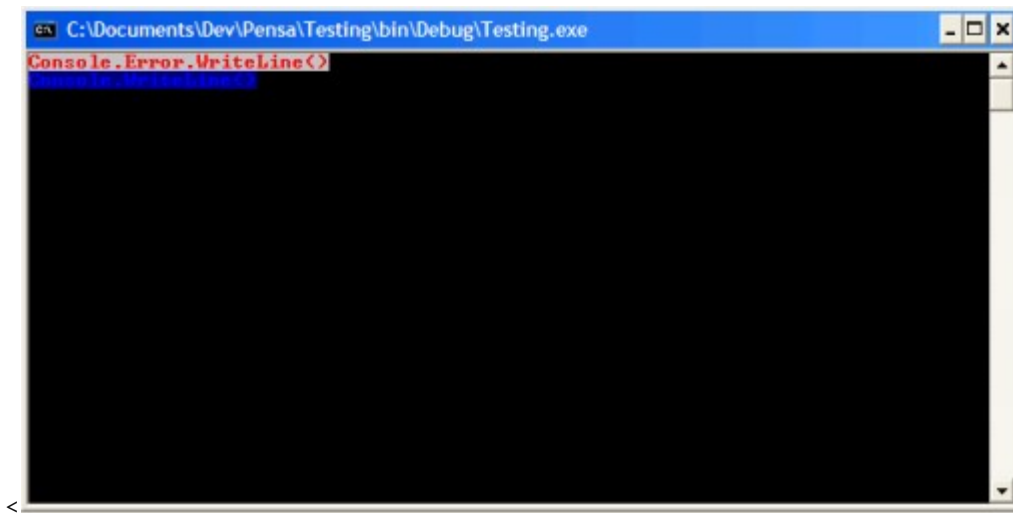
I initially intended to use it for debugging purposes and was going to call it **DebugConsole**, but it is also used as a replacement or extension for the existing **Console** class. I found that testing a low-level UI-less, data structure or class in a full-fledged Windows Application can be very difficult. Standard I/O no longer work. The Debugger Output Window is just a poor substitute for the console. It has fewer capabilities than the console and requires flipping between the application and the debugger. Now, with WinConsole, **Console.WriteLine()** and **Console.ReadLine()** are both available for WinForms apps.

Web applications are still out of luck. Sorry.

WinConsole in Console Application

In **Console** applications, **WinConsole** provides additional features such as the following:

- 1) Hiding and showing the console window (**WinConsole.Visible = false**)
- 2) Clearing the console window (**WinConsole.Clear()**)
- 3) Getting or setting the cursor location
- 4) Changing the color of text to one of 16 different console colors (**WinConsole.Color = ConsoleColor.Red**)



- 5) Having standard error output written with a different color as demonstrated above.
- 6) Automatically flashing window and beeping when an error message is written out. (**WinConsole.Beep()**)
- 7) Capturing asserts and traces to standard error each with a different color scheme

To read input and write output, you can use **Console.WriteLine** and **Console.ReadLine**. You can also use **WinConsole.WriteLine** or **WinConsole.ReadLine**, but it always defers to the **Console** functions.

WinConsole in Windows Application

WinConsole truly shines in a Windows application environment. It enables the use of an additional console window alongside the main application window, which can be hidden and shown anytime. It contains all the additional enhancements, listed above for Console applications, and more.

WinConsole is especially useful for debugging and tracing. Testing data structures becomes a lot harder when moving from a console app to a windows application, because of the omission of console window to report interesting messages. The standard **Debug.Write**, **Trace.Write** call **OutputDebugString**, which sends data to the limited Debugger's output window.

With **WinConsole**, calls from **Debug.WriteLine**, **Trace.WriteLine**, and even **Console.WriteLine** and **Console.Error.WriteLine** can output to the application console window, each in its own independent color. In addition to coloring, there is a flashing mode that can be selected (flash once, flash until response) to alert the developer/tester to warning and error messages.

Input can also be read from the console window through `Console.ReadLine()`. This allows the application to easily receive input from the user in a simple synchronous model instead of a more complex event-driven model. In this way, an entire command-line system can be developed for the application while it is running for performing various tests.

The best part is that the console window is owned and painted separately by the operating system in a separate thread, so that it is always viewable while debugging even after entering break mode or flipping back and forth between the debugger and the application. In contrast, normal application windows are frozen--they cannot be redrawn since event processing is suspended after a break.

WinConsole API

To use the `WinConsole` in a Windows application, `WinConsole.Visible` should be assigned at some point before the console is used. Of course, the assignment needs to be true in order to see any contents. Alternatively, `WinConsole.Initialize()` can be called, but the console may not be visible.

In console applications, `WinConsole.Initialize()` is sufficient. Most of the properties and methods in `WinConsole` will cause an initialization to occur as well, if it has not already occurred.

Below is a list of properties.

| Properties | Description |
|--------------------------------|--|
| <code>Buffer</code> | (<code>IntPtr</code>) Get the current Win32 buffer handle |
| <code>BufferSize</code> | (<code>Coord</code>) Returns the size of buffer |
| <code>Color</code> | (<code>ConsoleColor</code>) Gets or sets the current text and background color and other attributes of text |
| <code>CtrlBreakPressed</code> | (<code>bool</code>) indicates whether control break was pressed. Value is automatically cleared after reading. |
| <code>CursorPosition</code> | (<code>Coord</code>) Gets and sets the current position of the cursor |
| <code>Handle</code> | (<code>IntPtr</code>) Get the <code>HWND</code> of the console window |
| <code>MaximumScreenSize</code> | (<code>Coord</code>) Returns the maximum size of the screen given the desktop dimensions |
| <code>ParentHandle</code> | Gets and sets a new parent <code>hwnd</code> to the console window |
| <code>ScreenSize</code> | (<code>Coord</code>) Returns a coordinates of visible window of the buffer |
| <code>Title</code> | Gets or sets the title of the console window |
| <code>Visible</code> | Specifies whether the console window should be visible or hidden |

To change the color of text, `Color` must be assigned a `ConsoleColor`.

`ConsoleColor` is an `enum` consist of flags `Red`, `Blue`, `Green`, `Intensified`, `RedBG`, `BlueBG`, `GreenBG`, `IntensifiedBG`. The BG colors are for the background color. By using various combinations of each color flags, you can achieve 16 colors for text and 16 colors for the background. To produce white, `Red|Green|Blue`

Intensified must be set. When the intensified flag is missing, the unintensified color is midway between black and the chosen color. So, white becomes gray, red becomes dark red, and so on.

Below is a list of properties.

| Method | Description |
|---|---|
| <code>Beep()</code> | Produces a simple beep |
| <code>Clear()</code> | Clear the console window |
| <code>Flash(bool)</code> | Flashes the console window (Currently now working on my machine, if you can figure out why, I would be grateful) |
| <code>GetWindowPosition(out int x, out int y, out int width, out int height)</code> | Gets the Console Window location and size in pixels |
| <code>Initialize()</code> | Initializes <code>WinConsole</code> -- should be called at the start of the program using it |
| <code>LaunchNotepadDialog(string arguments)</code> | |
| <code>RedirectDebugOutput(bool clear, ConsoleColor color, bool beep)</code> | Redirects debug output to the console clear - clear all other listeners first color - color to use for display debug output |
| <code>RedirectTraceOutput(bool clear, ConsoleColor color) </code> | Redirects trace output to the console |
| <code>SetWindowPosition(int x, int y, int width, int height)</code> | Sets the console window location and size in pixels |

In addition to these methods and properties, **WinConsole** also includes all the standard **Console** methods and properties, which it simply redirects to the **Console** class, so that there is exactly no difference between calling a **Console** method and its corresponding **WinConsole** method. For example, **WinConsole.WriteLine** calls **Console.WriteLine**.

Coloring Standard Error Output, Debug Output or Trace Output

I have also included a **ConsoleWriter** class, which can be used to provide colored output, flashing, and/or beeping to the console window.

ConsoleWriter is constructed by calling `new` on the `ConsoleWriter(TextWriter writer, ConsoleColor color, ConsoleFlashingMode mode, bool beep)` constructor.

Redirecting Standard Error:

```
Console.Error = new ConsoleWriter(Console.Error, ConsoleColor.Red |  
ConsoleColor.Intensified, 0, true);
```

Redirecting Debug or Trace Output:

```
Debug.Listeners.Remove("default"); // Debug.Listeners.Clear();<BR>  
Debug.Listeners.Add( new TextWriterTraceListener( new  
ConsoleWriter(Console.Error, ...) ) );
```

You can also use the convenience function: `WinConsole.RedirectDebugOutput(...)`.

Launch Notepad Dialog

At Microsoft, virtually every group uses a command-line environment for building, testing and so on; this can sometimes make it difficult to get a complex set of information from a user, because of the UI-less environment.

Thus, the infamous notepad dialog was invented. A command would start a notepad.exe with a filename argument and wait for it to exit. The file is prepopulated with help comments and default information. The user edits the file and indicates OK by exiting notepad. The suspended command, which was waiting for notepad to exit, is now resumed and ready to process the notepad file.

The image shows a typical notepad dialog.

`WinConsole.LaunchNotepadDialog` takes a filename argument, and launches a notepad dialog, in which the user can modified the specified file.

Conclusion

This represents just one of my articles in the debugging series. There will be others.

I'd appreciate your vote; it is a powerful motivating force for me.

Version History

- June 28, 2003 - Original article.

License


This article, along with any associated source code and files, is licensed under [The Code Project Open License \(CPOLE\)](#)

About the Author



Wesner Moise



CEO SoftPerson; previously, Microsoft
United States 

I am a software entrepreneur and former Microsoft Excel developer

I founded SoftPerson LLC (softperson.com) to build software using artificial intelligence to perform tasks associated with people. My business plan was a finalist in a national competition.

I helped develop Microsoft Excel 97, 2000 and XP. I received a BA from Harvard College in Applied Mathematics/Computer Science and an MBA from UCLA in technology entrepreneurship. I also obtained an MCSE/MCSD certification in 1997. My IQ is in the 99.9 percentile. I received a Microsoft MVP award in 2006.

My technical blog on .NET technologies is wesnerm.blogs.com.

My personal website is <http://wesnermoise.com>.

My company website is <http://softperson.com>.

Comments and Discussions

 **64 messages** have been posted for this article Visit <https://www.codeproject.com/Articles/4426/Console-Enhancements> to post and view comments on this article, or click [here](#) to get a print view with messages.

[Permalink](#)
[Advertise](#)
[Privacy](#)
[Cookies](#)
[Terms of Use](#)

Article Copyright 2003 by Wesner Moise
Everything else Copyright © [CodeProject](#),
1999-2019

Web03 2.8.191213.1

