Multiple consoles for a single application

Zvika Ferentz

Rate me: 4.95/5 (23 votes)

10 Mar 2006 Public Domain

Sometimes, it's not enough to have only a single console window for your applcation... let's provide more!!

Download demo project - 19.2 Kb

Download source - 14 Kb

Introduction

The problem

Sometimes, a developer needs to display an application's output in several console-windows:

- You are part of a development-team, and each developer uses the `printf()` function to display some debug data.
- Your program outputs different types of data, and you would like to give each type a separate console.

But there's a problem - when you're writing a console-application, you can have only a single console-window! The MSDN documentation of the `AllocConsole()` says "A process can be associated with only one console", but what if we want some data to be displayed in another console-window, or even in another computer?

The solution

The solution is actually very simple - our process can start a new "helper" child-process, so the helper process will display whatever our process sends it. We can easily implement such a solution with pipes: for each new "console" (that I'll call logger), we'll open a pipe, and execute a "Console-Helper" application - the role of this application is very simple, it will print **everything** sent through the pipe.

If you want to display the output on another machine, you can easily convert the code to use TCP or UDP sockets instead of the pipe.

Using the code

The solution is divided into two main sections:

- **libProcessHelper**: a **very simple** library that you embed in your application. If you want a new console - you just create an instance of the `CConsoleLogger` class, call the `Create()` function, and print the output using one of the class-output-

functions. When you call the `Create()`, the function starts up a new child-process that uses a pipe to get data from our application. Notice that the child process is started with a single command-line parameter - the name of the pipe to open. All other parameters (title, size of the buffer, etc.) are passed via the pipe during its initialization.
- **ConsoleLoggerHelper.exe**: this is the small helper console-application. It just opens an existing pipe and waits for data to print.

Notice:

1. In order to use the suggested solution, you don't need to understand anything about the internal implementation. You just call the `Create()` function and it will do anything:

```
Copy Code
```

```
CConsoleLogger another_console;
another_console.Create("This is the first console");
another_console.printf("WOW !!! COOLL !!! another console ???");
```

2. We can use the same technique and provide a more sophisticated console that can display coloured data and even perform extra operations. In fact, the library contains a derived class `CConsoleLoggerEx` that implements coloured-operations with "extra" functionality, such as clear-screen, clear till the end of line, move cursor, ...

```
Copy Code
```

```
CConsoleLoggerEx coloured_console;
coloured_console.Create("This is the coloured console");
coloured_console.cls(CConsoleLoggerEx::COLOR_BACKGROUND_RED);
coloured_console.gotoxy(10,10);
coloured_console.cprintf( CConsoleLoggerEx::COLOR_WHITE |
      CConsoleLoggerEx::COLOR_BACKGROUND_BLUE,"White on Blue");
```

3. Once you've created a console, you can output data with the `printf()` member-function. However, you can use the `SetAsDefaultOutput()` member function to specify that even the CRT `printf()` will be redirected to this new console.

About the source code

The library: libProcessHelper

The library contains the `CConsoleLogger` and the derived class `CConsoleLoggerEx`. Their usage is (of course) very similar, so I'll stick to the `CConsoleLogger`. The `Create()` member function opens a new pipe using the `CreateNamedPipe` API. We're using the current process name and the current time so we can uniquely identify "our" console/pipe. Then, we create a new helper-child-process (using the `CreateProcess` API). This new process will open the pipe and display whatever we send it. Right after creating the helper process, we pass some data (console title, console size, user info, ...), each data is sent as a separate textual line (just like HTTP headers). We also call a virtual-function `AddHeaders()` so any derived-class can add new headers.

In order to print some data, the user calls the `printf` function (if you don't need any formatting, use the `print()` function which is more efficient).

You can redirect all the output (to `STDOUT`) to be passed to this console, using the `SetAsDefaultOutput` function.

The child process: ConsoleLoggerHelper

Its code is very simple:

- We receive the name of the pipe to open via the command-line.
- We open the pipe and read all "textual-headers" until we get `NULL` for each header, we check whether we can recognize it and operate accordingly.
- When we're done with the headers, we start the main loop. If this is an "extended console", we call the `ConsoleExLoop()` function; otherwise, we call the `ConsoleLoop()` function.

I won't get into the details about the helper's implementation due to the fact that the usage of pipes/console-windows are well documented inside the MSDN and other resources. In addition, this was written only as a "proof of concept", and I did'nt put too much effort to make it look better. It's just a single CPP file, with some very simple global functions, nothing more.

Building and running

Because it is implemented as a library (*.lib* file), you only need the header (*.h*) file, and the library itself. However, it's also possible to use the *.cpp*+*.h* files directly in your code.

By default, when you start a new console with this library, it looks for the default console helper filename:

Copy Code

```
#define DEFAULT_HELPER_EXE  "ConsoleLoggerHelper.exe"
```

The executable should be found in the current-working-directory. If you want to use another filename/location, you can use the environment variable or specify the filename as one of the `Create()` arguments.

Worth mentioning

The following section is for "advanced" (a.k.a. experienced) programmers. If it doesn't tell you anything, or it looks like Chinese, you can ignore it or study it. Actually, it's not complicated at all!!

- In order to prevent any race-condition (especially in the derived class, `CConsoleLoggerEx`), we need to use some kind of mutual-exclusion mechanism.
- If you are building the project with the Microsoft-Platform SDK, we'll be using the `InterlockedCompareExchange()` API to provide us the lock (see references in the source code itself!!).
- If you are building the project without the SDK, we'll be using the traditional `CRITICAL_SECTION` object, which is a little bit "expensive" (less efficient, time/CPU consuming).
- Nice to know: in order to detect the existence of the SDK (and display a nice warning if it does not exist), I'm using the following code:

  Copy Code

  ```
  #include "ntverp.h"
  #if !defined(VER_PRODUCTBUILD) ||  VER_PRODUCTBUILD<3790
  #pragma message ("****************************************"
                   "************************************************")
  #pragma message ("Notice (performance-warning):"
                   " you are not using the Microsoft Platform SDK,")
  #pragma message ("                              we'll use "
                   "CRITICAL_SECTION instead of InterLocked operation")
  #pragma message ("****************************************"
                   "************************************************")
  #else
  #define CONSOLE_LOGGER_USING_MS_SDK
  #endif
  ```

Future possible enhancements:

- Working over TCP/IP or UDP.
- Storing the data to a file once in a while.
- Why use a simple console? It can be a powerful graphics logger (what about MFC's `CListBox`??).
- Adding new functionality to the "extended console": set its position and scrolling attributes, ...

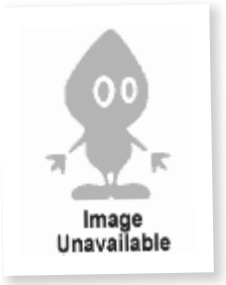
References

MSDN articles, MSDN documentation, MSDN magazine, and even some stuff from MSDN :)

License

Share

About the Author



Zvika Ferentz
Architect Protegrity
United States

No Biography provided