Microsoft Source Server – Git Extensions

The files found herein are to facilitate the debugging experience while using Microsoft Visual Studio with the git source control management system. The source indexing mechanism gives the advantage of being able to retrieve the exact source file used to compile a specific version of an assembly while stepping into the assembly code. While not an excuse to avoid TDD or other forms of automated testing, this does give the advantage to be able to step into a compiled library where the source would not normally be readily available.

Prerequisites

1. Microsoft Windows XP (or later)
2. Microsoft Visual Studio 2005 (or later), Express Edition (or better)
3. Microsoft Debugging Tools for Windows
4. ActiveState Perl (or another Perl interpreter)
5. Cygwin with git or msysgit
6. OpenSSH or Putty
7. SSH, password-less (private key) pull capabilities from remote “origin” git repositories
8. Perl, git, and OpenSSH configured in the PATH environment variable

Installation Instructions

1. Copy the files in SourceServer directory to:
   C**:\Program Files\Debugging Tools for Windows (x64)\srcsrv**
   (Actual installation path my vary depending upon your processer architecture.)
2. Optional: Add **C:\Program Files\Debugging Tools for Windows (x64)\srcsrv** to your PATH environment variable.
3. Copy the files in the VisualStudio directory to:
   C:\Program Files (x86)\Microsoft Visual Studio 9.0\Common7\IDE
   (Actual installation path my vary depending upon your processer architecture.)
4. Copy the files in the InSystemPath directory to:
   C:\Program Files (x86)\Microsoft Visual Studio 9.0\Common7\IDE
   (As long as the InSystemPath files are in the PATH environment variable, everything will work, but just during initial installation and to get everything going, put it into the Visual Studio directory. Once everything is working properly, feel free to change the directory.)
5. Open Visual Studio and in the Debugging Options and ensure that “Enable source server support” is checked. Enabling “diagnostic messages” is a good idea as well.

Usage Instructions

Setup:

1. Open Visual Studio.
2. Create a new C# library project.

3. Create a public class with a few public methods to call.
4. Build your project.
5. Add your project and source code to git (refer to git tutorials on the web on how to do this).
6. Commit your code.
7. Push your code to the origin repository. (In a normal scenario, the source indexing mechanism will occur on a continuous integration server, so you needn't worry about pushing changes to the origin repository.)

Indexing:

1. Run the following command from the command prompt while in your MyLibrary directory. gitindex.cmd /debug /source=. /symbols=.\MyLibrary\bin\Debug
2. The output will be something like this:

```
--------------------------------------------------------------------------------
ssindex.cmd [STATUS] : Server ini file: C:\Code\MyLibrary\srcsrv.ini
ssindex.cmd [STATUS] : Source root    : .
ssindex.cmd [STATUS] : Symbols root   : .\MyLibrary\bin\Debug
ssindex.cmd [STATUS] : Control system : GIT
--------------------------------------------------------------------------------
ssindex.cmd [STATUS] : Running... this will take some time...
ssindex.cmd [INFO  ] : ... indexing .\MyLibrary\bin\Debug\MyLibrary.pdb
ssindex.cmd [INFO  ] : ... wrote C:\Users\me\AppData\Local\Temp\indexFE66.stream to
.\MyLibrary\bin\Debug\MyLibrary.pdb ...
```

Debugging

1. Move the MyLibrary directory to MyLibrary.bak. (This is so the normal Visual Studio source file resolution mechanisms cannot find the appropriate file which then triggers the source server component within Visual Studio to do its job of pulling from the git repository.)
2. Setup a C# console application.
3. Add a reference to MyLibrary.dll
4. Write some code that uses a public method on your library.
5. Set a breakpoint at the first usage of your library code.
6. Run the console application.
7. When the breakpoint is hit, step into your library code (F11 on the keyboard).
8. Visual Studio will freeze for a few seconds while it talks through the source retrieval script to the remote git repository.
9. After a few moments, your source file will appear on the screen and you will be able to step through it as if it were part of your project.
10. Subsequent calls to this source file and other source files in your library will go much more quickly than the initial one because the git repository is now cloned to your PC in a location known to the source server. In my case, it's at: C:\Users\me\AppData\Local\SourceServer\<git repository SHA1 identifier>\.localRepo\.

Usage Notes

1. Although Cygwin's default Perl implementation could work, it seems to argue with Microsoft's source indexing mechanism. For this reason, we use ActiveState and we put it in the PATH environment variable BEFORE any the Cygwin bin path is listed (if you're using Cygwin+git instead of msysgit).
2. Please be sure that you have no trouble connecting and pulling from your remote "blessed" git repositories before you attempt to debug using the above instructions. This seems to be the most common problem.

Acknowlegements

1. Michael Collins at Imaginary Realties for writing the initial git.pm script.
2. John Robbins in MSDN Magazine's BugSlayer series.