Sourcepack

Author: Sebastian Solnica

Introduction

When using third-party libraries in your projects you often would like to know what's happening inside the library. For close-source projects there is no other way then using Reflector or ILSpy debugger (though we may end up with unreadable obfuscated code), but for open source projects there should be no problems with loading the valid source code into the debugger. There is a great initiative (http://www.symbolsource.org/) which aims at providing source-indexed symbol files for open-source projects, but unfortunately not many projects are available there and even for those accessible their version is not always accurate. Finally you are left with a manual setup and compilation of the source code (referencing the binaries using, for example, reference paths). But we could do it differently. I observed that it's more and more popular for open source project authors to provide the binaries with their pdb files. The pdb files in this form (especially for managed applications/libraries) are not very useful, although it's not difficult to change this situation. You will need a **sourcepack** script and source code zip file (which is usually provided by the author beside the binaries). **Sourcepack** modfies the symbols pdb files so that they will reference the source archive file and any debugger (which supports source server) will be able to extract the required source file on demand.

Script description and usage

The **sourcepack** script is a powershell script which examines and then modifies the pdb files in the given directory to make them reference the source code archive file. Firstly we must have a quick look at the PDB file structure.

PDB file structure

When you build your application/library either in the debug or pdb-only release mode the compiler will emit, beside the binaries, PDB files. In general PDB files contain information for the debugger how to bind the processor instruction addresses with the lines of the source code file (PDB files are even more important for native builds when they store metadata about the types and functions declared in the binaries). Normally PDB files contain only absolute paths to the source code files and thus are

usable only on machines which store the source code files in the same place as defined in the PDB file. To find those absolute addresses you may use the srctool application (with -r switch) which is a part of the Debugging Tools for Windows:

```
D:\lab\symbols\ConsoleApplication2\bin\Debug>srctool.exe -r ConsoleApplication1.pdb
D:\lab\symbols-lab\symbols\ConsoleApplication1\Program.cs
D:\lab\symbols-lab\symbols\ConsoleApplication1\AdvertQuickView.cs

D:\lab\symbols\ConsoleApplication2\bin\Debug>
```

However it's not the only way the PDB files may reference source code.

There is a special stream in the PDB file which can inform the debugger where to look for a source code file. The stream format is very extensible and you can actually put there any command you want under only one condition - it must extract the desired source code file into the target directory. Normally with the Debugging Tools for Windows you receive a bunch of scripts for different source code repositories (SourceSafe, CVS, Subversion).

The source indexing usually consists of following steps:
1. Index all source code files.
2. Index PDB files and match them with the already found source files.
3. Create a temporary stream file, which looks more or less like the one below (I marked with bold the mandatory fields, and in blue the section names):

```
SRCSRV: ini ------------------------------------------------
VERSION=1
INDEXVERSION=2
VERCTRL=Subversion
DATETIME=Sat Aug 13 08:36:36 2011
SRCSRV: variables ------------------------------------------
SVN_EXTRACT_TARGET=%targ%\%fnbksl%(%var3%)\%var4%\%fnfile%(%var1%)
SVN_EXTRACT_CMD=cmd /c svn.exe cat "%var2%%var3%@%var4%" --non-interactive > "%svn_extract_target%"
SRCSRVTRG=%SVN_extract_target%
SRCSRVCMD=%SVN_extract_cmd%
SRCSRV: source files ---------------------------------------
d:\lab\symbols-lab\symbols\consoleapplication1\program.cs*svn://localhost/*Program.cs*2
SRCSRV: end ------------------------------------------------
```

(more information about stream file format may be found here)

4. Write the temporary stream file to the PDB file.

As you may see in the snippet above, there is a special SRCSRVCMD variable which will be ran by the debugger if it does not find the source code file at the absolute path.

Sourcepack script

The sourcepack script may be considered as a just another tool for indexing the PDB files which uses the archive file (zip, 7z or any other) as a source code repository. So the extract operation will simply consists of calling one of the packer applications (7z, winzip, rar etc.) with correct arguments. For example for a 7z command the temporary stream file may look as follows:

```
SRCSRV: ini ------------------------------------------------
VERSION=1
INDEXVERSION=2
VERCTL=7z
DATETIME=08/19/2011 08:47:30
SRCSRV: variables ------------------------------------------
SRCSRVTRG=%targ%\%var2%
SRCSRVCMD=cmd /c 'C:\Program Files\7-zip\7z.exe' x C:\temp\ConsoleApplication1.zip
-o%targ% %var2%
SRCSRV: source files ---------------------------------------
c:\devwork\symbols\ConsoleApplication1\Program.cs*Program.cs
SRCSRV: end ------------------------------------------------
```

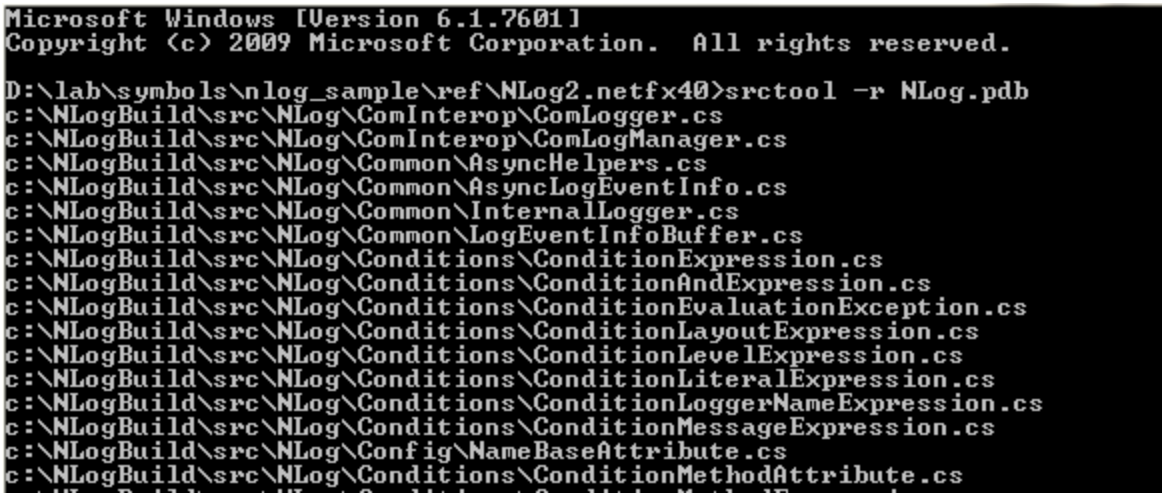
Usage description and examples

Let's assume that we have a very simple console application that uses the NLog logging library (this code snippet is actually taken from the NLog example directory):

```
class MyLogger
{
    private Logger _logger;

    public MyLogger(string name)
    {
        _logger = LogManager.GetLogger(name);
    }

    public void WriteMessage(string eventID, string message)
    {
        LogEventInfo logEvent = new LogEventInfo(LogLevel.Info, _logger.Name, message);

        logEvent.Properties["EventID"] = eventID;

        _logger.Log(typeof(MyLogger), logEvent);
    }
}

class Program
{
    static void Main(string[] args)
    {
        MyLogger l = new MyLogger("uuu");

        l.WriteMessage("1234", "message");
```

```
    }
}
```

To compile this code we need to download the binaries, from for example nlog.codeplex.com and reference them while compiling. Fortunately binaries come with PDB files, so let's have a look which files do they reference (below you can see a small snippet of the ouput):



We can see that the author kept the sources at the root location: c:\NLogBuild\ - we will need this information for further actions. We could stop here, download the source code, extract it to the c:\NLogBuild\ directory and start NLog source stepping. However, taking this approach for all the source projects you would like to debug, firstly might not always work out and secondly will result in a really messy directory tree and a big loss of your hard drive space (source files are kept uncompressed). **Sourcepack** was designed to resolve all those problems. It enables you to keep all the compressed source packages in one place and modify only the downloaded PDB files to reference them. In our NLog example let's create a C:\Sources folder and copy the NLog source package there. Now, let's run the **sourcepack.ps1** command (you may get it from sourcepack.codeplex.com):

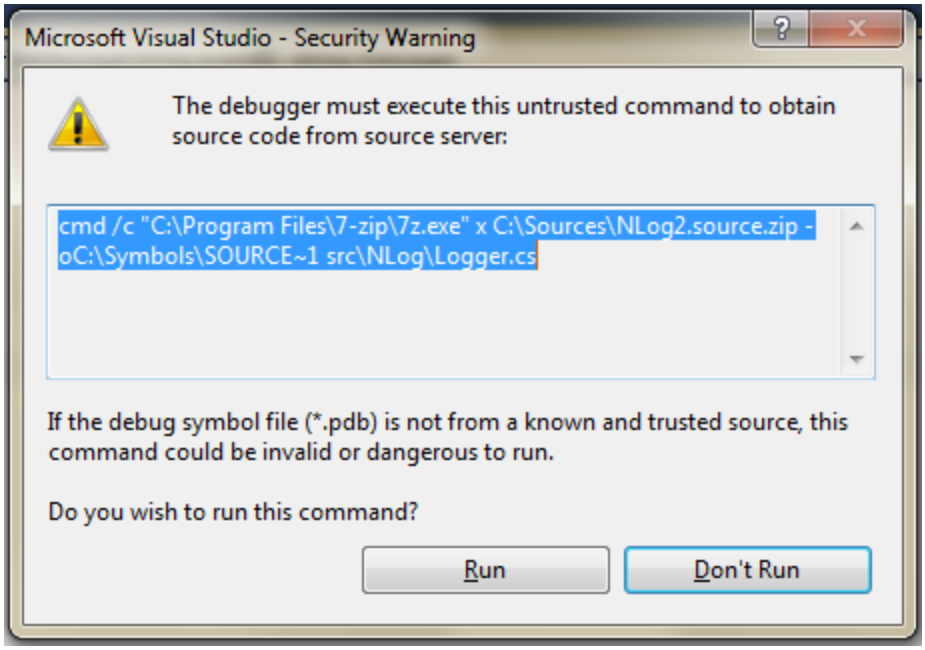
```
PS C:\> .\sourcepack.ps1 -symbolsFolder D:\lab\symbols\nlog_sample\ref\
-sourcesRoot c:\NLogBuild -sourceArchivePath C:\Sources\NLog2.source.zip
```

After this command finishes you may now rerun the srctool command on the NLog PDB files and find that they now contain source stream information embedded:

```
D:\lab\symbols\nlog_sample\ref\NLog2.netfx40>srctool.exe NLog.pdb
[c:\NLogBuild\src\NLog\ComInterop\ComLogger.cs] cmd: cmd /c "C:\Program Files\7-zip\7z.exe" x
C:\Sources\NLog2.source.zip -oD:\lab\symbols\nlog_sample\ref\NLog2.netfx40
src\NLog\ComInterop\ComLogger.cs
[c:\NLogBuild\src\NLog\ComInterop\ComLogManager.cs] cmd: cmd /c "C:\Program Files\7-zip\7z.exe" x
C:\Sources\NLog2.source.zip -oD:\lab\symbols\nlog_sample\ref\NLog2.netfx40
src\NLog\ComInterop\ComLogManager.cs
[c:\NLogBuild\src\NLog\Common\AsyncHelpers.cs] cmd: cmd /c "C:\Program Files\7-zip\7z.exe" x
C:\Sources\NLog2.source.zip -oD:\lab\symbols\nlog_sample\ref\NLog2.netfx40
```

```
src\NLog\Common\AsyncHelpers.cs
...
```

Now, start your favourite debugger (must support source server streams in PDB files) and try to step into for example Log method of the Logger object. The debugger should prompt whether you want to execute the source server command which was found in the PDB file. In Visual Studio 2010 this dialog looks as follows:



After you agree to run the command you should start source stepping the NLog code:

```
/// <summary>
/// Writes the specified diagnostic message.
/// </summary>
/// <param name="wrapperType">The name of the type that wraps Logger.</param>
/// <param name="logEvent">Log event.</param>
public void Log(Type wrapperType, LogEventInfo logEvent)
{
    if (this.IsEnabled(logEvent.Level))
    {
        this.WriteToTargets(wrapperType, logEvent);
    }
}
```

As this dialog may get a bit annoying after extracting several files you may get rid of it. What you need to do is to modify (or create) srcsrv.ini next to the srcsrv.dll file that is being used by your debugger (for VS2010 it's c:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\):

```
[trusted commands]
cmd.exe
```

This creates some risk as now all the commands in format of “cmd.exe <some command>” embedded in the PDB files will be executed without asking for permission so please take it into account.

Sourcepack argument reference

The table below describes all possible parameters that can be passed to the sourcepack script:

Parameter	Status	Default value	Description
-symbolsFolder	**MANDATORY**	N/A	The path of the root symbols directory. The directory is then recursively searched for any PDB files to be indexed.
-sourceArchivePath	**MANDATORY**	N/A	The path of the archive file in which all sources lie.
-outputZipFile (NOT IMPLEMENTED)	**OPTIONAL**	<none>	If this parameter is set the script will produce the zip file with source files referenced by the PDB files.
-sourcesRoot	OPTIONAL	guessing from PDB	The root of the source folder - usually it’s just a path to the folder from which the archive file was created.
-dbgToolsPath	OPTIONAL	<none>	Path of the Debugging Tools for Windows (the srcsrv subfolder) - if not specifed the script tries to find it. If you don’t have the Debugging Tools for Windows in PATH variable you need to provide this argument.
-archiverCommandPath	OPTIONAL	<script_path> \7za\7za.exe	With the script you probably also downloaded a 7za.exe application. If you unpack all the files into the same directory you don’t need to provide this argument. If not plase provide a path to the 7za.exe or 7z.exe including the exe file in it, eg. c:\program files\7-zip\7z.exe

Installation requirements

The following applications must be installed for the script to work:

- Powershell - to run the script :)

- Debugging Tools for Windows
- Any file archiver (free 7zip for instance)