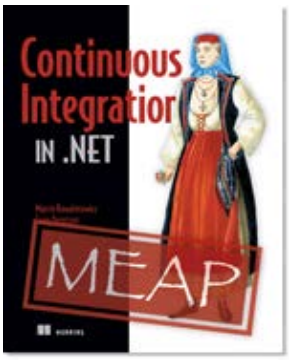
MANNING PUBLICATIONS



Continuous Integration in .NET
By Marcin Kawalerowicz and Craig Berntson

In this article, based on chapter 3 of Continuous Integration in .NET, the authors discuss the tools that help with software build automation, zeroing in on MSBuild as the best choice for the .NET developer using Windows and Visual Studio.

Build Automation Tools

In the Continuous Integration (CI) context, an automation platform is a tool or a set of tools that helps to automate the whole software build process including:

- Compiling the source code.
- Preparing the database.
- Performing various tests.
- Analyzing the code.
- Creating installation routines and deploying.
- Creating documentation.

What we're looking for should be easily maintainable. It should be stored in the source control system just like everything else that takes part in the CI process. The obvious way to automate the build process is to script it using human-readable text. You should avoid everything that doesn't use text as the description of a build process. All compile managers are a bad thing. Every automation tool that has only a user interface and requires you to manually click on it to make it run should get a boot!

Software automation using ordinary command-line commands organized in a batch file has been done for a long time. It is more of a curiosity now, but it shows the general idea. It makes the process faster in comparison to manually issuing the commands, reduces redundant tasks because you don't have to be personally involved in every build, and makes it possible for others to maintain the build. The knowledge of how to perform the build sits in a human-readable form in a file on a server and not in someone's head. Let's walk through some of the real automation tools and search for the one that suits us.

All the make children

Software automation platforms are older than the most active software developers. The great grandfather of almost all the tools we have now was the UNIX make utility. It was created at the end of the seventies and it has been successfully used mostly in the *ix world. It has a Windows version called nmake and a fairly good clone called OPus Make. All the make systems use a text file called a "make file" to describe the software build process. Later generation of tools like Jam and Cook changed this. They used more sophisticated statements to hide some of the lower level aspects of software automation. With time, the automation platforms became bigger and more complex. To emphasize this change, they began to be called automation systems. One of them is GNU Automake with the GNU Build System. GBS is a set of small tools that comes in handy while building software on *ix systems.

To close this short review of all the make ancestors we have to mention the automation tools that use a specific programming language to describe the build process. We have SCons, which uses Python, or Rake, which uses Ruby.

All of the tools we mentioned could be used to set up a continuous integration process. But, we will look at the vanguard of the build automation—the XML-based build systems Ant (NAnt) and MSBuild. The XML-based systems are step away from the tools that are using fairly complicated commands or even programming language to describe the build process. Now you can declare the steps in an XML build script. The steps are easy to extend and adopt.

NAnt and MSBuild are two of the tools to choose from if you are creating a build process in .NET environment. Both do the same job using similar techniques. NAnt is an open-source tool maintained by the community and MSBuild comes from Microsoft. Table 1 shows the most significant differences between them.

Table 1 NAnt vs MSBuild—most significant differences

Feature	NAnt	MSBuild
OpenSource	☺	☹
Cross-platform (Linux, Mono)	☺	☹
Good, if you already know Ant	☺	☹
Built into .NET Framework	☺	☺
Integrated with Visual Studio	☹	☺
Actively developed	☹	☺
Built-in features	☺	😐

Let's have a quick look at NAnt and explain why we will go with MSBuild.

It's not an ant

Once upon a time, there was an Ant. It was a good, established tool used to build applications in Java shops. It was ported to work in the .NET world and called NAnt (Not an Ant). From the Java ancestor, it inherited the XML declarative automation description language.

Let's try to use NAnt with this full-blown single-line C# program.

```
class c{static void Main(){System.Console.Write("Hello NAnt");}}
```

Let's place this program in a file called HelloNAnt.cs. Now, we will write a NAnt script to build our application. Let's call it HelloNAnt.build and specify according to listing 1.

Listing 1 A simple NAnt build script that will clean and compile a single file Windows application

```
<?xml version="1.0"?>
<project name="Hello NAnt" default="build" basedir=".">          #1
   <property name="debug" value="true" overwrite="false" />      #2
   <target name="clean">                                         #3
     <delete file="HelloNAnt.exe" failonerror="false" />         #3
     <delete file="HelloNAnt.pdb" failonerror="false" />         #3
   </target>                                                     #3
   <target name="build" depends="clean">                         #4
      <csc target="exe"                                          #5
           output="HelloNAnt.exe"                                #5
           debug="${debug}">                                     #5
        <sources>                                                #5
           <include name="HelloNAnt.cs" />                       #5
        </sources>                                               #5
      </csc>                                                     #5
   </target>
```

```
</project>
```

#1 Declares a project with default target
#2 Defines debug property
#3 Cleans target
#4 Build target
#5 C# compiler task

A NAnt script is an ordinary XML document. First (#1), we declare the project, specifying the name—`Hello NAnt`, the default target—`Build`, and the working directory—dot for the current directory. NAnt gives us the ability to define properties (#2). A property is a kind of named variable that we can assign a value. The overwrite attribute lets us set the variable from the command line. Our debug variable will be used by the C# compiler task (#5). The `Clean` target (#3) uses two delete tasks to erase unnecessary files. Setting the `failonerror` attribute tells NAnt to ignore possible errors; for example, if there is nothing to delete. The second target, #4 `Build`, will first run the `Clean` target because of the `depends` attribute, and then run the `csc` target to compile the source file.

One of the rules of continuous integration is to have everything you need to fully build a project inside the project directory/repository. To use the script that we just wrote we need NAnt executables[1]. Place the NAnt executables in the tools/nant folder. NAnt is ready to use after that.

Open a command window, navigate to the project folder and type `tools/nant/bin/nant.exe`. NAnt will launch (figure 1), run the script, and build our one-line program.

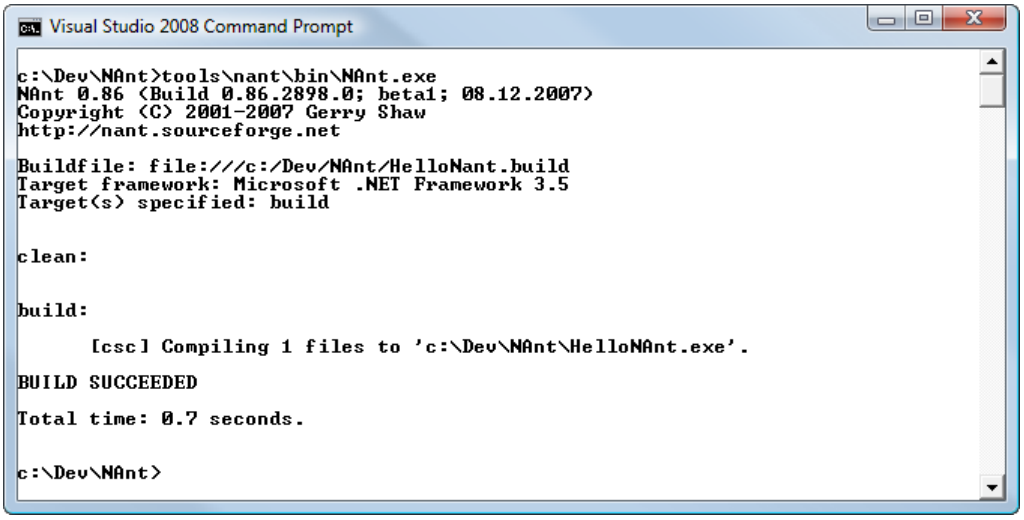


Figure 1 Starting a "Hello World"-style NAnt script. The Build performs a "clean" followed by a "build" task. As an artifact, we get a compiled executable.

Now that you have the script working, you can extend it, declare more steps and integrate more actions. NAnt is a good alternative for the software developers with a Java background that are familiar with its ancestor, Ant. Many developers use MSBuild only for compiling the source code and NAnt for integrating all other tools into the CI process.

The Microsoft worker MSBuild

Microsoft first shipped its own build tool for the .NET platform with the second version of .NET Framework. Updated versions were shipped with the .NET Framework versions 3.0, 3.5, and 4.0. You can check in *C:\Windows\Microsoft.NET\Framework*. All of the subfolders v3.0, v3.5, and v4.0 contain *MSBuild.exe*.

MSBuild means simply less work. You don't need to worry about third-party tools and how to integrate them with your environment. If you are like most of us, this would surely look appealing to you. You don't have to worry if your favorite build tool is installed on the integration machine because, if you have .NET Framework installed, it

[1] You can download the NAnt executables from http://nant.sourceforge.net.

will be there. A pragmatic person would find MSBuild quite appealing. Who knows how your business will grow? You might hit the scaling wall with only the free software and will have to think about something bigger. The whole Microsoft Team Foundation Server Build is set on top of MSBuild. Keep this in mind and you can feel prepared.

MSBuild is freely distributed with the .NET platform. It has the whole Microsoft machinery behind, so there is no worry about wide adoption and popularity. It won't die suddenly, leaving you alone without support and no one to ask about advice. MSBuild is extensible and well documented. MSBuild uses XML syntax similar to NAnt to perform build tasks. It is closely integrated with Visual Studio, it understands Visual Studio solution files and makes it possible to compile Visual Studio solution and projects without Visual Studio. It seems to be the build tool for .NET developers who want to set up a continuous integration assembly line.

Summary

Ideally the automation process is scriptable. Plain text makes it easy to check and change the things are done. The change can be done manually by you editing a project file or automatically by a tool like Visual Studio. It should allow you to define various execution paths. Using conditions and parameterization you should be able to perform various types of builds according to the situation. Your build process should be easily extensible. Sooner or later, you encounter a problem you will have to deal by extending your tool. It should be as easy as possible because the process itself doesn't make anyone richer; it is the software that emerges in the process that makes the money.

There are many tools that deal with the build automation. Right now, MSBuild seems to be the best choice for the .NET developer using Windows and Visual Studio. It is integrated with .NET Framework and used in the UI itself. But there are alternatives. NAnt is still something to look at. The choice is yours.

Here are some other Manning titles you might be interested in:

SQL Server MVP Deep Dives
Edited by Paul Nielsen, Kalen Delaney, Greg Low, Adam Machanic, Paul S. Randal, and Kimberly L. Tripp

ASP.NET MVC 2 in Action
Jeffrey Palermo, Ben Scheirman, Matthew Hinze, Jimmy Bogard, and Eric Hexter

ASP.NET 4.0 in Practice
Daniele Bochicchio, Stefano Mostarda, and Marco De Sanctis

Last updated: March 1, 2011