Team Development for Sitecore: Continuous Integration and Automated Builds

What is continuous integration

Continuous integration is the practice of having all developers in the team integrate their work with the work of others. You would want to do this process frequently and automatically leveraging a build server. By using continuous integration you can expect to reduce the number of issues that arise and hence be more efficient.

This is, obviously, a grossly simplistic description of continuous integration and you can certainly find tons of information about it on the web.

Microsoft Build Engine - MSBuild

Overview

MSBuild is the build system for Visual Studio. If you have ever looked at the contents of any visual studio project you are essentially looking at an XML file that conforms to the MSBuild schema. This XML format is very extensible and because TDS is a visual studio project it itself leverages MSBuild.

Targets

A .targets file contains items, properties and tasks for how the build should happen. If you look at the xml of a web application project you will notice that it imports some out-of-the-box targets. These targets allow you to modify the build to do things such as "`BeforeBuild`" and "`AfterBuild`" steps.

Being that TDS has some advanced features, such as deploying and packaging, we've created our own .targets file that ships with TDS. You can find the .targets file at `C:\Program Files (x86)\MSBuild\HedgehogDevelopment\SitecoreProject\v9.0\HedgehogDevelopment.SitecoreProject.targets`

TDS version 2.2 provides the following Targets that can be used in your builds.

- `BeforeSitecoreBuild` - happens before any TDS work is performed
- `BeforeSitecoreMerge` - happens just before sitecore items are deployed to your installation
- `AfterSitecoreBuild` - happens after all work has been performed
- `BeforeGeneratePackage` - happens before a package is generated
- `AfterGeneratePackage` - happens after a package is generated
- `BeforeFileReplacements` - happens before the file replacement process
- `AfterFileReplacements` - happens after the file replacement process

Example Scenario

For a very simple example let's try to make a copy of the web root files before any file replacements happen. This way we can compare the before and after of the file replacement process.

1. Download and open the Sample Project at http://www.hhogdev.com/Products/Team-Development-for-Sitecore/Documents.aspx
2. Right click on the "TDSProject1-VS20*XX*" project and "Unload" it
3. Right click on the "TDSProject1-VS20*XX*" project and edit it
4. At the bottom of the file, just above the closing `</Project>`, add in the following

```
<Target Name="BeforeFileReplacements" Condition=" '$(Configuration)' == 'Debug' ">
  <!-- define the new location -->
  <PropertyGroup>
    <_CopyPath>$(OutputPath)..\$(Configuration)_COPY</_CopyPath>
  </PropertyGroup>
  <!-- Grab the build output -->
  <ItemGroup>
    <_OutputFiles Include="$(OutputPath)\**\*.*"/>
  </ItemGroup>
  <!-- write a message to output window -->
  <Message Text="Destination: $(_CopyPath)" Importance="high" />
  <!-- clean it up -->
  <RemoveDir Directories="$(_CopyPath)" />
  <!-- perform the copy -->
  <Copy SourceFiles="@(_OutputFiles)"
DestinationFiles="@(_OutputFiles>'$(_CopyPath)\%(RecursiveDir)\%(FileName)%(Extension)')"/>
</Target>
```

5. Reload the project.
6. Rebuild the project
7. Look at the \TDS-Sample\TDSProject1\bin folder where your sample project is. You should now notice that there is a 'Debug' and a 'Debug_COPY' folder. If you compare these folders you will notice that the 'Debug' folder has the result of the file replacements while the Copy folder does not.

While this is a very simple scenario it demonstrates the power MSBuild and TDS gives you when trying to automate your builds.

Automating Your Builds

Now that we've covered the basics of Continuous Integration and MSBuild let's try to automate our builds. There are many tools such as Microsoft Team Foundation Server, NAnt, CruiseControl.net, Team City, etc... that will help automate your builds. Team Development for Sitecore will support any tool you choose as long as that tool can call MSBuild! Some tools support MSBuild natively, such as TFS and TeamCity, whereas other tools, such as NAnt, have their own build files and call out to MSBuild.

The command to use where trying to build your solution, or TDS project directly, would simply be `msbuild.exe [your solution.ext]`. There are optional switches you can pass to msbuild. It is very likely that you will need to pass a `/p:Configuration=[your config]` to msbuild to tell it which solution configuration you are building.