OpenMake Meister for Enterprise Level .Net Builds

A White Paper overview of how OpenMake Meister extends the features of the .Net ALM Suite for enterprise level software builds.

Email: **Request-info@OpenMakesSoftware.com**

www.OpenMakeSoftware.com

Summary

Introduction

OpenMake Meister supports the Visual Studio .Net, TFS and Team Build suite to deliver enterprise level cross solution, cross language and accelerated .Net builds. In a similar way that the Microsoft .Net IDE automates the “build” (compile and link process) on a solution level basis, Meister works to automate builds for multiple solutions and cross language builds (Java, UNIX, Linux). Meister also simplifies and accelerates the execution of .Net builds outside of the .Net IDE without requiring manually coded Nant scripts or modifying MSBuild scripts.

To provide these features, Meister leverages the .Net environment, and extends both TFS and Team Build.

Issues that Meister solves for .Net users can be summarized as follows:

- Multi-language and Multi-platform builds
- Builds across multiple solutions
- .Net Build Acceleration
- Dependency Discovery and Audit Reporting
- Build best practices regardless of the development language, or Microsoft IDE version.

The Self-Contained .Net ALM Suite

.Net developers enjoy the features of a fully integrated, self-contained development environment with automated compile and link capabilities, which does not require the use of manually coded make or Nant scripts. This means that as the developers make changes to the code within their solution, .Net auto-generates the MSBuild file needed to compile and link those changes. This auto-generation feature is a critical piece of the .Net technology. Within the .Net suite, TFS and Team Build can be used to execute the solution build outside of the .Net IDE. Developers’ check-in code to TFS and Team Build executes the solution build in what is called “headless mode. “

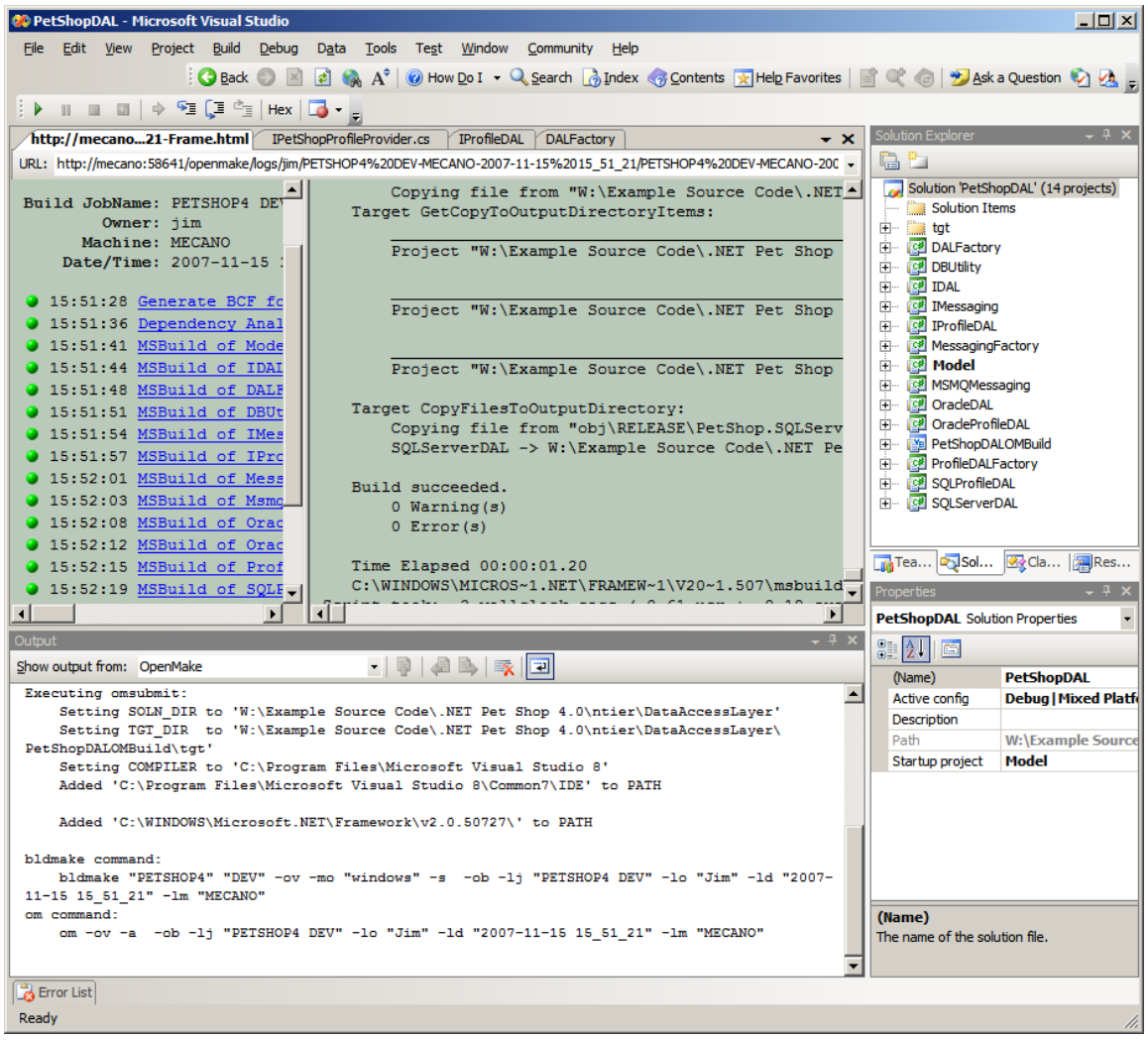
Meister Extends the .Net ALM Suite

When software applications that use the .Net suite grow in size beyond a single solution file, Meister extends the MSBuild process to treat multiple solution files as a single solution file. This allows development teams to break apart large applications into logical subsystems per solution,

but still compile the application as a single solution under the .Net ALM process calling TFS and Team Build in the same way as they would if the application used only one solution. By maintaining a single solution “appearance” to .Net, TFS and TeamBuild, the .Net self-contained ALM process is preserved.

Without Meister performing this function for multiple solution applications, developers often must turn to manually coding the MSBuild script to handle the shared dependency relationships between the solutions so that the application is always built in the correct order. Another approach is to code Nant scripts (.Net Build Tools – sourceforge.net) to manage the compile and link process. With either of these approaches, the .Net ALM automated process becomes dependent upon a manual process.

Meister also leverages MSBuild (and DevEnv), to auto-generate the MSBuild script file even when multiple versions of the .Net compiler are used, for example Visual Studio 2005 and Visual Studio 2010. This is a common requirement when a development organization has been writing applications in .Net and Visual Studio for many years. Common routines may have been developed in VS 2005 which must share components with newer applications written in VS 2010. Build decision making must be done between the newer application and the older common routines. Meister’s technology performs the auto-generation build feature so that the multiple version build can be executed either inside the .Net IDE using the add-in, or outside the IDE called by Team Build. Below is a screen shot of the Add-in showing how it can allow developers to use OpenMake Meister from inside of .Net for performing builds that are outside of .Nets current functionality.



Taking this logic one step further, Meister can support cross language builds. This means that an application written using both Java components and .Net components can be built using Meister. In a similar way that Meister solves a build using multiple versions of .Net, Meister can manage cross language builds providing the same automatic build features that are used when only a single .Net solution file is used. Meister extends this feature to support builds between Windows and Unix or Linux. In fact, Meister can even extend this functionality to the mainframe for applications that use mainframe components.

Understanding how Meister leverages the .Net technology and extends it to multiple solutions, multiple .Net versions, cross languages and cross platform is important in understanding the difference between what .Net build functionality does and what Meister provides. Meister acknowledges the importance of the build automation features that .Net offers and extends it beyond what the .Net IDE currently supports. Table 1.1 shows what objects are supported by MSBuild and how OpenMake Meister extends the build capabilities of Microsoft Visual Studio.

Language	MSBuild/Team Build	Meister
C#	Yes	Yes
J#	Yes	Yes
VB.NET	Yes	Yes
C .Net	Yes, via VCBuild.exe	Yes
ASPX	Yes, via ASPCompile.exe	Yes
Multiple Solutions	No	Yes
C – VS 6	No	Yes
C++ – VS 6	No	Yes
VB – VS 6	No	Yes
MSI Installers	Yes	Yes
Old .Net Project file formats, Biztalk 2005	No	Yes
Java	No	Yes
Cobol	No	Yes

Table 1.1

Meister's Added Benefits

Beyond Meister's core feature of managing multiple solutions and cross language compiles, Meister offers benefits around dependency management, build acceleration and audit reporting.

Meister's build engine includes source code dependency scanning which extends Meister's features to support the acceleration of builds and the audit reporting of builds. Because Meister controls the calls to the compilers, it can report on all dependencies and artifacts used by the compilers and linkers. This includes all artifacts, even when they are not managed by TFS. Meister's build audit watches the compile process and reports on all source modules and libraries, even when they come from a location defined by an environment variable and not from the TFS repository. Meister's audit report guarantees source to matching executables, which is the ultimate goal of any solid build to release process.

Because Meister understands the dependencies of a build, it also understands the ordering of the build. Using this information, Meister can improve build times of .Net builds in two ways. First, Meister supports incremental build processing, even when multiple solutions or cross languages are used. Incremental build processing, often called "Build Avoidance," prevents the re-building of objects that are already up to date. This means that an emergency fix made to a single source module will not initiate a full build of all binaries. Only the impacted binaries are re-built. This is by far the most effective method of building large .Net applications.

Meister can also accelerate your .Net builds using parallelization. Because Meister understands the source dependency relationships, it can call compilers and linkers in multi-threaded mode, drastically reducing the time it takes to execute full builds. Customer feedback on large .Net applications as demonstrated a reduction in build times of up to 40%.

Summary

Meister extends the functionality of the .Net ALM suite to support enterprise level .Net application builds. It is integrated into .Net, TFS and Team Build in a non-evasive method, maintaining the .Net end user experience and adding benefits that the .Net ALM suite alone cannot offer. From managing multiple solution file builds to accelerating the compile and link process Meister, extends the features of MSBuild to support larger enterprise applications that have grown beyond a single solution file process.