

# *Building Multi-language Applications with Visual Studio® and Openmake® Meister*

---

A White Paper and technical overview of integration between Openmake® Meister and Microsoft® Visual Studio® for the support of Multi-language and Multi-platform builds.



OpenMake Software  
213 W. Institute Place, #404  
Chicago, IL 60610  
Tel: 800.359.8049  
312.440.9545  
Email: [request-info@openmakesoftware.com](mailto:request-info@openmakesoftware.com)

<http://www.openmakesoftware.com>



## *Table of Contents*

---

|  |           |
|--|-----------|
| <b>TABLE OF CONTENTS</b>   | <b>1</b>  |
| <b>INTRODUCTION</b>  | <b>3</b>  |
| <b>Builds for the Visual Studio Suite</b>  | <b>3</b>  |
| <b>Openmake Meister Extends the Build Features of Visual Studio</b>  | <b>4</b>  |
| <b>Challenges Building Visual Studio and .Net Applications</b>   | <b>4</b>  |
| Build Challenges in Multi Platform Environments  | 4         |
| Build Decision Making when Building Multiple Solutions   | 5         |
| Build Decision Making for Build Avoidance  | 5         |
| Visual Studio Build Issues with the Web Project  | 6         |
| <b>Summary of Openmake Meister Added Value</b>   | <b>6</b>  |
| <b>HOW OPENMAKE MEISTER AND MICROSOFT VISUAL STUDIO WORK TOGETHER</b>  | <b>7</b>  |
| <b>Meister's Components</b>  | <b>7</b>  |
| <b>The Openmake Meister Knowledge Base</b>   | <b>7</b>  |
| <b>Extending Visual Studio Builds beyond the IDE</b>   | <b>8</b>  |
| <b>Meister Interacting with Team Foundation Server</b>   | <b>8</b>  |
| <b>Remote Build Servers for Multi-Platform Build Support</b>   | <b>9</b>  |
| <b>Meister, Team Build and MS Build</b>  | <b>9</b>  |
| <b>APPENDIX A - BUILDING PESHOP: A STEP BY STEP GUIDE FOR BUILDING APPLICATIONS USING THE OPENMAKE® MEISTER MICROSOFT® VISUAL STUDIO® ADD-IN</b> | <b>10</b> |
| <b>Introduction</b>  | <b>10</b> |
| <b>Developer Builds and the Meister Add-in</b>   | <b>12</b> |
| TGT files  | 14        |
| Builds   | 14        |
| <b>Integration Builds</b>  | <b>15</b> |
| <b>Advantages</b>  | <b>16</b> |
| Logging and Auditability   | 16        |

|                               |           |
|-------------------------------|-----------|
| <b>Additional Information</b> | <b>19</b> |
| <b>COMPANY OVERVIEW</b>       | <b>19</b> |

## Introduction

---

The purpose of this document is to present the benefits of using Openmake Meister to enhance the software build functionality of Microsoft Visual Studio 2005 and 2008 in conjunction with Microsoft Team Foundation Server. Openmake Meister standardizes the software build process and extends build features of MSBuild and Microsoft Team Build through innovative Build Forensics, linking production binaries back to their development origins. This whitepaper will be useful for software developers, build and configuration managers, or anyone building Visual Studio, or .Net objects outside of the Microsoft IDE.

*OpenMake Meister extends the functionality Visual Studio builds supporting the creation of multiple language and platform objects.*

### Buils for the Visual Studio Suite

Microsoft Visual Studio provides streamlined functionality of executing builds outside of the IDE using MSBuild. Because it is intended to support specifically Visual Studio, MSBuild's primary functionality is to support builds at the Solution Level, creating C#, and VB.Net objects. OpenMake Meister extends the functionality of Visual Studio 2005 and 2008 Builds supporting the creation of multiple language and platform objects including Microsoft Visual Studio 6, VS .Net (2003), as well as any extended Visual Studio objects such as Biztalk. 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                              | No                      | Yes     |
| Old .Net Project file formats, Biztalk 2005 | No                      | Yes     |
| Java  | No                      | Yes     |
| Cobol                                       | No                      | Yes     |

Table 1.1

*Meister allows VS developers to build applications that use diverse languages and tools.*

## **Openmake Meister Extends the Build Features of Visual Studio**

The main features of Openmake Meister important to Visual Studio .NET are:

- Multi-language and Multi-platform builds
- Builds across multiple solutions
- Dependency Discovery and Build Forensics
- Build best practices regardless of the development language, or Microsoft IDE version.
- Consolidation of MSBuild scripts, Nant Scripts and Nmake scripts into a set of standardized and reusable build methods.
- Centralized user management, on-demand workflow management, and self service information for viewing build logs and metric reports.

## **Challenges Building Visual Studio and .Net Applications**

Below are the challenges facing Visual Studio developers when managing project and solution level builds.

*Many enterprise development efforts utilizes development tools beyond the Microsoft Visual Studio suite.*

### **Build Challenges in Multi Platform Environments**

A heterogeneous build environment is one where multiple development tools are used to deliver applications, such as a company that has both Visual Studio and Java. It is important to realize that Visual Studio is a product built by and for Microsoft products. Indeed, it is a very useful tool for doing development within Microsoft's framework. It is likely that many enterprises will also need to build applications for other operating systems and other languages. These other applications may even depend on the Visual Studio developed projects.

On an enterprise level, this presents a problem at build time. Even if the Visual Studio portion of the development builds correctly, builds on other platforms will have to be done using some other build tool, or most likely by having to customize NANT or make scripts for that portion of development. This lack of standardization creates organizational and control problems when performing diverse builds on an enterprise level.

OpenMake Meister solves the problem of building applications using multiple tools and platforms. OpenMake Meister can generate a single Build Control File that will build, Java, Visual Studio .Net, Visual Studio 6, Visual Studio 2005 and Visual Studio 2008

objects. Through Build Services, standard reusable scripts provide the build framework for supporting builds that use multiple languages and tools. Build Services can be customized through reusable build scripts called Build Methods to cater to the unique requirements of each development environment.

*OpenMake Meister allows developers to continue using the Visual Studio IDE to build multiple solutions, the same as they would build a single solution.*

## **Build Decision Making when Building Multiple Solutions**

Developers working within Visual Studio can define multiple Project as part of a single Solution. This practice is important when developing using Microsoft Visual Studio and you want to execute builds using Microsoft Team Build and the generated MSBuild script. When managing multiple Projects using a single solution, the Visual Studio IDE will generate a MSBuild XML file that can be called by Team Build. Contained in this MSBuild XML file will be the required logic, called "Transforms", to build the Projects in the correct order.

In some larger enterprise efforts, it is not practical to manage the entire application under one solution. This means that multiple solutions are used to create a single application. For this effort, developers must write their own MSBuild XML scripts to handle the building of multiple solutions. This is where OpenMake Meister enhances the process. OpenMake Meister allows developers to continue using the Visual Studio IDE to build multiple solutions, the same as they would build a single solution. Meister generates a Build Control File that builds multiple solutions leveraging the Microsoft underlying framework. During the build process, Meister's deep dependency scanning links all of the dependent parts together. This means that regardless of how the solutions are dependent upon one another, even when the dependencies are circular, OpenMake Meister can accurately determine the order in which objects should be built. We call this good build "decision making".

*Meister uses machine intelligence to build applications using Build Avoidance – the fastest way to execute builds.*

## **Build Decision Making for Build Avoidance**

Building faster is critical for large projects. The most efficient way to build faster is through Build Avoidance. Build Avoidance simply means that your builds will be done incrementally, re-compiling only the objects that are out of date. With MSBuild you can define "Transform" relationships between objects and MSBuild will check the time and date of the relationships to determine if an object is out of date. To do this, you must code the "Transform" statements manually in the MSBuild script to define how you want to handle the date/time checking process. When using OpenMake Meister to handle Build Avoidance, this date/time process is handled automatically. Meister does deep dependency discovery prior to the compile/link process. It tracks this information during the build and determines what needs to be built and what can be avoided.

*As lean methodologies encourage the rapid changing of applications, builds must easily adapt*

Meister uses machine intelligence to track these dependencies. In large applications, it becomes harder and harder for a human to determine all of the possible dependent

to software updates.

relationships. Having to manage this process manually means that there is a high probability that the incremental build will miss a critical component resulting in a missed link step and ultimately a bad build. In addition, the process of manually tracking these relationships within the MSBuild script is time consuming and static. As lean methodologies encourage the rapid changing of applications, the process of building the applications must easily adapt to software updates. Removing the manual steps of the build is critical in pursuing truly automatic builds, regardless of language.

*The Openmake Meister solution for managing .NET Web Projects provides a development team based best practice approach for managing references.*

## Visual Studio Build Issues with the Web Project

A common build challenge for Visual Studio developers that needs to be addressed is the Web Project. The web project involves not only source and assemblies, but also a requirement to interact with the IIS web server. MSBuild can call the process that builds the Web Project, but developers must manually create these MSBuild scripts. In some cases, developers choose Nant, an Apache open source language to create the build logic for Visual Studio Web Projects.

*Visual Studio IDE based solutions for individual developers or Visual Studio Team Foundation Server based solutions for Team Build requirements.*

In either case, the build environment can be challenged by poorly managed and executed development practices, but the problem of managing assembly references for coded objects in Visual Studio seems to be unusually difficult for many development teams. It is a combination of the best practice approach not being widely known at the beginning of the development effort and from the lack of requirements for a concerted, manually organized code management process to be sustained. The overuse of local file references greatly hinders portability from one environment to the next. The Openmake Meister solution for managing .NET Web Projects provides a development team based best practice approach for managing references. Meister can provide successful builds regardless of machine and reference scheme. Clearly a simple, machine-independent method of finding references is desirable.

## Summary of Openmake Meister Added Value

- Meister supports multi-platform builds extending build features to non VS 2005 and 2008 objects.
- Meister manages MS 2005 and 2008 builds that is consistent across solution files, web projects and other development tools.
- Meister performs automatic dependency scanning linking dependencies of multiple solution level files for Build Audit, Impact Analysis and Build Avoidance.
- Meister allows any user to perform a build for one module, many modules or the entire project.
- Meister allows simple control of project and file references outside of the developer's desktop.
- Meister provides a portable method of rebuilding applications at any check pointed stage in the development lifecycle (REV 1.1. REV 1.2, etc).
- Meister enables the standardization of the INCLUDE and LIB settings,

removing the dependency of the build on machine specific configurations that can change the result of a compile.

## ***How Openmake Meister and Microsoft Visual Studio Work Together***

---

To achieve the benefits of extending multi-language and multi-platform builds to the Visual Studio developer, Meister leverages the Microsoft technology of Microsoft Visual Studio, Team Foundation Server and Team Build technology.

Meister collects and stores meta data about builds. It uses Build Services and Build Methods to standardize on how to build different types of binaries, for example VS .Net DLLs, VS 2005 and 2008 EXEs, Java Jars or Java Wars. Below is a review of the Meister architecture and how it leverages the Microsoft Technology.

### **Meister's Components**

#### **The Openmake Meister Knowledge Base**

Meister separates common build information from critical project specific information. Common build information is managed in the Meister Knowledge Base, while application specific information, that likely to change over time, is managed in target definition file, similar to a Microsoft Project file.

The Meister Knowledge Base server contains information on:

- **Build Services** – Build Services provides a process for defining build best practices for various languages and operating systems. Build Services use Build Methods for calling the development tools in a repeatable way. Build Services are defined for building Microsoft Visual Studio objects.
- **Build Methods** – Build Methods define the behavior of a compiler or linker that is needed to build different types of assemblies. Build Methods are reused across applications and projects. Example Build Methods are “.NET Solution Executable” and “.NET Solution Installer”.
- **Build Forensics** – Meister performs deep dependency scanning, spanning different languages and tools. This Build Forensic feature allows Meister to understand the inter-dependencies between objects, even when the objects are derived from different languages, such as C# and Java. This deep dependency scanning includes querying Team Foundation Server for revision information to be included in the Build Audit providing the link from the executables back to the source code revisions.
- **Workflows** – Workflows are a collection of commands that are performed before, during and after builds. Workflows can call version control tools, testing tools, build scripts and can be executed across different machines running different operating systems.

*Team Build can call the Meister Workflow that then executes builds and tasks across multiple operating systems.*

## Extending Visual Studio Builds beyond the IDE

*The Meister Visual Studio Add-in extends the Visual Studio IDE directly to the build, even when the build itself requires different versions of the Visual Studio compiler, requires the assembly of multiple solution files or incorporates other tools such as Java.*

The Meister Visual Studio Add-in provides a method for interacting with the Meister Knowledge base, directly from within the Visual Studio IDE. Developers need the ability to perform all of their day to day activities from within a central tool, i.e, Visual Studio. For this reason, the Meister Add-in extends the Visual Studio IDE directly to the build, even when the build itself requires different versions of the Visual Studio compiler, requires the assembly of multiple solution files or incorporates other tools such as Java.

The purpose of the Meister Visual Studio Add-in is as follows:

- To configure Meister Builds from within Visual Studio
- To automatically gather file specific names to pass to the Meister Build Services for generating the build scripts. This is done through the Meister Target Definition.
- To provide an IDE centric locations to generate build scripts and execute builds managed by Meister.
- To execute builds across solutions and diverse languages such as Java. The Meister Add-in can build all projects if they reside in one solution and are referenced through project references. But more importantly, Meister can still determine the correct build order when Projects make non-project based references to other assemblies that should still be built at the same time.
- To assign Projects and Solutions to a Meister Build Project.

Using Build Forensics, the Meister build engine enhances the Visual Studio build process by scanning the .csproj file for included .cs source code, and for included references. These references can be project references, file references, or COM references. Meister will recurse through the dependency tree resolving relationships between references, binaries and source. These Meister dependency relationships are stored as part of the Build Audit and Foot Printing. With the dependency relations stored, the executable, dynamic link library or assembly can be inspected and traced back to the source revision that created it.

## Meister Interacting with Team Foundation Server

Meister performs builds; it does not manage source code. For this reason, Meister leverages the Team Foundation Server to retrieve source code, build binaries and store binaries back into Team Foundation Server. In addition, Meister uses the Team Foundation Server item history for tracking precisely what object were used in the build, displayed via a comprehensive Build Audit Report. The Build Audit Report shows all items included in the build, even when it is not stored in Team Foundation Server. The Build Audit Report is a footprint of what occurred in the build and can

be used to audit the build against what is stored in Team Foundation Server.

## Remote Build Servers for Multi-Platform Build Support

A remote build server allows a developer to execute a build hosted on a machine other than the user's local desktop. The remote machine can be running a different operating system. This is useful for both controlled build environments or to run a single build that needs to execute on multiple operating systems. A Meister Workflow can be created to execute a build across multiple operating systems. Team Build can call the Meister Workflow that then executes builds and tasks across multiple operating systems.

*Meister passes to Team Build "execution commands" for Team Build to manage and orchestrate.*

## Meister, Team Build and MS Build

Meister passes to Team Build "execution commands" for Team Build to manage and orchestrate. These execution commands, called Meister Workflows, can include the execution of tasks across multiple machines and languages. This means that Team Build can use Meister to orchestrate a build across diverse operating systems and languages. A build could include the compiling of Visual Studio 2005 and 2008 objects on a Windows machine and Java objects on a Unix machine.

Meister has extended the MS Build functionality in order to enable Team Build interaction with Meister's Multi-platform and Multi-language support. MSBuild is the link between Team Build and Meister.

# Appendix A - Building Petshop: a Step by Step Guide for building applications using the Openmake® Meister Microsoft® Visual Studio® Add-in

## Introduction

This whitepaper provides a detailed description on how to set up and execute builds using Meister in conjunction with Visual Studio 2005 and 2008 using the Pet Shop 4.0 application as provided by Microsoft (<http://msdn2.microsoft.com/en-us/library/aa479070.aspx>). Pet Shop is described as an example for "showing the best practices for building enterprise, n-tier .NET 2.0 applications that may need to support a variety of database platforms and deployment scenarios."

Pet Shop 4 follows an n-tier architecture, as shown in the diagram 1.0

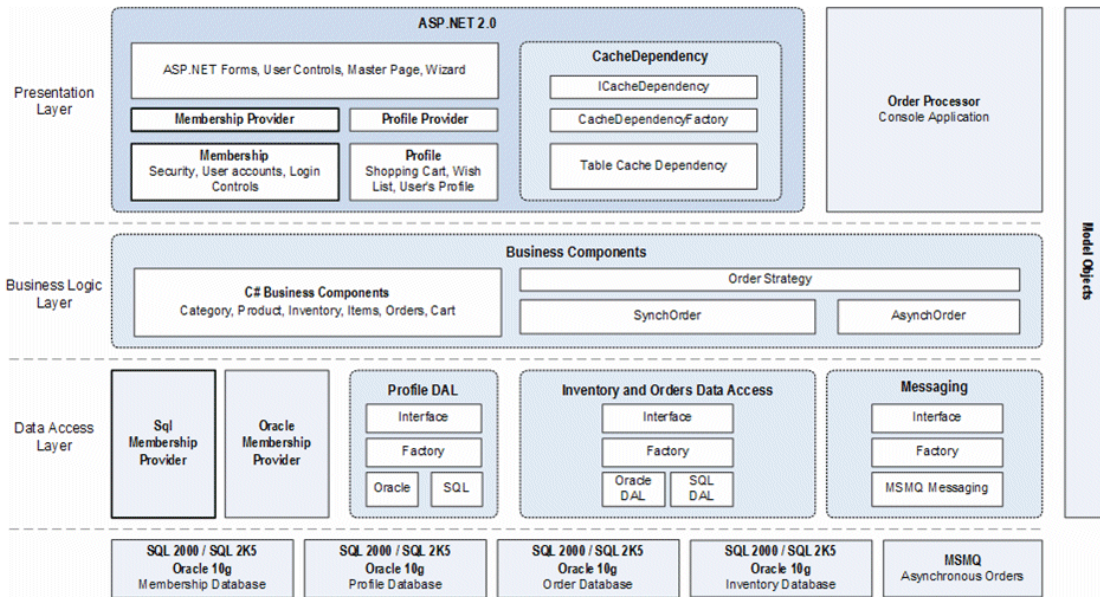


Diagram 1

[http://msdn2.microsoft.com/en-us/library/aa479070.bdassampet409\(en-us,msdn.10\).gif](http://msdn2.microsoft.com/en-us/library/aa479070.bdassampet409(en-us,msdn.10).gif)

The architecture can be split roughly into three tiers.

- Data Access, composed of
  - IDataAccess interface
  - Data Access Factory
  - IMessaging interface
  - Messaging Factory
  - Msmq Messaging
  - Oracle Profile Data Access Layer
  - Oracle

- SQL Server Profile Data Access Layer
- SQL Server
- DB Utilities
- IProfileDAL interface
- Profile Factory Data Access Layer
- Model objects
- Business Logic, composed of
  - IBllStrategy interface
  - BLL business logic
  - Profile
- Presentation Layer
  - ICacheDependency interface
  - Cache Dependency Factory
  - Membership
  - Order Processor console module
  - Table Cache Dependency
  - Web site

In the default configuration, these are provided in one solution (.sln) file containing all the project files.

The default Pet Shop configuration provides all the project files in one solution file. Each project refers to other required assemblies within the solution via project references. However, this approach may not be ideal for a larger application across multiple tiers. As the solution grows, there will be a cost in terms of loading all source code through one solution file, in terms of:

- Amount of source code to be retrieved,
- The complexity of the solution,
- Separation of concerns amongst team members responsible for different components.

This solution becomes too complex.

In the future, each layer of the tier may be separated into its own solution file. There will be one solution file for the Data Access Layer, one for the Business Logic, and one for the Presentation layer. Each layer depends on the layers previous to it through file references, and to projects within its layer through project references.

In the default MSBuild case, the second scenario of splitting the application into three solution files becomes problematic, because there is no longer any connection between the build order of items between each layer. It becomes dependent on a manual process to know to when each tier of the application must be rebuilt.

The Meister build engine is configured to build the Pet Shop application in either format. Meister can build all the projects if they reside in one solution and are referenced through project references. But more importantly, Meister can still determine the correct build order when Projects make non-project based references to other assemblies that should still be built at the same time.

Each assembly in a Meister build corresponds to a Target. The metadata for a Target is quite simple:

- Target: the assembly to be built

- Build Service: the Meister Build Service that will be used to create the Target.
- Build Dependencies: a listing of high-level source files that will be scanned by the Meister build engine and compiled to create the Target.

For a simple example of a Target definition, consider the PetShop.OrderProcessor.exe console application

- Target: OrderProcessor\bin\\$(CFG)\PetShop.OrderProcessor.exe
- Build Service: MSBuild Executable
- Build Dependencies: OrderProcessor\OrderProcessor.csproj

“MSBuild Executable” is a standard Build Service that Meister provides out of the box for building assemblies with MSBuild as the compiler. We provide 1000s of Build Services for 100s of compilers across eight different OS bases.

The Meister build engine will scan the OrderProcessor.csproj file for included .cs source code, but also for included references. These can be project references, file references, or COM references. If any reference corresponds to another target in the Meister build (as represented by another Meister Target File), Meister will build that Target first, recursing through the dependency tree.

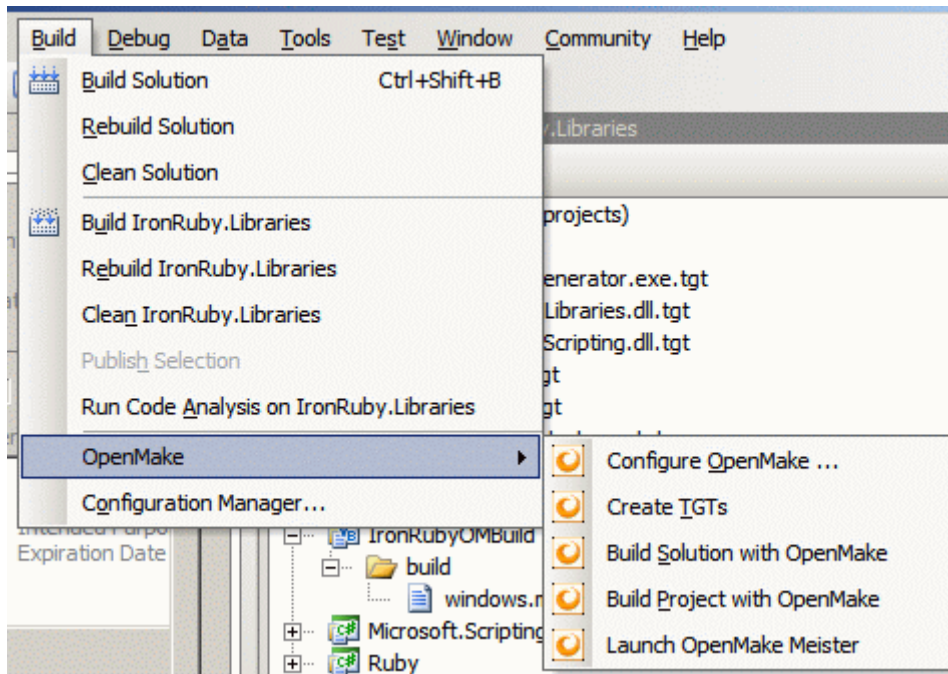
In this way, Meister resolves both project and file references, and is not tied to the solution file as a container object for building the application.

### **Developer Builds and the Meister Add-in**

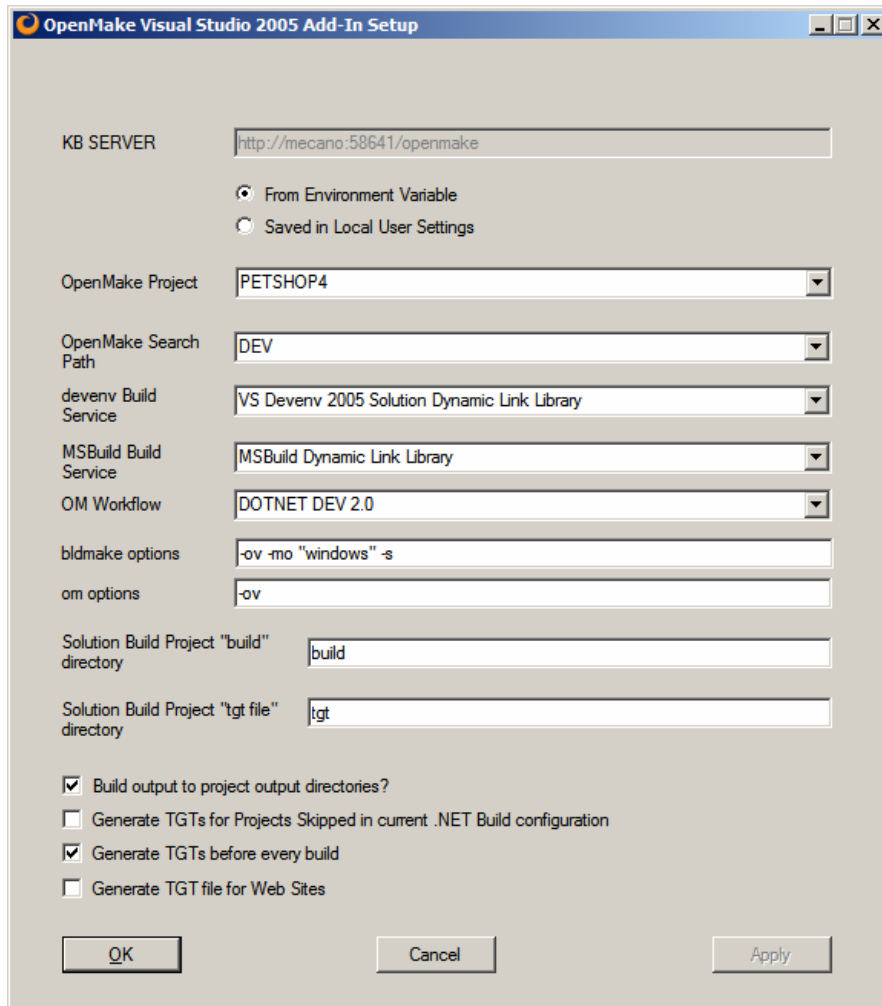
The default Pet Shop application comes with 22 projects. We do not want to have to create an Meister Target File (no matter how simple) for each project. Meister provides a Visual Studio 2005 and 2008 Add-in that will automatically create a Target File for each project with the solution.

The Add-in provides new menu items off the Build menu and the Solution and Project context menus. One can

- Configure Meister within the Visual Studio IDE
- Generate Meister Target Files (see above)
- Build the Solution or Project using Meister within the IDE
- Launch the Meister application.



The Add-in will connect to the centralized Meister server to determine the centralized Build Services and Build Projects. Once that connection is made, the Projects within the Solution can be assigned to a Meister project, and also assigned to a Build Service.



## TGT files

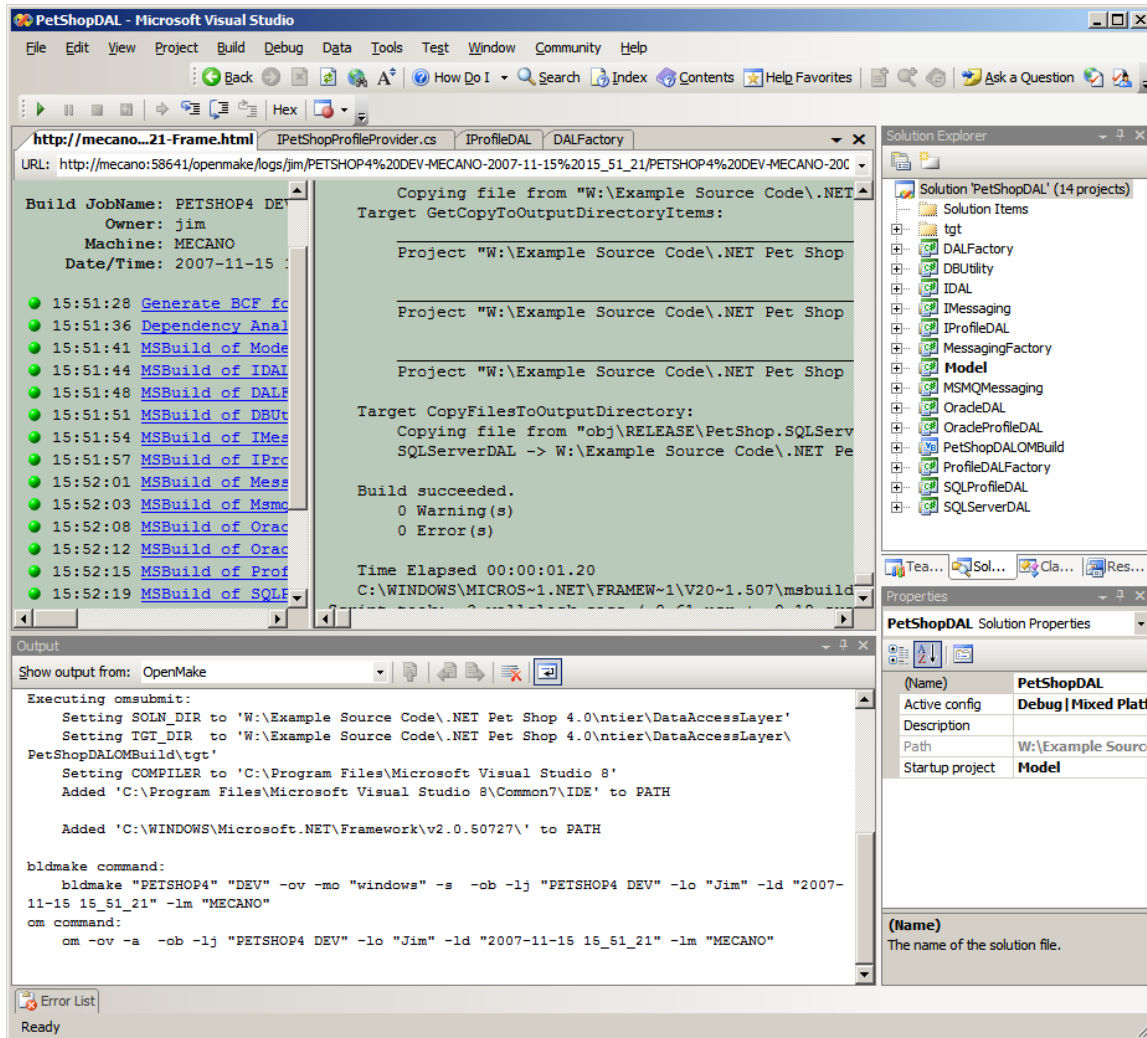
The Meister Add-in will automatically generate Meister Target Files for the projects listed in the solution file. By default, the Add-in will try to use the appropriate MSBuild Build Service (DLL or executable). However, certain project types, such as Setup projects, cannot be built by MSBuild and have to be built by the Visual Studio devenv.

The Target files are stored in a solution folder called "tgt". This solution folder places the target files on the file system in the same directory as the solution itself.

## Builds

The Meister Add-in can also execute developer-level Meister builds within the Visual Studio IDE. This build will occur in an Meister specific project within the solution, typically "<Solution Name>OMBuild/build". This build is segregated from the standard Visual Studio build, and is typically not meant as a replacement for the Visual Studio build. Instead, it represents a way to confirm that the developer Meister build will work when the Meister build is executed as integration build outside of the IDE.

Shown below is the Meister build within Visual Studio for the PetShop Data Access Layer solution. The build log is centrally stored on the Meister KB Server and shown in the Visual Studio IDE.



## Integration Builds

Once the source code and Meister Target files are checked into version control, an integration build can occur. This may be a nightly build or a continuous integration build. Meister supports scheduled builds and continuous integration build for a large number of SCM tools including Team Foundation Server.

The advantage of the integration build is that it will build all of the assemblies within the Meister project, as opposed to the developer build that will only build the assemblies within the developer's solution.

The integration build will allow for automatic determination of inter-solution dependencies. Most current build tools require human knowledge of inter-solution dependencies (i.e. "hey, remember to build Solution 1 before Solution 2"), or require maintenance of a separate "Build All" solution file. OpenMake avoids this problem.

## Advantages

There are a number of advantages to Meister builds above running MSBuild against a solution file, in addition to inter-solution dependencies:

- COM references and legacy libraries that are imported in .NET projects can be built as Meister Targets in the correct order in one build process,
- Legacy C or VB applications that incorporate new .NET functionality through COM interoperability can be automatically built in the correct order by adding the .NET dll as a dependency to the Legacy application.

## Logging and Auditability

All the logging of Meister builds is stored on the centralized KB server. Because the logs are centrally stored, this allows for team members to easily see what is happening with the latest builds and gives an immediate status of the health of the project.

In addition to build logs, there are separate Build Audit logs that detail the exact source code items that were compiled into an Assembly. Furthermore, Meister can run detailed impact analysis to show which files affect which Assemblies and how a change in one file can “ripple” through other projects. This impact analysis occurs both within the .NET sphere and also outside of it. Most impact tools are focused solely within the .NET sphere.

The Build Audit report details exactly which source code files were compiled into which assemblies. As well, it details the machine configuration (with OS patches) and the Environment variable settings at the time of the build

Dependencies:

In addition to file locations and timestamps, the Meister Build Audit report can integrate with many SCM and Versioning tools to provide customized information about the exact versions of each file that went into the build. This provides a deep audit trail linking source to assemblies that other tools do not provide.

Build Audit Report for Model\bin\RELEASE\PetShop.Model.dll

Project Variables:

Built on MECANO by jim at 11/15/2007 16:24:10  
System Info

Host Name: MECANO  
OS Name: Microsoft Windows XP Professional  
OS Version: 5.1.2600 Service Pack 2 Build 2600  
OS Manufacturer: Microsoft Corporation  
OS Configuration: Standalone Workstation  
OS Build Type: Multiprocessor Free

.....  
Logon Server: \\MECANO  
Hotfix(s): 239 Hotfix(s) Installed.  
[108]: Q147222

.....  
[211]: KB927779 - Update

NetWork Card(s): 1 NIC(s) Installed.  
[01]: Intel(R) PRO/100 VE Network Connection  
Connection Name: Local Area Connection  
DHCP Enabled: No  
IP address(es)  
[01]: 192.168.1.25

.NET Framework 2.0, service pack 0  
.NET Framework 1.1, service pack 1

Environment Variables:

ALLUSERSPROFILE=C:\Documents and Settings\All Users  
APPDATA=D:\Documents and Settings\Jim\Application Data  
APPL=PETSHOP4  
CFG=RELEASE

...

windir=C:\WINDOWS

Dependencies:

| Date                             | Time     | Size | Target                  | Dependencies      |
|----------------------------------|----------|------|-------------------------|-------------------|
| 11/09/2005                       | 16:49:44 | 4864 | W:\Example Source Code\ | .NET Pet Shop     |
| 4.0\ntier\DataAccessLayer\Model\ |          |      |                         | Model.csproj      |
| 11/09/2005                       | 16:44:50 | 3087 | W:\Example Source Code\ | .NET Pet Shop     |
| 4.0\ntier\DataAccessLayer\Model\ |          |      |                         | AddressInfo.cs    |
| 11/09/2005                       | 16:44:50 | 715  | W:\Example Source Code\ | .NET Pet Shop     |
| 4.0\ntier\DataAccessLayer\Model\ |          |      |                         | AssemblyInfo.cs   |
| 11/09/2005                       | 16:44:50 | 2354 | W:\Example Source Code\ | .NET Pet Shop     |
| 4.0\ntier\DataAccessLayer\Model\ |          |      |                         | CartItemInfo.cs   |
| 11/09/2005                       | 16:44:50 | 1178 | W:\Example Source Code\ | .NET Pet Shop     |
| 4.0\ntier\DataAccessLayer\Model\ |          |      |                         | CategoryInfo.cs   |
| 11/09/2005                       | 16:44:50 | 1478 | W:\Example Source Code\ | .NET Pet Shop     |
| 4.0\ntier\DataAccessLayer\Model\ |          |      |                         | CreditCardInfo.cs |

11/09/2005 16:44:50 2204 W:\Example Source Code\.NET Pet Shop  
4.0\ntier\DataAccessLayer\Model\ItemInfo.cs  
11/09/2005 16:44:50 1865 W:\Example Source Code\.NET Pet Shop  
4.0\ntier\DataAccessLayer\Model\LineItemInfo.cs  
11/09/2005 16:44:50 3231 W:\Example Source Code\.NET Pet Shop  
4.0\ntier\DataAccessLayer\Model\OrderInfo.cs  
11/09/2005 16:44:50 1637 W:\Example Source Code\.NET Pet Shop  
4.0\ntier\DataAccessLayer\Model\ProductInfo.cs  
11/09/2005 16:44:50 1430 W:\Example Source Code\.NET Pet Shop  
4.0\ntier\DataAccessLayer\Model\CustomProfileInfo.cs

## **Additional Information**

For more information on all the features Meister can bring to application build processes to enhance Visual Studio .NET, please review the Meister 7.1 datasheet available from a OpenMake Software representative or in downloadable form at <http://www.openmakesoftware.com/meister-7.1/>

## ***Company Overview***

---

OpenMake Software is a premiere provider of Build and Release Management software and consulting services dedicated to assisting customers with the implementation of standardized builds across the enterprise. OpenMake Software has specialized in the design and implementation of reliable and repeatable application builds for Global 2000 organizations since 1995.

### **North American Headquarters**

213 W. Institute Place, #404

Chicago, IL 60610

Ph: 800.359.8049 – 312.440.9545

<http://www.openmakesoftware.com>