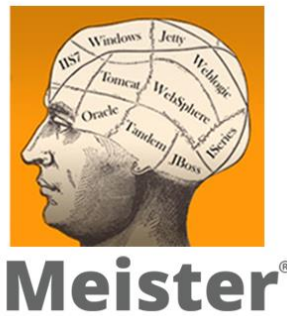


Building IBM WebSphere® Studio Applications with



A White Paper and technical overview of build management techniques using IBM WebSphere® Studio Application Developer integrated with OpenMake Meister



www.openmake.com

This document provides an overview of the use of Meister from OpenMake Software integrated with IBM Websphere. It is designed for software developers, configuration managers, and anyone concerned with managing the reliability of software applications.

Table of Contents

Introduction	1
Challenges Building Java and J2EE Applications without Meister using WebSphere Studio Advanced Developer	2
The Meister Solution	4
Meister Technical Overview.....	7
Meister Projects.....	7
The Meister Knowledge Base.....	7
Target Definition Files.....	8
Using Meister with WebSphere Studio Development	9
WebSphere Studio Development Assumptions	9
Meister Installation Assumptions	9
Meister Knowledge Base Setup.....	9
Installing the Meister Plug-In.....	10
Activating Meister Plug-In Menus within WebSphere Studio.....	10
Accessing Meister Plug-In Views in WebSphere Studio	11
Walk-Through Examples	14
Performing an Meister Build Outside of WebSphere Studio	27
Integrating the Meister Build with SCM-Managed WebSphere Studio Development	28
Extending the Meister Build Process.....	28
Company Overview	31
North American Headquarters.....	Error! Bookmark not defined.
Sales and Marketing.....	Error! Bookmark not defined.

Introduction

This document presents a discussion of how OpenMake Meister integrates into WebSphere Studio Application Developer (WSAD) and extends the build process.

In the first section, we discuss challenges faced in building Java and J2EE application outside of the WSAD IDE. We provide a general background of the build process within the WSAD IDE and general solutions for outside builds.

In the next two sections, we present an overview of OpenMake Meister where general terminology, features, and process are detailed.

In the fourth section, the OpenMake Meister build process for Java and J2EE applications is introduced. We discuss the setup and actual build steps used by OpenMake Meister.

The final section provides details on the Meister Plug-In for WebSphere Studio, the key integration point between Meister and WSAD. Typical example applications provide case studies for configuration and usage. We discuss how OpenMake Meister takes the build outside of the WSAD IDE and how the build process can be extended to provide additional functionality.

Challenges Building Java and J2EE Applications without Meister using WebSphere Studio Advanced Developer

WebSphere Studio Advanced Developer provides “point and click” build support in two steps, first, the creation of .class files and second, the creation of deployable archive files (.jar, .ear, .war). Builds are organized based on Workspaces allowing the entire Workspace to be built or allowing for the build of a single project within the Workspace.

For the creation of .class files, the user selects “Build” from the IDE menu to call the built-in Java Development Tooling (JDT) plug-in. The JDT simply compiles .java files into .class files. It will compile .java files into .class files, incrementally in an efficient and effective manner. In order to create the deployable archive files (.jar, .ear, .war), the developer points and clicks through the use of the built-in Export functionality.

Because the JDT and the Export functionality are managed through the IDE, the developers are shielded from the build steps. Information used to control the build such as project interdependencies and classpath may be “closed” and not readily accessible outside of the IDE.

Development efforts are managed with WebSphere based on Workspaces. A Workspace is a directory that contains objects associated with WebSphere projects. The Workspace also has associated metadata. This metadata includes developer preferences and build information about how the objects in the workspace get built and packaged. Each developer can define the metadata uniquely. This uniqueness means that the metadata associated with the workspace is specific to each developer’s workstation.

In a team setting, each developer has their own version of the workspace with the metadata being the main difference between developers. Each developer’s workspace contains a full source code tree. Developers coordinate the sharing of code by importing new versions of the code into their workspace as they see necessary based on team verbal communications or using a version management tool.

When builds occur, each developer builds their version of the workspace. Because each developer’s workspace can contain different source code and different workspace metadata, developers cannot confirm that a build of the workspace, regardless of developer, will produce the same results.

As the WebSphere application moves across the development lifecycle from development into production, build challenges are encountered. The following challenges must be overcome to ensure a repeatable, consistent WSAD build process at the developer workspace level and at a final production build level:

- Each developer is responsible for maintaining the correct level of code in their workspace. When a production build is required, inconsistencies between developer workspaces may cause the build to break.
- Developers may branch the source code managed in their workspace without realizing that there is a potential integration issue between shared code. These issues may not be realized until the final production build, often too late in the lifecycle to provide a simple fix.
- A single build expert must be used to manage the build and coordinate the team’s changes.
- To do the production build, the build requires a machine with the WebSphere Studio Application Developer IDE installed.

- The build expert must determine the correct version of the source code and manually confirm the correct settings of the workspace metadata to produce a stable version of the deployable objects.
- All builds are dependent upon the WebSphere IDE in a point and click method. The point and click method require multiple build steps.
- The workspace metadata including the buildpath information, the build machine and the build expert is critical in reproducing a single build.
- The ability to build parallel development efforts or support builds across a lifecycle (production, testing or emergency) becomes limited due to the dependency on a specific machine with a specific workspace configuration as well as the dependency on a particular build expert to perform a manual point and click process.

To avoid the point and click interface to the build process, the IBM recommended alternative is to develop an “outside” build using WebSphere in “headless” mode. Following is an excerpt from the WSDD Redbooks:

“... it would be best if the outside build could simply ask an Application Developer project to build itself using the existing project buildpath information. Such a build does not require the GUI to be running. You can have Application Developer launch itself "headless" and run a specified task, typically an Ant build. You need to create a wrapper `HeadlessAntRunner` that extends `AntRunner` and attaches a `HeadlessAntListener` as part of its run method. ...”

“Using Ant with WebSphere Studio Application Developer -- Part 1 of 3” [WSDD Library Support Downloads Redbooks Newsgroups All of IBM; Barry Searle](#)

Performing builds in headless mode only replaces the requirement have having a developer point and click through the process. All other challenges remain the same, i.e., team coordination of the WebSphere metadata, management of a specific build machine and heavy dependency on a particular build expert. This solution also adds the new challenge of maintaining build specific Java classes in order for the WSAD IDE to integrate with Ant and the creation of a build.xml Ant/XML script. Integration between ANT and WSAD is not simple. In addition, the supported Ant version (1.3 in WSAD 4.x, 1.4 in 5.0) may not support features that developers may require. Installing and using an updated version with WSAD will result in an unsupported installation.

The Meister Solution

Meister addresses the challenges that developers face when building their WebSphere Studio Application Developer workspaces in a team setting from development to production. Meister is a declarative software build automation tool designed to standardize application builds across the DevOps Pipeline. It is a unique tool in that it provides a standardized method to rebuild any executable module based on a platform type without being tied to a particular build location. Following is an explanation of how Meister resolves the critical challenges facing WSAD developers when building their applications.

Challenge:

Each developer is responsible for maintaining the correct level of code in their workspace. When a production build is required, inconsistencies between developer workspaces may cause the build to break.

Meister Solution:

OpenMake resolves this problem because it does not rely upon the workspace metadata. Instead, Meister manages Target Definition files (TGTs) that report the dependency information needed for the build. These TGTs can be shared, centralized and automatically updated depending on the development team requirements.

Challenge:

Developers may branch the source code managed in their workspace without realizing that there is a potential integration issue between shared code. These issues may not be realized until the final production build, often too late in the lifecycle to provide a simple fix.

Meister Solution:

Meister uses a Dependency Directory to allow developers to perform Unit builds against an approved build. This allows developers to build the code in their workspace against the last “approved” build. Because builds can be scheduled nightly, this more closely supports extreme programming where the continuous integration of source code is pursued.

Challenge:

A single build expert must be used to manage the build and coordinate the team’s changes.

Meister Solution:

The build knowledge base replaces the build expert. All build information is stored and managed by the Meister Knowledge Base and in the Meister Target Definition files (TGTs).

Challenge:

In order to do the production build, the build requires a machine with the WebSphere Studio Application Developer IDE installed.

Meister Solution:

Meister does not require the WSAD IDE installed. It does not use the WSAD IDE in “headless” mode. Meister comes with extensible PERL scripts that generate, dynamic, Ant/XML files. The generated Ant/XML files then call the appropriate Java tasks.

Challenge:

The build expert must determine the correct version of the source code and manually confirm the correct settings of the workspace metadata to produce a stable version of the deployable objects.

Meister Solution:

Meister integrates with version management tools to determine the correct version of source code to be included in the build. In addition, the workspace metadata is replaced by the Meister TGT files and Knowledge Base.

Challenge:

All builds are dependent upon the WebSphere IDE in a point and click method. The point and click method require multiple build steps.

Meister Solution:

Meister’s command line features allow you to automate the execution of the WSAD build. No additional Java coding or Ant scripting is required.

Challenge:

The workspace metadata including the buildpath information, the build machine and the build expert is critical in reproducing a single build.

Meister Solution:

Meister does not rely on workspace metadata as all target information is derived from the workspace and saved in a Target Definition file (TGT). The combination of the TGT and the Knowledge Base allows for a repeatable build process across any workstation and executed by any person.

Challenge:

The ability to build parallel development efforts or support builds across a lifecycle (production, testing or emergency) becomes limited due to the dependency on a specific machine with a specific workspace configuration as well as the dependency on a particular build expert to perform a manual point and click process.

Meister Solution:

Meister can dynamically generate a build for any level of the application based on an Meister Project and Dependency Directory. A developer can build any level of any WSAD Project from the command line or from the Eclipse plug-in.

Meister Technical Overview

Meister is a build process automation tool designed to standardize Client/Server and Internet application builds from development through production. It is a unique tool in that it provides a standardized method to rebuild any executable module based on platform type without the dependency of a particular development machine. This is critical for development teams during day-to-day compilations, as well as the change and configuration management teams during the production turnover process. Since Meister automates the entire build process and tracks dependencies between application components, the production turnover process can easily incorporate the rebuilding of all source modules turned over by the development teams. Development teams can easily implement nightly production builds without the need for a designated employee (the build master) who is dedicated to managing the application system make file.

Meister Projects

A Meister project defines a set of build targets and the directory locations in which all source files for all the project's build targets can be found. Meister projects typically correspond in a one-to-one relationship with the WebSphere Studio Workspace.

The 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 Files (e.g. target-dependency relationships).

The Meister Knowledgebase contains information on:

- **Build Types and Rules**
Build target types and all of the rules and compile flags necessary to create a target from the dependencies for that type.
- **Project Dependency Directories**
Source directories allowed for a particular configuration of an application including all source code, libraries and Meister target definition files.
- **Build Machines**
Information on remote build machines that may be used to build different components of a project.
- **Groups and Users**
Information on the users and their corresponding groups used to organize the availability and presentation of the Knowledge Base

Target Definition Files

Target Definition Files, identified by a .tgt file extension, are defined by developers and indicate build targets and high-level dependency information for those targets. These Target Definition Files are source code for the build. The creation of target definition files using the Meister Web Client represents the minimum required effort for developers.

Unlike other build methodologies, a target definition file does not require scripting of any kind by developers. It is simply a target name, Build Type and high-level dependency list.

Using Meister with WebSphere Studio Development

Meister provides full support of the use of Ant tasks when building Java based applications. When using Meister with IBM WebSphere Studio Application Developer, Ant tasks are derived based on the WSAD project file. Developers work within the WSAD IDE with Meister monitoring the changes being made to the project dependency through the Meister Eclipse plug-in. By monitoring the developer's work activities, Meister can maintain accurate Target Definition files (TGT). These TGT files are then used outside of the WSAD IDE to perform project builds at any release level or state of a development lifecycle.

This section provides an overview of how to set up and execute builds using Meister both within and outside the WSAD IDE. The key integration point is the Meister Plug-In that

- directly gathers information on the contents of the WSAD Projects.
- helps the developer set up a build using Meister.

Two examples are provided:

1. A standalone Java project: MetalWorks
Details the basics on setup and use of the Meister Plug-In
2. A J2EE application: MiniBank
A typical Enterprise Application consisting of:
 - MiniBank: EAR Project
 - MiniBankEJB: EJB Project
 - MiniBankWeb: Web Application Project

The Meister Plug-In is the same in both WebSphere Studio Application Developer 4.x and 5.0. MetalWorks is presented in 4.x and MiniBank in 5.0.

WebSphere Studio Development Assumptions

The developer should already be familiar with WSAD development processes. The WSAD Projects for the MetalWorks and MiniBank examples have already been imported into the developer's WSAD Workspace and can be built successfully within the IDE.

Meister Installation Assumptions

The Meister Knowledge Base Server is installed and running properly on a centralized server that the developer's workstation have network access to. The Meister Command Line Client is installed on the developer's local workstation.

Meister Knowledge Base Setup

In order to build a WSAD Project using Meister, it must be associated with an Meister Project. Typically, a Workspace consisting of multiple related WSAD Projects, will be associated to a single Meister Project. The Meister Project is configured using the Meister Web Client. Go to the Manage Projects menu option from the main menu on the Meister Web Client.

Meister Dependency Directory must be defined for each Meister Project. Suggested Dependency Directory configurations include:

- In WSAD or in the WSAD Workspace Root directory:
 - .
 - ./tgt
 - [WebSphere Studio Library Directory]
 - [Third-Party Library Directory]
 - [Java Home]/lib
 - [Java Home]/jre/lib
- Outside WSAD, referencing the WSAD Workspace Root directory:
 - .
 - [WebSphere Studio Workspace Root]
 - [WebSphere Studio Workspace Root]/tgt
 - [WebSphere Studio Library Directory]
 - [Third-Party Library Directory]
 - [Java Home]/lib
 - [Java Home]/jre/lib
- Outside of WSAD, referencing an SCM-managed directory
 - .
 - ./tgt
 - [SCM-Managed Directory]
 - [SCM-Managed Directory]/tgt
 - [WebSphere Studio Library Directory (SCM-Managed)]
 - [Third-Party Library Directory (SCM-Managed)]
 - [Java Home]/lib
 - [Java Home]/jre/lib

Installing the Meister Plug-In

The Meister Plug-In for WebSphere Studio Application Developer can be found in the WSAD directory on the Meister installation CD. Execute install.exe to initiate the install process.

The installer will automatically detect which version of WSAD is installed (4.x or 5.0) and the installation directory. These settings can be updated during the installation process, particularly if more than one version of WSAD is installed.

After installation is complete, a new subdirectory will be created in the WSAD plugins directory:

- 4.x: [WSAD]\plugins\com.openmake.eclipse_1.0.x
- 5.0: [WSAD]\eclipse\plugins\com.openmake.eclipse_1.0.x

where x is the current revision number of the Meister Plug-in.

To verify that the Meister Plug-In is recognized, start up WebSphere Studio Application Developer and go to Help->About.... The Plug-In Details button will list all installed plug-ins. The Meister Plug-In will have an entry:

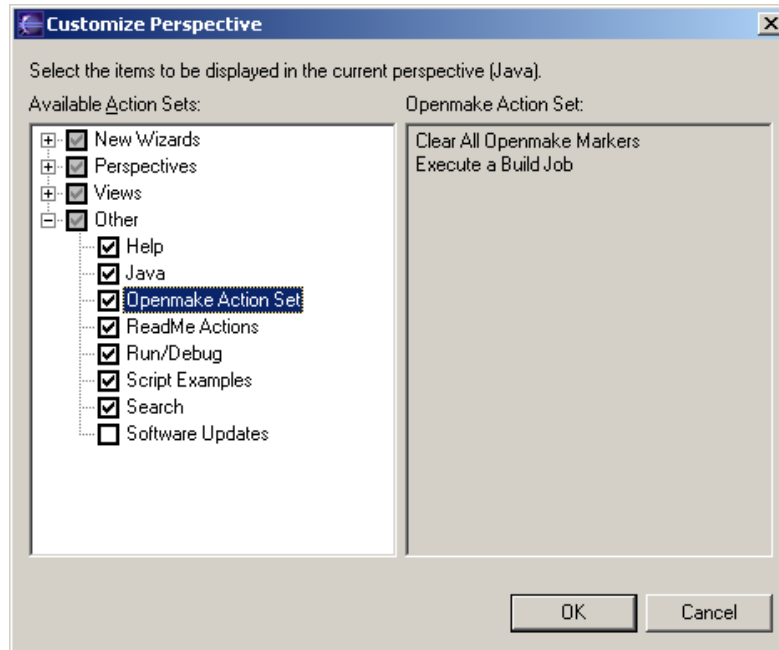
Catalyst Systems Corporation	Meister Plug-In	1.0.x
------------------------------	-----------------	-------

Activating Meister Plug-In Menus within WebSphere Studio

While context-sensitive popup menu options will be immediately available, Meister-specific menu-bar items will need to be activated through Perspective Customization:

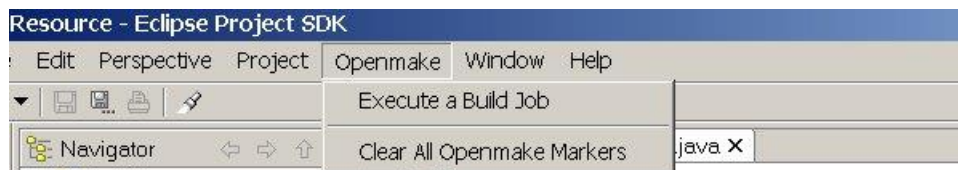
1. Go to

- a. 4.x: Perspective->Customize
- b. 5.0: Window->Customize Perspective...
2. In the Customize Perspective dialog, expand Other
3. Check the box next to "Meister Action Set":



4. Click OK

A Meister menu will be now be visible:

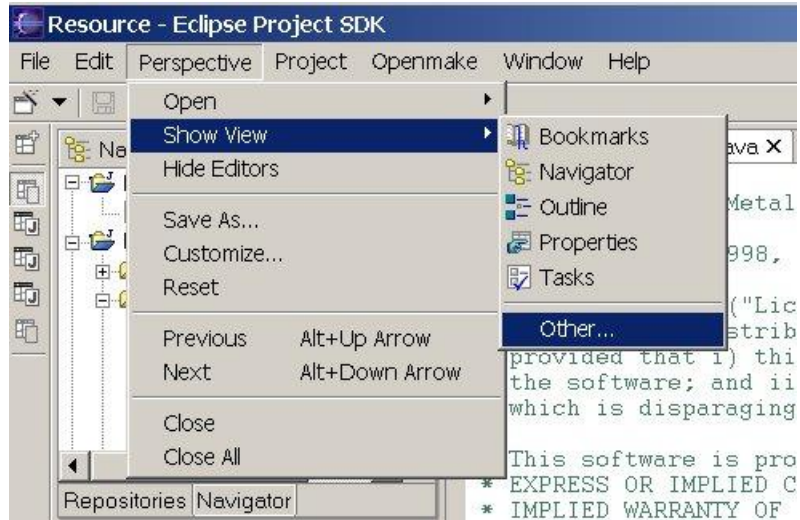


This process should be repeated in all Perspectives where the Meister Plug-In will be used. Typically, this will be in Java and J2EE.

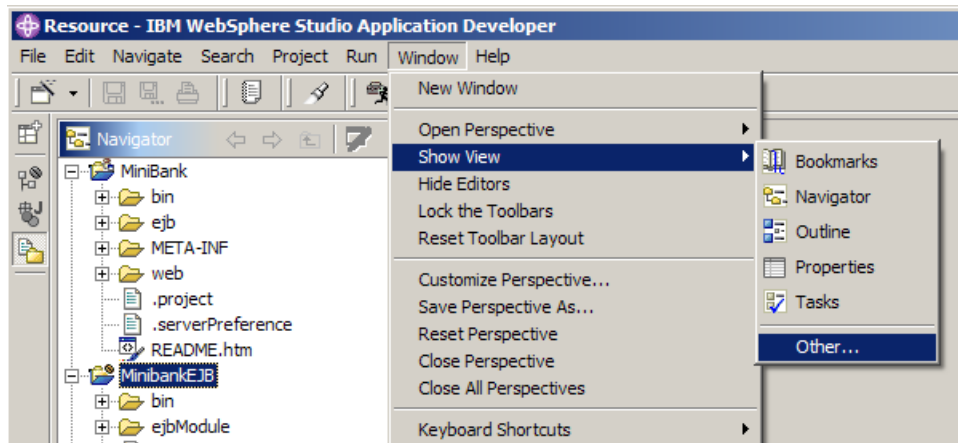
Accessing Meister Plug-In Views in WebSphere Studio

To open the Meister Plug-In Build Setup and Console Views:

1. Go to
 - a. 4.x: Perspective->Show View->Other



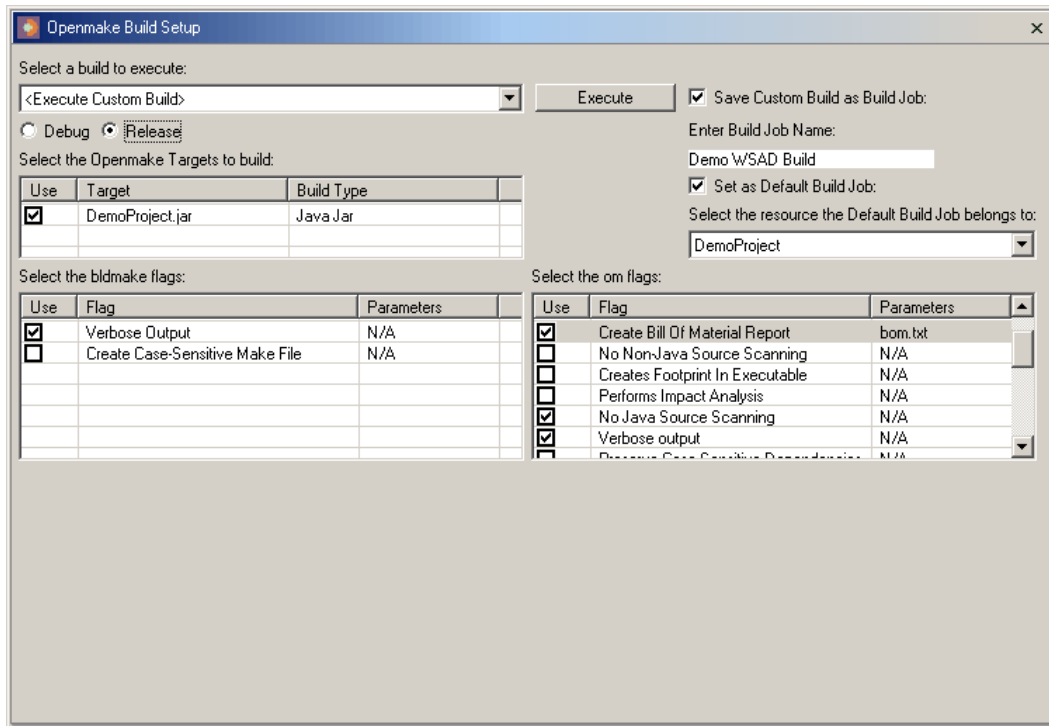
b. 5.0: Window->Show View->Other



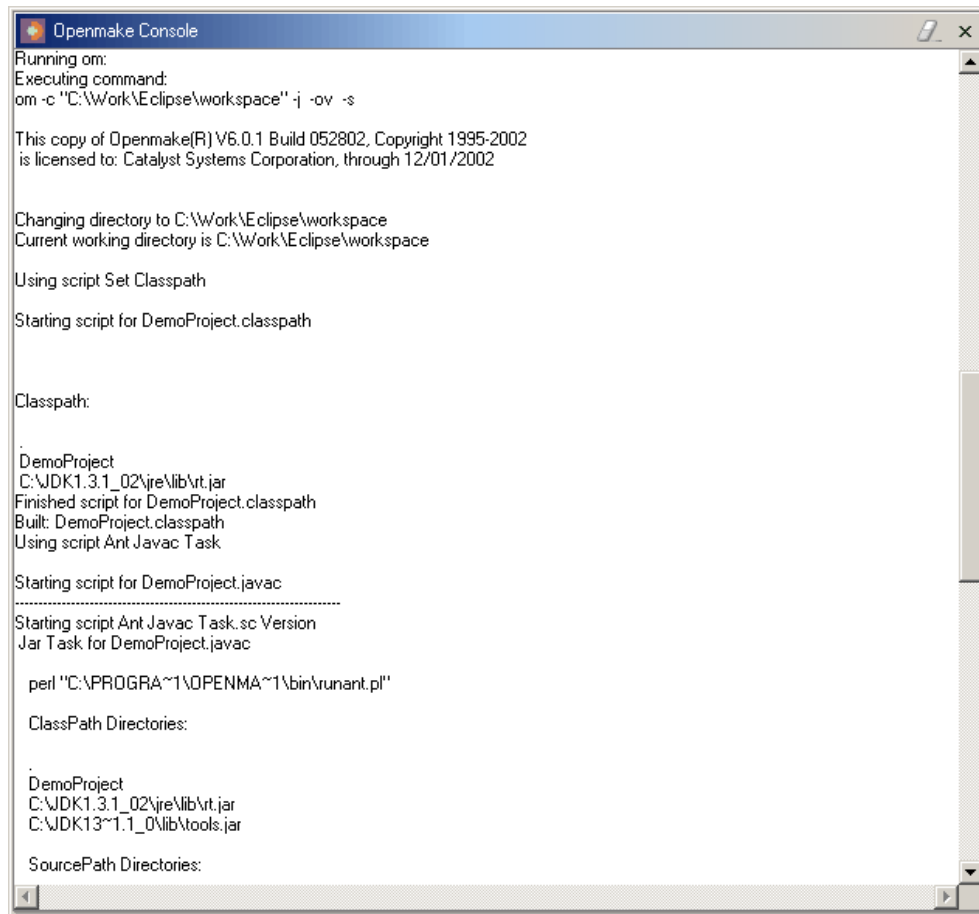
2. Expand the Meister section



- and choose either
- a. Meister Build Setup



- b. Meister Console



3. Click OK

Typically, only the Meister Build Setup View needs to be opened in this manner. The Meister Console will be opened automatically when a Meister build is being performed through the WebSphere Studio IDE.

Walk-Through Examples

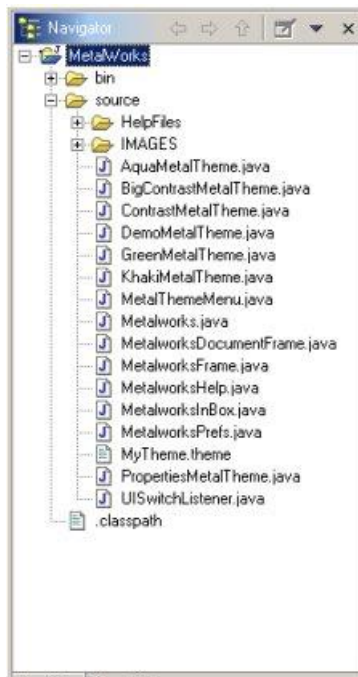
Example 1 – MetalWorks – a Standard Java Project

MetalWorks is an example application that comes with Meister. The Dev Dependency Directory is already configured for building within WSAD:

```
./tgt
$(REF_DIR)/MetalWorks/development
$(JAVA_HOME)/lib
$(JAVA_HOME)/jre/lib
```

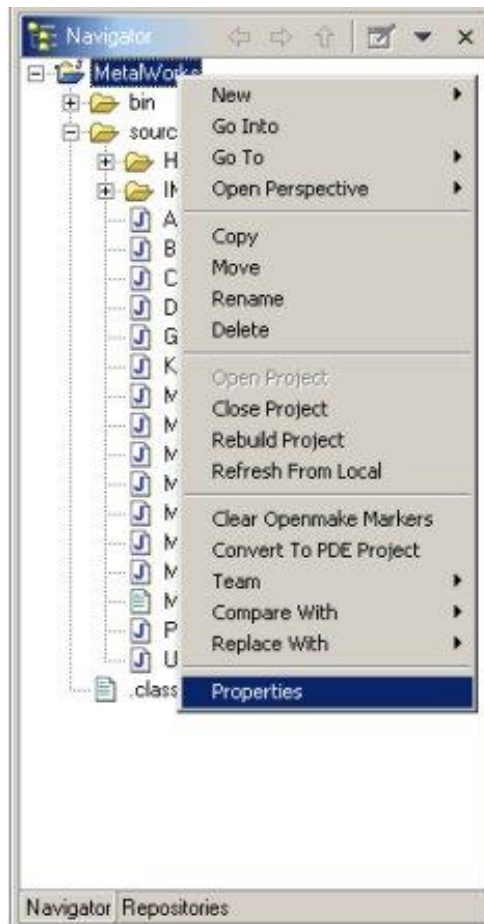
This Dependency Directory is a combination of the typical within WSAD and SCM-managed Dependency Directories. MetalWorks is a relatively simple straightforward Java application and only depends on rt.jar. WSAD J2EE libraries are not required.

The MetalWorks Java application has been loaded as follows:

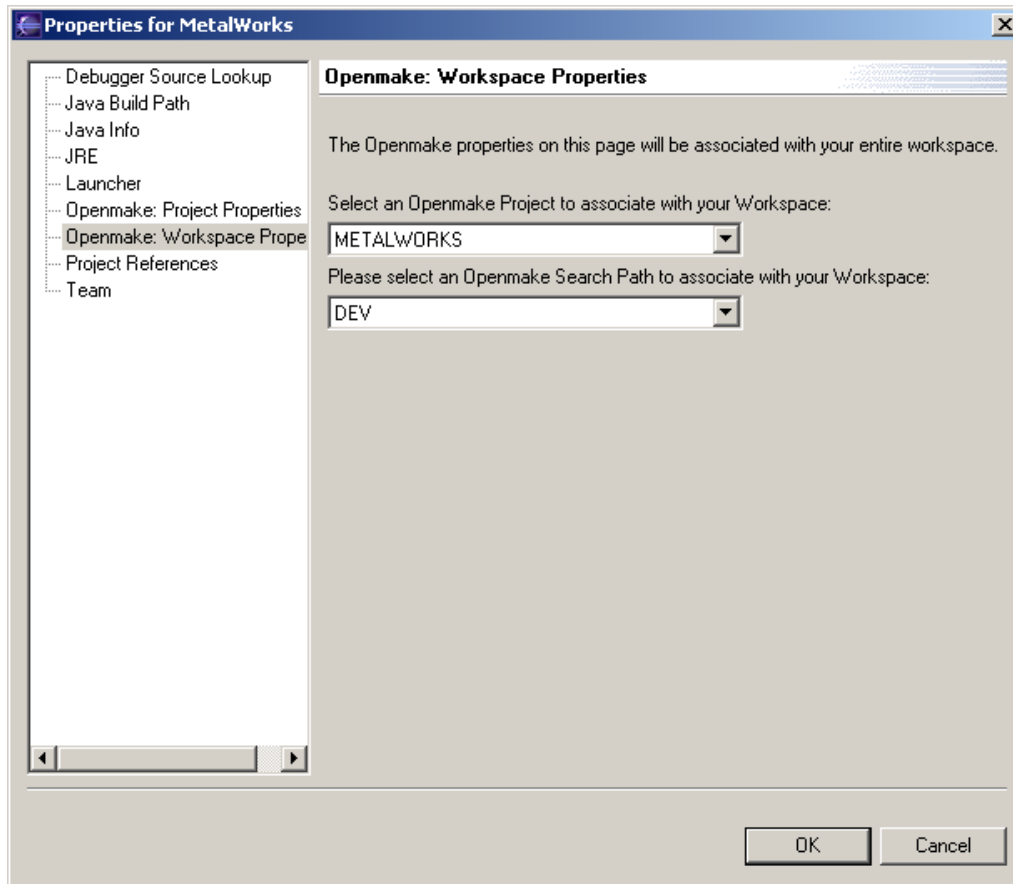


To configure the Meister Build:

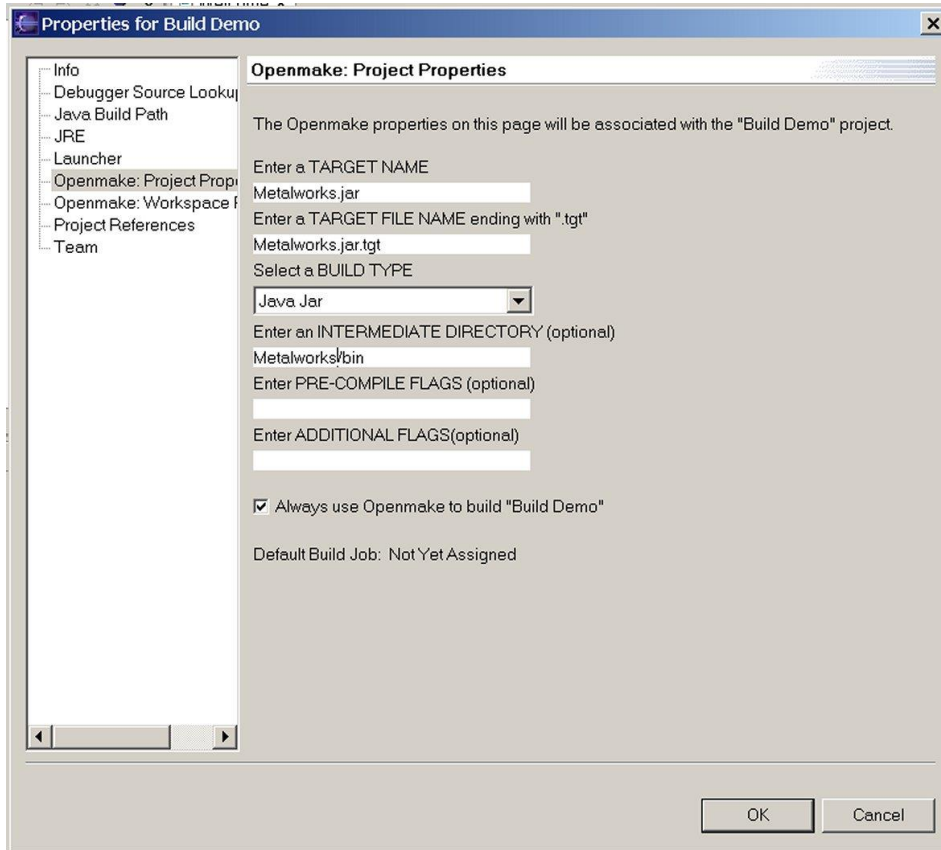
1. Open Project Properties for MetalWorks:



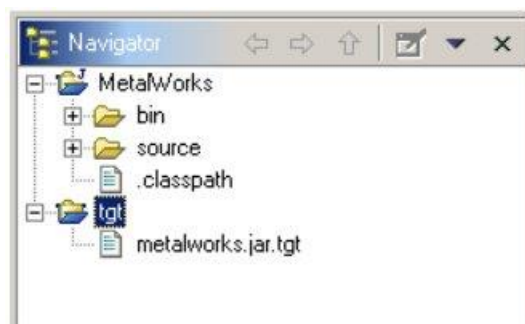
2. Go to the Meister: Workspace Properties page and select the METALWORKS Meister Project and DEV Meister Dependency Directory



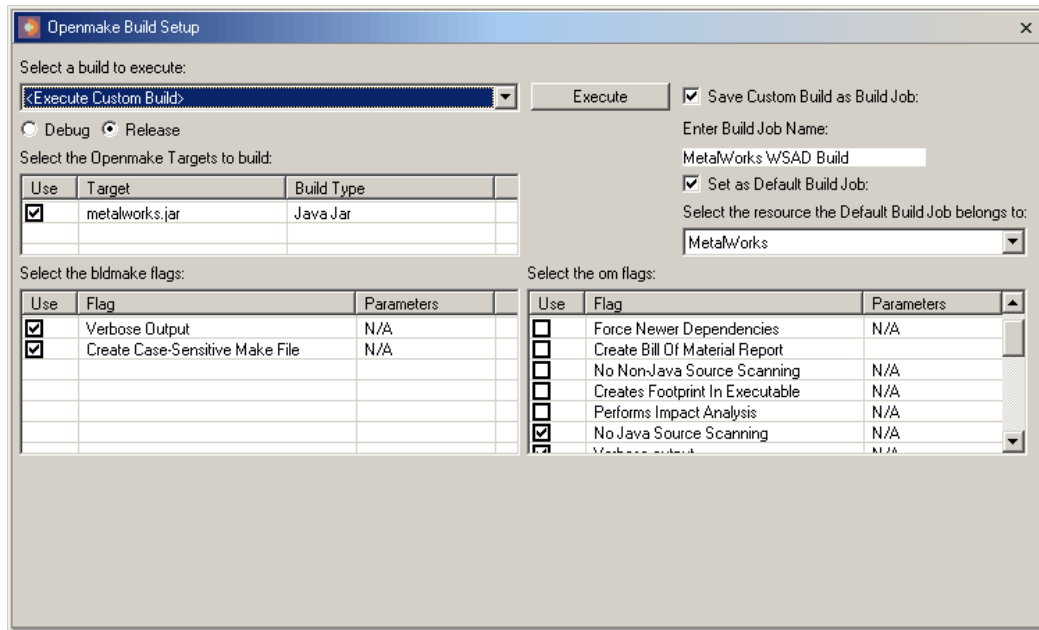
3. Go to the Meister: Project Properties page and enter in the following:
- Target Name: metalworks.jar
 - Target File Name: metalworks.jar.tgt
 - Build Type: Java Jar
 - Intermediate Directory: MetalWorks/bin
 - Always Use Meister To Build Note: This option will not be available at this point



After clicking OK, the Meister Plug-In will create a new WSAD Project “tgt” and create the .tgt file associated with the WSAD Project MetalWorks:

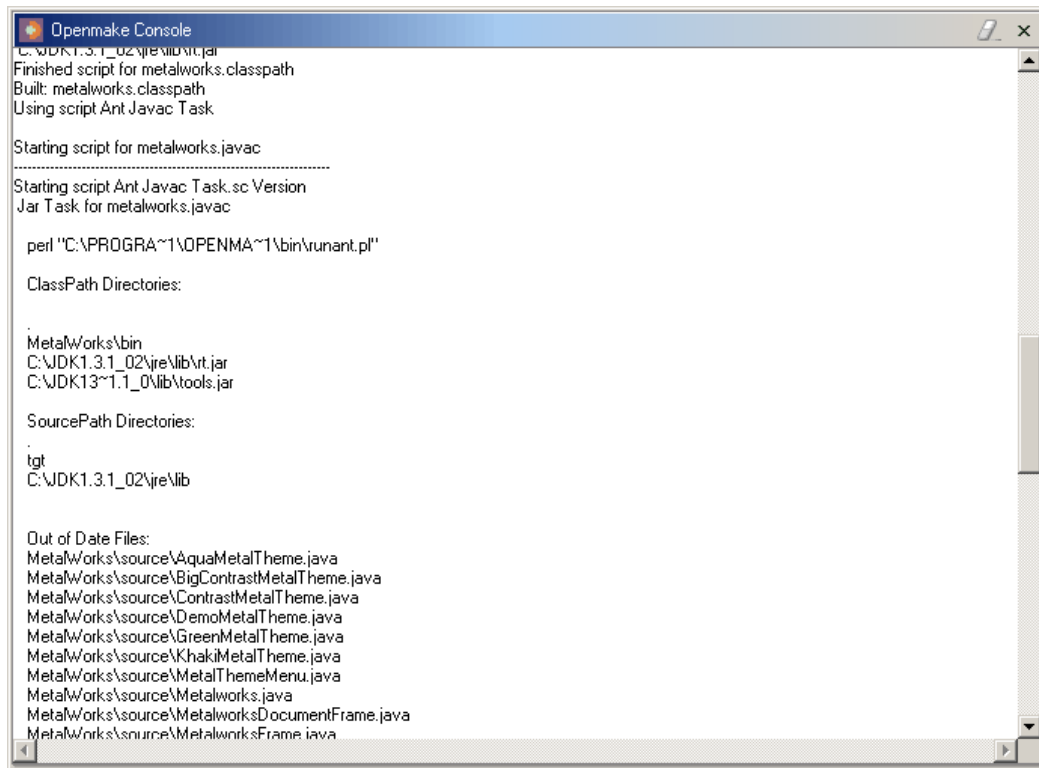


4. Open the Plug-in Build Setup Viewer and create a Custom Build with the following selections:
 - Target to build: metalworks.jar
 - bldmake flags: Verbose Output
 - om flags: Verbose Output
 - Save Custom Build as Build Job:
 - Enter “Metalworks WSAD Build” as Build Job Name
 - Set as Default Build for MetalWorks



Click Execute to perform build and save the Build Job.

The Console should be open and displaying the output log for the build:



```
Openmake Console
C:\JDK1.3.1_02\jre\bin\jar
Finished script for metalworks.classpath
Built: metalworks.classpath
Using script Ant Javac Task
Starting script for metalworks.javac
-----
Starting script Ant Javac Task.sc Version
Jar Task for metalworks.javac

perl "C:\PROGRAMS\OPENMAKE\bin\runant.pl"

ClassPath Directories:
.
MetalWorks\bin
C:\JDK1.3.1_02\jre\lib\rt.jar
C:\JDK13~1_0\lib\tools.jar

SourcePath Directories:
.
tgt
C:\JDK1.3.1_02\jre\lib

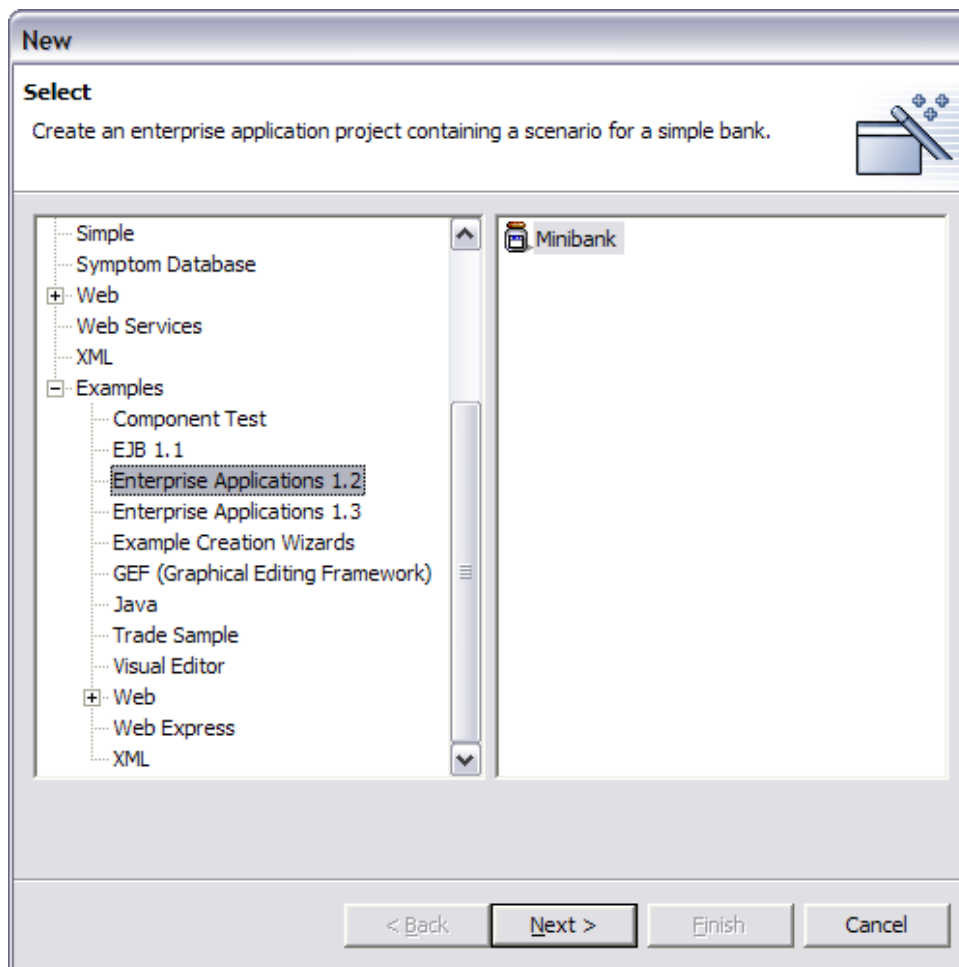
Out of Date Files:
MetalWorks\source\AquaMetalTheme.java
MetalWorks\source\BigContrastMetalTheme.java
MetalWorks\source\ContrastMetalTheme.java
MetalWorks\source\DemoMetalTheme.java
MetalWorks\source\GreenMetalTheme.java
MetalWorks\source\KhakiMetalTheme.java
MetalWorks\source\MetalThemeMenu.java
MetalWorks\source\Metalworks.java
MetalWorks\source\MetalworksDocumentFrame.java
MetalWorks\source\MetalworksFrame.java
```

A successful build will create the target metalworks.jar in the Workspace Root directory.

Example 2 - MiniBank

MiniBank is an example J2EE application that has provided with WSAD 5.0. To load the example into the WSAD Workspace:

1. Go to File->New->Other...
2. Select Examples->Enterprise Application 1.2 and choose MiniBank



Clicking Next> will allow you to select the EAR Project name

3. Change the Project name to MiniBank and click Finish.

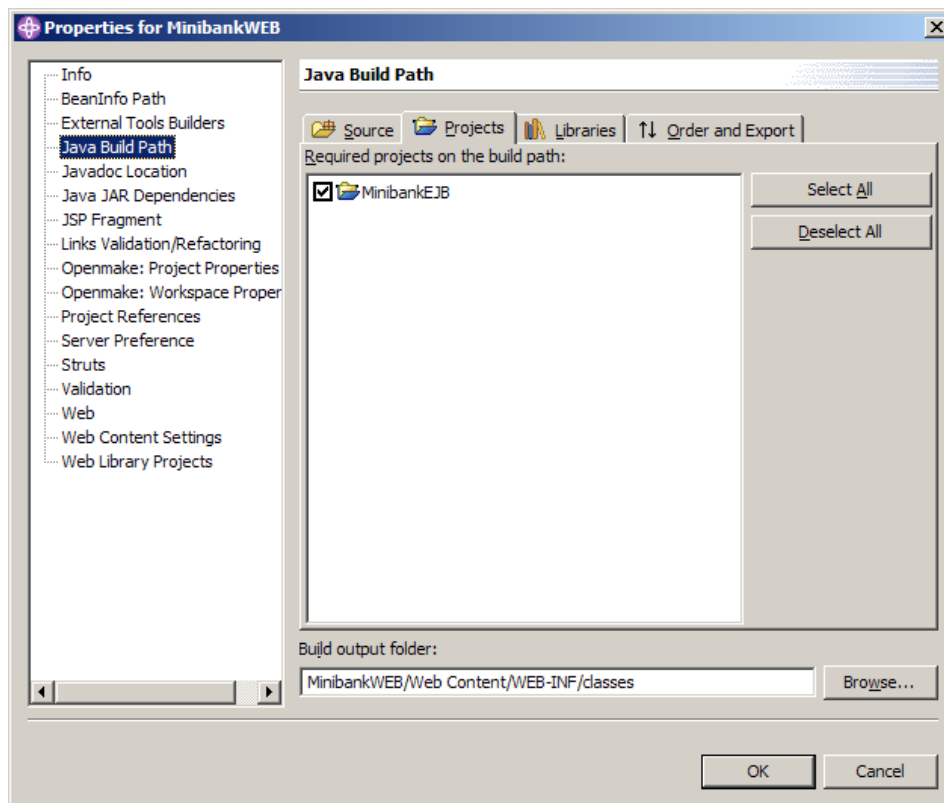
The MiniBank Enterprise Application Project along with MinibankEJB and MinibankWEB will be automatically be created in the Workspace.

A Meister Project MiniBank should be created along with a Dev Dependency Directory as follows:

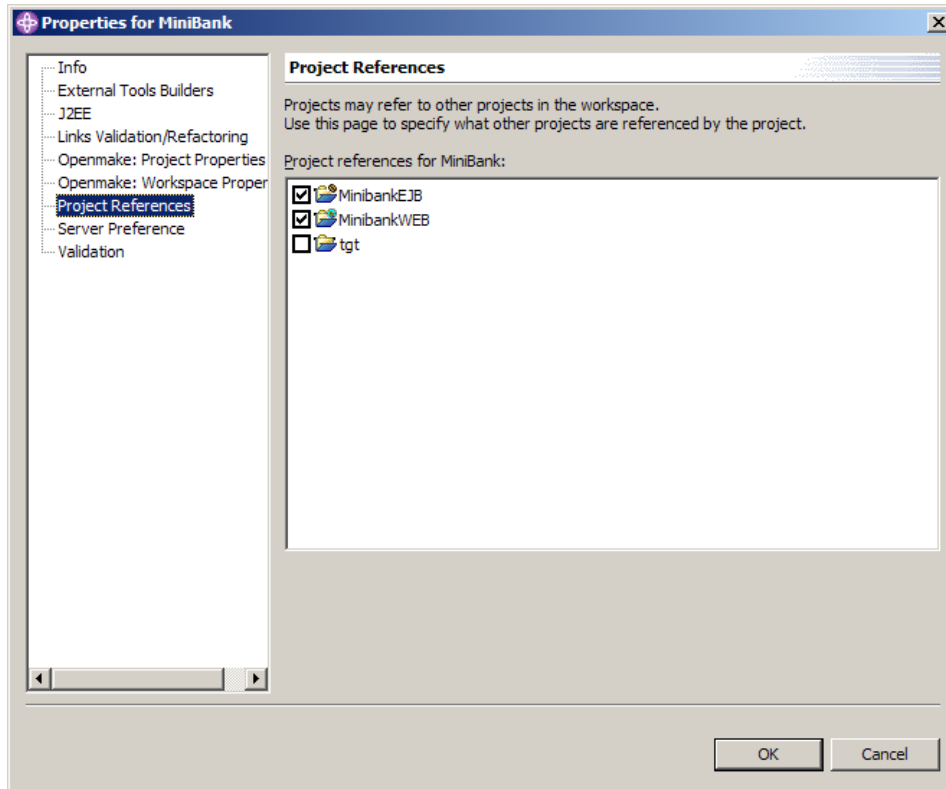
```
./tgt  
[WSAD 5.0 Install Directory]/runtimes/aes_v4_jars/lib  
[WSAD 5.0 Install Directory]/runtimes/base_v5/lib  
[Java Home]/lib  
[Java Home]/jre/lib
```

The process for configuring the Meister build is similar to MetalWorks example. However, WSAD Project dependencies exist that should be verified. Meister requires these settings in order to generate the associated .tgt's properly.

1. Verify WSAD Project dependencies
 - a. MinibankWeb
Java Build Path should specify a dependency on MinibankEJB



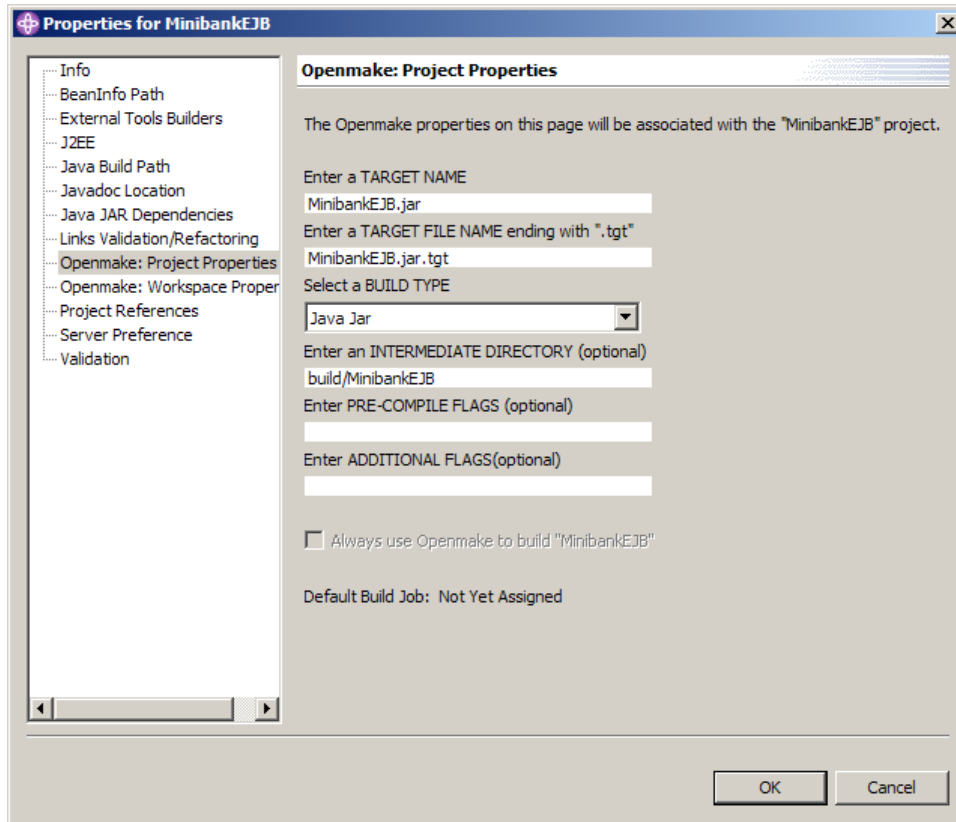
- b. MiniBank
Project References should specify dependencies on both MinibankEJB and MinibankWeb



2. Set Meister Properties

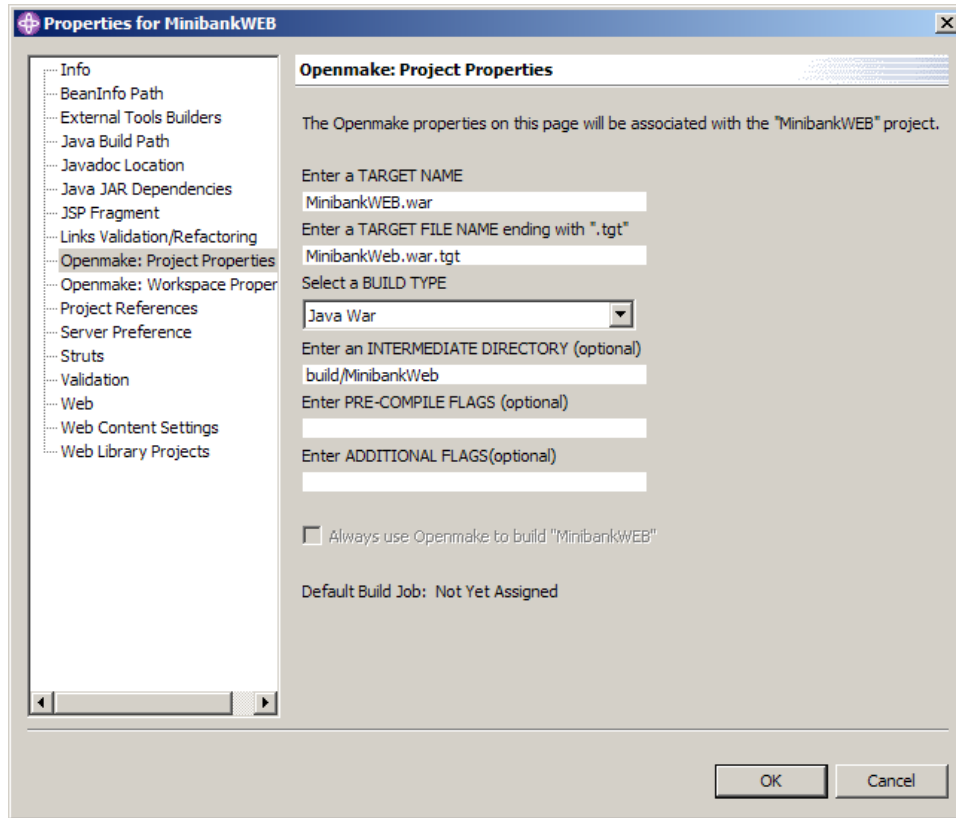
a. MinibankEJB:

- Target Name: MinibankEJB.jar
- Target File Name: MinibankEJB.jar.tgt
- Build Type: Java Jar
- Intermediate Directory: build/MinibankEJB



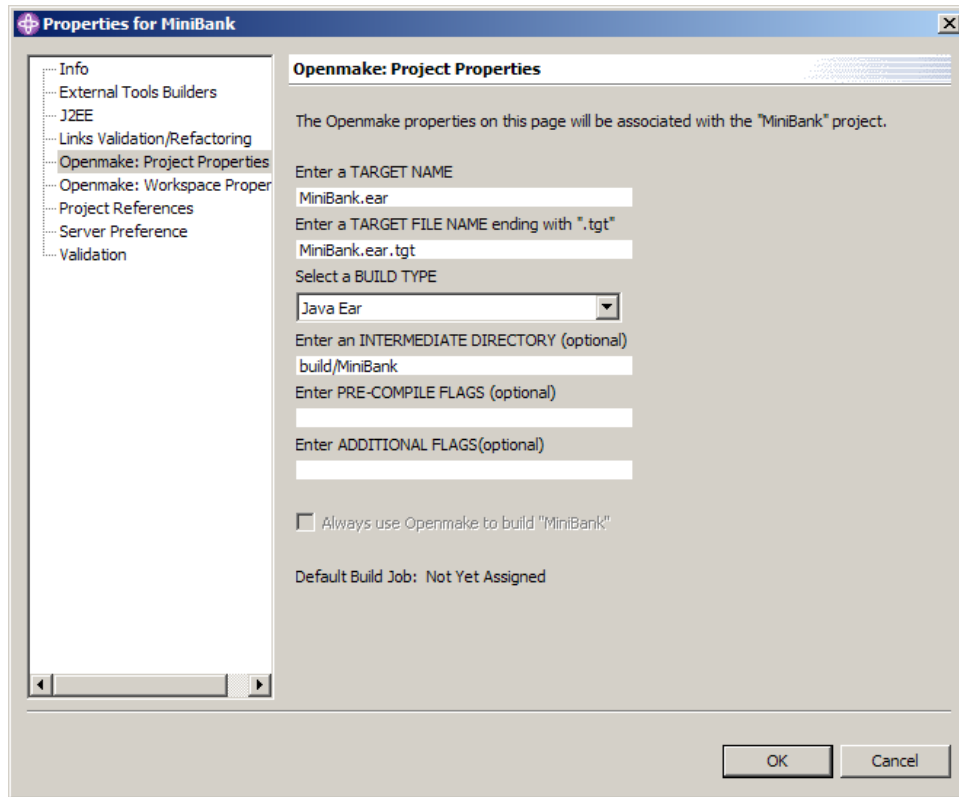
b. MinibankWeb:

- Target Name: MinibankWeb.war
- Target File Name: MinibankWeb.war.tgt
- Build Type: Java War
- Intermediate Directory: build/MinibankWeb

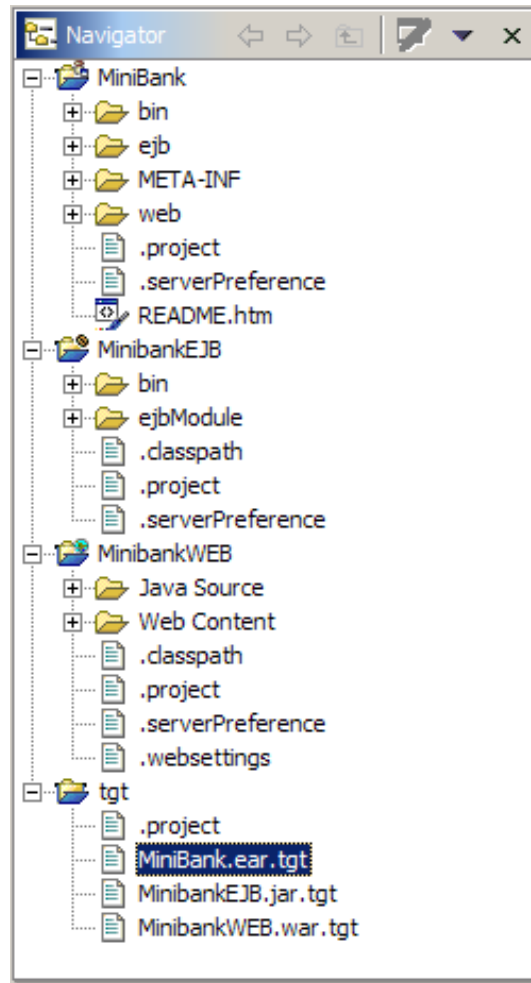


c. MiniBank:

- Target Name: MiniBank.ear
- Target File Name: MiniBank.ear.tgt
- Build Type: Java Ear
- Intermediate Directory: build/MiniBank



After the WSAD Projects have been configured, the Workspace will look like:



3. Configure and execute the Meister Build from the Build Setup View
 - Targets to build: MinibankEJB.jar, MinbankWeb.war, MiniBank.ear
 - bldmake flags: Verbose Output
 - om flags: Verbose Output
 - Save Custom Build as Build Job:
 - Enter “MiniBank WSAD Build” as Build Job Name
 - Set as Default Build for All Projects

A successful build will create the targets MinibankEJB.jar, MinbankWeb.war, and MiniBank.ear in the Workspace Root directory. MiniBank.ear will be deployable on a WebSphere Application Server.

Performing a Meister Build Outside of WebSphere Studio

The process Meister uses to perform builds outside of the WebSphere Studio IDE is identical to that used by the Plug-In. The only requirements are

- Meister .tgt files generated by the Meister Plug-In
- Source files either in the Workspace Root directory or in a sandbox/reference equivalent
- WebSphere J2EE libraries
- Java libraries

No additional interfaces to WSAD are required. Builds may also be performed on either the developers' workstations or a dedicated build server.

A typical build procedure is as follows:

1. Set up environment
Set JAVA_HOME, PATH, etc. as necessary to ensure a consistent build environment
2. Run bldmake to create Meister build file
bldmake [Meister Project] [Project Dependency Directory] -s

-s instructs bldmake to be case-sensitive
3. Run om to execute build
om -j -ov

-j turns off Java source code scanning by Meister. All necessary dependencies are assumed to have been already specified
-ov turns on verbose output

Integrating the Meister Build with SCM-Managed WebSphere Studio Development

The .tgt files generated by the Meister Plug-In can be versioned within the WSAD IDE in the same manner as other Workspace files. This allows a complete versioning of both the source code and the build process.

Automated builds can be set up for

- Nightly
- Snapshot
- Application Lifecycle (QA, Release, etc.)

Extending the Meister Build Process

Meister functionality can be readily extended through

- Additional Build Types and Rules
- New or updated Meister Build Scripts via the SDK

One particular scenario where extended Meister functionality has been used is the configuration of an Enterprise Application .ear for deployments to additional Application Servers. Typical development may tie the files in the WSAD Workspace to a particular network location.

Updates are typically required in

- Deployment descriptor files `ejb-jar.xml` and/or `web.xml`
- Application-specific configuration files

to point to location-specific databases, servers, or directories.

Two solutions have been implemented:

1. Generation and update of configuration files in an existing `.ear`
Meister Build Types and associated Build Scripts have been created using the SDK to

- Generate location-specific deployment descriptor and configuration files:

Source files:

- Original deployment descriptor and configuration files
- Location configuration data file

Output files:

- Location-updated deployment descriptor and configuration files

- Update `.jar`, `.war`, `.ear` with deployment descriptor and configuration files:

Source files:

- Original `.jar`, `.war`, and `.ear`
- Location-updated deployment descriptor and configuration files

Output files:

- Location-updated `.ear`

2. Dependency Directory selection of configuration files
Multiple versions of the deployment descriptor and other configuration files are maintained in separate directories. Standard Meister Build Types and Rules can be used.

For MyApp.ear
ejb-jar.xml versions:

```
/source/config/dev/MyEJB/META-INF/ejb-jar.xml  
/source/config/qa/MyEJB/META-INF/ejb-jar.xml  
/source/config/release/MyEJB/META-INF/ejb-jar.xml
```

For the Meister Project MyApp

Dev SearchPath

```
...  
/source/config/dev  
/source  
...
```

Build will use /source/config/dev/MyEJB/META-INF/ejb-jar.xml

QA SearchPath

```
...  
/source/config/qa  
/source  
...
```

Build will use /source/config/qa/MyEJB/META-INF/ejb-jar.xml

Release SearchPath

```
...  
/source/config/release  
/source  
...
```

Build will use /source/config/release/MyEJB/META-INF/ejb-jar.xml

Final build target is a location-specific .ear.

Company Overview

OpenMake Software started the evolution of builds in 1995, serving mainly the financial community with the mission of delivering a 100% insulated build process that were also fast. The OpenMake Software team understood the ins and outs of software compiles and links, and how easily a build could be the bottleneck of the software delivery process and be easily compromised on accident or on purpose. With this mission in mind, OpenMake Meister was created and has been serving large enterprises for over 25 years, the longest serving solution in the DevOps ecosystem. Meister has been sold and distributed by Broadcom for over 20 years.