



[serena.com](http://serena.com)

# **SERENA<sup>®</sup>** **BUILDER<sup>™</sup> FOR PROFESSIONAL<sup>™</sup> 6.4.1**

**User's Guide**

Serena Proprietary and Confidential Information



Copyright © 2003–2007 Serena Software, Inc. All rights reserved.

This document, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. Except as permitted by such license, no part of this publication may be reproduced, photocopied, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Serena. Any reproduction of such software product user documentation, regardless of whether the documentation is reproduced in whole or in part, must be accompanied by this copyright statement in its entirety, without modification.

This document contains proprietary and confidential information, and no reproduction or dissemination of any information contained herein is allowed without the express permission of Serena Software.

The content of this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Serena. Serena assumes no responsibility or liability for any errors or inaccuracies that may appear in this document.

### **Trademarks**

Serena, TeamTrack, StarTool, PVCS, Collage, Comparex, Dimensions, RTM, Change Governance, and ChangeMan are registered trademarks of Serena Software, Inc. The Serena logo, Professional, Version Manager, Builder, Meritage, Command Center, Composer, Reviewer, Mariner, and Mover are trademarks of Serena Software, Inc.

All other products or company names are used for identification purposes only, and may be trademarks of their respective owners.

### **U.S. Government Rights**

Any Software product acquired by Licensee under this Agreement for or on behalf of the U.S. Government, its agencies and instrumentalities is "commercial software" as defined by the FAR. Use, duplication, and disclosure by the U.S. Government is subject to the restrictions set forth in the license under which the Software was acquired. The manufacturer is Serena Software, Inc., 2755 Campus Drive, San Mateo, CA 94403.

Part number: MA-PROBUG-003

Publication date: March 2007

# Table of Contents

---

<b>Welcome to Version Manager</b> . . . . .	<b>7</b>
Typographical Conventions . . . . .	8
Contacting Technical Support . . . . .	8
<b>Introduction</b> . . . . .	<b>9</b>
What is Serena Builder for Professional? . . . . .	10
Audience for This Document . . . . .	10
Serena Builder Products . . . . .	10
Where to Find Information . . . . .	11
<b>Concepts</b> . . . . .	<b>13</b>
Understanding Serena Builder . . . . .	14
Serena Builder Components . . . . .	18
Serena Builder Clients . . . . .	19
Serena Builder Standalone Web Client . . . . .	20
Serena Builder Desktop Client . . . . .	21
Serena Builder Integrated Web Client . . . . .	22
Serena Builder PCLI Client . . . . .	23
Serena Builder Command-Line Client . . . . .	24
Advantages of the Integrated Clients . . . . .	24
Summarized Differences Between Clients . . . . .	24
Targets, Files, Projects, and Build Jobs . . . . .	25
Search Paths . . . . .	26
Different Search Path = Different Build Target . . . . .	26
Search Paths Look in Current Directory First . . . . .	26
Serena Builder Uses First-Found Copy of File . . . . .	27
Search Path Example . . . . .	27
Search Paths Can Contain Environment Variables . . . . .	28
Admin Controls Permissions to Modify Search Path . . . . .	28
Example Serena Builder Workflow . . . . .	28
Serena Builder vs. the Traditional Build Process . . . . .	29
Advantages of Serena Builder . . . . .	29
Serena Builder vs. Configuration Builder . . . . .	30
What to Do with Existing Build Processes . . . . .	30
<b>Installation</b> . . . . .	<b>31</b>
Installation Tasks for Serena Builder for Professional . . . . .	32
About Supported Platforms . . . . .	32
Installation Steps Done by the Administrator . . . . .	32
Remote Build Server Issues . . . . .	33
Installation Steps Done by You . . . . .	34

Installing the Command-Line Client . . . . .	36
Who Needs to Install the Command-Line Client. . . . .	36
Remote Build Server Licensing . . . . .	37
Licensing . . . . .	37
Knowledge Base Server Permanent Licenses. . . . .	37
Command-Line Client Permanent Licenses . . . . .	37
Remote Build Server Permanent Licenses. . . . .	38
PCLI Licensing Tied to Version Manager . . . . .	38
UNIX: Adding Setup Information to Accounts . . . . .	38
Upgrading from a Merant Build Installation . . . . .	39
Preserving Merant Build Projects and Build Jobs . . . . .	39
Converting Merant Build TGT Files to Serena Builder TGT File Format	39
Adapting Build Jobs to Changed Build Types . . . . .	40
<b>Tutorial . . . . .</b>	<b>43</b>
Creating a Java Build . . . . .	44
Set the Build Directory. . . . .	44
Import or Create Source Files . . . . .	44
Set Up the Build . . . . .	44
Add Targets . . . . .	46
Fill Out Target Overview. . . . .	47
Fill Out Target Detail . . . . .	48
Select Targets to be Built. . . . .	50
Select Options . . . . .	50
Execute the Build . . . . .	51
Check the Results . . . . .	51
Archive the Target Definition File. . . . .	51
Creating a Java Jar . . . . .	52
Add Targets . . . . .	52
Fill Out Target Overview. . . . .	52
Fill Out Target Detail . . . . .	52
Select and Submit Target. . . . .	53
Working With Multiple Source Directories . . . . .	54
Creating Development and QA Builds . . . . .	55
Creating Builds Based on Version Labels. . . . .	55
Creating Builds Based on Promotion Groups . . . . .	56
Complex Build Job Examples . . . . .	56
Example of Using the Serena Builder Desktop Client . . . . .	56
Create and Archive the .TGT File. . . . .	56
Select a Build Method and Build Machine . . . . .	57
Select Targets to Be Built. . . . .	57
Specify Build Parameters . . . . .	58
Submit the Build Job . . . . .	58
<b>BINGO Tutorial . . . . .</b>	<b>59</b>
Check Prerequisites . . . . .	60
Prepare the Build Environment . . . . .	60
Overview of the BINGO Project . . . . .	61

Set the Build Directory . . . . .	61
Copy Example Files to Working Directory . . . . .	61
Define Search Paths for the BINGO Project . . . . .	62
Set Up the BINGO Client Build. . . . .	63
The Set Classpath Task . . . . .	65
The Ant Javac Task . . . . .	65
The Ant Jar Task . . . . .	65
Option Groups and Source/Destination Directories . . . . .	65
Adding Dependencies. . . . .	66
Set Up the BINGO Server Build . . . . .	66
Ant RMIC . . . . .	66
Ant RMIC Jar . . . . .	66
Build BINGO in the Standalone Web Client . . . . .	66
Create a New Version Manager Project . . . . .	67
Add Workfiles to Version Manager . . . . .	67
Build BINGO from the Version Manager Integration . . . . .	68
Run the BINGO Game. . . . .	72
Apply Version Labels . . . . .	73
Build BINGO Using a Version Label. . . . .	73
<b>Serana Builder and Configuration Builder . . . . .</b>	<b>75</b>
Overview . . . . .	76
Prerequisites . . . . .	76
Phony Targets and Dummy Targets . . . . .	76
Running Existing Configuration Builder Scripts . . . . .	76
Example: Running a CB Script from the Standalone Web Client. . . . .	77
Start With a Working CB Script . . . . .	77
Set Up Build Directory and Search Paths . . . . .	77
Set Up the Build . . . . .	79
Define Target Detail. . . . .	80
Select Build Options. . . . .	81
Execute the Build Job. . . . .	82
Using CBMACROS, CBTARGETS, and CBFLAGS . . . . .	83
Entering Values Using Predefined Options . . . . .	84
Entering Values Using the Build Commands Field . . . . .	84
Parameter Syntax for CBMACROS, CBTARGETS, and CBFLAGS . . . . .	84
Running a CB Script from a Command Prompt Window . . . . .	85
Converting a Simple CB Script to a Serena Builder Script . . . . .	85
<b>Procedures . . . . .</b>	<b>87</b>
Using the Standalone Web Client . . . . .	88
Setting the Build Directory . . . . .	89
Setting Up a Build . . . . .	90
Defining a Target . . . . .	92
Java Developers: Using Double-Star Wildcards . . . . .	97
Using Options . . . . .	97
Archiving the Target Files. . . . .	98
Using the Version Manager Integration (Desktop Client). . . . .	98

Setting Up the Project . . . . .	99
Building the Project . . . . .	99
Selecting Source Revisions . . . . .	100
Creating a Build Job. . . . .	101
Selecting Targets to be Built. . . . .	102
Specifying Build Parameters . . . . .	103
Submitting a Build. . . . .	105
Removing Version Manager Integration Variables . . . . .	106
Workspace Info Not Used in Version Manager Integration . . . . .	107
Build Directory Embedded into Integrated Build Jobs. . . . .	107
Incorporating an Ant Task into Serena Builder. . . . .	107
Using Remote Build Machines . . . . .	109
Setting Up Remote Build Machines . . . . .	109
Setting Up Remote Builds. . . . .	109
Remote Builds and Permissions. . . . .	109
Creating a Remote Build . . . . .	109
Viewing Reports. . . . .	110
Viewing the Log for a Previous Build . . . . .	111
Impact Analysis . . . . .	111
Viewing Footprint Information . . . . .	113
The Serena Builder for Professional Web Client . . . . .	113
Customizing Build Jobs. . . . .	114
Public vs. Private Build Jobs . . . . .	114
Changing the Build Machine . . . . .	114
Changing the Build Job Name . . . . .	114
Changing Bldmake or Om Flags . . . . .	115
Serena Builder and Cross-Platform Development. . . . .	115
Building from Different Source Revisions . . . . .	115
Building from the Tip . . . . .	116
Building from a Version Label . . . . .	116
Building from a Promotion Group . . . . .	116
Building From Reference Directories. . . . .	116
Using Serena Builder with PCLI . . . . .	117
<b>Troubleshooting. . . . .</b>	<b>119</b>
Checklist. . . . .	120
Troubleshooting. . . . .	121
Tips and Suggestions . . . . .	123
<b>Index. . . . .</b>	<b>125</b>

# Welcome to Version Manager

---

Thank you for choosing Serena® Version Manager™, a powerful and versatile version control system that will revolutionize the way you develop software. Version Manager helps you organize, manage, and protect your software development projects on every level—from storing and tracking changes to individual files, to managing and monitoring an entire development cycle.

Serena® Builder™ for Professional™ integrates the Openmake build management system into Version Manager. Builder provides a way to control your organization's build process, yet is flexible enough to allow building for different groups, different release versions, or different life cycle stages.

With Builder, you can launch builds from within Version Manager, from the web, or from the PCLI command-line.

Purpose of this manual	This manual describes how to use Builder for Professional, including conceptual and "how to" information on basic and advanced Serena Builder for Professional tasks, a summary of the ways to work with Serena Builder for Professional, and instructions for accessing the online help.
For more information	This manual is one of several manuals related to Builder. See <a href="#">"Where to Find Information" on page 11</a> .

## Typographical Conventions

The following typographical conventions are used in the online manuals and online help. These typographical conventions are used to assist you when using the documentation; they are not meant to contradict or change any standard use of typographical conventions in the various product components or the host operating system.

Convention	Explanation
italics	Introduces new terms that you may not be familiar with and occasionally indicates emphasis.
<b>bold</b>	Emphasizes important information and field names.
UPPERCASE	Indicates keys or key combinations that you can use. For example, press the ENTER key.
monospace	Indicates syntax examples, values that you specify, or results that you receive.
<i>monospaced italics</i>	Indicates names that are placeholders for values you specify; for example, <i>filename</i> .
monospace bold	Indicates the results of an executed command.
vertical rule	Separates menus and their associated commands. For example, select File   Copy means to select Copy from the File menu. Also, indicates mutually exclusive choices in a command syntax line.
brackets []	Indicates optional items. For example, in the following statement: SELECT [DISTINCT] , DISTINCT is an optional keyword.
. . .	Indicates command arguments that can have more than one value.
	Shows you which shortcut button to click. Shortcut buttons are placed in the margin.

## Contacting Technical Support

Registered customers can log in to <http://support.serena.com/>.

# Chapter 1

---

## Introduction

What is Serena Builder for Professional?	10
Audience for This Document	10
Serena Builder Products	10
Where to Find Information	11

## What is Serena Builder for Professional?

Serena® Builder™ for Professional™ is a build management system. It provides a way to control the build process, preventing multiple copies of source code from proliferating. Yet, it also provides the flexibility to organize your build process around different teams, different release versions, or different life cycle stages.

Based on the Openmake product from Catalyst Systems Corporation, Serena Builder for Professional provides integration with Version Manager. From the Version Manager desktop client or the Version Manager web client, you can launch build jobs, inspect the results, and repeat the process after modifying your code, without leaving Version Manager.

## Audience for This Document

The audience for this document is the individual developer or other user of Serena Builder for Professional. This user will be creating individual build targets, submitting build jobs for execution, and examining build logs. This user will not be installing the product, and will not be setting up search paths or build types—generally, the Administrator does that before the individual users begin work.

If you are using the Enterprise Edition of Builder, your Administrator can define user groups that allow granting specific permissions, such as the permission to modify search paths. If that is the case and you have such permissions, you can find more detail on setting up search paths in the *Builder Administration Guide*.

## Serena Builder Products

Serena Builder for Professional comes in two forms: Serena Builder and Serena Builder Enterprise Edition.

- Serena Builder allows you to build Windows applications, including pre-defined build types for many popular Windows development tools such as Microsoft .NET and Microsoft Visual Studio.
- Serena Builder Enterprise Edition adds support for Java and for UNIX. In addition, Enterprise Edition allows you to generate impact analysis reports, execute remote builds, and define user groups, roles, and privileges.

Every installation of Serena Builder includes a free trial of the Enterprise Edition. If you have not purchased the Enterprise Edition, Serena Builder will remind you of how many days remain until the evaluation period expires.

In addition to Serena Builder for Professional, there are other Serena Builder products:

- Serena Builder for Dimensions enables you to build applications from within Serena Dimensions.

- Serena Builder for z/OS enables you to build applications on z/OS mainframe UNIX System Services (USS) and native environments.



**NOTE** You should understand that in this manual, Serena Builder for Professional may be informally referred to as just "Builder".

## Where to Find Information

This document describes Serena Builder for Professional from a user's point of view. If you are installing the Serena Builder Knowledge Base server, or if you are the Builder administrator, you may require additional documentation. Here is a table describing where you can find different types of information.

Source of information	Description
Serena Builder for Professional User's Guide	Contains information about the special features of the Serena Builder integration with Version Manager.
Builder Installation Guide	Contains information about installing the Builder servers.
Builder Administration Guide	Contains information about topics such as setting up the servers, creating command scripts, defining build types, and defining search paths.
Builder Development Guide	Contains information about topics such as creating targets, performing builds, and converting makefiles to TGT files.
Builder Getting Started Guide	Contains information about the features and architecture of Builder, and some sample scenarios.
Builder for Dimensions User's Guide	Contains information about the special features of the Builder integration with Serena Dimensions.
Builder z/OS Getting Started Guide	Contains information about using Builder on the z/OS platform.
Builder z/OS Administration Guide	Contains information about managing build types, rules, and projects on the z/OS platform.
Version Manager help	The Builder dialog boxes integrated within Version Manager have help available from the Help buttons or from the F1 key.
Builder web client help	The Builder web client home page contains a link to the online help. This help is produced by Catalyst Systems Corporation.
Builder web client online user's guide	The "i" icon on the Builder web client home page is a link to many different online documents, including a Development Guide and a Getting Started Guide.



**NOTE** The Builder Installation Guide, Administration Guide, Development Guide, and Getting Started Guide are installed with the software (see ["Installing the Command-Line Client" on page 36](#)), and will not be found in the same directory as this user's manual.



# Chapter 2

---

## Concepts

Understanding Serena Builder	14
Serena Builder Components	18
Serena Builder Clients	19
Targets, Files, Projects, and Build Jobs	25
Search Paths	26
Example Serena Builder Workflow	28
Serena Builder vs. the Traditional Build Process	29
Serena Builder vs. Configuration Builder	30
What to Do with Existing Build Processes	30

## Understanding Serena Builder

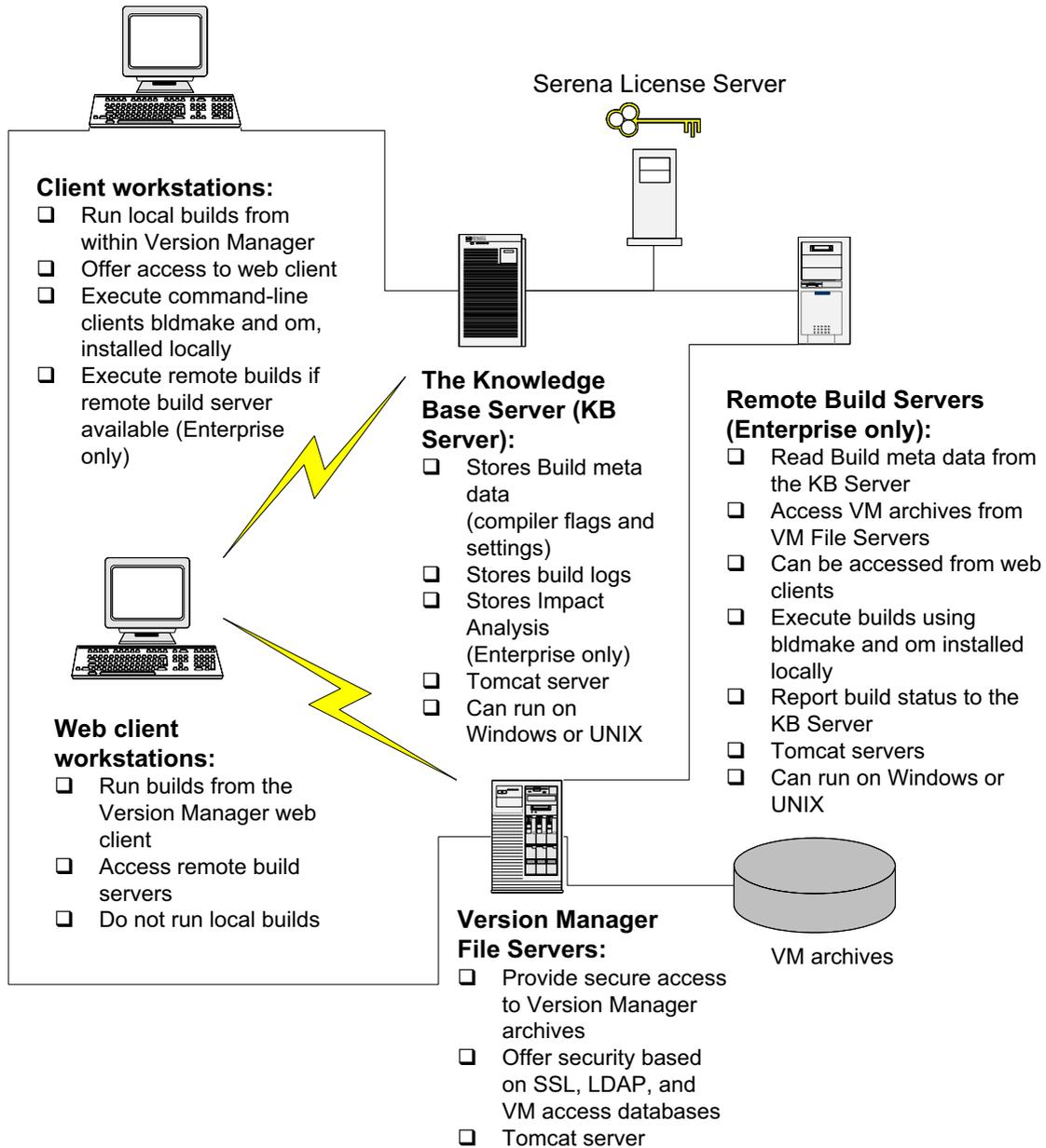
The following diagrams will help you understand Serena<sup>®</sup> Builder<sup>™</sup> for Professional<sup>™</sup>.

The first diagram shows the Serena Builder architecture.

The second diagram shows the variety of Serena Builder clients.

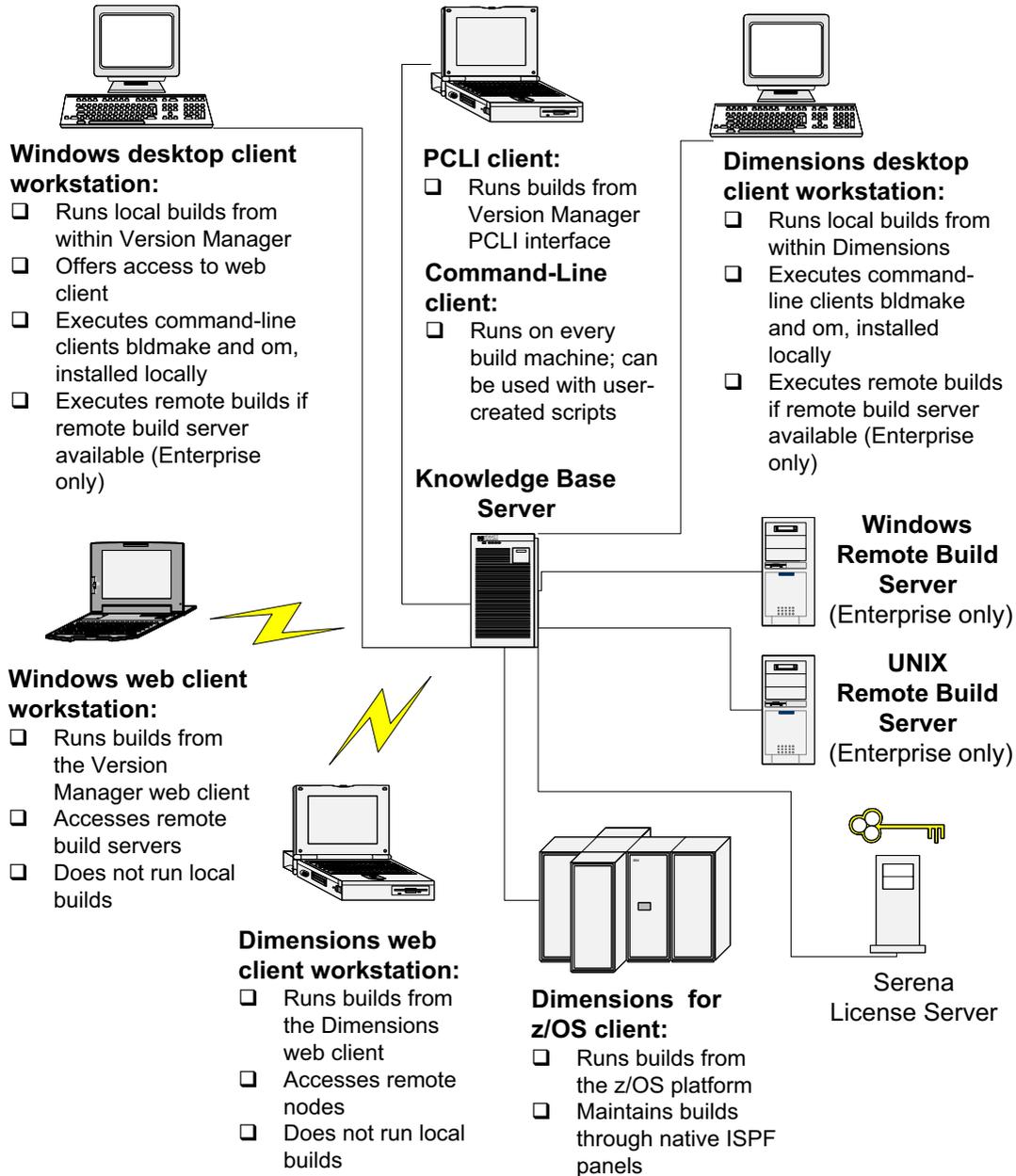
The third diagram shows how Serena Builder can be used with integrated development environments (IDEs).

# Serena ChangeMan Builder Architecture



# Serena ChangeMan Builder Clients

Not shown: Version Manager or Dimensions archives



# Serena ChangeMan Builder with IDEs

Not shown: Version Manager or Dimensions archives

Client Using .NET 2003



Use the .NET integration to allow automatic creation of target definition files from .NET solution or project files. Execute builds from within the .NET IDE.

Client Using WSAD 4.x.x (Eclipse 1.x)



Use the ChangeMan Builder Plug-In for WebSphere Studio 4.x.x to automatically create and update target definition files. Execute builds from inside or outside the IDE.

Client Using WSAD 5.x (Eclipse 2.x)



Use the ChangeMan Builder Plug-In for WebSphere Studio 5.x to automatically create and update target definition files. Execute builds from inside or outside the IDE.

Knowledge Base Server



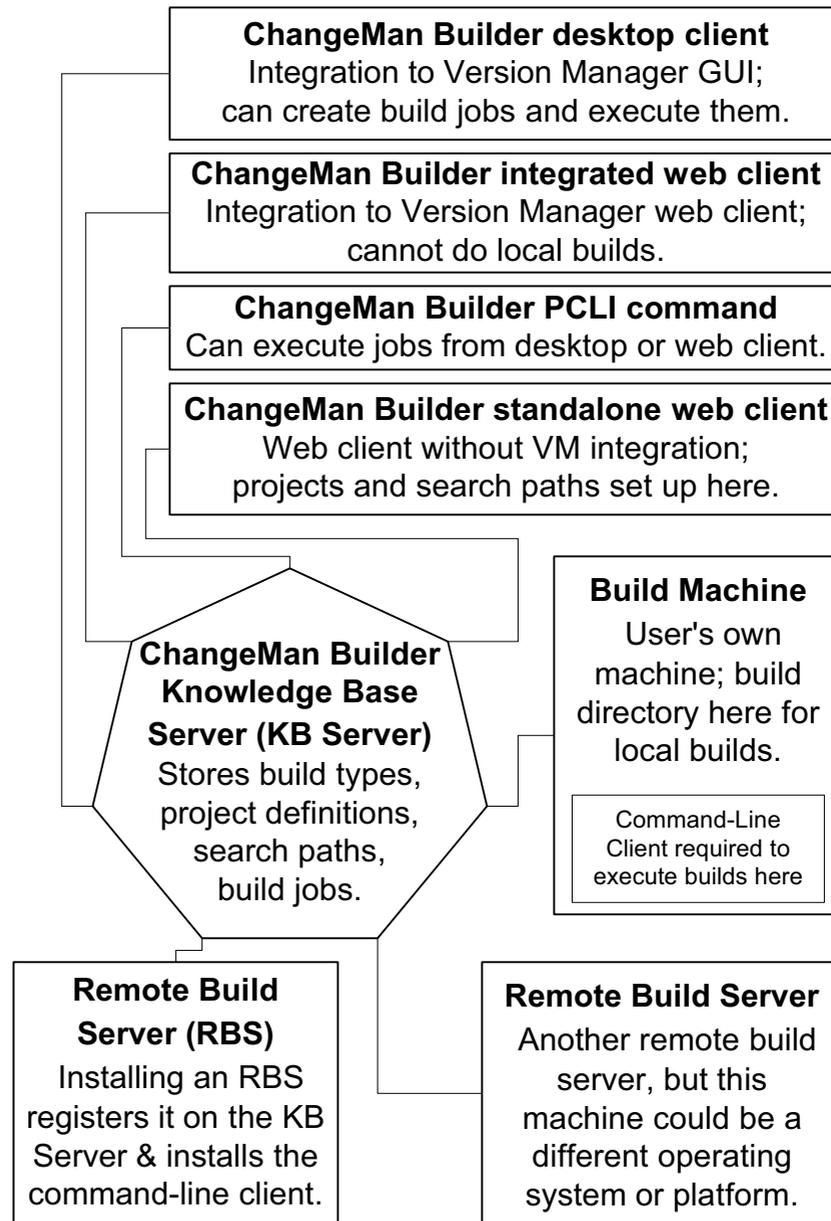
Remote Build Server (Enterprise only)



Serena License Server

# Serena Builder Components

Serena® Builder™ components:



The preceding diagram illustrates several important Serena Builder concepts:

- Serena Builder has many different clients with which you can interact—clients integrated into Version Manager, a PCLI client, and a standalone web client. Each client has slightly different capabilities—for example, the standalone web client is the only client in which you can define build dependencies. Each client will be discussed more fully later.
- Serena Builder has a central Knowledge Base server. This server stores the data that identify each build job. The KB server also stores data collected as a result of the build, and contains information that you want to be uniform across your group or

enterprise, such as definitions for the types of build jobs that can be compiled, or what to do with a `.java` file versus a `.cpp` file.

The Serena Builder administrator sets up the Knowledge Base server, creates build projects and search paths, and does other work necessary to make the KB server transparent to the individual developers.

- Remote build servers are optional machines set up to be available on the network for performing builds. After the Serena Builder administrator installs the remote build server on a given machine, that machine is registered with the Knowledge Base server, which then makes it visible to others on the network. Users can then specify that a build job should execute there.
- Build machines that are not remote build servers are typically individual users' machines. Users can perform a local build on their own machines, as long as they have installed the command-line client.
- The command-line client is a special client that goes on every build machine, remote or local, but is generally not used directly. Composed of the `bldmake` and `om` executables, the command-line client creates a makefile from your build job, then executes the makefile. Normally, however, you interact with it through one of the other clients. (It is sometimes useful to call `bldmake` and `om` directly, for debugging purposes. See [Chapter 8, "Troubleshooting" on page 119](#)).

## Serena Builder Clients

As mentioned earlier, Serena Builder for Professional offers several different clients for you to use:

- Builder standalone web client  
This client is available from any web browser, and is not integrated into Version Manager.
- Builder desktop client  
This client is integrated into the desktop version of Version Manager.
- Builder integrated web client  
This client is integrated into the web version of Version Manager.
- Builder PCLI client  
This client is available from the Version Manager PCLI interface.
- Builder command-line client  
This client is installed on all build machines and on machines that wish to use the PCLI client. You do not generally interact with this client directly, but it is necessary to perform builds.

All of these clients communicate with the central Knowledge Base server; each has slightly different capabilities. A discussion of each client follows.

## Serena Builder Standalone Web Client

The Serena Builder standalone web client is available from a web browser such as Microsoft Internet Explorer.



**NOTE** In this book the client is referred to as the standalone web client in order to distinguish it from the web client that is integrated into the web version of Version Manager.

The client has a central display area, a set of icons along the bottom for operations such as Back, Undo, and Save, and a menu of commands along the right.

The menu of commands contains three headings:

- Administration
- Development
- Information

The Administration menu items permit the Serena Builder administrator to modify build types, to create projects, to manage users, and, in the Enterprise Edition, to manage groups. These tasks are described in the *Builder Administration Guide*.

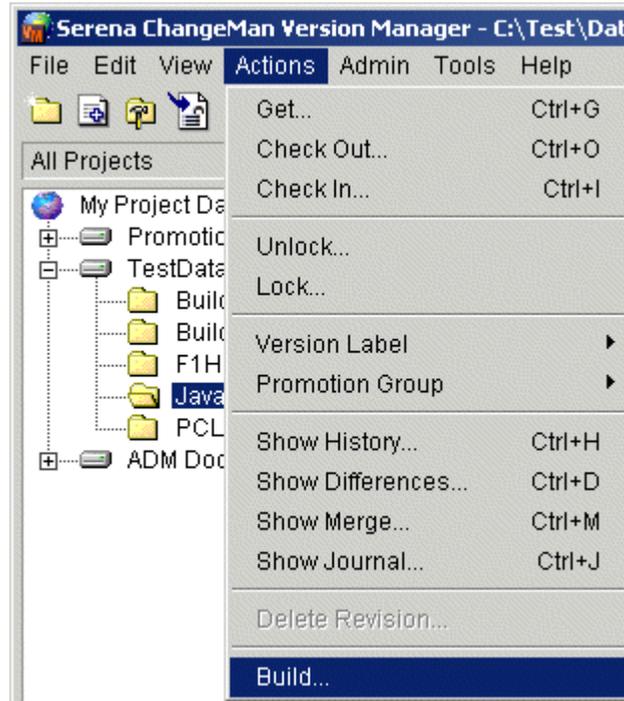
The Development menu items permit you to set up a build, to define build dependencies, to execute or modify existing build jobs, to view reports from completed builds, and to set the directory where the build should be executed. This is the only place where you can set up a build; it is not available from any of the other clients.

The Information menu items offer information about the version of Serena Builder you are using, on-line help, and an on-line User's Guide (a set of documents different from the one you are reading). The command to install the command-line client is also under this menu.

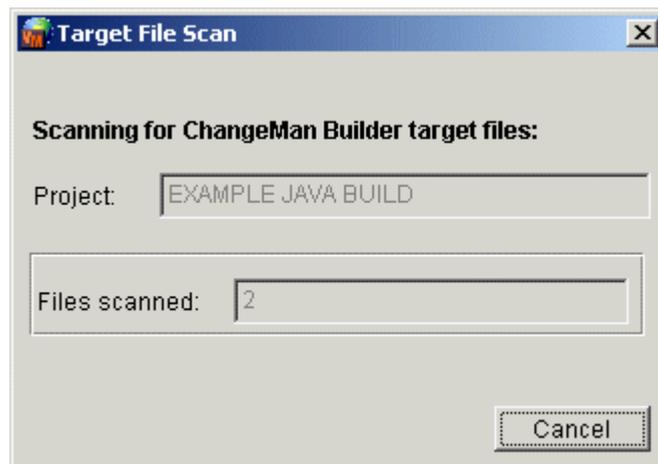
Examples of using the standalone web client appear in [Chapter 4, "Tutorial"](#) and [Chapter 7, "Procedures"](#).

## Serena Builder Desktop Client

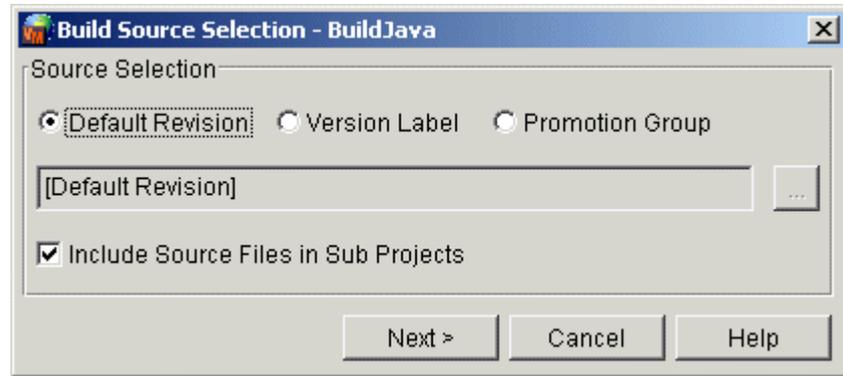
The Serena Builder desktop client is available from the Actions menu of the Version Manager GUI:



Once you select the Serena Builder menu item, Builder scans the selected Version Manager project, looking for files that indicate that a build job has been set up for that Version Manager project:



Builder then presents you with a dialog asking how you would like to select source files:



From this client, you can execute previously-defined build jobs, give new names to existing build jobs, change from an incremental to a clean build, and do other tasks necessary in the edit-compile-test cycle.

Examples of using the desktop client appear in [Chapter 4, "Tutorial"](#) and [Chapter 7, "Procedures"](#).

### ***Differences: Standalone Web Client and Desktop Client***

Using the standalone web client and the desktop client differ in some areas:

- All projects must be set up in the standalone web client, as you cannot set up projects in the desktop client.
- In the standalone web client, you must set a build directory and search path so that Builder can locate your source files.
- In the integrated or desktop client, the `-s` switch for case sensitivity is always on.
- In the integrated or desktop client, you check out your source files to a directory of your choosing, then tell Builder where to build by choosing the build directory from the Build Parameter dialog box.
- When you save a build job from the desktop client, it must be a private build job (see [page 26](#)). You can run a public build job from the desktop client, but when you save a build job, it is always saved as a private build job.
- If you wish to run a build job using PCLI, you must have created that build job in the desktop client.

## **Serena Builder Integrated Web Client**

The Serena Builder integrated web client is available from the icons in the web version of Version Manager:



The integrated web client, like the desktop client, begins by scanning the files in the selected Version Manager project. It then shows you a dialog box asking you how source files should be selected:

From this client, you can perform any task that you can perform in the desktop client, with two exceptions:

- You cannot browse for version labels or promotion groups.
- You cannot execute a local build.

## Serena Builder PCLI Client

The Serena Builder PCLI client is called from the PCLI command-line, in a Command Prompt window:

```

C:\> Command Prompt
C:\>pcli build -h
Serena ChangeMan Version Manager (PCLI) v8.1.0.0 (Build 739) for Windows NT/80x86
Copyright 1985-2004 Merant. All rights reserved.
Build - Submit the specified ChangeMan Builder job.
Syntax: Build [<options>]
        The project database and project name which were used to create the
        Build Job must be specified in order to verify access
        to the build job.
Options:
-bj<job_name>      Existing ChangeMan Builder job name.
-h                Displays help for the command. The action terminates
                  after it processes the -h option even if you specify
                  other options.
-id<user id>[:<user_password>]
                  Specifies a user ID and password if necessary. This
                  option is valid only for project databases and projects
                  that have the ULOGIN (Login Dialog) login source
                  specified and an access control database enabled. In
                  this case, you must specify a value for this
                  option.
-pp<project_path> Specifies the project or folder to be used.
-pr<project_database>
                  Specifies the location of the project database. A project
                  database must be specified with either this option or
                  the PCLl_PR variable.
-sp/<@/<user ID> ! Public/<parent_workspace>/<child_workspace>
                  Specifies a public or private workspace. Note
                  that user IDs are case-sensitive. To specify the Root
                  Workspace, enter /@/RootWorkspace for the workspace
                  value. If a workspace is not specified the user's default
                  workspace is used.
C:\>

```

From this client you can execute any build job that was previously defined in either the desktop client or the integrated web client. You cannot execute a build job that was defined solely on the standalone web client; nor can you define a new build job using the PCLI client alone.

## Serena Builder Command-Line Client

The Serena Builder command-line client is the name used to refer to two executables that perform the work of creating makefiles and executing them. These executables are `bldmake` and `om`. While you generally will not use `bldmake` and `om` directly, it is useful to know of them, because you will see them listed in the Build Commands field that is visible when you submit a build job, and in the build log that appears after a build job has been submitted.

Also, there are various options to the commands that can provide debugging or footprinting information. The options to `bldmake` and `om` are described in the *Builder Development Guide*.

## Advantages of the Integrated Clients

The integrated clients have the advantage of automatically retrieving source and target definition files (.TGT files—see ["Targets, Files, Projects, and Build Jobs" on page 25](#)) from Version Manager archives into the specified build directory.

The standalone web client expects you to have placed all source files and .TGT files into the build directory before submitting the build.

## Summarized Differences Between Clients

Here is a table summarizing the different capabilities of each of the Serena Builder for Professional clients.

Type of Client	Define Search Paths?	Set Up Projects?	Create Build Jobs?	Submit Build Jobs?
Desktop client	No	No	Yes	Yes
Version Manager web client	No	No	Yes	Yes (cannot run local builds)
Standalone web client	Yes	Yes	Yes	Yes
PCLI client	No	No	No	Yes (if created in desktop client)
Command-line client (normally not interacted with directly)	No	No	No	No

Essentially, you need to set up search paths and projects in the standalone web client, and from there you can use any of the other clients except the PCLI client, which can only submit previously-defined build jobs.

---

# Targets, Files, Projects, and Build Jobs

All of the Serena Builder clients and servers described earlier exist for the purpose of generating or executing build jobs. Build jobs contain the definitions of which files and libraries make up the target—the thing being built. Here are descriptions of the concepts at this level.

- Target

The target is the desired result of the build action. For example, in a *project* containing a set of source files, header files, and libraries, the target is the software being built, such as an .EXE file. Defining a target means specifying what the final product is, and what *dependencies* are necessary to create that product.

- Dependency

A dependency is a file or other item that is used to create a *target*. Any source files mentioned in a *target definition file* are examples of dependencies, because the final target depends on their existence and proper compilation.

- Target definition file (.TGT file)

A target definition file, or .TGT file, is the file that Builder creates to store the definition of what the target is.

Some parts of the Builder documentation refer to a .TGT file as a target file, but do not confuse a .TGT file with a target. The .TGT file defines what the target is.

- Search path

The search path is a set of directory locations that Builder uses to locate files needed to complete the build. These files are usually source files, but could also be third-party libraries. Setting up search paths is usually the task of the Builder administrator.

By default, every search path includes a "." to represent the local build directory. Otherwise Builder will not be able to find the intermediate products of the build (such as an .obj file).

Once a local copy of a file is found, Builder stops looking for that file. All other files are retrieved from the search path, making it unnecessary for the developer to check out the entire development tree. This avoids source code duplication and proliferation.

Builder uses search paths for every build job, so it is critical to understand how they work, and to set them up properly. See "[Search Paths](#)" on page 26.

- Project

A Builder project is a named collection of search paths that define where source files are to be found. It does not include the target definitions, nor the source files themselves. Because the project only contains search paths, individual developers can define their own targets.

- Build types

A build type defines what type of compiler, source files, and platform Builder should use. An example of a build type is "Windows - Visual Basic", indicating that the platform is Windows, and the compiler to be used is Visual Basic.

Build types appear on the Setup Build page and serve to indicate how a given target will be built. The Builder administrator can define custom build types if necessary, but most build types are predefined and it is very likely that no custom build types will be necessary.

- Build tasks

A build task is a logical stage within a build. For example, a Java Jar build project would have the build tasks Set Classpath, Ant Javac, and Ant Jar. The Set Classpath task corresponds to pre-compile activities, the Ant Javac task corresponds to the compile operations, and the Ant Jar task corresponds to the .jar file packaging operations.

- Build rules

A build rule defines what steps a given compiler should take to build the target, given a known type of input file and a known type of output file. For example, a build rule could specify that a .c file should be compiled to an .obj file by the C compiler.

- Build directory

The build directory is where Builder will place the final products of the build. By default, Builder looks for source files here, and also in directories named in the search path. Specifying the build directory is one of the first things an individual developer should do.

- Build job

Build jobs are the saved specifications of the build directory, targets, build options, build machines, and other parameters that determine how targets are to be built. The build jobs appear on the Execute Build Jobs page.

Build jobs can be either public, meaning they are visible to and executable by everyone, or private, meaning that they are only visible to and executable by the build job's owner.

## Search Paths

Each project in Serena Builder is associated with a search path. A search path is a collection of directory paths. Just as the PATH variable tells your operating system where to find applications, the search paths tell Builder where to find source files or other needed files such as third-party libraries.

### **Different Search Path = Different Build Target**

Projects in fact are commonly associated with multiple search paths. Each search path defines a different build deliverable. A project may have one search path for developers and another search path for quality assurance, or one search path for a Windows platform and another search path for UNIX.

### **Search Paths Look in Current Directory First**

By default, each search path in Serena Builder begins with ".", meaning that the current directory is to be searched first.

## Serena Builder Uses First-Found Copy of File

Once Serena Builder finds a copy of a file, it stops looking for that file name. This means that a developer's local copy of a source file can take precedence over the copy checked in to the release area, allowing for testing of changes.

For example, when an individual developer wants to test his own `accounting_module.cpp`, Builder finds the local `accounting_module.cpp` and builds the project with that, ignoring the checked-in version found elsewhere in the search path directories.

So while it is true that search paths generally are set up by the Builder administrator, individual developers can still test their files locally by defining local build directories and placing their test files there.

## Search Path Example

Consider a hypothetical project named Project1 that needs to be built by the developers, by quality assurance, and by a release engineer. Developers want to test their changes before passing them on to QA. QA needs to build their own debug versions, whereas the release engineer must build only what will be shipped to customers.

In this situation, the search paths might be defined as follows:

Search path	Defined as
DEV	.
	\$project1\dev\src
	\$project1\dev
	\$project1\qa\src
	\$project1\qa
	\$project1\release\src
	\$project1\release
QA	.
	\$project1\qa\src
	\$project1\qa
	\$project1\release\src
	\$project1\release
RELEASE	.
	\$project1\release\src
	\$project1\release

Notice the following:

- Each search path begins with ".", meaning that Builder will search the current directory first.
- The released version of the product (using the search path RELEASE) will be built only from files in the `release\src` or `release` directories.

- Quality Assurance can test proposed changes by placing those changes in the qa\src or qa directories and then building with the QA search path. This is possible because a) the order of directories listed is significant, and b) **Builder will stop looking for a file once it finds one copy**. This allows local copies of a file to supersede the "final" checked-in versions in the RELEASE area.
- Developers can test their changes by placing local copies of their changed files in the dev\src or dev directories and then building with the DEV search path. They do not need to check out the entire build tree, because Builder will automatically look for the remaining files in the other directories named in the search path. This allows individual developers or QA engineers to test locally, yet still be working with essentially the same source tree, rather than multiple copies of the source tree.
- Builder also uses the search path to locate .jar files, .zip files, and other third-party auxiliary files. It does not use the search path to locate compilers—those are located using the standard PATH variable.

## Search Paths Can Contain Environment Variables

Another way to allow for variation in search paths is to use environment variables. If a search path contains an environment variable such as COMPILER, JAVA\_HOME, or REFDIR, it can be modified by each developer as needed—to account for different drive mappings, for example.

See the example projects provided with Builder for more examples of how to use search paths.

## Admin Controls Permissions to Modify Search Path

If it is essential for members of your group to modify the search paths, the Builder administrator can grant that privilege, provided that you are using the Enterprise Edition of Serena Builder for Professional. See the *Builder Administration Guide*.

## Example Serena Builder Workflow

Here is a description of a typical workflow for Serena Builder; this will be helpful when you come to the specific instructions on how to use the product.

- 1 The administrator installs the product and sets up the Knowledge Base server. This machine stores the configuration data such as build types, projects, search paths, and build jobs.
- 2 The administrator examines the predefined build types that come with Builder. These include many popular development environments, such as .NET, Visual Studio, Ant Java, and so on. If necessary, the administrator modifies the command scripts and build rules associated with each build type. It is possible that no customization is needed.

When the administrator is done, there are a set of build types such as "Windows MSVC Application Static" that are ready to be used by the developers.

- 3 The administrator sets up the needed projects. This includes defining search paths for each project. Web client users in particular need to do this, as the search path

determines where to locate source files, and in the case of Java programs, where to locate .jar or .zip files.

- 4 The administrator sets up groups, users, and privileges.
- 5 The individual users create their accounts by logging in to the web client, then installing the command-line client.
- 6 The users set a build directory to identify where the products of the build should be placed, or the users accept the default build directory.
- 7 The users set up their builds by defining targets (the end products of a given build) and by specifying dependencies (the files that are used to create the target). The users also specify which operating system and compiler shall be used to build each target.
- 8 The users save all this information into a build job, which they can submit immediately, or save for later. Users can execute their build jobs from the web client, the desktop client, or from PCLI.
- 9 When the build completes, the users review the build log. If the Enterprise Edition is installed, users may also review the Impact Analysis reports.

## Serena Builder vs. the Traditional Build Process

Serena Builder is designed to avoid problems inherent in the traditional build process:

- Variations in compiler flags and configurations
- Multiple copies of source code
- Reliance on only one or two people who can execute the build process, using complex build scripts
- Lack of cross-platform support

### Advantages of Serena Builder

Builder improves on the traditional build process in the following ways:

- Standardization of compiler flags

The Builder administrator can control how compilers are configured and which flags are used. While individual developers can still specify debug flags and release flags, can still have a choice of compilers, and can still pass special flags to the compilers, the Builder administrator sets the default compiler flags and defines which versions of each compiler are to be used. This avoids launching builds that have the wrong compiler version or have wrong compiler settings.
- Standardization of third-party libraries

Because the Builder administrator controls the search path used to locate third-party libraries, proliferation of multiple copies of libraries is reduced or eliminated.
- Source code is always in known locations, yet allows individual developers to test locally

Because Builder will stop looking for a specific file once a copy is found, an individual developer can make changes to a small set of files, then trigger a build that uses his local files and the official release files for everything else. This reduces the multiple local copies of source code that rapidly grow out of date.

- Standardized build scripts allow anyone to build

Once a Builder project has been set up with the proper build types, project settings, and search paths, these values are stored on the Builder Knowledge Base server as a specific build job. If necessary, anyone can execute a public build job.

- Cross-platform support

Builder can build products that are constructed partially on one platform and partially on another. The server component of a product, for example, might be built on a UNIX machine, while the client is built on a Windows machine.

Furthermore, the creator of a build job may define a job on a Windows machine that will run on a UNIX machine, using a remote build capability (Enterprise Edition only).

## Serena Builder vs. Configuration Builder

If you have used Merant Configuration Builder, you should be aware of a difference between Serena Builder for Professional and Configuration Builder:

- Configuration Builder can access Version Manager archives and can examine the revision dates of archives, for example, to determine whether a file needs to be checked out.
- The desktop client version of Builder can access Version Manager archives, but the web client version of Builder cannot open Version Manager archives, and expects to find all necessary files either in the build directory or in directories specified by the Search Path.

## What to Do with Existing Build Processes

If you have existing build processes, you have the following choices:

- Continue to use your existing build system
- Use Builder to create a build job that encapsulates your existing build scripts (that is, use a wrapper script)
- Migrate your existing Configuration Builder or pcsmake build scripts to Builder
- Create new build jobs using Builder

# Chapter 3

## Installation

---

This section discusses the installation of Serena® Builder™ for Professional™. Most of the installation steps should have been done for you by the Builder administrator, but some of the steps you have to do.

Installation Tasks for Serena Builder for Professional	32
About Supported Platforms	32
Installation Steps Done by the Administrator	32
Installation Steps Done by You	34
Installing the Command-Line Client	36
Remote Build Server Licensing	37
Licensing	37
UNIX: Adding Setup Information to Accounts	38
Upgrading from a Merant Build Installation	39

## Installation Tasks for Serena Builder for Professional

Although this is not a substitute for the installation and administration guides, here is a table of the Serena Builder installation tasks that must be completed, and where the instructions for those tasks can be found.

Task	Responsible	Where described
Install Serena Builder servers	Administrator	Builder Installation Guide
Customize Build Types and Build Rules for the compilers you use	Administrator	Builder Administration Guide
Define search paths	Administrator	Builder Administration Guide
Define build types	Administrator	Builder Administration Guide
Command-line client must be installed	You	This document (see <a href="#">"Installing the Command-Line Client" on page 36</a> )
Create account by opening a browser session to standalone web client for the first time	You	This document (see <a href="#">"Installation Steps Done by You" on page 34</a> )
Create target files	You	This document (see <a href="#">"Setting Up a Build" on page 90</a> )
Add target files to Version Manager archives	You	Version Manager User's Guide
Define build job and submit it	You	This document

## About Supported Platforms

Serena Builder for Professional supports the same platforms that are supported by Serena PVCS Version Manager. You can find the most recent list of supported platforms in the Version Manager Readme.

## Installation Steps Done by the Administrator

Here are the steps that the Builder administrator should do in order to get Builder running. These steps are described in greater detail in the *Builder Installation Guide*, the *Builder Administration Guide*, and in Version Manager documentation, but this list should give you a sense of what needs to be done and who should do it.

- 1 Install the Builder Knowledge Base server. See the *Builder Installation Guide* and also check the Readme for the latest information.

On UNIX, be sure to capture the information that appears at the end of the install transcript; this needs to be added to the `.profile` or `.cshrc` file of the account that

will run the Knowledge Base server. See ["UNIX: Adding Setup Information to Accounts" on page 38](#)

- 2 If the Knowledge Base server is also to be used as a build machine, the command-line client needs to be installed. Install the command-line client from the Install Client link on the web client home page.
- 3 Define the build types and build rules necessary to use the compilers needed at your company. See the *Builder Administration Guide* for instructions.
- 4 Create Builder projects corresponding to the projects at your company.
- 5 If you are using the web client, define search paths to the locations where source code for the projects can be found. (If you are using the integrated client, this is not necessary.) Again, see the *Builder Administration Guide* for instructions.
- 6 Set up users, groups, and privileges.
- 7 Once you have local builds working properly, set up remote build servers if desired. On UNIX, be sure to capture the information that appears at the end of the install transcript; this needs to be added to the `.profile` or `.cshrc` file of the account that will run the remote build server. See ["UNIX: Adding Setup Information to Accounts" on page 38](#)

## Remote Build Server Issues

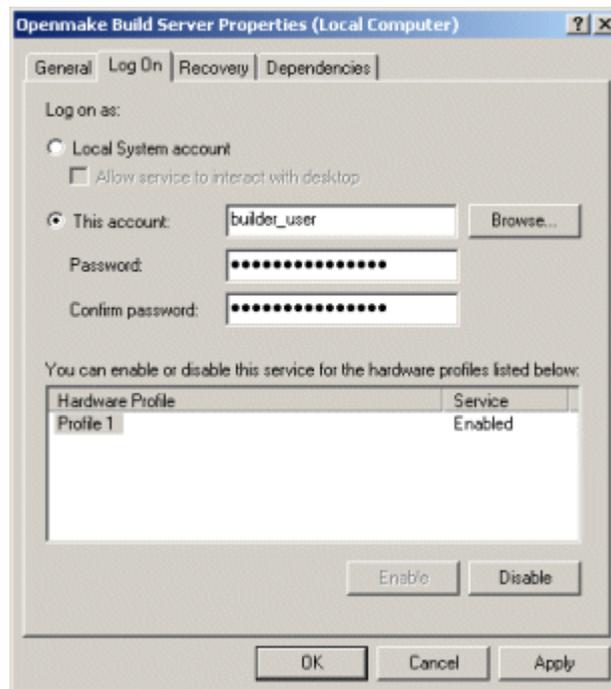
Following are two issues related to remote build server installation. For the main set of instructions on how to set up a remote build server, see the *Builder Administration Guide*.

### ***Remote Build Server Service Must Run Under "This Account"***

If you choose to install a remote build server on a Windows machine, a service named Openmake Build Server is installed on that machine.

It is important that this service not be run under "Local System Account". Instead, it should be run under a specific user that you create.

After creating the user for the remote build server, go to the Properties dialog box for the Openmake Build Server service and specify the user under **This account**:



### ***Drive Mappings Can Be Lost if User Logs Out***

The previous section established the need for a user ID under which the remote build server could run. You should be aware that if that user logs out—for example, inadvertently, because of a system crash—the drive mappings under that user could be lost. This can cause a loss of access to a Version Manager project database (PDB) that Builder needs.

If this happens, log the remote build server back in as the user ID under which the proper drive mappings have been defined.

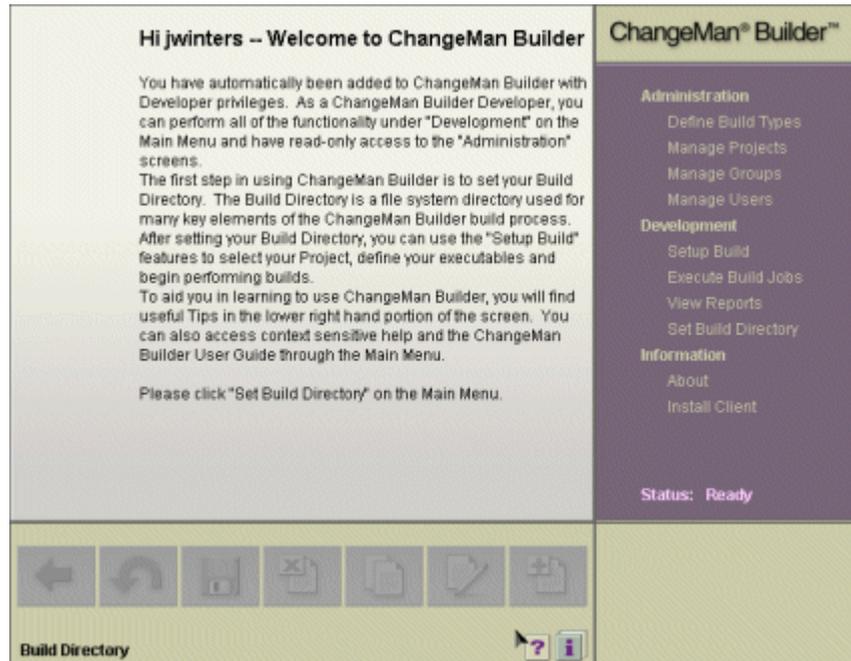
## **Installation Steps Done by You**

- 1** From your Administrator, get the DNS host name of the Knowledge Base server, and the port number to the server (default 58080).
- 2** Using the information from step 1, open a browser session using this URL:  
`http://<KB server DNS host name>:<port number>/openmake/index.html`
- 3** If you do not have the Java runtime client installed, you will see dialog boxes asking if you wish to install. Follow the installation instructions.

- 4 When the Java runtime client install completes, you see a dialog box asking if you wish to accept the Catalyst security certificate:



Click **Always**. A welcome page appears:



- 5 Click **Set Build Directory** as directed, and set a suitable location for placing builds (this can be easily changed later).
- 6 If you should have Administrator privileges, contact your Builder administrator to request them.
- 7 Return to the web client home page, and click **Install Client** to install the command-line client. See "[Installing the Command-Line Client](#)" on page 36. (On Windows machines, you will be asked to reboot after installing.)
- 8 On UNIX, be sure to capture the information that appears at the end of the install transcript; this needs to be added to the `.profile` or `.cshrc` file of the account that

will run the command-line client. "[UNIX: Adding Setup Information to Accounts](#)" on [page 38](#).

## Installing the Command-Line Client

The installation of Serena Builder is primarily carried out by the Builder administrator, but end-users must install the command-line client. Installing this client allows you to execute builds locally.

For both Windows and UNIX, you must install Perl separately. The Windows installation no longer installs Perl.



**NOTE** You can find ActiveState Perl, including a link to the ActivePerl documentation, at <http://www.activestate.com>.

For Windows, after installing the command-line client, you will be asked to reboot your computer.

To install the command-line client:

- 1 On the Builder standalone web client home page, click **Install Client**. You are taken to a page containing instructions for installing the client, and links to the downloadable install file. (Or, you can use the installation CD.)
- 2 Print the install instructions, and then download the install file.
- 3 Execute the install file and follow the instructions in the install wizard, and in the printed install instructions.
- 4 (Windows) When the install completes, reboot your computer.



**NOTE** If you install a command-line client after permanent licenses have been installed (for a new user, for example), you will have to convert the temporary license installed by default to a permanent license. You can do this by editing the value of the `MERANT_LICENSE_FILE` environment variable. See "[Command-Line Client Permanent Licenses](#)" on [page 37](#).

## Who Needs to Install the Command-Line Client

You must install the command-line client on the following machines:

- Any machine that will be performing local builds
- The machine hosting the Version Manager web server
- Any machine using Builder from PCLI

It is not necessary to install the command-line client on the Knowledge Base server, unless that machine will also be used to perform builds.

Remote build servers need the command-line client also, but the process of installing a remote build server automatically installs the command-line client.

# Remote Build Server Licensing

To set up a remote build server, you must have a remote build server named license. These licenses can be ordered through your Serena sales representative or through web fulfillment and are expressly designated for remote build server machines. Without one of these named licenses, you will not be able to set up a remote build server, regardless of the number of other Builder licenses you have purchased. This licensing arrangement differs from previous Serena Builder for Professional releases (that is, from the initial Serena Builder for Professional release).

Note that the remote build server license enables the remote build machine to be accessed from within the Version Manager integration, but does not license Version Manager for general use from the remote build server machine.

To install the remote build server license, follow the instructions in the *Serena Version Manager Installation Guide* or the *Serena License Manager QuickStart Guide*.

## Licensing

Serena Builder automatically installs an evaluation version of the Enterprise Edition, good for thirty days without a permanent license. After that period, you or your Builder administrator will have to take some steps to upgrade the temporary licenses to permanent licenses. The steps for doing this are slightly different depending on which component of Builder you are upgrading. Here are the steps for all components.

### Knowledge Base Server Permanent Licenses

Upgrading the Knowledge Base Server license is usually done by the Builder Administrator.

- 1 Get a permanent license key using web fulfillment or by calling Support. This procedure is detailed in the *Version Manager User's Guide*.
- 2 Install this key on the Serena license server.
- 3 Edit the `license.ini` file on the Knowledge Base server so that it points to the Serena license server. To do this, you replace the line in `license.ini` that begins like this:

```
http://<name of machine>...
```

with a line that looks like this:

```
@<name of license server>
```

where `<name of license server>` is replaced by the name of your Serena license server.

### Command-Line Client Permanent Licenses

Upgrading the command-line client license can be done by the individual user, assuming the permanent license key has been installed on the Serena license server.

To upgrade the command-line client license: Edit the value of the `MERANT_LICENSE_FILE` environment variable so that it points to the Serena license server. To do this, you replace the value of the environment variable, which begins like this:

```
http://<name of machine>...
```

with a line that looks like this:

```
@<name of license server>
```

where `<name of license server>` is replaced by the name of your Serena license server.

## Remote Build Server Permanent Licenses

Upgrading the remote build server licenses is usually done by the Builder administrator. The procedure is the same as that described for upgrading command-line client licenses: Edit the `MERANT_LICENSE_FILE` environment variable so that it points to the Serena license server.

## PCLI Licensing Tied to Version Manager

It is unnecessary to separately license the Builder command that is callable from PCLI. The PCLI commands are tied to Version Manager licensing.

# UNIX: Adding Setup Information to Accounts

UNIX installations need to capture setup information, such as environment variable definitions, to individual users' accounts, so that the proper settings are in place when the users log in. To accomplish this, the Builder administrator should follow these steps:

- 1 At the end of a UNIX install, environment variable information that appears at the end of the install needs to be sourced into individual users' environments. The Builder administrator should copy this information into a separate file—for example, `buildervars`. Then, for each user, the administrator should add to the `.profile` or `.cshrc` file a source command calling the `buildervars` file.
- 2 The Builder administrator also should source the appropriate file listed below to each `.profile` or `.cshrc`:
  - If you are using Bourne shell, Korn shell, or bash, source the `vmprofile` file.
  - If you are using C shell, source the `vmcshrc` file.

The `vmprofile` and `vmcshrc` files are in the `/bin` directory of your Version Manager install; that is:

```
<install location>/vm/<platform>/bin
```

The default install location is `/usr/serena`.

# Upgrading from a Merant Build Installation

If you upgrade from a Merant Build installation, you will have to consider three migration issues:

- How to preserve the build jobs and projects you have created with Merant Build
- How to convert the TGT files from those build jobs to the Serena Builder TGT file format
- How to adapt build jobs to changes in build types

## Preserving Merant Build Projects and Build Jobs

The default installation directory for Merant Build was:

- Windows: C:\Program Files\Merant\Build
- UNIX/Linux: <USER\_HOME>/Merant/Build

The default installation directory for Serena Builder is:

- Windows: C:\Program Files\Serena\ChangeMan\Builder
- UNIX/Linux: <USER\_HOME>/Serena/ChangeMan/Builder

If you install Serena Builder into the default directory, your existing build jobs and projects will not be moved to the new installation.

If you wish to preserve the existing projects and build jobs, follow this procedure:

- 1 On the Knowledge Base server of the existing installation, create the following directory structure:
  - Windows: C:\Program Files\Serena\ChangeMan\Builder\tomcat\webapps\openmake.ear
  - UNIX/Linux: <USER\_HOME>/Serena/ChangeMan/Builder/tomcat/webapps/openmake.ear

These directories will be filled in by the Builder install program, but you will move over the build job and build project information so that it will be preserved.

- 2 From the existing installation, copy this directory
 

```
<install dir>\tomcat\webapps\openmake.ear\openmake.war
```

 into the new directory structure you just created.
- 3 Install Serena Builder into the new directory structure.

The existing projects will appear in the new Knowledge Base server.

## Converting Merant Build TGT Files to Serena Builder TGT File Format

If you try to execute a build job from a project that has been migrated to the new Knowledge Base server, the build job will likely fail to execute, and you will see a build log similar to this:

Serena ChangeMan Builder powered by Openmake(R) V6.3 Build 101504,  
Copyright 1995-2004

```
Application BUILD_JAVA from DEVELOPMENT for all targets.  
Search Path=.;C:\j2sdk1.4.2\jre\lib;  
ERROR 390: Could not find Build Task for dependency HelloWorld.java  
ERROR 49: Could not convert target file  
          C:\BuildJava\Test2\HelloWorld.class.tgt to 6.3 format. Please  
          convert the target file from the Web Client.  
ERROR 42: Could not find any targets in the Search Path.  
Done loading target lists.
```

Builder will attempt to convert some TGT files for you, during the execution of a given build job. If Builder is not able to convert the TGT file, then Error 49 appears and you need to convert the TGT file.

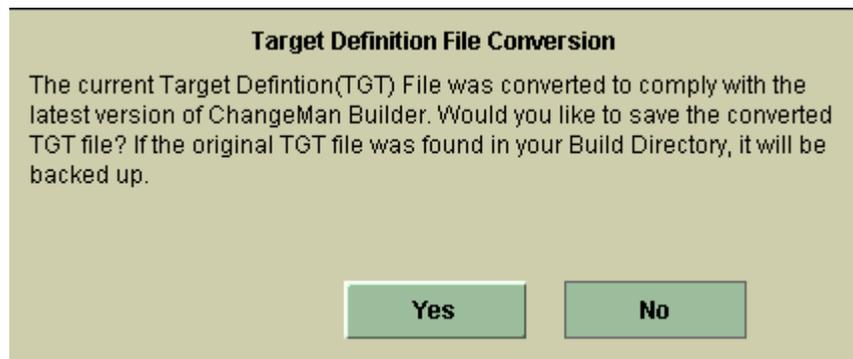
### **Batch Convert TGT Files**

If you have many TGT files to convert, you can use a batch conversion utility available in the omcmdline.jar file. See the section "Converting Target Formats" in the *Serena Builder Development Guide*.

Afterwards, you should inspect the batch-converted TGT files in the standalone web client.

### **Convert Files in the Standalone Web Client**

For an individual build job, Builder will automatically convert the TGT file of the build job when you open the Target Definition page for a given target. Builder then displays the following message:



If you click No, the converted TGT file is not saved, and you will not be able to execute the build job.

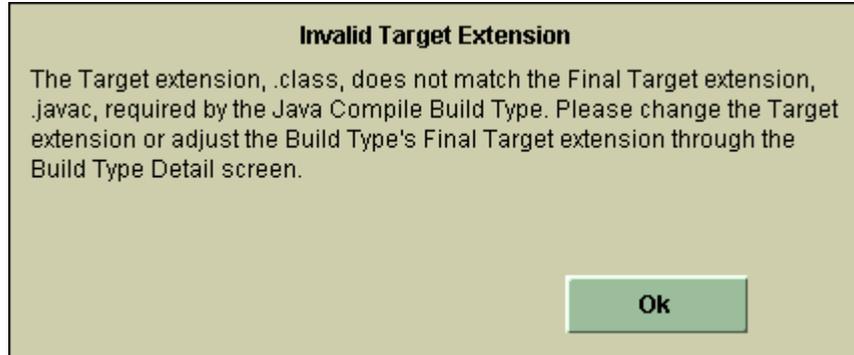
If you click Yes, the converted TGT file is saved, and the old TGT file is saved as <old filename.tgt>.pre63. However, you will likely see an additional error message referring to an invalid target extension. That is discussed in the following section.

## **Adapting Build Jobs to Changed Build Types**

Even after you have converted a build job to the new TGT format, you may still have to modify a given build job because of changes to the build types in Serena Builder.

For example, the older build jobs used a task called Ant Classic Javac. This task would accept an individual .java file and compile it to a .class file, and the corresponding TGT file was named <target>.class.tgt.

If you try to execute this build job in Serena Builder, the following error message appears:



The new build types automatically include subtasks such as Set Classpath and Ant Javac. This is a convenience when creating a new build job, but it means that the file extensions of some targets have changed from the previous version.

To modify the build job, open the Target Definition page for each target, and change the target extension as appropriate.



# Chapter 4

---

## Tutorial

This section will show you several examples of how to use Serena® Builder™ for Professional™, from setting up new projects to executing builds.

Creating a Java Build	44
Creating a Java Jar	52
Working With Multiple Source Directories	54
Creating Development and QA Builds	55
Creating Builds Based on Version Labels	55
Creating Builds Based on Promotion Groups	56
Complex Build Job Examples	56
Example of Using the Serena Builder Desktop Client	56

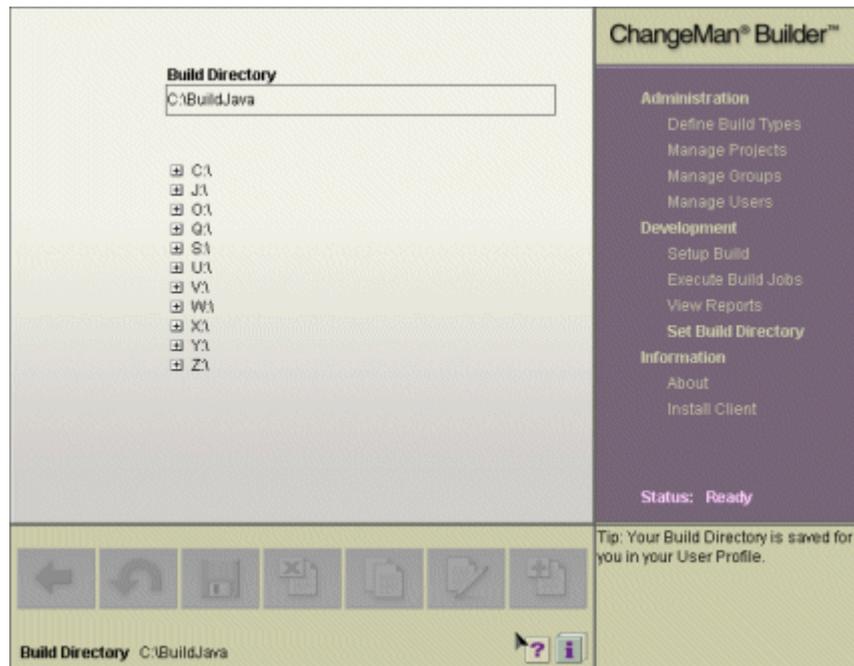
## Creating a Java Build

Here is an example of how a Java developer using Ant would begin using Serena Builder for Professional.

The developer has already installed the Builder command-line client and has explored the Builder menus. Now he wants to see if he can create a small build.

### Set the Build Directory

Deciding that his build directory will be C:\BuildJava, he enters this value into the field that appears when he clicks **Set Build Directory** on the main screen:



The developer clicks the floppy-disk icon to save the build directory definition.

### Import or Create Source Files

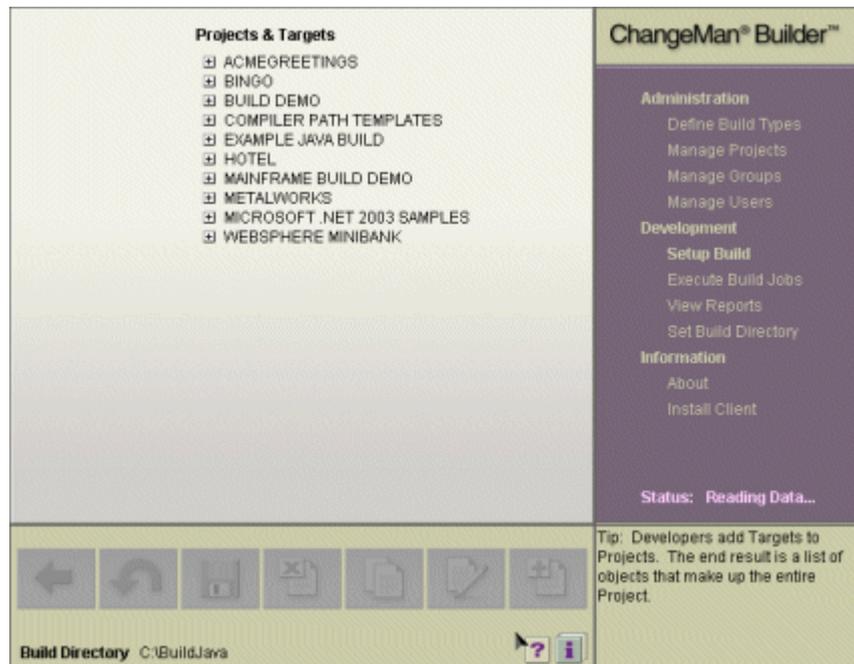
Now the developer is ready to move his source files into the build directory. He checks out a small Version Manager test project that he has already created, placing the test file into the build directory. This file is named `helloworld.java`, so the build directory currently contains the following files:

```
helloworld.java
```

### Set Up the Build

Now the developer wants to specify how the project should be built. In this small test case, `helloworld.java` should become `helloworld.class`.

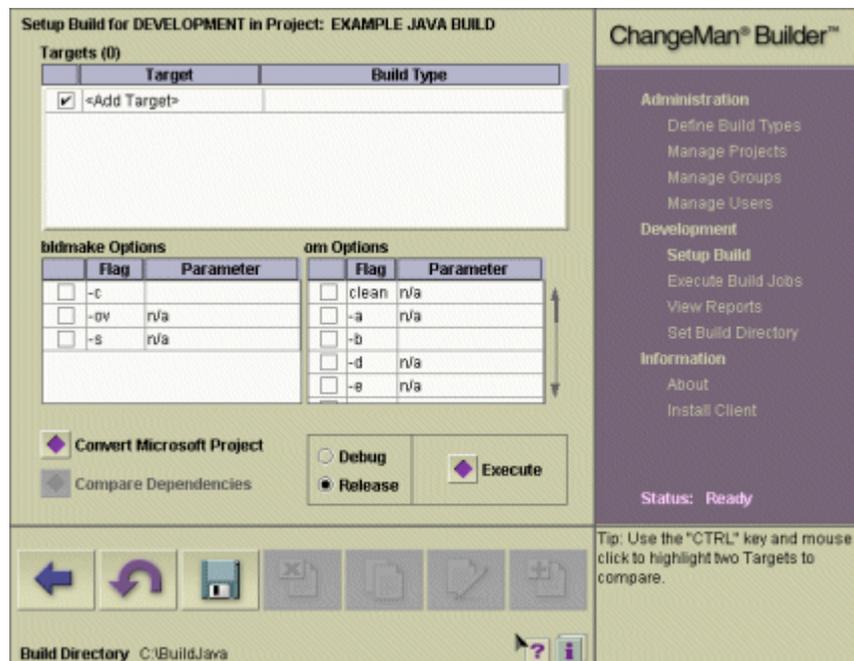
To accomplish this, the developer clicks **Setup Build** on the main page, and the Projects and Targets page appears:



The developer knows that he needs to find a project that is already set up for the Java compilers in use at his company. He finds a project named Example Java Build.

He clicks on the name of this project to see what it contains, and finds a single item named DEVELOPMENT. This represents a search path that has already been defined by the Builder administrator. That is, DEVELOPMENT specifies which directories Builder should look in to find the source files.

Knowing that the search path contains the current directory by default, the developer double-clicks **DEVELOPMENT** to display the Setup Build page:



## Add Targets

As mentioned earlier, the developer has only a single source file and knows what it should produce when compiled.



**NOTE** The end-product is the target, and the source file is the dependency.

The developer double-clicks **<Add Target>**, and the Target Overview page appears.

Target Definition for Project: EXAMPLE JAVA BUILD

Overview **Detail**

**Operating System**  
Windows

**Build Type**  
NET C# Dynamic Link Library

**Final Target Name**

**Final Target Directory**

**Intermediate Directory**

**Target Definition File**

Exclude From Full Project Build

**Build Type Information**

**Final Target Extension:**  
.dll

**Description:**  
Creates a DLL using the Microsoft .NET C# compiler. The dependency should be the .NET Project file (.csproj). Must run bldmake with -s to preserve case sensitivity.

Tip: Tasks that begin with the word "Ant" call the Jakarta Project Apache ANT tasks. Those that do not have the word "Ant" are ChangeMan Builder specific Java tasks.

Build Directory C:\Build\Java

## Fill Out Target Overview

The first thing the developer does is select Java for the Operating System, and Java Compile for the Build Type. The screen changes to display the Java build type information:

Target Definition for Project: EXAMPLE JAVA BUILD

Overview Detail

Operating System  
Java

Build Type  
Java Compile

Final Target Name  
[Empty]

Final Target Directory  
[Empty]

Intermediate Directory  
[Empty]

Target Definition File  
[Empty]

Exclude From Full Project Build

**Build Type Information**

**Final Target Extension:**  
.javac

**Description:**  
Uses the Ant "javac" task to compile Java source code files. All dependencies added to this Build Task will be compiled using the compiler specified in the "compiler" Option parameter.

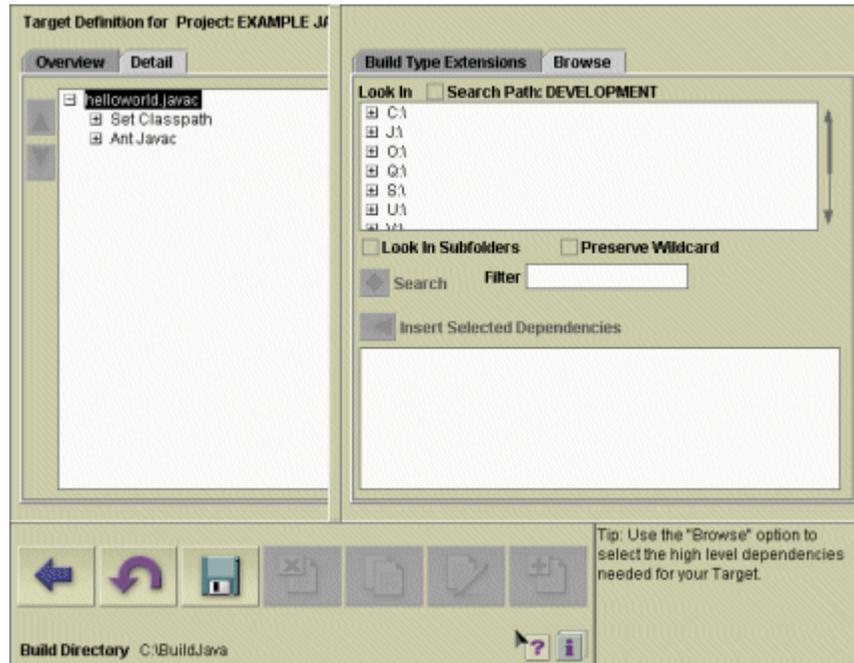
Tip: Tasks that begin with the word "Ant" call the Jakarta Project Apache ANT tasks. Those that do not have the word "Ant" are ChangeMan Builder specific Java tasks.

Build Directory C:\BuildJava

For Target Name, the developer types `helloworld.javac`, as recommended by the Build Type Information. Builder automatically fills in `helloworld.javac.tgt` for the Target File Name. The remaining fields are left blank.

## Fill Out Target Detail

Now the developer clicks the **Detail** radio button at the top of the page, and the Target Detail page appears:



This version of Builder automatically includes a template for the appropriate build tasks—in this case, Set Classpath and Ant Javac—for you. The next step is to fill in the detail for those build tasks.

### Set Classpath Task

Here the developer needs to specify the relationship of source file to final product. Because this is a Java program, the first task listed is Set Classpath. That task appears in the task list, with a plus sign next to its name. By clicking on the plus sign, or by double-clicking the task name, the developer reveals the <Add Dependency> item:



The developer double-clicks on **<Add Dependency>**, and the text becomes editable. The developer types `rt.jar`, since that file contains many of the standard Java classes.



**NOTE** The IBM Java Development Kit, version 1.4.0 and later, does not contain `rt.jar`. Therefore, on AIX, or with IBM WebSphere Developer Studio or AppServer, you should use `core.jar` and `graphics.jar`.

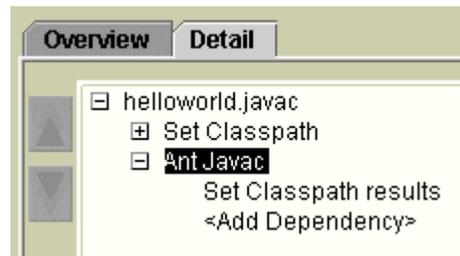
If the developer needed a specific .jar file, he could have used the Browse tab to locate the file.

If you enter the name of a file, use a non-fully-qualified filename; that is, do not include drive letters in the name of the file.

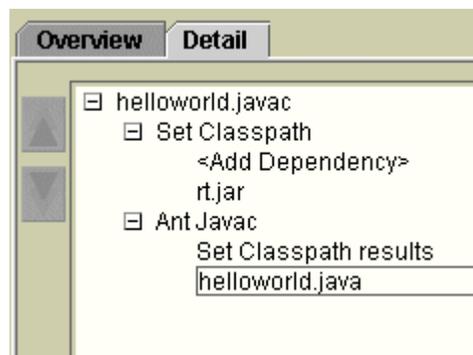
### **Ant Javac Task**

The next build task, also provided by Builder, is Ant Javac, indicating an Ant Java compile task.

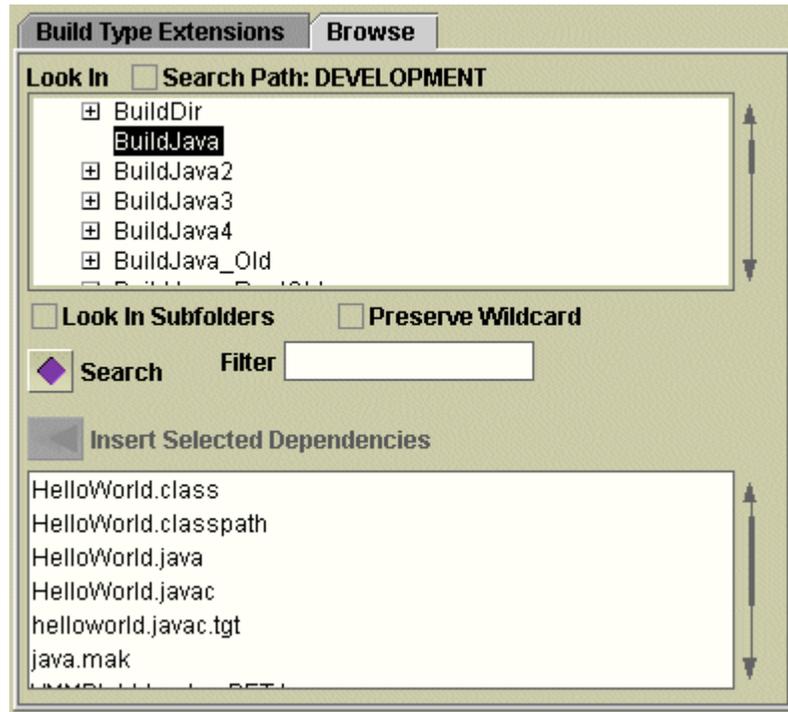
The developer clicks the plus sign next to Ant Javac, or double-clicks the task name. The Ant Javac task hierarchy expands:



The "Set Classpath results" indicate that the Set Classpath task will be performed first. To also indicate that the source file `helloworld.java` is a dependency and must be compiled to complete the build, the developer double-clicks the **<Add Dependency>** item, and types the name of the file:



The developer could also have clicked the **Browse** tab to browse locate the directory containing the source file, then clicked **Search**. The source file `HelloWorld.java` appears in the list of search results:



The developer selects `HelloWorld.java`, then uses the **Insert Selected Dependencies** button to move the file to the Detail area.



**NOTE** If the Ant Javac task is not selected first, the **Insert Selected Dependencies** button will have no effect.

Since there are no other dependencies, the developer clicks the floppy-disk icon to save the target detail, and then clicks the Back arrow icon to return to the Target Overview page. When asked if he would like to save the target, the developer clicks **Yes**.

## Select Targets to be Built

Since there is only one target, the developer clicks the check box next to `helloworld.javac`, indicating that that target should be built. The **<Add Target>** check box automatically receives a check box also; this indicates that all targets have been selected.

## Select Options

The developer considers the various options under `bldmake Options` and `om Options`, but as this is only a small test project, he leaves them all unselected, which is the default setting.



**NOTE** The options for the `bldmake` and `om` programs are detailed in the *Builder Development Guide*.

## Execute the Build

To execute the build, the developer clicks the **Execute** button on the Setup Build page. Immediately a new browser window appears, displaying a two-paned build log window.

Initially the build log window displays the message "Waiting for the log file to be created," but after a few moments, the build log window shows the results of the compile operation:

```

 Automatically Refresh

Build JobName: EXAMPLE JAVA
Owner: jting
Machine: My Computer
Date/Time: 2005-01-20 10

10:29:58 License Valid
10:30:00 builds finished
10:30:00 License Valid
10:30:01 Building: hellow
10:30:04 Built: helloworl
10:30:04 om finished with

Serena ChangeMan Builder powered by Opensake(R) V6.3 Build 101504,

Application EXAMPLE JAVA BUILD from DEVELOPMENT for all targets.
Done loading target lists.

Done processing rules.
Creating java.mak
Creating windows.mak

Construction Complete.
builds finished successfully

Serena ChangeMan Builder powered by Opensake(R) V6.3 Build 101504,

Building: helloworld.javac
Creating helloworld.javac
Buildfile: build_helloworld.javac.xml

compile:
Built: helloworld.javac
om finished with no errors

```

## Check the Results

The developer looks in the build directory. Whereas before there was only the `helloworld.java` file, now he finds the following files:

File	Purpose
<code>helloworld.class</code>	Java class file.
<code>helloworld.classpath</code>	Classpath definition file.
<code>helloworld.java</code>	Original source file.
<code>helloworld.javac</code>	File defining compile task in Ant.
<code>helloworld.javac.tgt</code>	Target definition file.
<code>java.mak</code>	Java makefile or build control file.
<code>windows.mak</code>	Windows makefile or build control file.

The developer checks the code defined by `helloworld.class` to verify that the build produced the desired results.

## Archive the Target Definition File

Although this example is very simple, if the developer wanted to preserve all the setup work done just described, he would create a new source control archive for the target

definition file (.TGT). This file would then be checked in and checked out along with the source files.

## Creating a Java Jar

In this example, another developer takes the example used in the previous sections and sets up Builder to create a Java .jar file.

The developer in this example:

- 1 creates on her hard drive a new build directory named C:\BuildJavaJar
- 2 enters that location on the Set Build Directory page
- 3 uses the same EXAMPLE JAVA BUILD project
- 4 uses the same DEVELOPMENT search path

However, when she double-clicks **DEVELOPMENT**, no targets appear on the Setup Build page. This is because she is in a different build directory, and does not have the target definition file created by the other developers.

### Add Targets

The developer double-clicks **<Add Target>** on the Setup Build page to begin creating a new target named `helloworld.jar`. The Target Overview page appears.

### Fill Out Target Overview

On the Target Overview page, she sets the Operating System to Java, but sets the Build Type to Java Jar. She also fills in the target name as `helloworld.jar`. Filling in the target name allows her to move to the Target Detail page.

### Fill Out Target Detail

On the Target Detail page, the developer sees that Builder has automatically entered three build tasks: Set Classpath, Ant Javac, and Ant Jar.

#### ***Set Classpath Task***

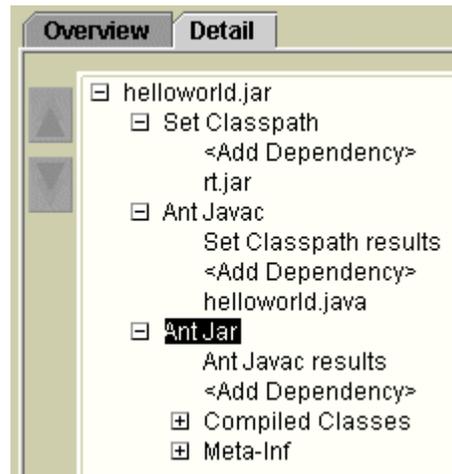
The developer specifies the name and location of .jar files and other classpath-related dependencies for her program, as described earlier. See ["Set Classpath Task" on page 48](#).

#### ***Ant Javac Task***

The developer specifies which files are to be compiled, as described earlier. See ["Ant Javac Task" on page 49](#).

## Ant Jar Task

The developer clicks on the plus sign next to the Ant Jar task (or double-clicks the task name). The Ant Jar hierarchy appears:



As seen earlier, the previous task (in this case, Ant Javac results) is listed first, indicating that the Ant Jar task is dependent on that task's results.

To complete the definition of the .jar file, the developer does the following:

By using <Add Dependency> underneath Meta-Inf, the developer can add a specific manifest for the .jar file.

By using <Add Dependency> under Compiled Classes, the developer can add specific compiled classes to the .jar file.

By using <Add Dependency> immediately under Ant Javac results, the developer can add any other files needed by the .jar file.

When all needed dependencies have been specified, the developer clicks on the floppy-disk icon to save the target definition file. Then, to return to the Target Overview page, she clicks on the Back arrow icon.

## Select and Submit Target

On the Target Overview page, the developer selects the helloworld.jar target to be built, and then clicks the **Execute** icon.

A browser session appears immediately, and after a few moments, the build log informs her that helloworld.jar was built.

The developer inspects the build directory and finds that the helloworld.jar file is indeed present. When she examines the manifest of the .jar file, it shows the proper contents.

To preserve her work, the developer adds helloworld.jar.tgt to her source control archives.

## Working With Multiple Source Directories

In this example, two developers collaborate on the same project, illustrating how Builder handles source files in multiple locations.

The Builder installation directory contains the example `Examples\ref\metalworks`. A partial listing of this directory is:

```
metalworks
  \dev
  \dev\src
  \dev\src\MetalworksInbox.java
  \release
  \release\HelpFiles
  \release\IMAGES
  \release\src
  \release\src\MetalWorks.java
  \release\src\
  MetalWorksDocumentFrame.java
  \release\src\MetalWorksFrame.java
  \release\src\MetalWorksHelp.java
  \release\src\MetalWorksInbox.java
  \release\src\MetalWorksPrefs.java
```

The two directories `\dev` and `\release` represent an individual developer's working directory, and the official release directory, respectively. Each contains a source directory named `src`.

Notice the two files named `MetalWorksInbox.java`. The copy in the release directory represents the official version of the file, whereas the copy in `\dev\src\` represents work that an individual developer might be doing.

Through the use of different search paths, Builder makes it possible for the individual developer to test his changes before checking them in to the official release directory.

Also, it is unnecessary for the developer to make local copies of all source files. Builder looks in the directories named in the search path, and stops looking once it has found an instance of the file. This allows local developer files to take precedence over release directory files.

The developer working in the `\dev\src` directory builds the project using the search path named `Development`, which is set by the Administrator to look at the `\dev\src` directory. Once Builder finds the `MetalWorksInbox.java` file in `\dev\src`, it stops looking for that file. The copy in `\release\src` is never used. This allows the developer to build the entire project while testing his changes to `MetalWorksInbox.java`, yet not affect the final release until he is ready.

The release engineer building the final release version uses the `Release` search path. This search path never looks at the `\dev\src` directory, and considers only the `\release\src` directory.



**NOTE** The previously described method depends on the Builder administrator setting up the search paths to point to the `\dev` and `\release` directories.

## Creating Development and QA Builds

In the previous example, you saw how using different search paths made it possible for individual developers to test their code without checking it into the main release directory, and without requiring multiple copies of all files.

The Metalworks example also shows how code may be separated for development and Quality Assurance.

The Metalworks project has three search paths set up: Dev, QA, and Release. These search paths are defined as follows:

Search path	Defined as
DEV	.
	./tgt
	\$(REFDIR)\metalworks\dev\src
	\$(REFDIR)\metalworks\dev
	\$(REFDIR)\metalworks\qa\src
	\$(REFDIR)\metalworks\qa
	\$(REFDIR)\metalworks\release\src
	\$(REFDIR)\metalworks\release
QA	\$(REFDIR)\metalworks\qa\src
	\$(REFDIR)\metalworks\qa
	\$(REFDIR)\metalworks\release\src
	\$(REFDIR)\metalworks\release
RELEASE	\$(REFDIR)\metalworks\release\src
	\$(REFDIR)\metalworks\release

This indicates that the DEV search path will cause Builder to search first in the current directory (.), then in a directory named tgt (for target files), then in the \dev\src directory. If the search does not find all of the files named in the build, Builder continues on to the \qa and \release directories, in that order. This allows the individual developer to have her local copies of files take precedence.

The QA search path, by contrast, does not even look in the \dev directory but instead starts with the \qadev directory. This allows the QA engineers to build only the version of the product that has been released to them for testing, or to the final release directory.

Finally, the RELEASE search path searches only in the release directory, insuring that when the release version of the product is built, Builder will take no source files from development or QA.

## Creating Builds Based on Version Labels

In this example, a developer uses Builder with Version Manager version labels.

To support version labels, the Builder administrator must set up a search path corresponding to each version label. The version control software takes care of checking out each version, but different versions must be placed in different directories.

For an example of a project set up to use version labels, see the Acme Greetings project in the Examples directory.

## Creating Builds Based on Promotion Groups

In this example, a developer uses Builder with Version Manager promotion groups to manage the different stages of a project.

To support promotion groups, the Builder administrator must set up a search path corresponding to each promotion level.

For an example of a project set up to use promotion groups, see the Build Demo project in the Examples directory.

## Complex Build Job Examples

The previous examples used only simple examples. In real-world projects, many source files are used, including third-party libraries, non-Java files such as HTML files, and so on.

When you install the command-line client of Builder, you also install some examples of more complex build jobs that incorporate those elements. You can find those examples at:

```
<install directory>\Serena\ChangeMan\Builder\examples\ref
```

These examples are discussed in the *Builder Getting Started Guide*.

## Example of Using the Serena Builder Desktop Client

Here is an example of the process of using Serena Builder for Professional from within Version Manager.

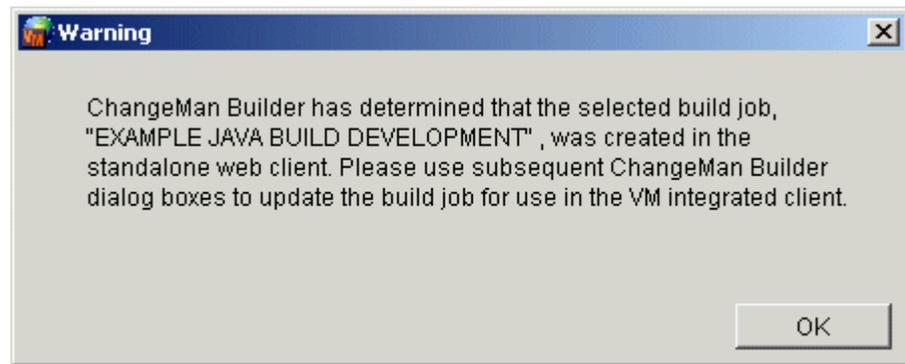
### Create and Archive the .TGT File

- 1 Make sure that the source files for your project are archived in Version Manager.
- 2 Have your administrator create a suitable Builder project for your source files.
- 3 Optional: Have your administrator create a Builder search path to the location to which your source files will be checked out.
- 4 Optional: Set the build directory in the web client. This step is not absolutely necessary, but it does determine where you can find the .TGT file to be created in the next step.

- 5 Use the Builder web client to set up the build. This creates a target definition file (.TGT file). Make sure that the target is set to build (that is, the check box for the target is selected).
- 6 Locate the .TGT file and copy or move it as necessary to make sure it is in the same directory as your source files. Add the .TGT to the Version Manager archives.

## Select a Build Method and Build Machine

- 1 From Version Manager, select the project folder containing your source files.
- 2 Check out the files to the directory where you wish the build to occur. You do not need to check out the .TGT file, but you need write access to the source files and to the files that are the end products of the build.
- 3 Choose Actions | Build. The Target File Scan dialog searches for target definition files. If it finds a .TGT file, the Build Source Selection dialog box appears.
- 4 The first time you run the build job, the desktop client will announce that the build job was created in the standalone web client:



Once you fill out the rest of the desktop client dialog boxes (covered in the rest of this section), the build job will be available from within Version Manager, and the dialog above will not appear in the future.

Click **OK**. The Build Source Selection dialog box appears.

- 5 Select whether the build should be done by default revision, version label, or promotion group, then click **Next**. The Build Job Selection dialog box appears.
- 6 Select the build machine to use. For a local build, select "My Computer", then click **Add**. The chosen build machine appears in the large field.
- 7 Click the build machine name in the large field. The Next button becomes enabled.
- 8 Click **Next**. The Target Selection dialog box appears.

## Select Targets to Be Built

- 1 From the Project Targets Selection field, select the targets to be built, then click the > button to move the targets to the Project Targets Selected field.
- 2 Click **Next**. The Build Parameter dialog box appears.

## Specify Build Parameters

- 1 Use the Browse button (...) to choose a directory where the build should take place. You must select a directory or the Next button will not become enabled.
- 2 Specify any parameters for bldmake or om (see the *Builder Development Guide* for possible parameters).
- 3 Specify whether the build is to be Debug or Release.
- 4 Select a build mode (see "[Selecting a Build Mode](#)" on page 104).
- 5 Select whether Footprinting and Project Processing should be enabled (see [page 104](#)).
- 6 Click **Next**. The Submission dialog box appears.

## Submit the Build Job

- 1 Make any last-minute changes on the Submission dialog box.
- 2 If you wish, save the build job. This will make it unnecessary to re-enter the build directory and other information if you submit the build job again.
- 3 When you are satisfied with the parameters, click **Submit**. The build job is submitted for execution, and the Builder log browser appears. After a delay, the results of the build operation appear in the log browser.

# Chapter 5

---

## BINGO Tutorial

Check Prerequisites	60
Prepare the Build Environment	60
Overview of the BINGO Project	61
Set the Build Directory	61
Copy Example Files to Working Directory	61
Define Search Paths for the BINGO Project	62
Set Up the BINGO Client Build	63
Set Up the BINGO Server Build	66
Build BINGO in the Standalone Web Client	66
Create a New Version Manager Project	67
Add Workfiles to Version Manager	67
Build BINGO from the Version Manager Integration	68
Run the BINGO Game	72
Apply Version Labels	73
Build BINGO Using a Version Label	73

This tutorial shows you how to set up and build BINGO, a client-server game written in Java. This game ships with Serena<sup>®</sup> Builder<sup>™</sup> for Professional<sup>™</sup>, in the examples directory.

You will perform the tutorial exercises in the standalone web client and in the Version Manager integration. The PCLI command-line interface is not used in this tutorial.



**NOTE** The example BINGO source files include pre-defined TGT files that you will add to Version Manager; therefore, you will not be defining TGT files in this tutorial.

## Check Prerequisites

Before beginning work on the BINGO tutorial, be sure you have the following in place:

- License Manager available
- Version Manager installed
- Serena Builder installed
- Java Development Kit installed
- Define environment variables
- Administrator rights, so that you can modify the search path

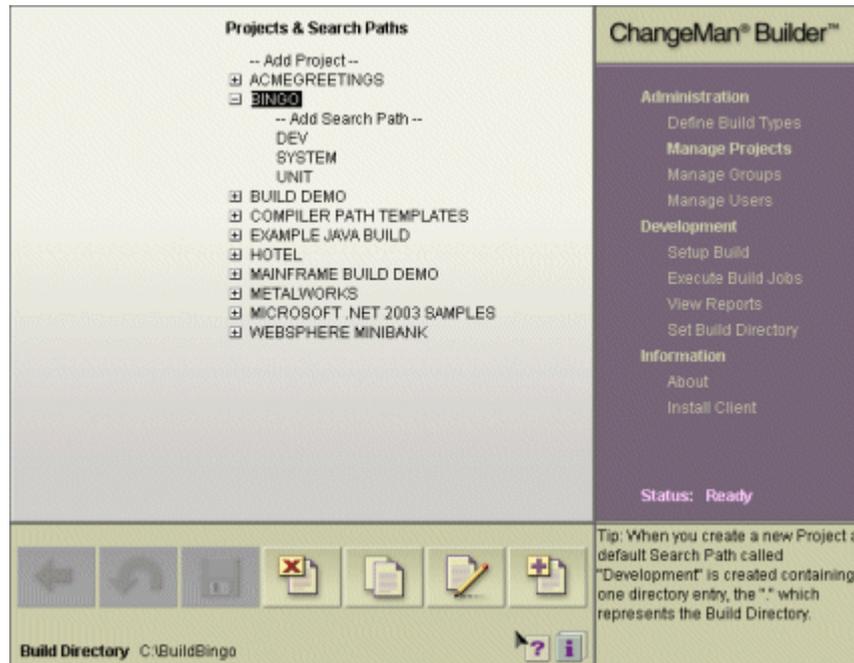
## Prepare the Build Environment

As with any Serena Builder project, you need to prepare the build environment. The BINGO project is a Java project, so you should make sure that the Java Development Kit is available and that it is recognized by Builder.

If you have not already done so, go through the smaller examples in the previous chapter; that will verify that the Java development environment is up and available.

## Overview of the BINGO Project

The BINGO project is a client-server application written in Java. It appears in the list of projects on the Builder standalone web client:



You can see the list of search paths—DEV, SYSTEM, and UNIT—each meant to capture the requirements for a different type of build. For example, a developer wants to use files from his local directory, and the system build will look for files in a common system turnover area.

This example covers the DEV build, representing an individual developer.

## Set the Build Directory

At the start of every project, you must set the build directory. In this example, the build directory is C:\BuildBingo.

## Copy Example Files to Working Directory

- 1 Create a working directory (in this example, BINGO\_WorkingDir) so that the original source files remain unchanged.
- 2 Copy the example files from
 

```
<install directory>\examples\ref\bingo-game
```

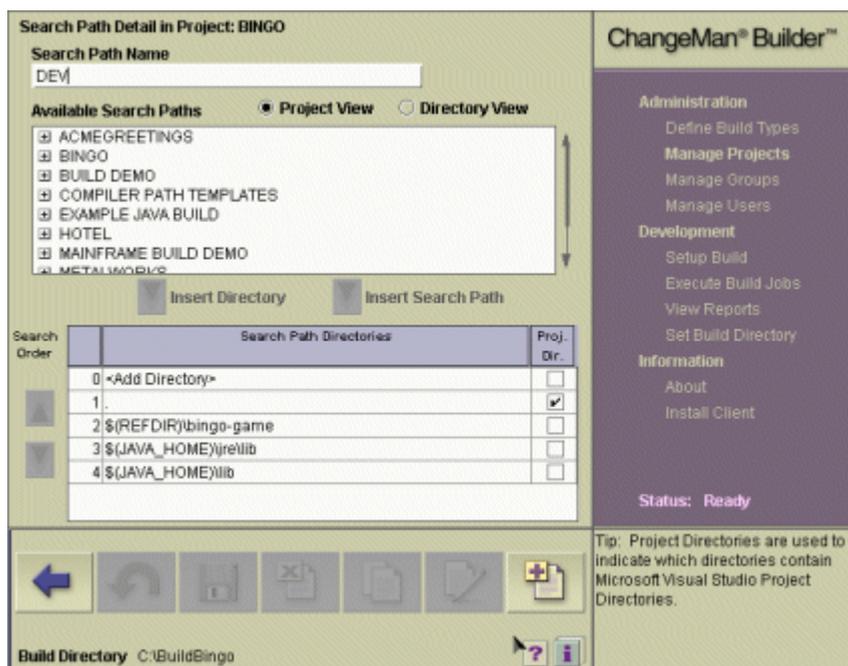
 into the working directory. The default install directory is C:\Program Files\Serena\ChangeMan\Builder.

- Return to the BINGO project in the Builder standalone web client.

## Define Search Paths for the BINGO Project

Now you will modify the search path for the project.

- From the Builder standalone web client menu, click **Manage Projects**. The list of projects appears.
- Expand the **BINGO** entry. The **DEV**, **SYSTEM**, and **UNIT** search paths appear, along with the **Add Search Path** task.
- Double-click the **DEV** search path. The Search Path Detail page appears:



The list of search path directories shows the directories that Builder will search when it evaluates dependencies in the project. Since "." is the first entry, the build directory will be the first location that Builder looks in, followed by the bingo-game directory in the reference directory tree, and finally the two locations underneath JAVA\_HOME.

To add the working directory you created earlier to the list of search path directories, follow these steps:

- Click the **Directory View** radio button.
- Browse for the desired directory.
- Click **Insert Directory** to add the directory to the list.
- Use the Search Order buttons to adjust the directory's search priority.
- Click the Save icon (floppy disk) to save the search path information.

# Set Up the BINGO Client Build

Once you have the project search paths set up, you can set up the build.

- 1 Click **Setup Build**.
- 2 As before, expand the **BINGO** item. The **DEV**, **SYSTEM**, and **UNIT** items again appear.
- 3 Double-click **DEV**. The **DEV** project structure, with a client target and a server target, appears in the Setup Build page:

**Setup Build for DEV in Project: BINGO**

Targets (2)

Target	Build Type
<input checked="" type="checkbox"/> <Add Target>	
<input checked="" type="checkbox"/> bingo-client.jar	Java - Java Jar
<input checked="" type="checkbox"/> bingo-server.jar	Java - Java RMI/Jar

blidmake Options

Flag	Parameter
<input type="checkbox"/> -c	
<input checked="" type="checkbox"/> -ov	n/a
<input checked="" type="checkbox"/> -s	n/a

om Options

Flag	Parameter
<input checked="" type="checkbox"/> clean	n/a
<input type="checkbox"/> -a	n/a
<input type="checkbox"/> -b	
<input type="checkbox"/> -d	n/a
<input checked="" type="checkbox"/> -e	n/a

Convert Microsoft Project  
 Compare Dependencies

Debug  
 Release

**Execute**

Build Directory C:\BuildBingo

ChangeMan® Builder™

Administration

- Define Build Types
- Manage Projects
- Manage Groups
- Manage Users

Development

- Setup Build
  - Execute Build Jobs
  - View Reports
  - Set Build Directory

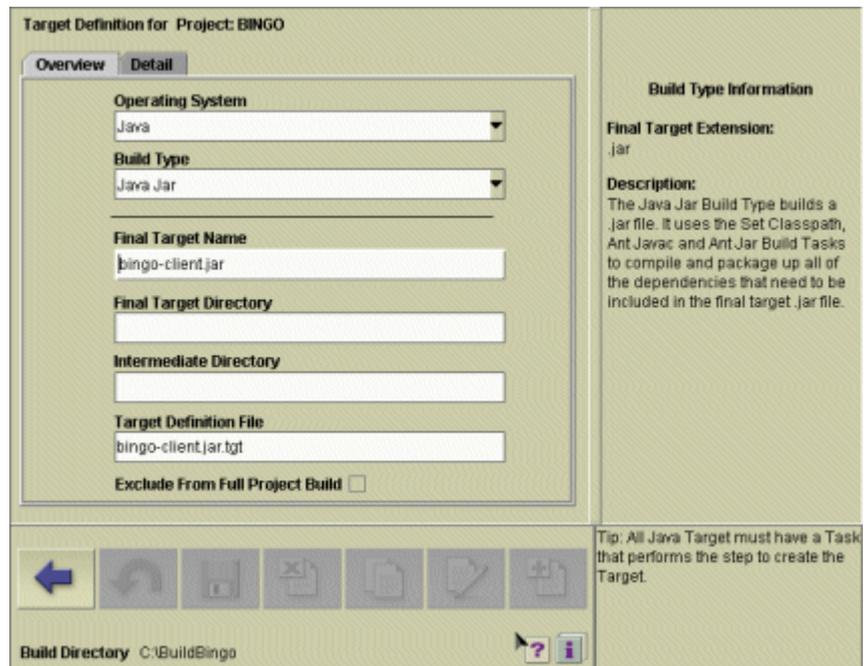
Information

- About
- Install Client

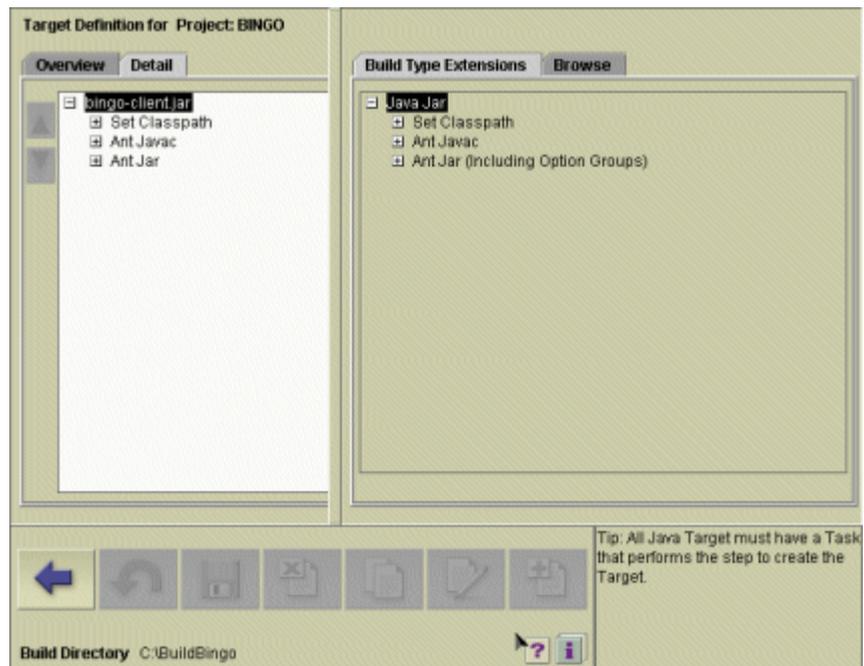
Status: Ready

Tip: Select the check box next to the <Add Target> option to select all Targets for a build.

- 4 Double-click bingo-client.jar. The Target Definition page appears, displaying the Overview tab.



- 5 Click the Detail tab. You see a new hierarchy tree with bingo-client.jar at the top, and the three tasks underneath.



The build tasks are automatically selected based on the build type of the target selected.



**TIP** The lower right-hand corner of the screen displays a brief explanation of the selected target type or selected build task.

For a Java .jar, the build tasks are Set Classpath, Ant Javac, and Ant Jar.

## The Set Classpath Task

In this task you can see `rt.jar` and `bingo-server.jar`. These are both dependencies of `bingo-client.jar`. The `rt.jar` file contains many of the standard Java runtime classes, and the `bingo-server.jar` is the server half of the BINGO program. If there were another file that Builder needed to know about before beginning the compilation of source files, you would add it here.

## The Ant Javac Task

In this task you can see Set Classpath results, meaning that this task depends on the results from Set Classpath, and two other tasks that list all `.java` files in the `bingo\player` and `bingo\shared` directories, meaning that Builder should compile all of those files.

## The Ant Jar Task

This task lists the items and tasks necessary to create the `bingo-client.jar` file itself.

- Ant Javac results—results of the previous task
- `RegistrarImpl_Skel.class`—RMIC skeleton file
- `RegistrarImpl_Stub.class`—RMIC stub file
- `chit.gif` and `invisible.gif`—Image files used in the BINGO game
- Compiled Classes—Option group for compiled classes (see ["Option Groups and Source/Destination Directories,"](#) following)
- Meta-Inf—Option group for manifest info (see ["Option Groups and Source/Destination Directories,"](#) following)

## Option Groups and Source/Destination Directories

Option groups such as Compiled Classes allow you to set options in an identical way for an entire group of files. The Compiled Classes option group in the Ant Jar Task contain two pre-defined options, for example:

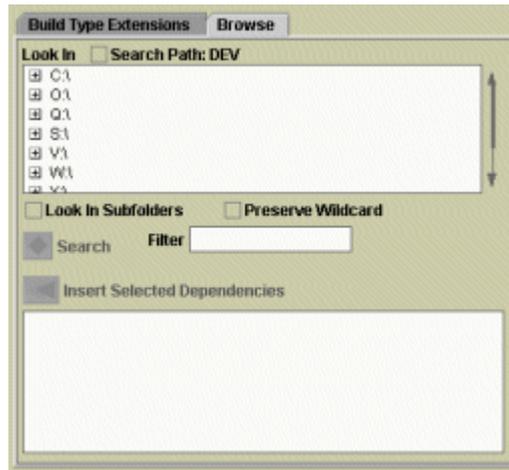
Pre-Defined Options				
	Option Name	Parameter	REL	DBG
<input type="checkbox"/>	<code>dir=</code>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	<code>prefix=</code>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

The **dir** option allows you to specify where the compiled classes should be pulled from; the **prefix** option allows you to specify a directory where the classes added to the `.jar` file should be placed, rather than the default root directory of the `.jar` file.

Similarly, the **prefix** option in the Meta-Inf option group is already filled in with `META-INF`, indicating that dependencies you add to the Meta-Inf group will be placed by default in the `META-INF` directory.

## Adding Dependencies

If you need to add a dependency, the Browse tab allows you to do this:



This tab allows you to search for a directory, then insert it underneath the selected build task, and then adjust its priority within the items for that task.

## Set Up the BINGO Server Build

Setting up the BINGO server build is similar to the process just described for the BINGO client, but there are two new tasks in the BINGO server: Ant RMIC and Ant RMIC Jar.

### Ant RMIC

The Ant RMIC task compiles the Remote Method Invocation classes. Notice that there is an option for the type of compiler to use. The default is "sun", but there are also choices for Kaffe and for Weblogic.

### Ant RMIC Jar

The Ant RMIC Jar task specifies which files shall be placed in the RMIC jar. The task contains the following options:

- basedir—specify a directory from which to collect the files
- compress—apply compression to the jar file
- manifest—specify the name of the manifest.mf file

## Build BINGO in the Standalone Web Client

Once you have set up the BINGO client and server, you can build the project.

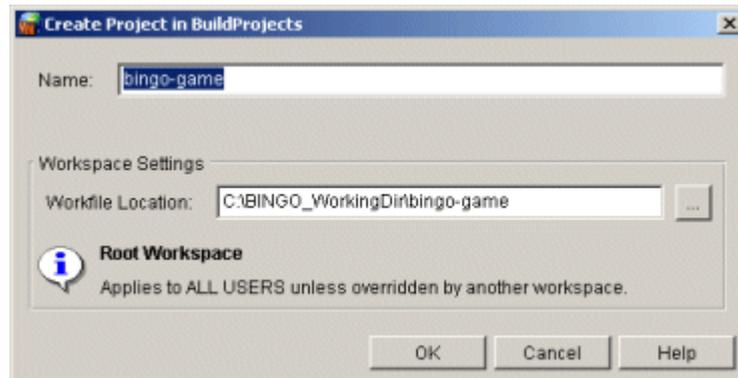
- 1 Click **Execute Build Jobs**.

- 2 Select the build job from the list of available build jobs.
- 3 Click **Execute**. The build job should execute successfully.

## Create a New Version Manager Project

Once you have successfully built the project in the Builder standalone web client, you can add the project to Version Manager.

In the Version Manager desktop, use **File > Create Project** to create a new Version Manager project.

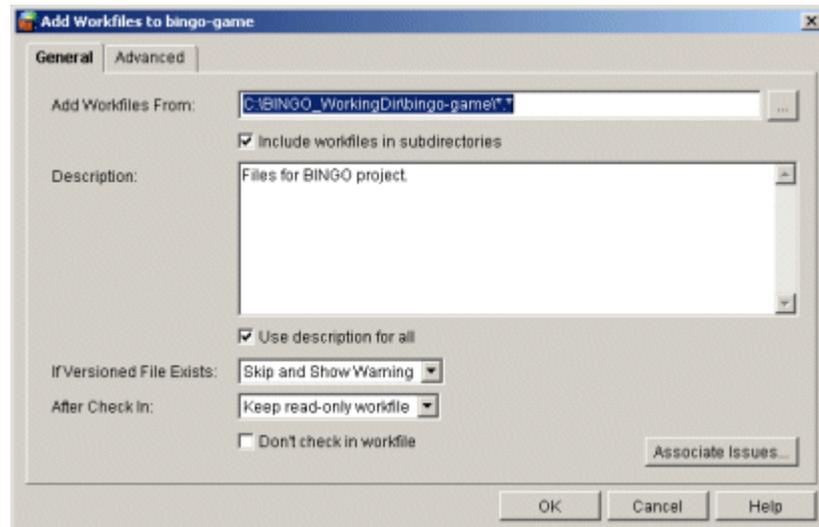


Set the Workfile Location to the working directory that you created in ["Copy Example Files to Working Directory"](#) on page 61.

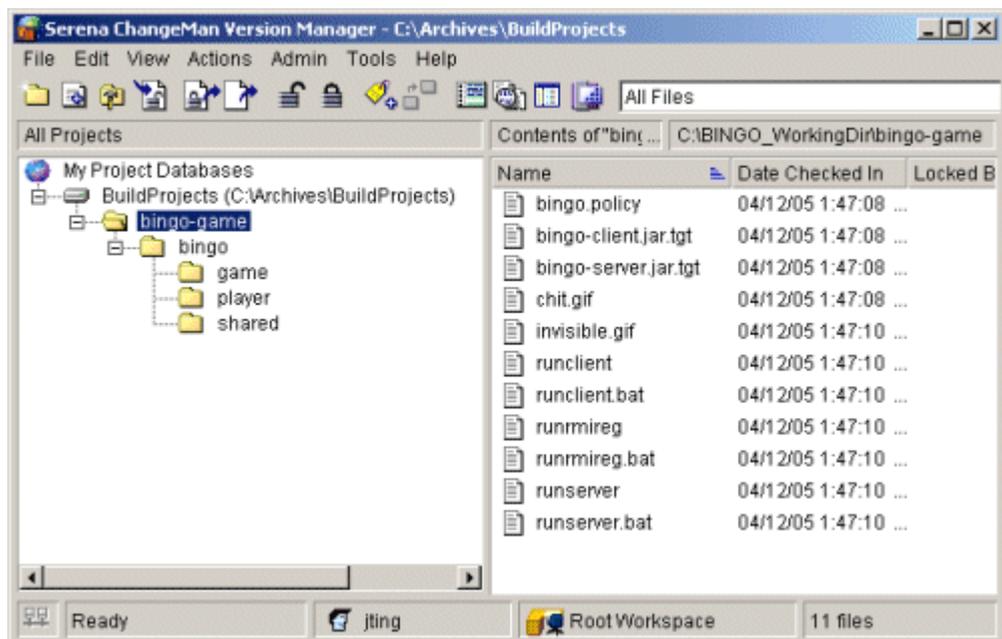
## Add Workfiles to Version Manager

After you have created the Version Manager project, add the workfiles for the BINGO example to the project.

Use **File > Add Workfiles** to add the contents of the bingo-game directory to Version Manager.



When you are done, the project should look like this:

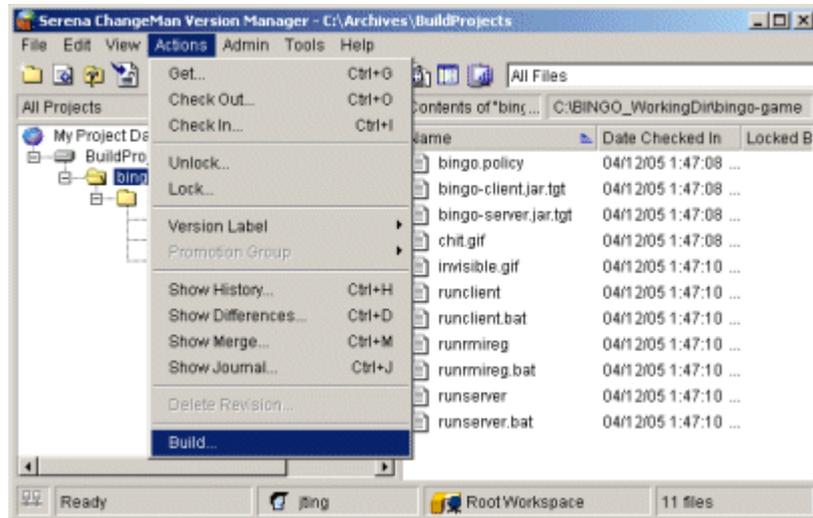


As shown in the above illustration, the TGT files need to be archived in Version Manager before you can build the project using the Version Manager integration.

## Build BINGO from the Version Manager Integration

Now you can configure the build job to run from the Version Manager integration.

- 1 In Version Manager, select the project and then choose **Actions > Build**.

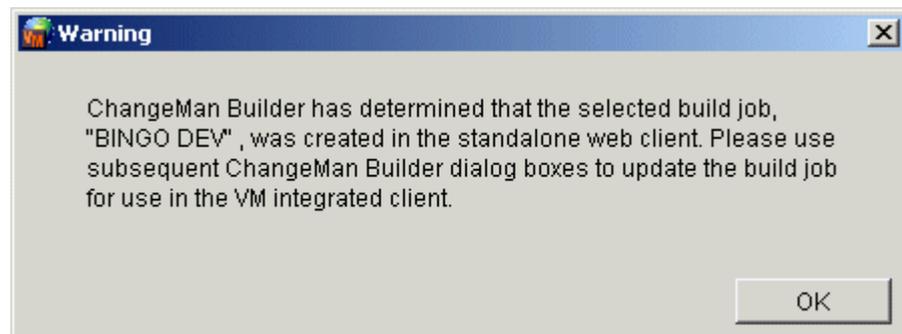


- 2 There is a slight delay as the Builder integration searches the project for TGT files. After that, the Build Source Selection dialog box appears:



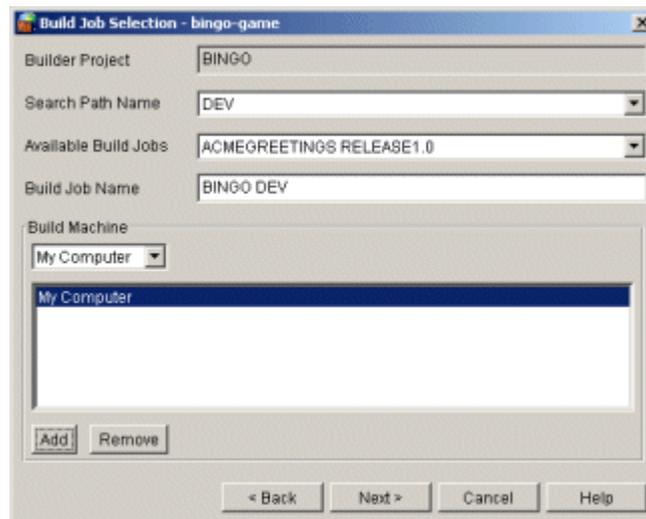
**NOTE** The Build Source Selection dialog box is the first screen in what is referred to as the Builder wizard.

- 3 You can choose to build the project from the default revision, from a version label, or from a promotion group. Click **Next**.
- 4 Because this is the first time the project has been built from the Version Manager integration, a warning dialog box appears:

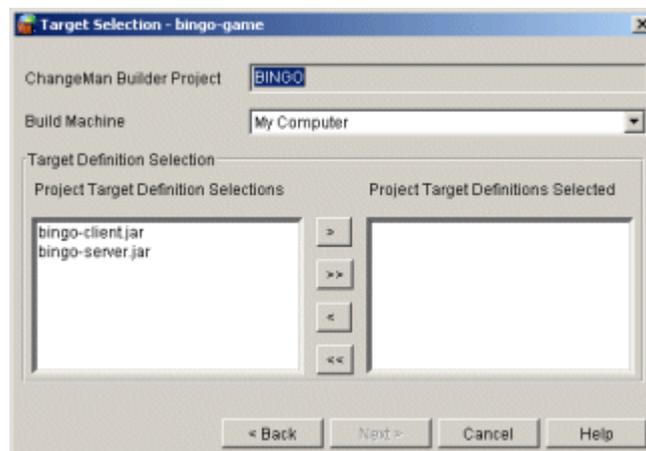


All that you need to do is finish completing the rest of the dialog boxes. Once you save the build job from the Version Manager integration, this message will no longer appear. Click **OK**. The Build Job Selection dialog box appears.

- 5 Click **Add** to add "My Computer" to the list of build machines.

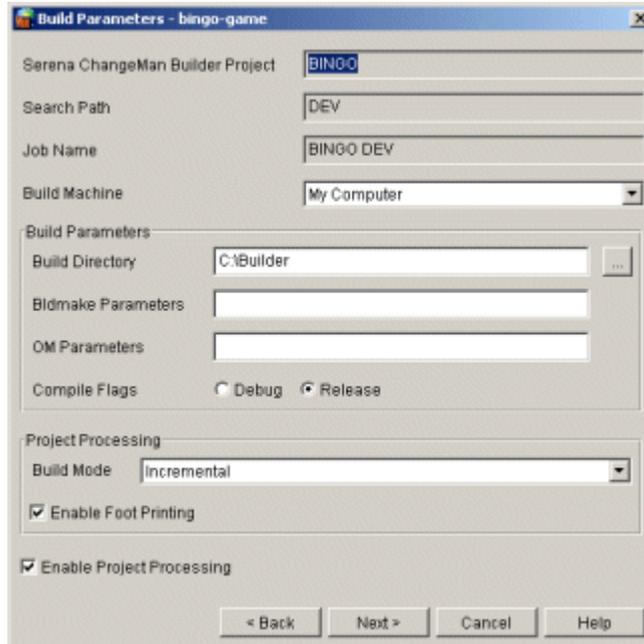


- 6 Click **Next**. The Target Selection dialog box appears.



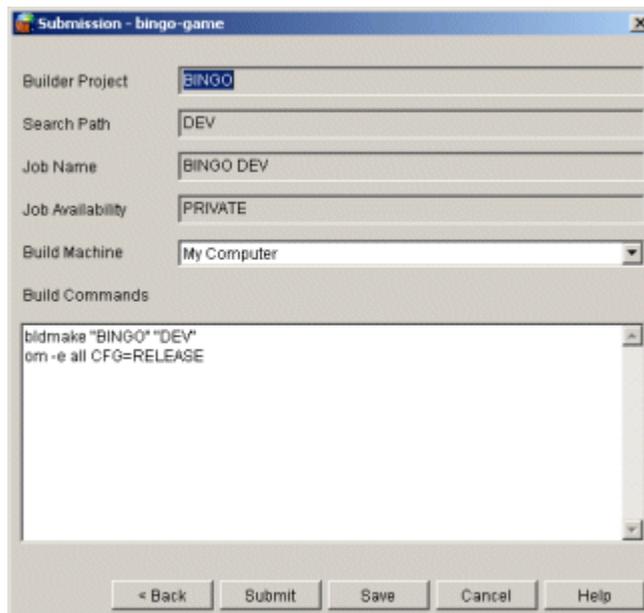
- 7 Click the **>>** button to move all of the targets to the Project Target Definitions Selected area.

- 8 Click **Next**. The Build Parameters dialog box appears.



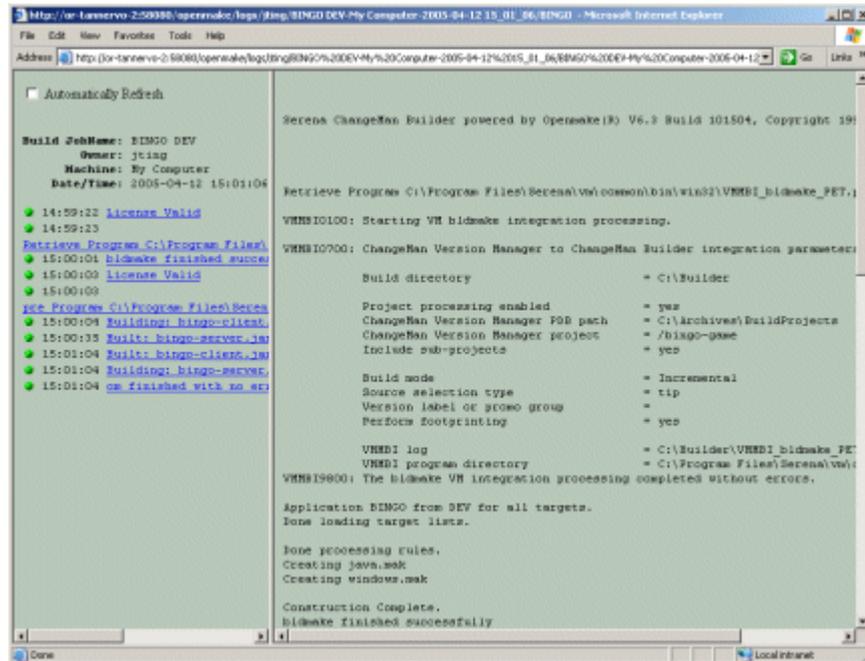
Notice that the build directory is set to the default location—C:\Builder in Windows, \$HOME/Builder in UNIX. If you wish to change the build directory to a different location, use this dialog box to change it.

- 9 Optional: To enable more output in the build log from either bldmake or om, enter -ov (for "output verbose") as a bldmake parameter or an om parameter.
- 10 Click **Next**. The Submission dialog box appears.



Notice the Build Machine field. Although this field also appears on the Build Parameters dialog box, you can use the drop-down list on this dialog box to change the build machine to other available build machines (a remote build server, for example).

- 11 Click Save to save the build job.
- 12 Click Submit to submit the build job. The build log appears in a window of your default Internet browser (you may have to bring the window to the front).



Unlike the build logs from jobs submitted to the standalone web client, the build log will contain messages such as this:

```
VMMSI0100: Starting VM bldmake integration processing.
```

You are looking for the following confirmation messages within the build log:

```
VMMSI9800: The bldmake VM integration processing completed without errors.
```

```
...
bldmake finished successfully
```

```
...
BUILD SUCCESSFUL
Total time: 1 second
```

```
Built: bingo-client.jar
```

```
Building: bingo-server.jar
om finished with no errors
```

These messages indicate that the VM integration completed successfully, and that the project built successfully.

## Run the BINGO Game

To run the BINGO game, execute the following file from a command prompt window:

```
C:\<build directory>\bingo-game\runmireg.bat
```

Wait a few seconds, then start the server by executing this file, again from a command prompt:

```
C:\<build directory>\bingo-game\runserver.bat
```

Wait a few more seconds, then start the client by executing this file, from a command prompt:

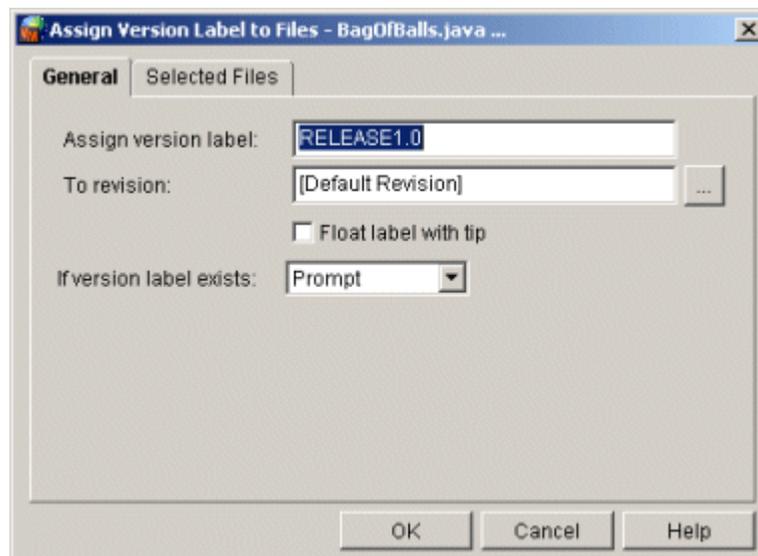
```
C:\<build directory>\bingo-game\runclient.bat
```

## Apply Version Labels

You can optionally add version labels to the deliverables and then build the project according to version label.

To apply version labels:

- 1 In Version Manager, select the files or projects to receive the label.
- 2 Right-click and choose **Assign Version Label**:

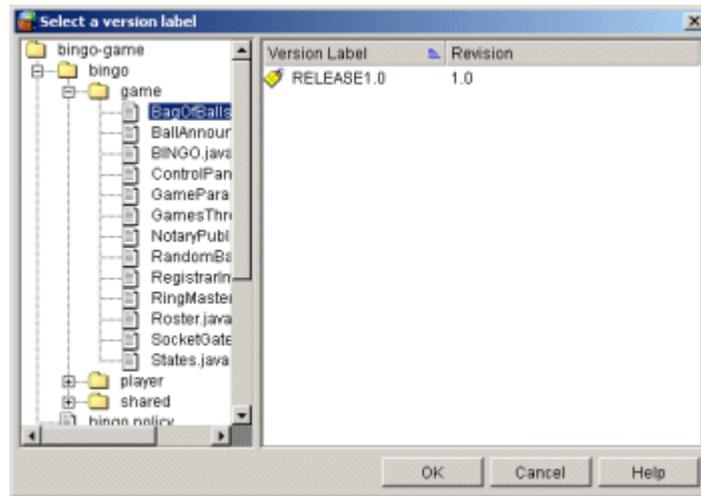


## Build BINGO Using a Version Label

Once you have assigned version labels, you can build the project by version label instead of building the latest revision.

- 1 Select **Actions > Build**.
- 2 In the Build Source Selection dialog box, choose the **Version Label** radio button.

- 3 Enter the name of the version label, or use the **Browse** button (...) to display a list of the available labels.



- 4 Continue with the rest of the build procedure as detailed in the earlier parts of this chapter.

## Chapter 6

---

# Serana Builder and Configuration Builder

This section discusses how to use Serena® Builder™ for Professional™ with Configuration Builder.

Overview	76
Prerequisites	76
Phony Targets and Dummy Targets	76
Running Existing Configuration Builder Scripts	76
Example: Running a CB Script from the Standalone Web Client	77
Using CBMACROS, CBTARGETS, and CBFLAGS	83
Parameter Syntax for CBMACROS, CBTARGETS, and CBFLAGS	84
Running a CB Script from a Command Prompt Window	85
Converting a Simple CB Script to a Serena Builder Script	85

## Overview

You can use scripts developed for Merant Configuration Builder with Serena Builder for Professional. Through the use of a "wrapper" script, Builder can call a Configuration Builder build job and execute it from within Builder's standalone web client, or even from within Version Manager.

## Prerequisites

To use Configuration Builder with Serena Builder, you should first make sure your Configuration Builder installation is set up properly and that the build jobs you wish to call from Serena Builder are working.

## Phony Targets and Dummy Targets

The Builder documentation refers to "phony" targets, which are TGT files that do not actually result in a built file, but which cause a particular task to be done—files are copied from one location to another, for example. The "phony" targets in Builder are the same as "dummy" targets in Configuration Builder. Using a phony target allows you to list a Configuration Builder script as a dependency.

## Running Existing Configuration Builder Scripts

Running an existing Configuration Builder script is accomplished with the use of a "wrapper" script. To use this script, you define a project in Builder using the "Merant CB" build type, and set the CB script parameters using the values in the Options fields on the Build detail page.

Pre-Defined Options				
	Option Name	Parameter	REL	DBG
<input type="checkbox"/>	cbmacros		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	cbtargets		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	-All		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	-B		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	-Batch		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	-Compile		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	-Debug		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	-DebugOn		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Before using the Configuration Builder script, you still have to do the following:

- 1 Set the Build Directory to the location of the CB script.
- 2 Have the Builder administrator create a search path to the location of the files named in the CB script.

- 3 Create a phony target that calls the build script as one of its dependencies. This target does not have to do anything else. You should not define the final result of the CB script as the target of the Builder script. That is, if the CB script creates an executable file named "hello.exe", do not make "hello.exe" the target of the Builder script.
- 4 If there are any flags or macro definitions that you want to pass to Configuration Builder, use the Options Detail page.

Once you have completed these steps, you can run the Configuration Builder script using Builder.

## Example: Running a CB Script from the Standalone Web Client

Here is an example of the process of incorporating a Configuration Builder script into a Builder build job.

### Start With a Working CB Script

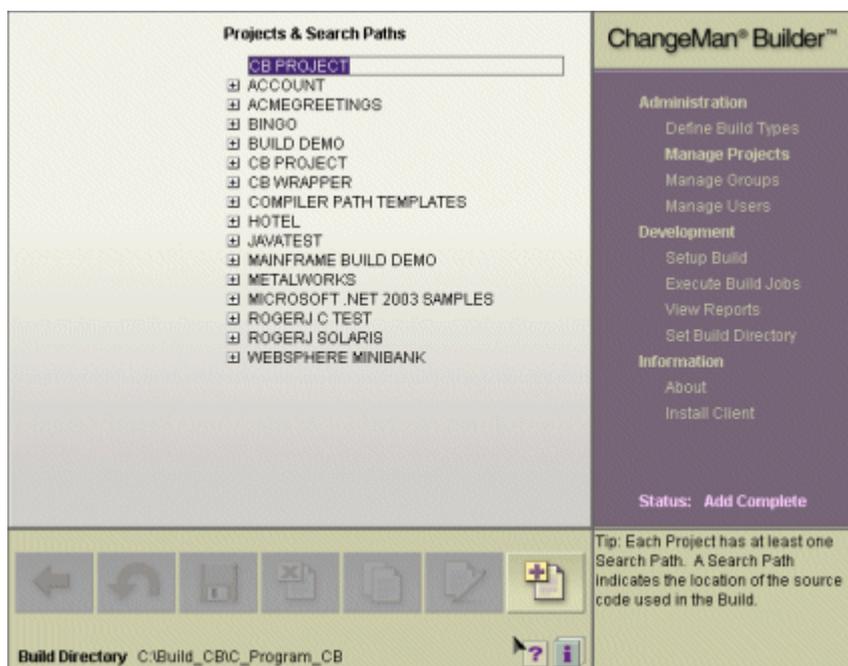
You should verify that the Configuration Builder script.bld (or equivalent) file successfully builds your product in Configuration Builder:

```
C:\Build_CB\C_Program_CB>build
PVCS Configuration Builder 7.5.0.0 (Build d230) for Windows NT/80x86
Copyright 1998-2002 MERANT. All rights reserved.
Today's date is 28 Mar 2005 11:08:26.
gcc -c -g -o hello.o C:\Build_CB\C_Program_CB\hello.c
echo Linking object modules . . .
Linking object modules . . .
gcc -o hello hello.o
build complete.
```

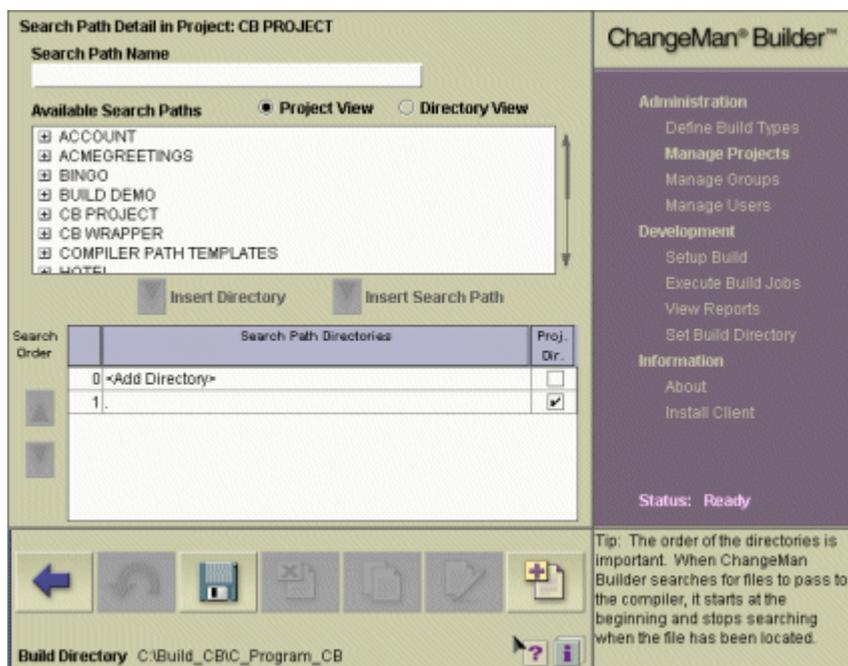
### Set Up Build Directory and Search Paths

- 1 Start the Builder standalone web client.
- 2 Set the build directory to the desired location.

- 3 From the main page, click **Manage Projects**. The Projects & Search Paths page appears. Add a new project (in this example, the project is named CB PROJECT).



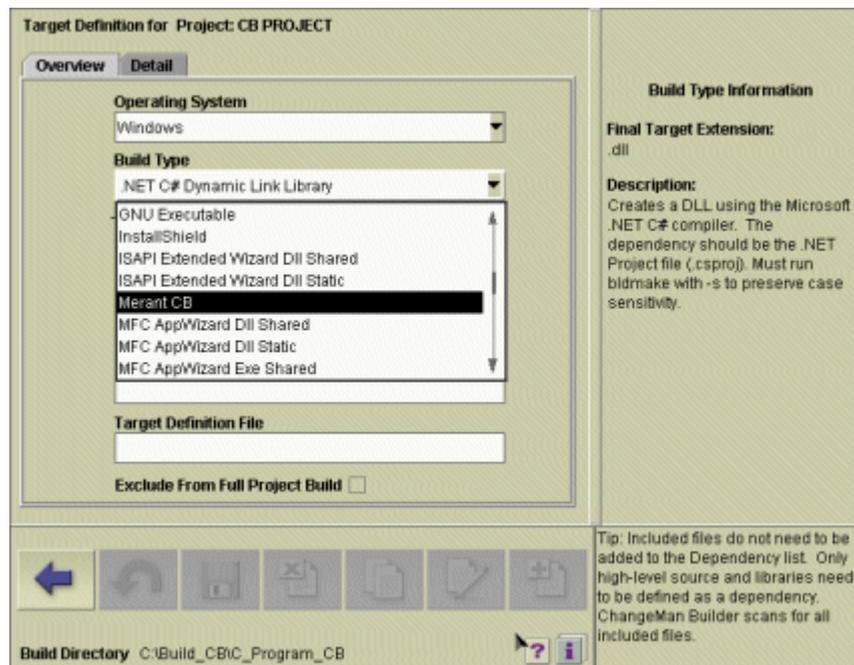
- 4 Expand CB PROJECT so that you can see the Add Search Path item.
- 5 Double-click **Add Search Path** to display the Search Path Detail page.



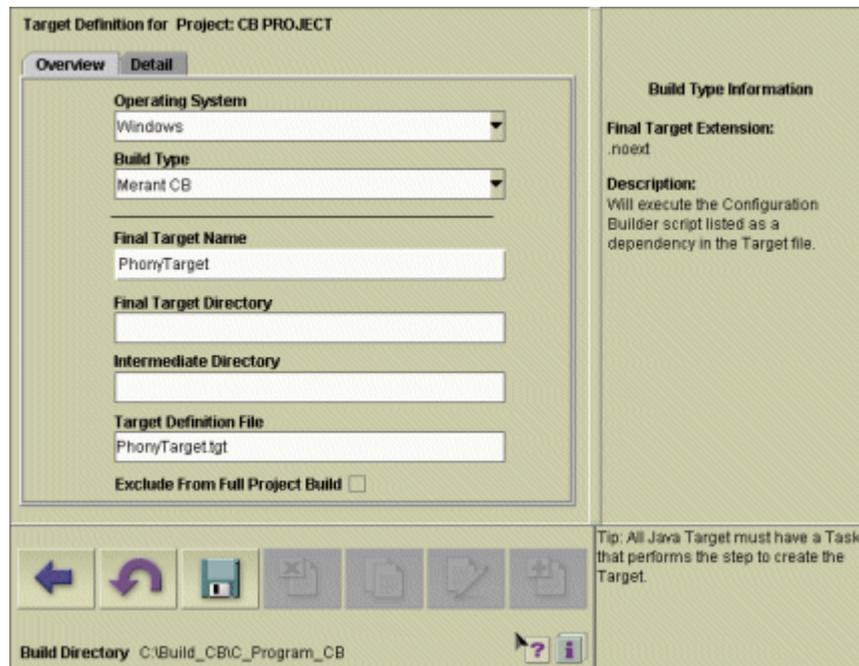
- 6 Name the Search Path (in this example, the search is named DEVELOPMENT).
- 7 Add any necessary directories to the search path and save it.

## Set Up the Build

- 1 Click **Setup Build** to display the Projects and Targets page.
- 2 Expand the new project (CB PROJECT) to display the newly-created search path (DEVELOPMENT).
- 3 Double-click the name of the search path to display the Setup Build page.
- 4 Double-click **<Add Target>**. The Target Definition page appears.
- 5 For the operating system, accept the default choice of Windows.
- 6 For the build type, select Merant CB.



- 7 Enter a "phony target" name in the Final Target Name field. This example uses the name "PhonyTarget".



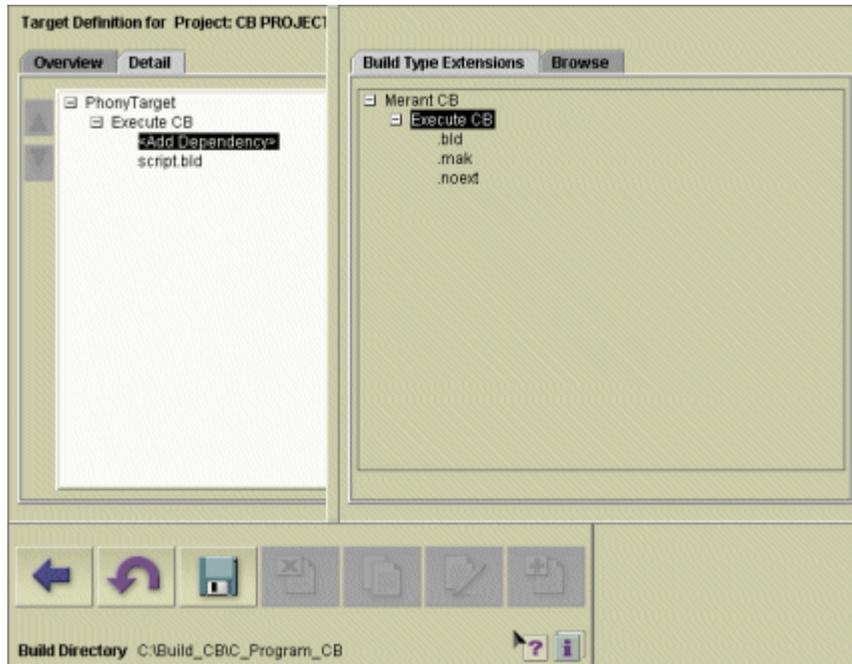
Builder fills in the name of the Target Definition file for you.

- 8 Do not click the Save icon (the floppy disk) yet because you have to define the target dependencies. You will do this in the next section.

## Define Target Detail

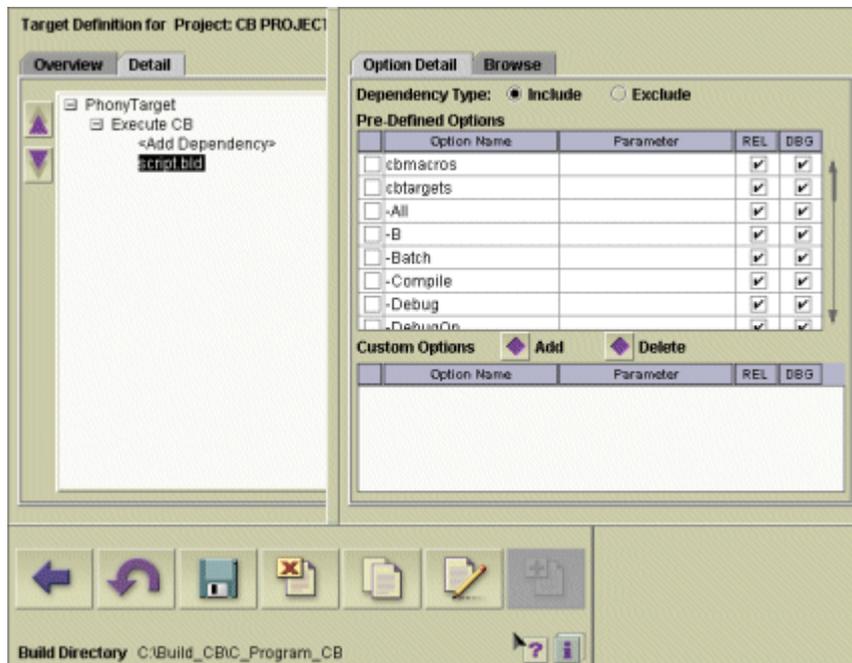
- 1 Click the **Detail** tab to start defining dependencies for the project. You see the name of the phony target, and the Execute CB task underneath it.
- 2 Expand the Execute CB task to display the <Add Dependency> item.

- 3 Double-click **<Add Dependency>** to add the name of the Configuration Builder script file. In this example, the file is named script.bld.



## Select Build Options

- 1 Select the name of the dependency to display the Option Detail tab.



- 2 If you wish to enter values for CBMACROS or CBTARGETS, use the corresponding entries in the list of option names. (See ["Using CBMACROS, CBTARGETS, and CFLAGS"](#) on page 83 for important usage information.)

- 3 If you wish to enter values for Configuration Builder flags, select them from the list of Pre-Defined Options.
- 4 Click the floppy disk icon to save the target definitions.

## Execute the Build Job

- 1 Click **Execute Build Jobs**.
- 2 Select the build job from the list of available build jobs.
- 3 Click **Execute**. The Configuration Builder build job should execute successfully. Here is a sample transcript:

```
Serena ChangeMan Builder powered by Openmake(R) V6.3 Build 101504, Copyright 1995-2004
```

```
Application CB WRAPPER from DEV for all targets. Done loading target lists.
```

```
Done processing rules.  
Creating windows.mak
```

```
Construction Complete.  
bldmake finished successfully
```

```
Serena ChangeMan Builder powered by Openmake(R) V6.3 Build 101504, Copyright 1995-2004
```

```
Building: phonytarget
```

```
Creating phonytarget *****
```

```
*  
* Executing Configuration Builder:  
*  
* Build type version = 1.1  
*  
* Verbose = YES  
* Config = DEBUG  
*  
* Build program = C:\PROGRA~1\PVCS\CB\nt\build.EXE  
* Build script flag = -script "C:\Build_CB\C_Program_CB\script.bld"  
*  
* TGT file:  
* flags =  
* cbmacros =  
* cbtargets =  
* OM command line:  
* cbflags = -NoSilent  
* cbmacros =  
* cbtargets =  
*  
* CB command:  
* C:\PROGRA~1\PVCS\CB\nt\build.EXE -script "C:\Build_CB\C_Program_CB\script.bld" -  
NoSilent  
*  
*****
```

```
PVCS Configuration Builder 7.5.0.0 (Build d230) for Windows NT/80x86  
Copyright 1998-2002 MERANT. All rights reserved.  
Today's date is 28 Mar 2005 11:04:20.  
gcc -c -g -o hello.o C:\Build_CB\C_Program_CB\hello.c  
echo Linking object modules . . .  
Linking object modules . . .  
gcc -o hello hello.o  
build complete.
```

```
*****
*
* Configuration Builder Script Completed:
*
* Return Code = 0
*
*****
```

om finished with no errors

## Using CBMACROS, CBTARGETS, and CBFLAGS

The Merant CB build type allows you to enter values for Configuration Builder macros, targets, or flags.



**NOTE** If you are unfamiliar with Configuration Builder macros, flags, or targets, see the Configuration Builder documentation.

You can enter values for CBMACROS, CBTARGETS, and CBFLAGS in two ways:

- On the Build Detail tab, using Pre-Defined Options
- From Execute Build Jobs, using the Build Commands field

If you define values for CBMACROS, CBTARGETS, or CBFLAGS using the Pre-Defined Options, you will have to save the build job before you can execute it, so the values will be stored in the TGT file, and will apply to all build machines.

If you define values for CBMACROS, CBTARGETS, or CBFLAGS using the Build Commands field, you can execute the build job without saving it. Also, you can apply different values for each build machine.

In both cases, you should be aware of requirements for spaces and double quotes in the parameters to CBMACROS, CBTARGETS, and CBFLAGS. See ["Parameter Syntax for CBMACROS, CBTARGETS, and CBFLAGS" on page 84](#).

Finally, note that you can redefine a macro on the command-line, and it will take precedence over the value defined for CBMACROS on the Build Detail tab. However, you will usually not be able to override values for CBFLAGS and CBTARGETS in the same way.

## Entering Values Using Predefined Options

In the list of predefined options on the Option Detail tab, there are entries for CBMACROS and CBTARGETS. (CBFLAGS are listed individually.)

Pre-Defined Options				
	Option Name	Parameter	REL	DBG
<input type="checkbox"/>	cbmacros		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	cbtargets		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	-All		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	-B		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	-Batch		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	-Compile		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	-Debug		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	-DebugOn		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

To use any of these options, simply check the box to the left of the desired option, and then enter the parameter to the option in the Parameter field. Values for CBMACROS, CBTARGETS, and CBFLAGS entered in this way are recorded in the TGT file.

## Entering Values Using the Build Commands Field

If you wish, you can enter values for CBMACROS, CBTARGETS, and CBFLAGS in the Build Commands field of the Build Detail page. Use the same syntax as in the Pre-Defined Options fields.

## Parameter Syntax for CBMACROS, CBTARGETS, and CBFLAGS

When you enter values for the parameters to CBMACROS and CBTARGETS, you must be careful with spaces and quotation marks.

- Only double quotes are allowed.
- Spaces are not permitted after or preceding the equal sign ("=").
- If you need to pass a parameter that contains a quoted string itself, use the backslash character to escape the nested double quotation marks:

```
CBMACROS="x=\"nested string\" "
```

- When there is a nested string, you must include a space between the two quotation marks at the end of the parameter (as shown in the previous example).

If you make a mistake with quotation marks, a Perl syntax error is likely to result. This may appear in the build log as a failure to run the build script.

## Running a CB Script from a Command Prompt Window

- 1 Navigate to the directory where the Configuration Builder script is and execute bldmake from there.

You must execute bldmake using the syntax that appears on the Build Detail page. That is, if you are using bldmake to execute a Configuration Builder script named Script.bld that creates an executable file named MyApplication.exe, you still have to know the name of the Builder project and search path in order to call bldmake.

- 2 When bldmake has finished creating the build control file, execute om.

Again, you should execute om using the syntax on the web client's Build Detail page. That is, if you have parameters to pass to the CB script, add them to the end of the om command. Use the following syntax:

```
cbflags="-NoSilent" cbmacros="NewMessage=Done."
```

- 3 Inspect the command window transcript.

The command window transcript should show that the target named in the Configuration Builder script was actually completed.

## Converting a Simple CB Script to a Serena Builder Script

The instructions for performing this task are in the *Builder Development Guide*. See the section "Make to TGT Conversion".



# Chapter 7

## Procedures

---

This section discusses procedural information for Serena® Builder™ for Professional™.

Using the Standalone Web Client	88
Setting the Build Directory	89
Setting Up a Build	90
Using the Version Manager Integration (Desktop Client)	98
Incorporating an Ant Task into Serena Builder	107
Using Remote Build Machines	109
Viewing Reports	110
Viewing Footprint Information	113
The Serena Builder for Professional Web Client	113
Customizing Build Jobs	114
Serena Builder and Cross-Platform Development	115
Building from Different Source Revisions	115
Building From Reference Directories	116
Using Serena Builder with PCLI	117

## Using the Standalone Web Client

After you have installed the command-line client, you can use the full functionality of the standalone web client.



**NOTE** This section is designed to get you started with the standalone web client. The Builder Development Guide goes into much more detail on this topic.

- 1 Start your browser and open the standalone web client home page.
- 2 You see the Catalyst security certificate dialog box:



Click **Always**. The Java applet loads (this may take several seconds).

- 3 You see the Builder standalone web client home page.



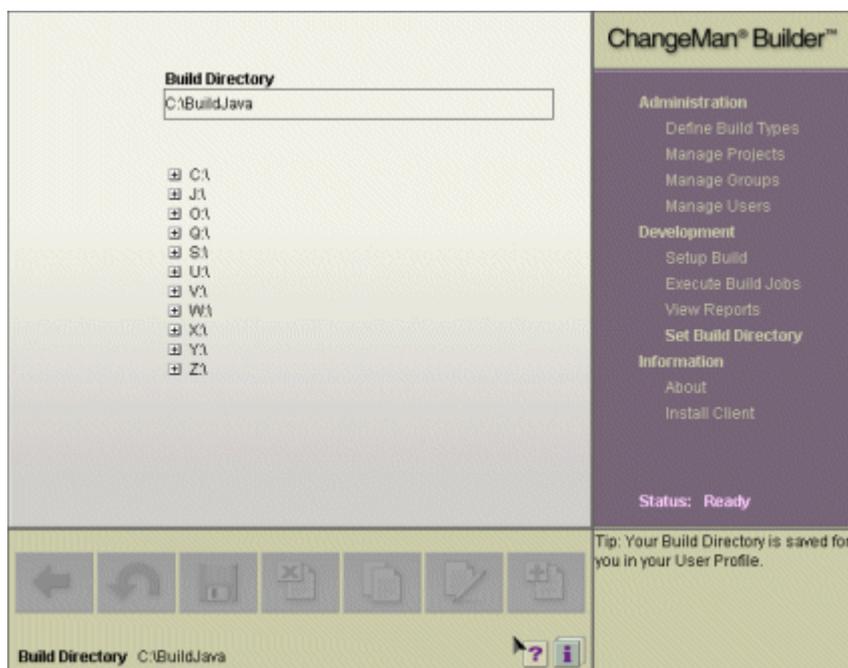
**NOTE** Enterprise Edition vs. Standard Edition—All users receive the Enterprise Edition for a thirty-day evaluation period. If after the evaluation period you choose not to purchase the Enterprise Edition, you can still use the Standard Edition.

## Setting the Build Directory

After you have installed the command-line client and before you attempt to create any new build jobs, you must first set the build directory. This is a directory into which Builder can write the results of each build.

To set the build directory:

- 1 Under the Development commands, click **Set Build Directory**. You see the following screen:



- 2 Type the location for the build directory into the text field, or browse for it using the directory trees underneath. If you change your mind, the Undo icon



will return the build directory to the last saved value.

- 3 Save the build directory by clicking the Save icon.



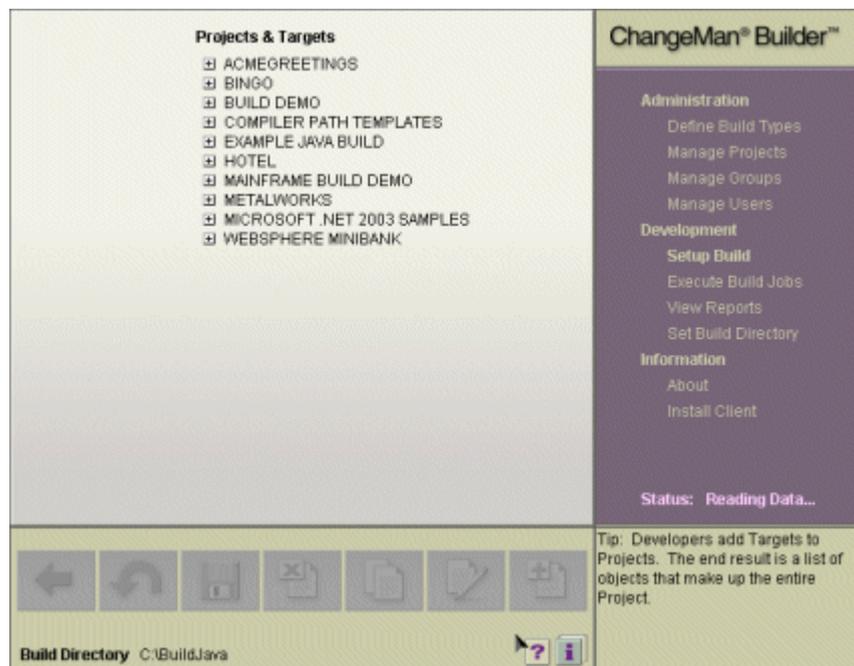
If you change your mind and wish to use a different location, simply use Set Build Directory again.

## Setting Up a Build

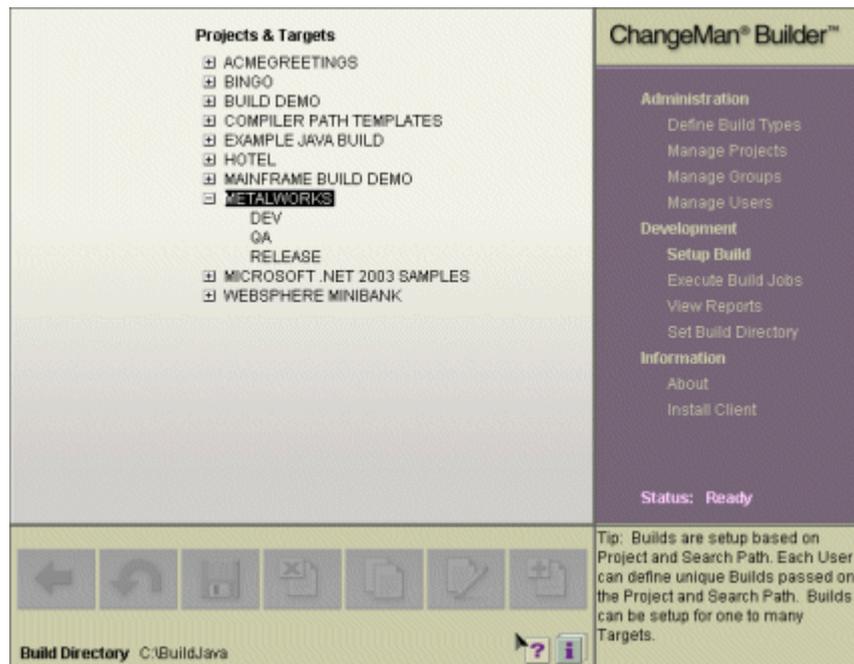
After you have set a build directory, you can set up a build.

To set up a build:

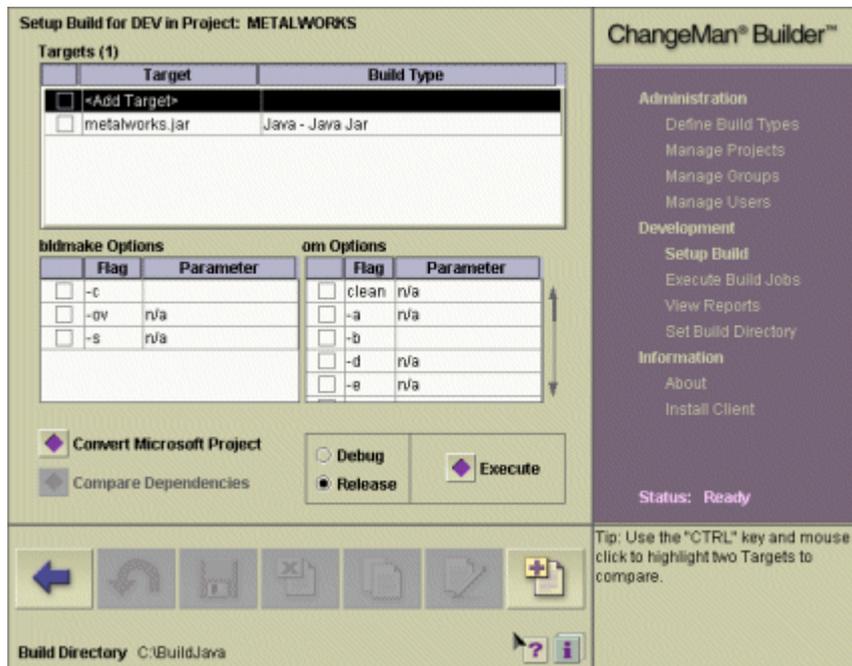
- Under the Development set of commands, click **Setup Build**. You see a list of Projects and Targets.



- Expand the project for which you wish to define a target. You see the search paths defined for that project:



- 3 Double-click the Search Path for which you wish to create a target. You see the Setup Build screen:



In the list of targets, you can see that the first entry is <Add Target>. Now you are ready to define a target.

## Defining a Target

To define a Target, you must fill out the fields on the Target Overview page and the Target Detail page.

## Filling out the Target Overview page

- 1 Double-click the **<Add Target>** entry on the Setup Build page. You see the Target Overview page:

- 2 Type a name for the target. The name for the target file is automatically filled in, though you can change it if you want to.
- 3 Select the operating system for the target. The default is the operating system you are using, but if the target is to be built on a remote machine, you should select the operating system of that machine. (If you are using Serena Builder Standard Edition, the operating system will default to Windows.) Also note that Java is listed as its own operating system.
- 4 Select the build type for the target. These build types are already set up for you by your Administrator.
- 5 If you wish to save intermediate targets (for example, object files before linking) in a directory separate from the build directory, type that name into the Intermediate directory field.



**NOTE** You should use relative paths—such as `project1\foo.java`—because Builder will concatenate these paths with the Search Path directories when looking for dependencies.

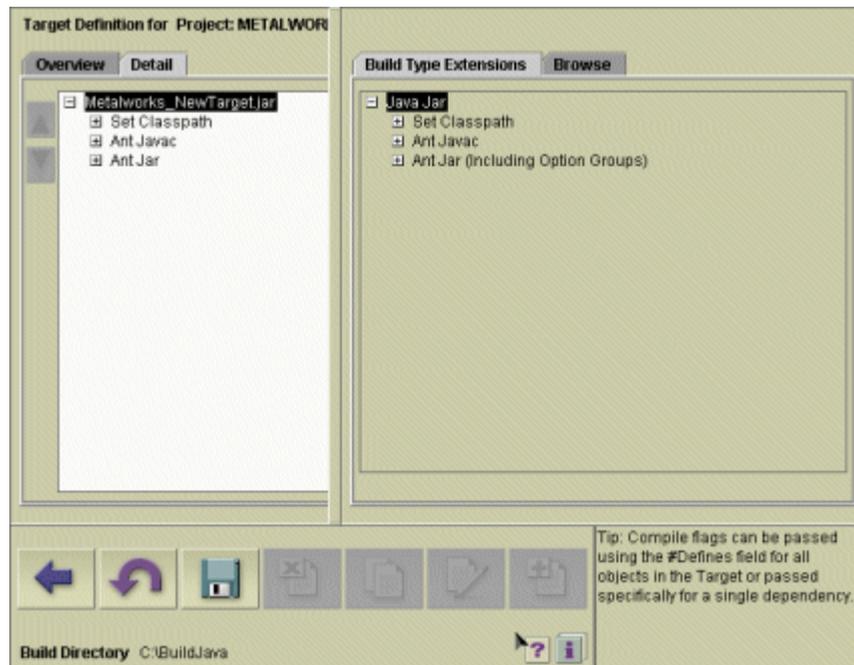
- 6 You can optionally enter any pre-compile parameters. Be aware that any parameters entered here are for pre-compile steps, and are passed to every dependency. If you wish to specify individual parameters for each dependency, you can do that on the Target Detail page.
- 7 You can also optionally enter any additional flags meant for the compiler, such as `#defines`. Once again, parameters entered here are passed to every dependency. To specify compiler flags specific to each dependency, enter those flags on the Target Detail page.

- 8 Finally, if the target should not be included every time there is a full build, you can select the **Exclude from full build** check box. An example of this would be if the target defines a large file such as a releaseable disk image--it may not be necessary to build this every night.

After you have entered at least the target name and target file name on the Target Overview page, you can fill out the fields on the Target Detail page. You don't have to save the target file first.

### **Filling out the Target Detail page**

- 1 Click the **Detail** radio button on the Target Overview page. You see the Target Detail page.



You see a hierarchy with your target at the top, and the build tasks associated with that type of target underneath.

On the right, the Build Type Extensions tab displays. This tab shows you the file extensions that are permitted for the selected build type. If you selected "Java Jar",

for example, and expanded the "Ant Javac" build task in the Build Type Extensions hierarchy, you would see this:



These represent the types of files that can be used as input files for the type of target you have chosen.

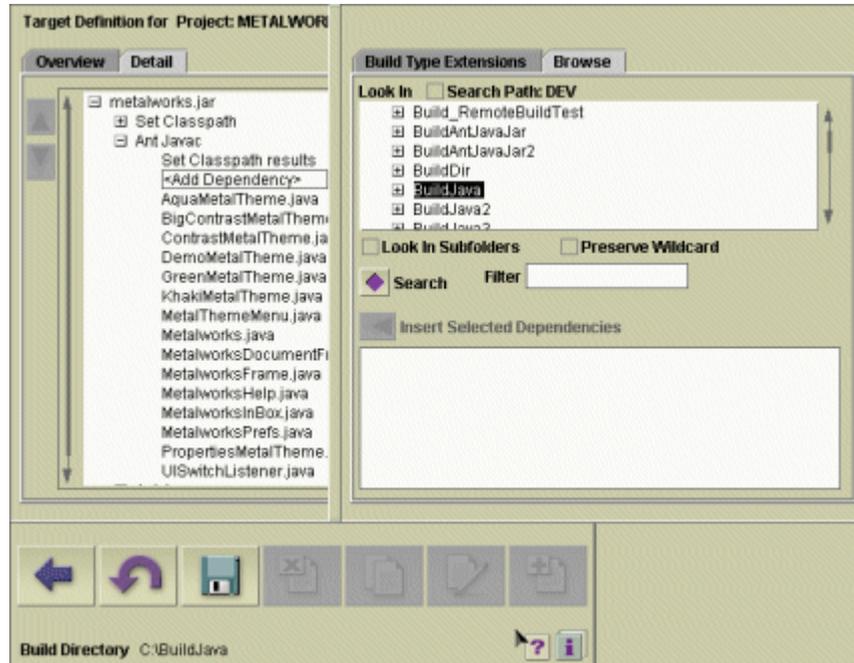
- 2** Go back to the Detail tab, on the left side of the page. Expand the hierarchy. You should observe the following behavior:
  - a** When you select an item such as <Add Dependency> , the right-hand tabs change to Build Type Extensions and Browse.
  - b** When you select the name of a build task, the name of a file, or an item representing the results of a previous build task (such as Set Classpath results), the right-hand tabs change to Option Detail and Browse.

The Option Detail page allows you to enable the pre-defined options for a build task or for a specific file. The pre-defined options are those that have been defined by your Administrator.

You can also enable custom options, if you know the names of the options. The names follow the Ant conventions, and can be seen by examining the example build jobs.

See ["Using Options" on page 97](#).

- 3 To browse for a specific file, use the Browse tab on the right-hand side of the screen:



**NOTE** If you know the name of the item, you can type it in instead of browsing for it.

- 4 Use the Look In tree and Search Filter to browse for the source files that make up this dependency. Select the directory that contains the source files, then click **Search**. The possible items appear in the lower list box.
- 5 Select the dependencies that should be included, then click the Insert Selected Dependencies arrow. The dependencies appear in the Detail tab, under the current build task.



**TIP** When you add

- 6 If you change your mind about using a dependency, select it from the Detail tab, then click the Delete icon. This will remove the dependency from the build task. (It will not delete the item from disk.)
- 7 When you are done adding the dependencies, click the Save icon, then click the Back icon to return to the Setup Build page. From there you can execute the build job.



**NOTE** You may want to rearrange the order of the dependencies. Use the up and down arrows next to the build task hierarchy to accomplish this.

### **Which Files Should Be Dependencies?**

You do not have to specify every single source file as a dependency. Here are some tips for specifying dependencies.

- You do not have to specify any file that is named as an included file.

- You can use wildcards to specify a group of files. (Java developers can use the "double-star" wildcards familiar to Ant users. See ["Java Developers: Using Double-Star Wildcards" on page 97.](#))

## Java Developers: Using Double-Star Wildcards

Several of the examples use the traditional "\*" wildcard to indicate that all files of a certain extension should be compiled:

```
<path>/top_level_dir/intermediate_directory/*.java
```

In Java packages, however, there are many intermediate-level directories, and you might end up with this:

```
<path>/top_level_dir/second_level_dir1/*.java
<path>/top_level_dir/second_level_dir2/*.java
<path>/top_level_dir/second_level_dir3/level3_dirA/*.java
<path>/top_level_dir/second_level_dir3/level3_dirB/*.java
```

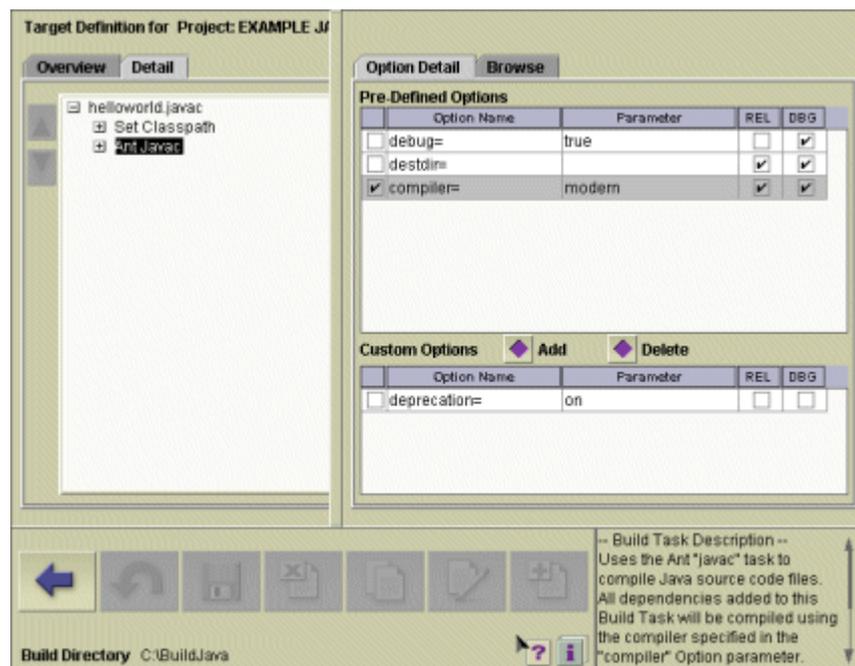
Serena Builder allows you to specify a "double-star" wildcard, to allow the search for source files to explore multiple directories without having to name them individually:

```
<path>/top_level_dir/**/*.java
```

This dependency specification will match all of the intermediate directories given in the previous example.

## Using Options

Serena Builder allows you to specify options for a given build task. The Detail tab on the Target Definition page allows this:



When you select a build task or dependency, the right-hand side of the screen shows you the Option Detail tab. This tab has sections for Pre-Defined Options and for Custom Options.

Pre-defined options are those that have been defined by the Builder administrator. These options can be applied to a build task, a build rule, or to a build type. If you do not have Administrator privileges, you will not be able to delete or add to these options.

Custom options are options that you add, to a build task or individual dependency. These options follow the Ant format. For example, the switch to list deprecated classes in Java should take this form:

Option Name	Parameter
deprecation=	on

Your Builder administrator defines which options are available. If your administrator has permitted it, you can choose whether or not to use the options, and whether or not they apply to Release builds, Debug builds, or both. See the *Serena Builder Administration Guide* for details.

## Archiving the Target Files

Once you have created the target files, you must add them to your Version Manager archives if you wish to use the Version Manager integration. Use the standard process of adding workfiles to archives; if you are unfamiliar with this, see the *Version Manager User's Guide* or online help.

### **Use 1:1 Ratio for Version Manager Projects and Serena Builder Projects**

In each Version Manager project, you should store only the target files for that one project. Builder does not examine sub-projects for target files, so the target file should define which subdirectories need to be examined for source files.

## Using the Version Manager Integration (Desktop Client)

Once you have set up target files and have checked them into Version Manager, you can then use the Version Manager integration to submit build jobs from within Version Manager.

There are some limitations:

- You cannot create or edit search paths or target files from within Version Manager.
- You can create or submit build jobs.
- The integration will not use workspace settings when retrieving files.

Even if you have a Version Manager workspace defined that causes files to be checked out to a specific workspace, the integration will always use the project hierarchy when retrieving files. The Build Directory becomes the root of the project, and the project hierarchy applies starting from there.

Following are procedures detailing the steps for using the Builder desktop client Version Manager integration. As you will see, the search path strategy in the desktop client is a

little different than in the web client. For an overview of the process, see ["Example of Using the Serena Builder Desktop Client"](#) on page 56.

## Setting Up the Project

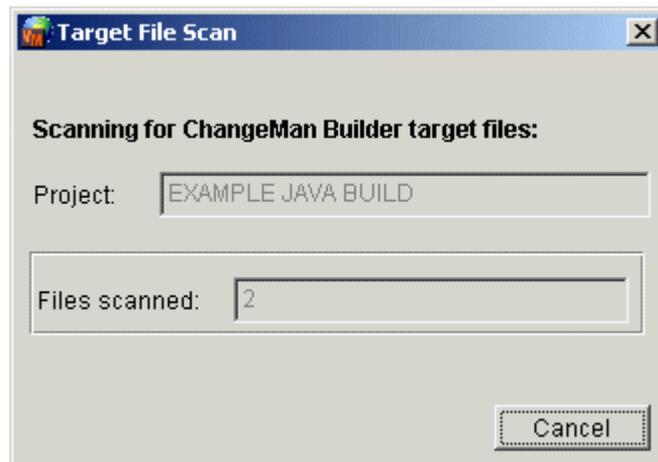
Before you can build a project, you must make sure certain conditions are met:

- 1 Verify that your Builder administrator has set up a proper project for your build process. That is, if you wish to build a Java project, be sure that there is a project that calls the desired Java compiler and has the proper search path and other parameters defined.
- 2 In the web client, define the targets you wish to build. This will create a target definition file (.TGT file).
- 3 Check the target definition file into Version Manager, alongside the source files. You should not need to check it out again unless you wish to change it.

## Building the Project

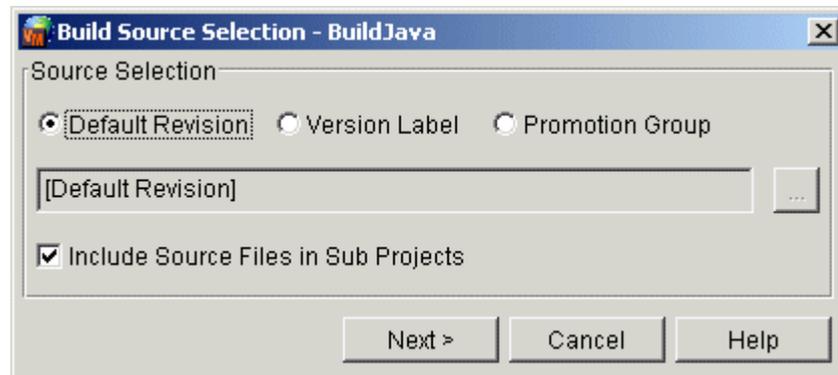
After you check in a set of changes to a given project, follow these steps to build the project:

- 1 From Version Manager, choose the Actions | Build menu item. Builder scans the project for Builder target definition files:



**NOTE** If you have not checked in the target definition files, Builder will display a message at this point indicating that it could not find any target definition files in the project.

- 2 After the scanning completes, you see the Builder Source Selection dialog box:



## Selecting Source Revisions

When you are looking at the Builder Source Selection dialog box, your aim is to specify whether you wish to use the default revision, a revision with a particular label, or a particular promotion group.



**NOTE** If you are unfamiliar with the concepts of labeling and promotion groups, see the *Version Manager User's Guide*.

Choose **Default revision** when you have not used labels nor promotion groups, and are interested in using the latest revision. When you select this option, the text field displays [Default revision]. Both the text field and the browse button are disabled.

Choose **Version label** when you have applied labels to your source code and wish to build only files with that label. When you select this option, the text field and browse button become enabled. Clicking the browse button displays the "Select a version label" dialog box, from which you can select a file and then choose from the available labels.

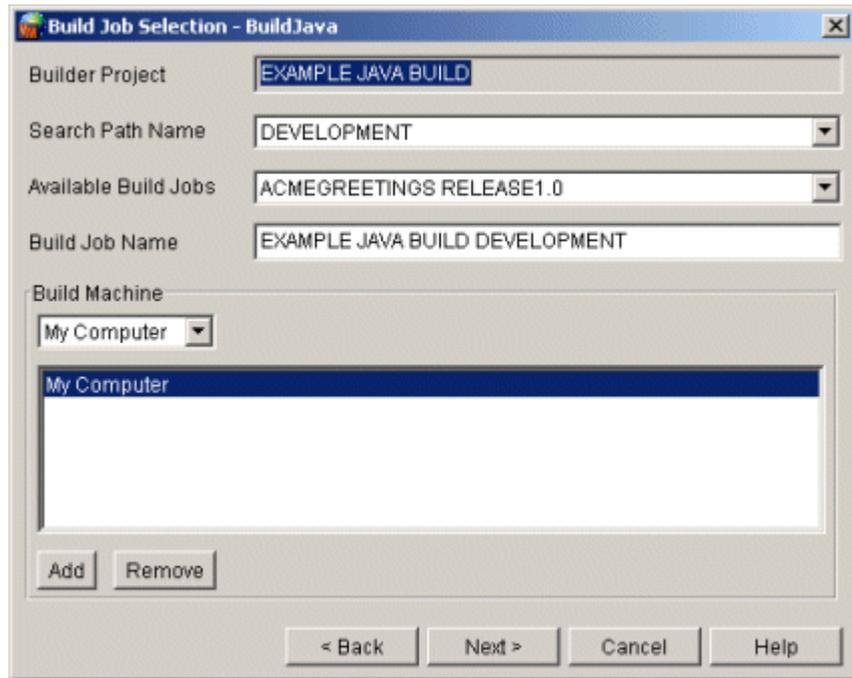
Choose **Promotion group** when you have created promotion groups and wish to build only files in a particular promotion group. (Note that the .TGT file must also be assigned to the promotion group.) When you select this option, the text field and browse button become enabled.

Choose **Include source files in subprojects** when you wish to build all of the files in the subprojects, as opposed to just those at the top level Version Manager project.

When you have finished, click **Next** to move to the Build Job Selection dialog box.

## Creating a Build Job

After you have specified how source files should be selected, you are ready to create a build job. This is done with the Builder Job Selection dialog box:



The dialog box allows you to select from previously-created build jobs. It also allows you to create a build job if none currently exist; the steps for this task follow.

To create a new build job:

- 1 Verify that the **Project** field lists the correct project.
- 2 Select a search path name from the list. These search paths are already set up for you by the Administrator. They point to specific directories, which may represent different operating systems or different release versions.



**NOTE** Once you select a search path, the **Build job name** field is populated, because the default build job name is the project name followed by the search path name. If there were existing build jobs, selecting one from the **Available build jobs** list would replace the default build job, and all subsequent fields would be filled in with the previous settings.

- 3 If you wish to change the name of the build job, type the new name into the **Build job name** field.
- 4 Use the **Build machine** list to locate a build machine from the list. This list displays all available machines on the Builder Knowledge Base server.
- 5 Use the **Add** button to add the selected machine to the large list box.

- Repeat until all desired machines for this build job have been added. The machines in the list box are the ones that will appear in the **Build machine** lists in subsequent dialog boxes.

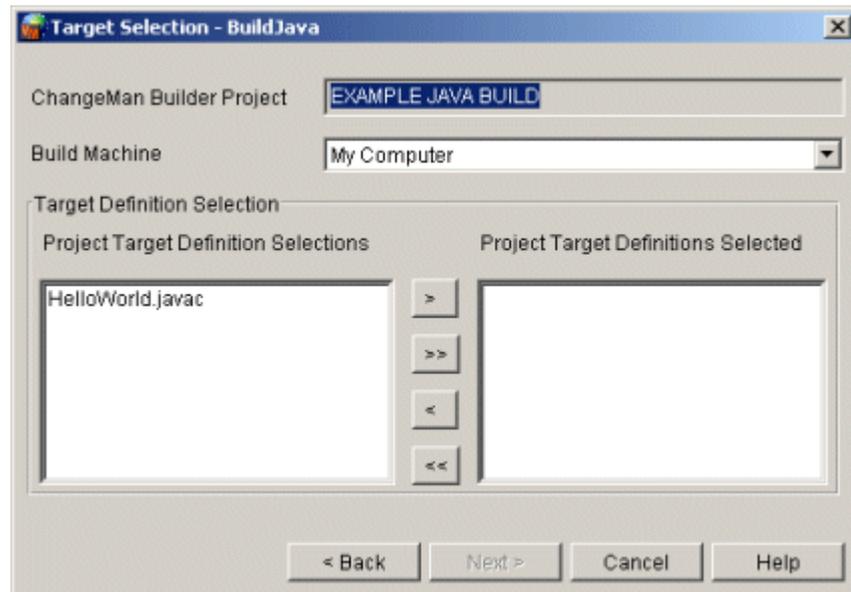


**NOTE** Settings in subsequent dialog boxes are based on the currently selected build machine. If you change build machines, you should revisit other settings as well.

- Click **Next** to move to the Builder Target Selection dialog box. (If Next is grayed out, try clicking the name of the build machine.)

## Selecting Targets to be Built

After you have created or selected a build job, you are ready to select the targets to be built. This is done with the Target Selection dialog box:



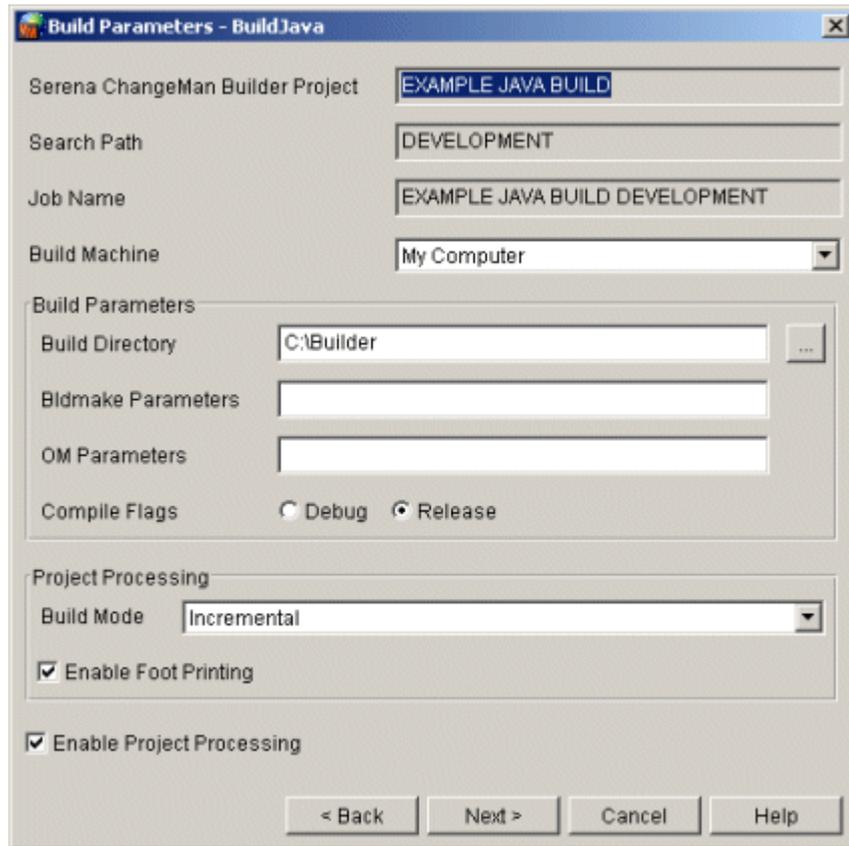
Be aware that the targets change with each build machine—targets are specific to a particular operating system and set of tools.

To select targets:

- Select a build machine from the list. The selected build machine has associated targets; these targets appear in the **Project Target Definition Selections** list.
- Use the arrow buttons to move specific targets from the **Project Target Definition Selections** list to the **Project Target Definitions Selected** list. To add all targets, click the >> button.
- If you wish to remove any targets, use the < button, or the << button to remove all targets.
- When you have finished selecting targets, click **Next** to move to the Parameters dialog box.

## Specifying Build Parameters

After you have selected the targets to be built, you are ready to specify build parameters. This is done with the Build Parameters dialog box:



To specify parameters:

- 1 Select a build machine from the list.
- 2 Make sure that the build directory points to the location of your source files. If necessary, type the directory path, or use the browse button to locate the correct build directory.

The build directory field defaults to C:\Builder on Windows, and \$HOME/Builder under UNIX.

If you are creating a remote build, you must make sure the build directory path is one that can be recognized by the remote machine. See ["Archiving the Target Files" on page 98](#).

- 3 Accept the current value for bldmake parameters, or change them as desired.

As described earlier, bldmake is one of the executables that make up Serena Builder for Professional. It has its own set of parameters, which are described in the *Builder Development Guide*.

The parameters that display by default are those that were specified during the creation of the build job. Showing the bldmake parameters here gives you a chance to modify them.

- 4 Accept the current value for om parameters, or change them as desired.

As with bldmake, om is another executable that is part of Serena Builder for Professional. Its parameters are also described in the *Builder Development Guide*.

- 5** Accept the current setting for **Debug** or **Release** compiler flags, or change the setting as desired.

The setting of this control is again specified initially during the creation of the build job, but you can modify it here.

- 6** Accept the default setting of **Enable project processing**, or disable it.

When you leave project processing enabled, the following is true:

- A bldmake pre-event trigger retrieves the selected Builder target file revisions into the build directory.
- The bldmake pre-event trigger also retrieves all project source file revisions into the build directory.
- You can select the build mode (see ["Selecting a Build Mode" on page 104](#)).
- You can enable footprint processing, which extracts revision information from the Version Manager archives and embeds it into the final build product (for .EXE and .DLL files only). This can then be queried later; see ["Viewing Footprint Information" on page 113](#).

If you do not enable project processing, then the following is true:

- A bldmake pre-event trigger retrieves the selected Builder target file revisions, but not the source file revisions, into the build directory.
- You cannot select the build mode, which means that Builder will use only the files already populated in the build directory to perform the build.
- You cannot enable footprint processing, so no revision information will be embedded into the .EXE or .DLL files.

### **Selecting a Build Mode**

When Project Processing is enabled, you can select a Build mode. This allows you to control how the build handles existing work files and revisions.

<b>Build Mode</b>	<b>Description</b>
Incremental	Checks out revisions from the archives if the date and time of the archive revisions are newer than the existing work files.
Exact	Checks out revisions from the archives if the date and time on the archive revisions does not match exactly the date and time on the existing work files.
Clean	Deletes existing build directory structure before retrieving revisions from the archives. Note: if you have a file editor or command shell open to the existing directory, that may prevent Clean from completing.

Build Mode	Description
New	Builds only if the named build directory does not exist. This keeps a user from overwriting existing directories. If you wish to perform another build on a build marked New, you must delete any existing build directories first.
Rebuild	Rebuilds all intermediate and final targets. Does the same check for revisions as the incremental build, but then forces a rebuild of the final targets and deletes intermediate files.

Be careful when using a Clean build! Be sure you know where the build directory is, since that directory will be emptied.



**NOTE** You might be tempted to add a "cd" command to the Build Commands field, in order to change to a specific directory. If you do this, do not place the "cd" command before the bldmake command, or the Windows version of Builder will report a "Directory deletion failed" error.

After you have specified all of the parameters, click **Next** to move to the Builder Submission dialog box.

## Submitting a Build

After you have specified build parameters, you are ready to submit the build. This is done with the Submission dialog box:

Submission - BuildJava

Builder Project: EXAMPLE JAVA BUILD

Search Path: DEVELOPMENT

Job Name: EXAMPLE JAVA BUILD DEVELOPMENT

Job Availability: PRIVATE

Build Machine: My Computer

Build Commands:

```
bldmake "EXAMPLE JAVA BUILD" "DEVELOPMENT"
om -e all CFG=RELEASE
```

< Back Submit Save Cancel Help

To submit a build job:

- 1 Select whether the job should be **Private** or **Public**. Private build jobs can be executed only by the person who created the build job. Public build jobs can be executed by and are visible to everyone. You can save a job only as a private build job.
- 2 For each **Build machine** used in the build:
  - a Verify that the build machine is correct.
  - b Examine the **Build commands** for the correct bldmake and om commands. This field can contain other commands. For example, if you wanted to automatically launch a text editor when the build completes, you could add a command to do that.



**NOTE** Be aware that deleting the existing bldmake and om commands can cause the build to fail. You can change, delete, or add to the parameters to either bldmake or om. Also, Serena adds some commands and parameters (to bldmake and om) that are not visible in this field.

- 3 When you are satisfied with the contents of the dialog box, submit the build job by clicking **Submit**. If you wish to save the build job and submit it later, click **Save**.
- 4 If there are no problems with the submission of the build job, a dialog box indicates that the build job has been successfully submitted. This does not indicate that the build job has completed successfully. The dialog box indicates how you can view the log (one per build machine) and reports when the build has completed (see "[Viewing Reports](#)" on page 110).



**NOTE** After you save a build job in the Version Manager integration, it is no longer modifiable from the standalone web client. Although the standalone web client will still allow you to make changes to the build directory and other parameters, the build job will still execute according to the parameters defined in the Version Manager integration. These parameters are stored in the VMMBI variables that appear in the Build Commands field. If you remove the VMMBI variables (and some other additions to the Build Commands field), then you can modify the build job in the standalone web client again. See "[Removing Version Manager Integration Variables](#)" on page 106.

## Removing Version Manager Integration Variables

If you wish to return a build job to a state where it can be modified in the standalone web client, you must remove the Version Manager integration variables from the Build Commands:

- 1 In the standalone web client, click **Execute Build Jobs**.
- 2 Select the build job to be modified from the list of build jobs. The Build Detail page appears.
- 3 In the Build Commands field, remove every line that begins with SET VMMBI.
- 4 If you are using a UNIX platform, you should also remove the line that begins with "export". It follows all of the SET VMMBI definitions.
- 5 In the line that begins with bldmake, delete everything from -rp to the end of the line.

- 6 In the line that begins with om, delete everything from -vp to the end of the line.
- 7 Save the build job.

The build job now will be modifiable in the standalone web client.

## Workspace Info Not Used in Version Manager Integration

The Builder integration does not make use of workspace information. That means that for a given Version Manager project, Builder will retrieve files based on the project hierarchy, and will disregard any specific workfile locations that may have been set in a workspace.

For example, consider the following Version Manager project:

```
FoucaultApplication
+Module1
--File1.c (in workspace, checks out to C:\Test)
--File2.c (in workspace, checks out to C:\Test)
--Module1.TGT
+Module2
```

In this example, a user has defined a workspace and within that workspace, has set two of the files to check out to a specific workfile location (C:\Test). Using the Serena Builder integration to Version Manager, however, the files would be retrieved to the top of the project structure. In this case, that would be the build directory for Module1.

The top of the project structure is considered to be wherever the .TGT file is.

## Build Directory Embedded into Integrated Build Jobs

When you save a build job in the Version Manager integration, the build directory is saved with that build job. This means that under certain situations, if you try to change the build directory, you will not be able to.

- If you try to change the build directory by adding a "cd" command to the Build Commands field of the Submission dialog, that will not work (and may cause problems; see ["Build directory deletion fails on clean build in Windows Version Manager Integration" on page 123](#)).
- If you use the standalone web client to open a build job that previously has been saved in the Version Manager integration, the build directory set in the Version Manager integration will be retained. If you want to change the build directory, you must change it in the Version Manager integration, or you must delete all of the Version Manager integration variables in the Build Commands field (see ["Removing Version Manager Integration Variables" on page 106](#)).

# Incorporating an Ant Task into Serena Builder

This feature is available only in Serena Builder for Professional Enterprise Edition.

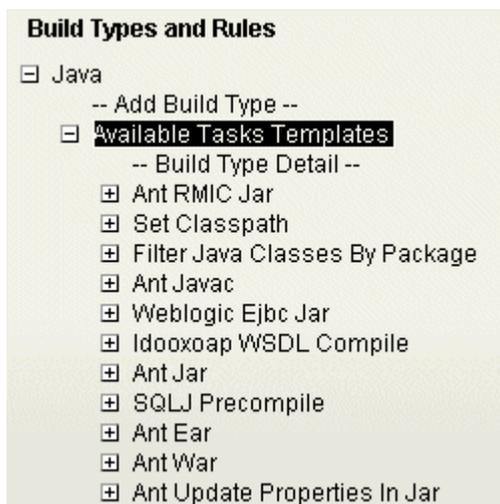
In earlier versions of Builder, many Ant tasks were available under the Target Detail page. Now, however, the build tasks for each type of build job are templated, and only the appropriate tasks appear. Each build task corresponds to an Ant task.

If you have Administrator privileges, you can still view the list of Ant tasks and add them to a particular build type.

To view the list of available tasks, click **Define Build Types** in the Administration menu on the Builder standalone web client home page. A list of build types and rules appears.



If you expand the Java entry, and then the Available Tasks Templates entry, you will see the list of available Ant tasks:



If you require a task that is not included in the existing set of tasks, it is possible to add the task. This requires creating a Perl command script and an XML file to describe the task to the Knowledge Base server.

Creating your own command script requires Perl proficiency and is beyond the scope of this manual. See the "Scripting" section of the *Builder Administration Guide*.

# Using Remote Build Machines



**NOTE** This feature is available only on the Enterprise Edition of Serena Builder, and also requires the purchase of remote build server licenses.

If you are using the Enterprise Edition of Serena Builder, you can run builds on machines other than your own. These machines are known as remote build machines or remote build servers.

## Setting Up Remote Build Machines

Remote build machines must be set up at install time. Once this is done, the Builder Knowledge Base server becomes aware that they are available, and henceforth they appear in the list of available build machines. The details of the remote build server install process are covered in the *Builder Installation Guide*.

## Setting Up Remote Builds

When creating a build that will run on a remote machine, the build directory path must be in a form that can be recognized by the remote machine.

In Windows, if the remote build machine is mapped to a local drive on your machine, you can use the browse button to locate the build directory. However, you will have to edit the path so that it is recognizable to the remote machine.

Another way to specify build directories is to use an environment variable. For example:

```
%SERENA_CM_BUILD_DIR%\build_project\dev
```

## Remote Builds and Permissions

Permissions can cause a remote build to fail. If the process running the remote build does not have permission to reach the Version Manager archive databases (PDBs) or the directories where tools such as compilers are located, the build will fail.

To avoid this problem, you should do the following:

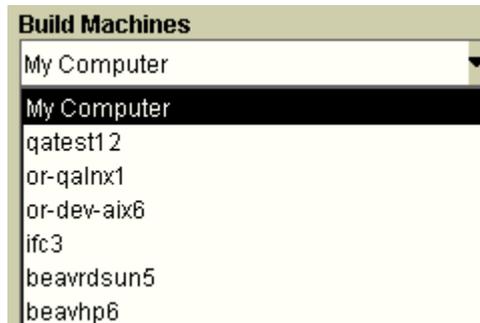
- Make sure there is a particular user ID that has access to the Version Manager PDB files; for example, the pvcs user ID.
- On Windows, open the Services dialog box (from Control Panel | Administrative Tools) and check the properties of the "Openmake Build Server" service.
- You should log in as the user ID of the remote process to check permissions.

## Creating a Remote Build

To create a remote build:

- 1 Find out which machines on your network are available as remote Builder servers, and verify that they have Version Manager, the proper compilers, and any other necessary software installed.

- 2 If you are using the web client, verify that you have permission to copy your source files to these machines; if you are using the desktop client, this is unnecessary.
- 3 Have the Builder administrator define a search path that points to a build directory on the remote build machine.
- 4 Set this directory as the build directory in Builder.
- 5 Set up the build job normally.
- 6 When you execute the build job, select the remote build machine from the list of available machines:



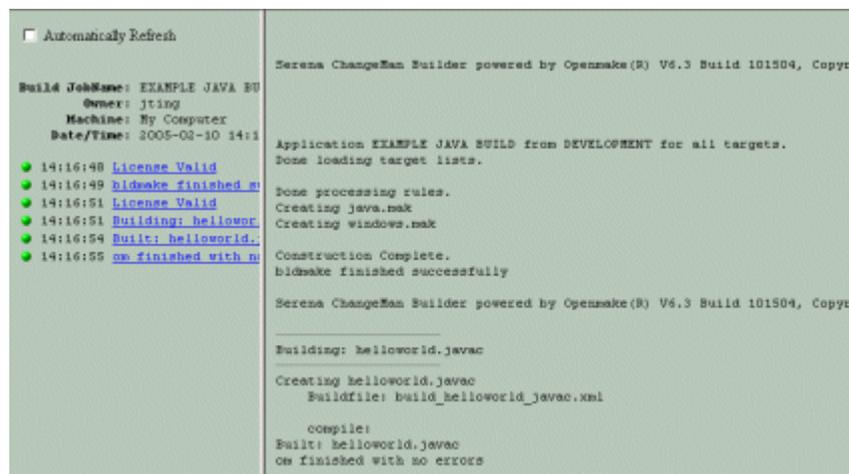
- 7 In the Build Commands field, add the commands necessary to change the current directory of the remote machine to the directory containing your source files. Add these commands before the line containing bldmake. For example:

```
cd /d C:\data\Build_Directory\Test1
bldmake "EXAMPLE_TEST1" "RELEASE"
```

- 8 Save the build job.
- 9 Execute the build job.

## Viewing Reports

After you have successfully submitted a build, Builder launches a log browser for the just-submitted build job:



The browser has two panes:

- The left-hand pane shows the name of the build job, a summary of the build activities, and a check box controlling whether or not the browser should be automatically refreshed.
- The right-hand pane displays messages describing the progress of the build. If you see the message "No information has been logged at this time", it commonly means that the build has not yet completed. If the build has completed, you will likely see the message "om completed successfully."

## Viewing the Log for a Previous Build

If you want to see the log for a build performed in the past, click **View Reports** on the Builder web client, then locate and double-click the build in the tree of reports. A new log browser window displays the build log.

To view the log for a build performed in the past:

- 1 Click **View Reports** on the Builder web client.
- 2 Double-click **Build Logs** to expand its list of entries. You see a list of date entries (Today, Yesterday, and so on). Date entries that have build logs appear with plus signs next to the name.
- 3 Locate and expand the desired date entry. You see a list of build machines. If you are building only on your own computer, "My Computer" will be the only entry.
- 4 Double-click the desired build machine entry to find the desired build log. The build logs are named with timestamp information.
- 5 Double-click the build log to view it again.

## Impact Analysis

This feature is available only with the Enterprise Edition of Serena Builder for Professional.

Impact analysis allows you to predetermine which files will be affected by a change. You can see which files are used to create the result of the build, or you can see which build result is affected by the file in question.

Builder calls these two types of impact analysis reports **implosion reports** and **explosion reports**:

## Enabling Impact Analysis

To view impact analysis reports, you must generate the information at the time of the build. To do this, edit the build job so that the `-g` parameter is passed to om:

om Options		
	Flag	Parameter
<input type="checkbox"/>	-d	n/a
<input type="checkbox"/>	-e	n/a
<input checked="" type="checkbox"/>	-g	n/a
<input type="checkbox"/>	-j	n/a
<input type="checkbox"/>	-n	n/a

## Implosion and Explosion Reports

Both types of reports provide information about what files will be affected by a change to the source code. The main difference is where the reports begin.

Implosion reports start with the files that were used in the build and show you how those files affect the build result.

Explosion reports start with the build result and expand to show you the files that made up the build result.

To view either type of report:

- 1 Click **View Reports** on the Builder web client.
- 2 Double-click **Impact Analysis** to expand its list of entries.
- 3 Double-click the date entries (Today, Yesterday, and so on) to find the desired impact analysis report. The reports are named with timestamp information.
- 4 Select the desired report, then click either the Implosion or Explosion icon.

Here is a sample Implosion report:

Start imploding from here:				
File	Size	Date	Time	View Source
<a href="#">HelloWorld.java</a>	353	02/10/2005	14:17:20	<a href="#">C:\BuildJava\HelloWorld.java</a>

Start imploding from here:				
File	Size	Date	Time	View Source
<a href="#">helloworld.java</a>	353	02/10/2005	14:17:20	<a href="#">C:\BuildJava\helloworld.java</a>

Start imploding from here:				
File	Size	Date	Time	View Source
<a href="#">rt.jar</a>	26347933	05/30/2003	12:32:37	<a href="#">C:\j2sdk1.4.2\jre\lib\rt.jar</a>

Start imploding from here:				
File	Size	Date	Time	View Source
<a href="#">helloworld.classpath</a>	32	02/08/2005	15:40:21	<a href="#">C:\BuildJava\helloworld.classpath</a>

The blue links lead to pages with more detail. On the detail pages you will see the following notes:

When you see...	It means:
Current File	The starting point of the analysis.
Impacts Current File	These files affect the Current File.
Current File Impacts	These files are affected by the Current File.

Here is how the report (in this case an Explosion report) is expanded after you click on a blue link::

Current File:				
File	Size	Date	Time	View Source
<a href="#">helloworld.javac</a>	9	02/10/2005	14:17:44	<a href="#">C:\BuildJava\helloworld.javac</a>

Current File Impacts:				
File	Size	Date	Time	View Source
<a href="#">helloworld.java</a>	353	02/10/2005	14:17:20	<a href="#">C:\BuildJava\helloworld.java</a>
<a href="#">helloworld.classpath</a>	32	02/08/2005	15:40:21	<a href="#">C:\BuildJava\helloworld.classpath</a>

## Viewing Footprint Information

Footprint information in .EXE or .DLL files may be viewed using the `omident` utility in the `\bin` directory underneath the Builder command-line client install directory.

From a command prompt, type:

```
omident <name of final target executable>
```

This will print the footprinting information to standard output.

## The Serena Builder for Professional Web Client

The Serena Builder for Professional Web client is to be used with the Version Manager web client. Its dialogs and functionality should be identical to the desktop Version Manager integration, except that you cannot execute a local build.

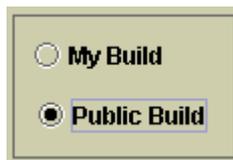
Therefore, if in the Web client you define a build job that names "My Computer" as the build machine, you will not be able to execute that job.

## Customizing Build Jobs

When you click **Execute Build Jobs**, two lists of build jobs appear, Public Builds and My Builds. Normally you select a build job from one of the lists, and then click the Execute icon. However, if you edit the build job, either by double-clicking the name of the build job or by clicking the Edit icon, you can customize the build job in the following ways.

### Public vs. Private Build Jobs

By default, build jobs that you create are private, meaning that they are not executable by anyone other than you. You can make your build jobs available to others by specifying that they are Public:

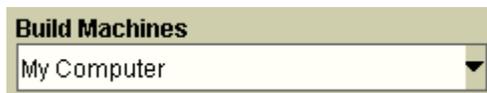


Some caveats:

- Once a build job has been made public, it cannot be made private again.
- If a build job requires access to specific archives, those archives must also be available to everyone, or at least to users executing the build job—otherwise the build job will fail.
- If you are the creator of a private build job that has been made public, you retain your ability to modify that build job, even if you do not have permissions to modify any other public build jobs.

### Changing the Build Machine

If you have the Enterprise Edition of Serena Builder, your Builder administrator has the ability to set up multiple build machines. If you click **Execute Build Jobs**, select a build job, and then expand it, you will see a pulldown list of build machines on the Build Detail page :



By selecting a different build machine, you can cause the build to be executed on that machine, but:

- The specified build directory must exist on that machine.
- If the build machine uses a different operating system or platform, you may have to specify new build targets.

### Changing the Build Job Name

You can also change the build job name on the Build Detail page. You might want to do this to help with debugging a build job, or if you are trying to create a build job by modifying an existing one.

## Changing Bldmake or Om Flags

In the Build Commands field on the Build Detail page, you can see the bldmake and om commands that Builder will use to execute your build job. These two commands have many options, and it is beyond the scope of this manual to discuss them all.



**TIP** You can read about these options in the *Builder Development Guide*; also, typing `bldmake -?` or `om -?` at a command prompt will display the options for the command.

However, here is an example of how you might want to modify the build commands: If you want to add Impact Analysis to a build job (Enterprise Edition only), you could do that by modifying the om command to `om -g`.

To do this, click to set the insertion point on the command containing om, then edit the command. When you modify the build commands, the disk icon becomes active, allowing you to save the build job if desired.

## Serena Builder and Cross-Platform Development

If you plan to use Builder to develop programs on multiple platforms at once, please be aware of the following two suggestions:

- If you use Builder with either of the Version Manager-integrated clients, you must make sure that the Version Manager project databases (PDBs) have been set up properly for cross-platform access. See the *Version Manager Administrator's Guide* for instructions on this process.
- Builder comes with a file that can help with path mapping issues that sometimes arise in cross-platform development. Check the `VMMBI_bldmake_PET_INI.pl` file, which can be found in:

Windows:

```
<Version Mgr Install Dir>\Serena\vm\common\bin\win32
```

UNIX:

```
<Version Mgr Install Dir>\Serena\vm\<platform>\bin
```

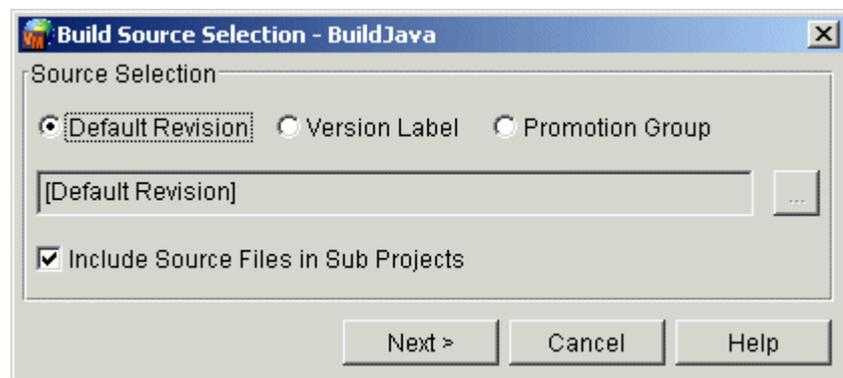
This file will contain the most up-to-date information regarding path mapping for cross-platform development.

## Building from Different Source Revisions

You can use Serena Builder for Professional to build the same set of source files in different ways:

- You can build the default revision (usually the tip, or default version label)
- You can build from a specific version label
- You can build from a specific promotion group

The choice is determined by choosing the appropriate radio button on the Source Selection dialog box.



## Building from the Tip

To build from the tip or default version label, click **Default Revision** on the Source Selection dialog box.



**NOTE** In Version Manager, the default revision is typically the latest revision; however, it can be a version label. The default revision is defined in the workspace setting or a configuration file. See the Version Manager online help.

## Building from a Version Label

To build from a specific version label, click **Version Label** on the Source Selection dialog box, then enter the name of the version label, or locate it using the Browse button.

## Building from a Promotion Group

To build from a specific promotion group, click **Promotion Group** on the Source Selection dialog box, then enter the name of the promotion group, or locate it using the Browse button.

Be aware that the .TGT file must be assigned to the same promotion groups as the source files that you are building, or the build will fail.

## Building From Reference Directories

Serena PVCS Version Manager is able to define reference directories. A reference directory provides a central repository of workfiles for browsing, printing, or copying. These directories automatically store a copy of each workfile each time the workfile is checked into an archive.

Thus, if you add the reference directory to your search path, you can build your project without having to check out revisions.

Some of the examples in the Builder examples directory use an environment variable named REFDIR to define the root of the reference directory. Subdirectories underneath the reference directory then refer to QA or dev, allowing different teams to build different versions of the product.

Your Builder administrator should let you know if you need to define a REFDIR environment variable.

## Using Serena Builder with PCLI

Serena Builder for Professional is available from the PCLI interface. A single command, `build`, allows you to execute builds from the PCLI command line, provided that the build job was created using the desktop client. (The build job will run under the user ID that submits the build, not the user ID that created the job.)



**NOTE** If you have created a build job using the web client only, you will not be able to run it using PCLI.

Here is an example `build` command line:

```
C:>pcli build -prC:\Test\YourVersionMgrDatabaseName  
-pp/YourVM_Project -bjPCLI_BUILDJOB
```

where the `-pr` flag specifies the location of the Version Manager database, the `-pp` flag specifies the location of the Version Manager project, and the `-bj` flag specifies the name of the Builder build job.

For full usage details on the build command, use the help flag:

```
C:>pcli build -h
```

The command is also documented in the *Version Manager PCLI User's Guide and Reference*.



# Chapter 8

---

## Troubleshooting

This section discusses troubleshooting information, tips, and suggestions for using Serena® Builder™ for Professional™.

<a href="#">Checklist</a>	120
<a href="#">Troubleshooting</a>	121
<a href="#">Tips and Suggestions</a>	123

## Checklist

Here is a checklist for the user starting to use Serena® Builder™ for Professional™.

- Have the correct type and number of Serena Builder licenses
- Have received the URL for the standalone web client from Administrator
- Have verified that the Knowledge Base server is up and running (use ping)
- Have installed a compatible version of the Java Development Kit

---

- Have set environment variables (UNIX: **TMP**; Windows: **tmp** and **temp**)
- Have installed the Java runtime client
- Have seen Welcome screen that indicates I have been added to user list
- Have set a build directory
- Have installed a compatible version of Perl
- Have installed the command-line client
- UNIX: Have received changes to `.profile` from Administrator (to set environment variables; see ["UNIX: Adding Setup Information to Accounts" on page 38](#))

If all of the above are true, you should be able to start building projects as described in the ["Tutorial" on page 43](#).

---

# Troubleshooting

Here are some commonly-experienced problems, along with suggestions for what to do.

- Compiler complains of no input files  
This may be because the directory containing the source files are not specified in the Search Path.
- Project was not found on the KB (Knowledge Base) server  
This may mean that your target definition file makes reference to a project that has changed or was deleted.
- Error connecting to KB server  
This may mean that there are network problems, but it may also mean that there are no valid Serena Builder licenses. Serena Builder performs a license check when any of the clients, including the PCLI client, tries to connect to the Knowledge Base server.
- Can't find compiler  
Serena Builder does not use the search path to locate your compiler. Use your PATH variable to specify the path to your compiler.
- Could not find any targets in the target list  
Serena Builder cannot find any targets. Check the Setup Build page to make sure that at least one target is selected.  
  
If you see this message when trying to do a remote build, it may mean that Serena Builder is not aware of where your source files are on the remote machine. You must either add the source directory to the search path in a form recognizable to the remote machine, or use commands such as "cd <source directory>" preceding the bldmake command on the Build Detail page of Execute Build Job.
- Metalworks example: class BigContrastMetalTheme is public, should be declared in a file named BigContrastMetalTheme.java  
OR
- Bingo example: ERROR 40: The source dependency bingo-server.rmic was not found.  
Add the -s case sensitivity switch to bldmake and try the build job again. In the Version Manager integration, this switch is always on, so this problem should occur only in the standalone web client.
- Targets were defined, but have disappeared  
If you change the build directory, the .TGT file is still in the old build directory and thus Serena Builder cannot find it. You should either recreate the .TGT file or copy it from the old build directory.
- The source dependency <.jar file> was not found.  
If you see an error complaining that a .jar file that you used in the Set Classpath task was not found, even though you listed it as one of the dependencies, you may need to add the path to the .jar file into the Search Path for that project. Serena Builder uses the Search Path to locate .jar or .zip files in the Set Classpath task.

- Serena Builder complains that it can't open a \*.mak file  
This may be because you added the \*.mak files as Version Manager archives. This is unnecessary and will cause the build to fail, since Builder does a Get on the archives, resulting in read-only workfiles. The only files you need to archive are the source files and the .TGT files.
- Builder complains that it cannot create a \*.mak file  
This may be because there are existing \*.mak files that are set to read-only. Reset the permissions on these files and try the build job again.
- Log Report Browser opens in same window as web client  
This is a setting controlled by your web browser. For example, in Internet Explorer, deselecting the setting "Reuse windows for launching shortcuts" in Tools | Internet Options | Advanced | Browsing causes the log report browser to open in its own window.
- Log Report Browser opens, but nothing displays  
There are several possible reasons for this:
  - Local builds: Command-line client is not installed
  - Local builds: Command-line client directory is not in the path
  - Remote builds: Remote build server is not running
  - Remote builds: Command-line client is not installed
  - Remote builds: Command-line client directory is not in the pathThe common thread in all of these is that Builder cannot find bldmake in the path.
- PDB path mismatch  
In the Version Manager integrated clients, two situations can cause a PDB path mismatch error to display. In both cases, the error is most likely caused by a build job that has the same name as another build job that references a different Version Manager project (that is, a different PDB).  
Scenario #1: Default build job  
If the default build job is from a different Version Manager project, you will see the error message right away.  
Scenario #2: User selects build job from wrong PDB.  
If you select a build job that has the same name as a build job from a different Version Manager project, then you will see the PDB path error message.  
In both cases, you should select the build job that matches the associated Version Manager project, or click to dismiss the error message and create a new build job.
- Builder asks if you would like to install an update  
If another user (with administrative privileges) has changed the build type you are using, you will be asked if you would like to install an update from the Knowledge Base server.
- ERROR 538: om could not open translated pre script file  
If you see this error, it is possible that the TMP environment variable is not defined, or the user has insufficient privileges to write to or execute scripts in the directory.

- Build directory deletion fails on clean build in Windows Version Manager Integration

If you see a Build directory deletion error, it is possible that it is caused by the addition of a "cd /d <directory>" command in the Build Commands field of the Submission dialog in the Version Manager integration.

Using a clean build means that the build directory will be deleted each time the build job begins to execute. If you add a "cd" command into the build directory, Windows considers the directory busy and Builder is unable to delete it.

If you must add a "cd" to the Build Commands, place it after the bldmake command.

- Error running Retrieve program

If you see this error, it may mean that Builder is not able to find a supported version of Perl. (The Retrieve program is a Perl script.) Check the value of the PERLLIB environment variable, and check to see that you have a supported version of Perl.

- Unable to get OPENMAKE\_SERVER environment variable value

If you see this error, it could mean that the OPENMAKE\_SERVER variable is not set, but it could also mean that Perl is not installed or in your path.

## Tips and Suggestions

- Check log files in the build directory

The integrated version of Builder creates log files that remain in the build directory. If you are having difficulty with a build, check the build directory for the presence of these log files as they may contain information that can help you. The newest data is appended to the end of the log file.

- Use verbose flags

The build options on the Setup Build page of the web client contain several options that can help you determine what is wrong with a troublesome build.

- Check for conflict with other programs during clean build

If you tell Builder to perform a clean build, and another program such as a text editor is editing a file in a subdirectory of the build directory, Builder will be unable to delete the contents of the build directory and the build will fail.

- Check for permission to access Version Manager archives

Even if a build job is Public, if you do not have permission to access the Version Manager archives referenced in the build job, the build job will not complete. That is, if the user who created the build job made it Public, but the archives used in the build job are not available to all user IDs, the build job will fail.

- Check value of OPENMAKE\_SERVER environment variable

This should point to your Knowledge Base server. Normally it is set when you install the command-line client. If you changed the name of your Knowledge Base server machine, or have switched from one KB server to another, you need to reset the value of OPENMAKE\_SERVER.

- UNIX: Be aware that Bourne shell is the default shell

On UNIX, Builder uses Bourne shell as the default shell. This can affect you in two situations:

- If you define /bin/sh to something other than Bourne shell
- If you add your own commands to the Build Commands field on the Submission dialog box

Be aware that Builder is using Bourne shell and adjust your customizations accordingly.

If you are using both UNIX and Linux, you may find that the bash shell allows the greatest compatibility between platforms.

# Index

---

- A**
- audience for manual 10
- B**
- Bingo example 59
  - error #40 121
- bldmake
  - changing flags 115
  - part of command-line client 19, 24
- build directory
  - defined 26
  - setting 89
- build directory deletion error 123
- build job
  - changing the name of 114
  - defined 26
  - public vs. private 26, 114
- build machine
  - changing 114
- build modes
  - Clean 104
  - Exact 104
  - Incremental 104
  - New 104
  - Rebuild 104
- build types
  - defined 25
- building from a promotion group 100, 116
- building from a version label 100, 116
- building from the default revision or tip 100, 116
- building to a specific location 89
- C**
- can't create \*.mak file 122
- can't find compiler 121
- can't open \*.mak file 122
- case sensitivity in desktop client 22
- case sensitivity switch 121
- change directory command ignored in integration 107
- changes in TGT file format 39
- Clean build
  - defined 104
  - warning about 105
- clients
  - different types of 19
- command-line client
  - defined 24
  - installing 36
  - who needs to install 36
- compiler complains of no input files 121
- Configuration Builder
  - comparison to Serena Builder 30
  - converting CB scripts to Serena Builder (see Builder Development Guide) 85
  - differences from Serena Builder 30
  - used with Serena Builder 75
- contacting technical support 8
- conventions, typographical 8
- could not convert target file 39
- could not find any targets 121
- cross-platform development 115
- D**
- dependency
  - defined 25
- desktop client
  - can save only private build jobs 22
  - defined 21
  - differences from web clients 22
- documentation
  - list of all Serena Builder manuals 11
- drive mappings
  - lost if user logs out 34
- E**
- enable project processing 104
- Enterprise Edition of Serena Builder 10
- error
  - cannot create .mak file 122
  - error #538 122
  - error connecting to KB server 121
  - error running Retrieve 123
- eval license
  - upgrading 37
- F**
- first-found file 27
- footprint information
  - omident 113
  - viewing 113

- I**
- impact analysis 111
  - install update from KB server 122
  - Installation
    - administrator-performed steps 32
    - of command-line client 36
    - user-performed steps 34
  - integrated web client
    - cannot browse for version labels or promotion groups 23
    - cannot execute local build 23
    - defined 22
  - invalid target extension 40
- K**
- Knowledge Base server
    - defined 19
- L**
- license
    - upgrading an evaluation 37
  - license.ini 37
  - licensing 37
  - log report browser
    - nothing displays 122
    - opens in same window as web client 122
- M**
- menus
    - Administration 20
    - Development 20
    - in standalone web client 20
    - Information 20
  - Merant Build
    - upgrading from 39
  - Metalworks example 121
- O**
- om
    - changing flags 115
    - part of command-line client 19, 24
  - omcmdline.jar 40
  - omident 113
  - online help
    - accessing 8
    - for the command-line interface 8
    - for the GUI 8
  - OPENMAKE\_SERVER environment variable 123
- P**
- PCLI client
    - defined 23
    - requirement that build job was defined in integrated clients 24
  - PDB path mismatch 122
  - Perl
    - ActiveState 36
    - if not installed 123
    - no longer installed on Windows 36
    - PERLLIB environment variable 123
    - syntax error if Configuration Builder quote marks wrong 84
  - permanent licenses
    - command-line client 37
    - Knowledge Base server 37
  - platforms, about supported 32
  - procedures 87
  - project
    - defined 25
  - project not found on KB server 121
- R**
- remote build server
    - defined 19
    - must run under "this account" 33
  - remote builds
    - and permissions 109
  - reports
    - implosion and explosion 112
    - log browser 111
    - viewing for previous build 111
  - requirement to install command-line client for local builds 19
  - requirement to set up projects in the standalone web client 22
  - Retrieve 123
  - rt.jar
    - not found in IBM Java Development Kit 48
- S**
- search path
    - and environment variables 28
    - defining different build targets 26
    - example 27
    - looks in current directory first 26
    - right to modify 28
    - uses first-found copy 27
    - using environment variables 28
  - search paths
    - defined 26
  - Serena Builder
    - compared to Configuration Builder 30
    - compared to traditional build process 29

---

- components diagram 18
- concepts 13
- defined 10
- difference between Enterprise and standard editions 10
- Enterprise Edition 10
- workspace info not used 107
- Serena, contacting 8
- setting the build directory 89
- setting up builds only possible from standalone web client 20
- source dependency not found 121
- workspace information
  - not used in Serena Builder integration 107
- wrapper script
  - and Configuration Builder 76

## T

- target
  - defined 25
  - excluding from full build 94
- target definition file 25
- targets disappeared 121
- TGT file
  - batch conversion 40
  - defined 25
  - must be archived in Version Manager 98
  - not the same as a target 25
- Troubleshooting 119
- troubleshooting 121
- tutorial 43
- typographical conventions 8

## U

- upgrading evaluation licenses 37
- upgrading from Merant Build 39
- Using Serena Builder with Configuration Builder 75

## V

- Version Manager
  - workspaces 107
- vmcshrc 38
- vmprofile 38

## W

- warning
  - build job created in standalone web client 57
- web client
  - differences from desktop client 22
  - integrated 22
  - standalone 20
- workflow
  - example of typical 28

