WIND RIVER

Wind River® Workbench

USER'S GUIDE

2.6

VxWorks Version

Copyright © 2006 Wind River Systems, Inc.

All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means without the prior written permission of Wind River Systems, Inc.

Wind River, the Wind River logo, Tornado, and VxWorks are registered trademarks of Wind River Systems, Inc. Any third-party trademarks referenced are the property of their respective owners. For further information regarding Wind River trademarks, please see:

http://www.windriver.com/company/terms/trademark.html

This product may include software licensed to Wind River by third parties. Relevant notices (if any) are provided in your product installation at the following location: *installDir*/*product_name*/**3rd_party_licensor_notice.pdf**.

Wind River may refer to third-party documentation by listing publications or providing links to third-party Web sites for informational purposes. Wind River accepts no responsibility for the information provided in such third-party documentation.

Corporate Headquarters
Wind River Systems, Inc.
500 Wind River Way
Alameda, CA 94501-1153
U.S.A.

toll free (U.S.): (800) 545-WIND
telephone: (510) 748-4100
facsimile: (510) 749-2010

For additional contact information, please visit the Wind River URL:

http://www.windriver.com

For information on how to contact Customer Support, please visit the following URL:

http://www.windriver.com/support

1 Oct 06
Part #: DOC-15844-ND-00

Contents

PART I: INTRODUCTION

1 Overview 3

1.1 Introduction 3

1.2 Wind River Documentation 4

1.3 Road Map to the Wind River Workbench User's Guide 4

1.4 Understanding Cross-Development Concepts 5

1.4.1 Hardware in a Cross-Development Environment 5

1.5 Basic Eclipse Concepts 7

1.5.1 Window 7

1.5.2 Workspace 7

1.5.3 Perspectives 8

1.5.4 Views 10

1.5.5 Editors 11

1.5.6 Projects 11

1.6 Accessing Additional Interface Information 12

2 Wind River Workbench Tutorials 13

2.1 Introduction 13

2.2 Starting Wind River Workbench 14

2.3 Tutorial: Creating a Project and Running a Program 15

2.3.1 Before You Begin 15
2.3.2 Creating a Project 16
2.3.3 Importing Source Files Into Your Project 16
2.3.4 Building Your Project 16
2.3.5 Creating a Connection Definition to the VxWorks simulator 17
2.3.6 Downloading the Program and Attaching the Debugger 18
2.3.7 Setting Up the Device Debug Perspective 18
2.3.8 Setting and Running to a Breakpoint. 20
2.3.9 Modifying the Breakpoint 21

2.4 Tutorial: Editing and Debugging Source Files 22

2.4.1 Before You Begin 22
2.4.2 Introducing an Error into the Source Code 22
2.4.3 Tracking Down a Build Failure 22
2.4.4 Displaying File History 23
2.4.5 Rebuilding the Project 23

2.5 Tutorial: Using the Editor's Code Development Features 24

2.5.1 Using Code Completion to Add Symbols to Your File 24
2.5.2 Using Parameter Hints 25
2.5.3 Using Bracket Matching to Clarify Syntax 25
2.5.4 Finding Symbols in Source Files 26

2.6 Tutorial: Tracking Items of Interest in Your Files 26

2.6.1 Creating a Bookmark on a Source Line in a File 26
2.6.2 Creating a Bookmark for an Entire File 27

2.6.3 Locating and Viewing Your Bookmarks 27

2.7 Tutorial: Using Workbench to Debug a VxWorks 5.5.*x* Target 27

2.7.1 Before You Begin 28

2.7.2 Creating a Project 28

2.7.3 Creating a VxWorks 5.5.*x* Target Server Connection 29

2.7.4 Launching a Kernel Task and Attaching the Debugger 29

2.7.5 Setting and Running to a Breakpoint 30

2.7.6 System Mode Debugging 31

2.7.7 Using Core Dump Files 32

2.7.8 Using Already Available Tornado 2 Projects 32

3 Setting Up Your Development Environment 35

3.1 Introduction 35

3.1.1 Overview of Host and Target Configuration Tasks 36

3.1.2 Understanding Target Servers and Target Agents 37

3.2 Configuring Your Cross-Development System 40

3.2.1 Configuring Host Software 40

3.2.2 Verifying Serial Setup and Power 45

3.3 Setting Up a Boot Mechanism 49

3.4 Booting VxWorks 50

3.4.1 Default Boot Process 50

3.4.2 Entering New Boot Parameters 52

3.4.3 Boot Program Commands 53

3.4.4 Description of Boot Parameters 55

3.4.5 Booting With New Parameters 58

3.4.6 Alternate Boot Methods 59

3.4.7 Rebooting VxWorks 60

3.5 Configuring Host-Target Communication for Workbench 61
3.5.1 Ethernet Connections 61
3.5.2 Serial-Line Connections 64

3.6 Troubleshooting VxWorks Problems 67

PART II: PROJECTS

4 Projects Overview 71

4.1 Introduction 71

4.2 Workspace/Project Location 72

4.3 Creating New Projects 73
4.3.1 Subsequent Modification of Project Creation Wizard Settings 74
4.3.2 Projects and Application Code 74

4.4 Overview of Preconfigured Project Types 74
4.4.1 Workbench Sample Projects 75
4.4.2 VxWorks Image Project 75
4.4.3 VxWorks Boot Loader Project 76
4.4.4 VxWorks Downloadable Kernel Module Project 76
4.4.5 VxWorks Real-time Process Project 77
4.4.6 VxWorks Shared Library Project 78
4.4.7 VxWorks ROMFS File System Project 78
4.4.8 User-Defined Projects 79
4.4.9 Native Application Project 79

4.5 Projects and Project Structures 80
4.5.1 Adding Subprojects to a Project 80
4.5.2 Project Structures and Host File System Directory Structure 81
4.5.3 Project Structures and the Build System 82

4.5.4 Project Structures and Sharing Subprojects 83
4.5.5 Customizing Build Settings for Shared Subprojects 83

4.6 Project-Specific Execution Environments 83
4.6.1 Using a project.properties file with a Shell 85
4.6.2 Limitations When Using project.properties Files 85

5 Creating VxWorks Image Projects 87

5.1 Introduction 87

5.2 Creating a VxWorks Image Project 88
5.2.1 Specifying a Non-Default Driver 91

5.3 Importing a VxWorks Image Project 92
5.3.1 Migrating a VxWorks Image Project 92

5.4 Configuring Kernel Components 93
5.4.1 The Kernel Configuration Editor 94

5.5 VxWorks Image Projects in the Project Navigator 94
5.5.1 Global Project Nodes 95
5.5.2 Project Build Specs and Target Nodes 95
5.5.3 Build Output Folders 96
5.5.4 Makefile Nodes 97
5.5.5 Project File Nodes 97

5.6 Adding Application Projects to the VxWorks Image Project 99

5.7 Notes on Board Support Packages (BSPs) 100
5.7.1 Using the Simulator BSP 100
5.7.2 Using a Wind River BSP 100
5.7.3 Using a Custom BSP for Custom Hardware 100

6 Creating Boot Loader Projects 103

6.1 Introduction 103

6.2 Creating a Boot Loader Project 104

6.3 Creating a Customized Boot Loader 105

6.3.1 Selecting Boot Loader Drivers 106

6.4 Boot Loader Projects in the Project Navigator 106

6.4.1 Global Project Nodes 106

6.4.2 Project Build Specs and Target Nodes 107

6.4.3 Makefile Nodes 107

6.4.4 Other Project Files 107

7 Creating VxWorks ROMFS File System Projects 109

7.1 Introduction 109

7.2 Creating a VxWorks ROMFS File System Project 110

7.3 Configuring the VxWorks ROMFS File System 110

7.4 VxWorks ROMFS File System Projects in the Project Navigator 111

7.4.1 Global Project Nodes 111

7.4.2 Project File Nodes 112

8 Creating VxWorks Real-time Process Projects 113

8.1 Introduction 113

8.2 Creating a VxWorks Real-time Process Project 114

8.3 Configuring VxWorks Real-time Process Projects 115

8.3.1 Configuring Build Support and Specs 115

8.3.2 Configuring Build Tools 116

8.3.3 Configuring Build Macros 117

8.3.4 Configuring Build Paths 118

8.4 VxWorks Real-time Process Projects in the Project Navigator 120

8.4.1 Global Project Nodes 120

8.4.2 Project Build Specs and Target Nodes 120

8.4.3 Makefile Nodes 121

8.4.4 Project File Nodes 121

8.5 Application Code for a VxWorks Real-time Process Project 122

8.6 Linking to VxWorks and Using Shared Libraries 122

9 Creating VxWorks Shared Library Projects 123

9.1 Introduction 123

9.2 Creating a VxWorks Shared Library Project 124

9.3 Configuring VxWorks Shared Library Projects 124

9.3.1 Configuring Build Support and Specs 125

9.3.2 Configuring Build Tools 126

9.3.3 Configuring Build Macros 126

9.3.4 Configuring Build Paths 128

9.4 Shared Libraries in the Project Navigator 130

9.4.1 Global Project Nodes 130

9.4.2 Target Node 130

9.4.3 Makefile Nodes 130

9.4.4 Project File Nodes 131

9.5 Source Code for the Shared Library 131

9.6 Making Shared Libraries Available to Applications 131

9.6.1 Configuring the Application Projects 132

10 Creating VxWorks Downloadable Kernel Module Projects 133

10.1 Introduction 133

10.2 Creating a VxWorks Downloadable Kernel Module Project 134

10.3 Configuring VxWorks Downloadable Kernel Module Projects 134

10.3.1 Configuring Build Support and Specs 135

10.3.2 Configuring Build Tools 136

10.3.3 Configuring Build Macros 137

10.3.4 Configuring Build Paths 138

10.4 Downloadable Kernel Modules in the Project Navigator 140

10.4.1 Global Project Nodes 140

10.4.2 Project Build Specs and Target Nodes 140

10.4.3 Makefile Nodes 141

10.4.4 Project File Nodes 141

10.5 Application Code for a VxWorks DKM Project 142

11 Creating User-Defined Projects 143

11.1 Introduction 143

11.2 Creating and Maintaining Makefiles 144

11.3 Creating a User-Defined Project 144

11.4 Configuring a User-Defined Project 145

11.4.1 Configuring Build Support 145

11.4.2 Configuring Build Targets 146

11.4.3 Configuring Build Specs 147

11.4.4 Configuring Build Macros 147

11.5 Creating an Application for VxWorks 148

12 Creating Native Application Projects 151

12.1 Introduction 151

12.2 Creating a Native Application Project 152

12.3 Configuring Native Application Projects 152

12.3.1 Configuring Build Support and Specs 153

12.3.2 Configuring Build Tools 154

12.3.3 Configuring Build Macros 155

12.3.4 Configuring Build Paths 156

12.4 Native Applications in the Project Navigator 158

12.4.1 Global Project Nodes 158

12.4.2 Project Build Specs and Target Nodes 158

12.4.3 Makefile Nodes 159

12.4.4 Project File Nodes 159

12.5 Application Code for a Native Application Project 160

13 Working in the Project Navigator 161

13.1 Introduction 161

13.2 Creating Projects 162

13.3 Adding Application Code to Projects 162

13.3.1 Importing Resources 162

13.3.2 Adding New Files to Projects 163

13.4 Opening and Closing Projects 163

13.4.1 Closing a Project 163

13.5 Scoping and Navigation 164

13.6 Moving, Copying, and Deleting Resources and Nodes 166

13.6.1 Resources and Logical Nodes 166
13.6.2 Manipulating Files 167
13.6.3 Manipulating Project Nodes 167
13.6.4 Manipulating Target Nodes 168

14 Advanced Project Scenarios 171

14.1 Introduction 171

14.2 Resource Locations 172

14.3 Multiple, Unrelated Software Systems 173
14.3.1 Using Different Workspaces for Different Systems 173
14.3.2 Using the Same Workspace for Different Software Systems 174

14.4 Complex Project Structures 174
14.4.1 Project Assumptions 175
14.4.2 Infrastructure Design 176
14.4.3 Development 181
14.4.4 Finalization 186

PART III: DEVELOPMENT

15 Navigating and Editing 195

15.1 Introduction 195

15.2 Wind River Workbench Context Navigation 196
15.2.1 The Symbol Browser 197
15.2.2 The Outline View 197
15.2.3 The File Navigator 198

15.3 The Editor 198
15.3.1 Code Templates 199

15.3.2 Configuring a Custom Editor 200

15.4 Search and Replace 201

15.4.1 Initiating Text Retrieval 201

15.5 Static Analysis 201

15.5.1 Sharing Static Analysis Data with a Team 202

16 Building Projects 205

16.1 Introduction 205

16.2 Configuring Workbench Managed Builds 208

16.2.1 Configuring Standard Managed Builds 208

16.2.2 Configuring Flexible Managed Builds 208

16.3 Configuring User-Defined Builds 214

16.4 Accessing Build Properties 214

16.4.1 Workbench Global Build Properties 215

16.4.2 Project-specific Build Properties 215

16.4.3 Folder, File, and Build Target Properties 215

16.4.4 Multiple Target Operating Systems and Versions 216

16.5 Build Specs 216

16.6 Makefiles 217

16.6.1 Derived File Build Support 217

17 Building: Use Cases 221

17.1 Introduction 221

17.2 Adding Compiler Flags 222

17.2.1 Add a Compiler Flag by Hand 222

17.2.2 Add a Compiler Flag with GUI Assistance 223

17.3 Building Applications for Different Boards 224

17.4 Creating Library Build-Targets for Testing and Release 225

17.5 Architecture-Specific Implementation of Functions 228

17.6 Executables that Dynamically Link to Shared Libraries 229

17.7 User-Defined Build-Targets in the Project Navigator 232

17.7.1 Custom Build-Targets in User-Defined Projects 232

17.7.2 Custom Build-Targets in Workbench Managed Projects 233

17.7.3 User Build Arguments 233

17.8 A Build Spec for New Compilers and Other Tools 233

17.9 Developing on Remote Hosts 235

17.9.1 General Requirements 236

17.9.2 Remote Build Scenarios 236

17.9.3 Setting Up a Remote Environment 237

17.9.4 Building Projects Remotely 238

17.9.5 Running Applications Remotely 239

17.9.6 Rlogin Connection Description 240

17.9.7 SSH Connection Description 240

18 RTPs and Shared Libraries from Host to Target 241

18.1 Introduction 241

18.2 A VxWorks Real-time Process from Host to Target 242

18.2.1 Set Up the Project Structure for Real-time Processes 242

18.2.2 Add Code to the Real-time Process Project 243

18.2.3 Add the Real-time Process to the VxWorks ROMFS Target File System 244

18.2.4 Build the System 246

18.2.5 Set up the Target Connection 246

18.2.6 Run the Real-time Process on the Simulator 247

18.3 A VxWorks Shared Library from Real-time Process to Target 247

18.3.1 Set Up the VxWorks Shared Library Project 247

18.3.2 Add Code to the Shared Library Project 248

18.3.3 Add the Shared Library to the Run-Time Process 249

18.3.4 Modify the Code in the Real-time Process Project 251

18.3.5 Generate Include Search Paths 251

18.3.6 Add Libraries to the VxWorks ROMFS Target File System 252

18.3.7 Build the System Again 253

18.3.8 Run the RTP with the Shared Library on the Simulator 253

PART IV: TARGET MANAGEMENT

19 Connecting to Targets 257

19.1 Introduction 257

19.2 The Target Manager View 258

19.3 Defining a New Connection 258

19.4 Establishing a Connection 259

19.4.1 Assumptions 259

19.4.2 Connecting to the Target 259

19.4.3 Specifying an Object File 260

19.4.4 The Kernel Shell 261

19.5 The Registry 262

19.5.1 Launching the Registry 262

19.5.2 Remote Registries 263

19.5.3 Shutting Down the Registry 263

19.5.4 Changing the Default Registry 264

20 New Target Server Connections 265

20.1 Introduction 265

20.2 Defining a New Target Server Connection 265

20.2.1 Wind River Target Server 266

20.2.2 Target Server Connection Page 266

20.2.3 Object Path Mappings Page 270

20.2.4 Target State Refresh Page 273

20.2.5 Connection Summary Page 274

20.3 Kernel Configuration 275

21 New VxWorks Simulator Connections 277

21.1 Introduction 277

21.2 Defining a New Wind River VxWorks Simulator Connection 277

21.2.1 VxWorks Boot Parameters Page 278

21.2.2 VxSim Memory Options Page 278

21.2.3 VxWorks Simulator Miscellaneous Options Page 278

21.2.4 Target Server Options Page 279

PART V: DEBUGGING

22 Launching Programs 283

22.1 Introduction 283

22.2 Launching a Kernel Task or a Process 284

22.2.1 Defining the Target Connection 285

22.2.2 Defining the Kernel Task or Process to Run 285

22.2.3 Specifying a Build Target to Download 286

22.2.4 Specifying The Projects to Build 286

22.2.5 Defining Debug Behavior 287
22.2.6 Specifying Where Workbench Should Look for Source Files 288
22.2.7 Configuring Access Methods 288
22.2.8 Using Your Launch Configuration 289

22.3 Reset & Download: Hardware Debugging Launches 289

22.4 Launching a Native Application 290
22.4.1 Specifying the Location and Arguments for Your Application 290
22.4.2 Specifying Remote Settings 290
22.4.3 Setting Environment Variables 291
22.4.4 Configuring Access Methods 291
22.4.5 Running Your Native Application 292

22.5 Relaunching Recently Run Programs 292
22.5.1 Increasing the Size of the Launch History List 293

22.6 Controlling Multiple Launches 293

22.7 Launches and the Console View 298

22.8 Using Attach-to-Target Launches 300
22.8.1 Attaching the Debugger to a Running Task or Process 301
22.8.2 Attaching the Debugger to the Kernel 301
22.8.3 Attaching the Kernel in Task Mode 301
22.8.4 Attaching the Kernel in System Mode 302

22.9 Suggested Workflow 302

23 Managing Breakpoints 305

23.1 Introduction 305

23.2 Types of Breakpoints 305
23.2.1 Line Breakpoints 306

23.2.2 Expression Breakpoints 307
23.2.3 Hardware Breakpoints 307

23.3 Manipulating Breakpoints 309
23.3.1 Importing Breakpoints 309
23.3.2 Exporting Breakpoints 310
23.3.3 Refreshing Breakpoints 310
23.3.4 Disabling Breakpoints 310
23.3.5 Removing Breakpoints 310

24 Debugging Projects 313

24.1 Introduction 313

24.2 Using the Debug View 314
24.2.1 Understanding the Debug View Display 315

24.3 Coloring Views 318

24.4 Stepping Through a Program 319

24.5 Using Debug Modes 320
24.5.1 Setting and Recognizing the Debug Mode of a Connection 323
24.5.2 Debugging Multiple Target Connections 324
24.5.3 Disconnecting and Terminating Processes 324
24.5.4 Configuring Debug Settings for a Custom Editor 324

24.6 Understanding Source Lookup Path Settings 326

24.7 Using the Disassembly View 326
24.7.1 Opening the Disassembly View 327
24.7.2 Understanding the Disassembly View Display 327

24.8 Using the Kernel Objects View 328
24.8.1 Understanding the Kernel Objects View Display 329

24.9 Remote Kernel Metrics 331

24.10 Run/Debug Preferences 336

25 Troubleshooting 337

25.1 Introduction 337

25.2 Startup Problems 338

25.2.1 Pango Error on Linux 341

25.3 General Problems 341

25.3.1 Java Development Tools (JDT) Dependency 341

25.3.2 Help System Does Not Display on Solaris or Linux 342

25.3.3 Help System Does Not Display on Windows 342

25.3.4 Removing Unwanted Target Connections 342

25.4 Error Messages 343

25.4.1 Project System Errors 343

25.4.2 Build System Errors 345

25.4.3 Target Manager Errors 348

25.4.4 Launch Configuration Errors 353

25.4.5 Debugger Errors 354

25.4.6 Static Analysis Errors 354

25.5 Troubleshooting VxWorks Configuration Problems 355

25.5.1 What to Check 355

25.6 Error Log View 358

25.7 Error Logs Generated by Workbench 358

25.7.1 Creating a ZIP file of Logs 358

25.7.2 Eclipse Log 359

25.7.3 DFW GDB/MI and Debug Tracing Logs 360

25.7.4 Debugger Views GDB/MI Log 360
25.7.5 Debugger Views Internal Errors Log 361
25.7.6 Debugger Views Broadcast Message Debug Tracing Log 361
25.7.7 Target Server Output Log 362
25.7.8 Target Server Back End Log 362
25.7.9 Target Server WTX Log 363
25.7.10 Target Manager Debug Tracing Log 364
25.7.11 Static Analysis Parser Logs 364

25.8 Technical Support 365

PART VI: USING WORKBENCH WITH OTHER TOOLS

26 Integrating Plug-ins 369

26.1 Introduction 369

26.2 Finding New Plug-ins 370

26.3 Incorporating New Plug-ins into Workbench 370

26.3.1 Creating a Plug-in Directory Structure 370
26.3.2 Installing a ClearCase Plug-in 371

26.4 Disabling Plug-in Functionality 373

26.5 Managing Multiple Plug-in Configurations 373

27 Using Workbench in an Eclipse Environment 375

27.1 Introduction 375

27.2 Recommended Software Versions and Limitations 375

27.3 Setting Up Workbench 376

27.4 Using CDT and Workbench in an Eclipse Environment 377

27.4.1 Workflow in the Project Navigator 377
27.4.2 Workflow in the Build Console 379
27.4.3 Workflow in the Editor 379
27.4.4 Workflow for Debugging 380

28 Using Workbench with Version Control 381

28.1 Introduction 381

28.2 Using Workbench with ClearCase Views 381
28.2.1 Adding Workbench Project Files to Version Control 382

PART VII: REFERENCE

A Command-line Updating of Workspaces 387

A.1 Overview 387

A.2 wrws_update Reference 388

B Command-line Importing of Projects 391

B.1 Overview 391

B.2 wrws_import Reference 392

C Configuring a Wind River Proxy Host 395

C.1 Overview 395

C.2 Configuring wrproxy 397

C.3 wrproxy Command Summary 399

D Glossary 403

D.1 Introduction 403

D.1.1 Refining a Search 403

D.2 Terms 404

Index 409

PART I

Introduction

1 Overview 3

2 Wind River Workbench Tutorials 13

3 Setting Up Your Development Environment 35

1 Overview

1.1 Introduction 3

1.2 Wind River Documentation 4

1.3 Road Map to the Wind River Workbench User's Guide 4

1.4 Understanding Cross-Development Concepts 5

1.5 Basic Eclipse Concepts 7

1.6 Accessing Additional Interface Information 12

1.1 Introduction

Welcome to the *Wind River Workbench User's Guide*. Wind River Workbench 2.6 is an Eclipse-based development suite that provides an efficient way to develop real-time and embedded applications with minimal intrusion on the target system.[1]

Wind River Workbench is available on Windows, Linux, and Solaris hosts, but in this guide, screenshots and paths will be shown as on Windows.

1. Eclipse is an industry-standard framework for building development suites.

1.2 Wind River Documentation

A wide variety of documentation in many different formats is available to Workbench customers. See the getting started for your platform for a description of the full document set.

1.3 Road Map to the Wind River Workbench User's Guide

- *Part I. Introduction* provides an introduction to basic Eclipse terminology and functionality, walks you through a set of tutorials that introduce the major features of Workbench, and explains how to set up your development environment in order to run your programs on real target hardware.
- *Part II. Projects* explains the **Project System**, including creating new projects, importing and exporting existing projects, and creating VxWorks images and user applications.
- *Part III. Development* describes the **Editor**, **Static Analysis**, and **Build System** features of Workbench.
- *Part IV. Target Management* provides details about using the **Target Manager**, including how to configure a target server, and how to create and manage your target connections.
- *Part V. Debugging* explains **Debugger** functionality, including launch configurations, attaching the debugger to processes, working with breakpoints, displaying processes in the Debug and Disassembly views, and examining registers and memory. This section also provides **Troubleshooting** information, explaining how to respond to error messages you may see while using Workbench.
- *Part VI. Using Workbench with Other Tools* describes how to incorporate **plug-ins**, such as ClearCase, into Workbench.
- *Part VII. Reference* provides information about updating your workspace with the command-line, as well as a **Glossary** of Workbench and Eclipse terms for which you may want more information.

1.4 Understanding Cross-Development Concepts

Cross-development is the process of writing code on one system, known as a *host*, that will run on another system, known as a *target*.

Cross-development allows you to write code on a system that you have available to you (such as a PC running Linux, Windows, or Solaris) and produce applications that run on hardware that you would have no other convenient way of programming, such as a chip destined for a mobile phone.

1.4.1 Hardware in a Cross-Development Environment

A typical host is equipped with large amounts of RAM and disk space, backup media, printers, and other peripherals. In contrast, a typical target has only the resources required by the real-time application with perhaps some small amount of additional resources for testing and debugging.

Working on the Host

You use the host just as you would if you were writing code to run on the host itself—to manage project files; edit, compile, link, store multiple versions of your real-time code, and configure the operating system destined to run on the target.

Connecting the Target to the Host

A number of alternatives exist for connecting the target system to the host, such as Ethernet, serial, and JTAG. See *3. Setting Up Your Development Environment* for more information about setting up your hardware.

Running Your Application Code

Run-time code is the code that is intended for the final application. The run-time includes the kernel, your application-specific code, and some selected library code. The term run-time does not usually refer to the target agent, although you will typically include it during development and debugging. See *3.1.2 Understanding Target Servers and Target Agents*, p.37 for more information about the target agent.

Workbench allows you to avoid the cumbersome process of downloading your complete run-time code each time you make a change by allowing you to download and run individual application modules as they are developed. You can even run application modules on the host in the target simulator, Wind River VxWorks Simulator, if target hardware is not available.

Advantages of Using Wind River Workbench

Wind River Workbench ensures the smallest possible difference between the performance of the target you use during development, and the performance of the target after deployment, by keeping most development tools on the host.

A fundamental advantage of using Wind River Workbench is that your application does not need to be fully linked. Code that is only partially completed can be downloaded for incremental testing and debugging; application modules do not need to be linked with the run-time system libraries, or even with each other. The host-resident shell and debugger can be used interactively to invoke and test either individual application routines or complete tasks.

Workbench loads the relocatable object modules directly, and maintains a complete host-resident symbol table for the target. This symbol table is incremental: the target server incorporates symbols as it downloads each object module. You can examine variables, call subroutines, spawn tasks, disassemble code in memory, set breakpoints, trace subroutine calls, and so forth, all using the original symbol names.

Wind River Workbench shortens the cycle between developing an idea and implementing it by allowing you to quickly download your incremental run-time code and dynamically link it with the operating system. Your application is available for symbolic interaction and testing with minimal delay.

The Workbench debugger allows you to view and debug applications in the original source code. Setting breakpoints, single-stepping, examining structures, and so on are all done at the source level, using a convenient graphical interface.

1.5 Basic Eclipse Concepts

Wind River Workbench is based on the Eclipse Platform, an industry-standard framework for building development suites.

This section provides a very brief overview of some of the Workbench components inherited from Eclipse.

1.5.1 Window

The term *window* refers to the desktop development environment. You can open more than one window at a time by selecting **Window > New Window** (each window will see the same projects and workspace.) A Workbench window can contain one or more *perspectives*.

1.5.2 Workspace

Workbench uses a *workspace* to store your current working environment. Some of the items that are saved with the workspace include the set of open projects, and the size and location of views.

The workspace also contains information about the current session, including the types and positions of your views when you last exited Workbench, current projects, and installed breakpoints.

The default location of your workspace is *installDir***workspace**, but it can be located elsewhere if necessary. If you want to run two or more copies of Workbench, each must have its own workspace.

Maintaining More Than One Workspace

If you want to run two independent copies of Workbench (to keep some projects and files completely separate from others) you must establish a second workspace. This is not a required step for the tutorial in *2. Wind River Workbench Tutorials*.

1. Launch Workbench as described in *2.2 Starting Wind River Workbench*, p.14.
2. Select **File > Switch Workspace** to open the **Select a workspace** dialog.
3. Select the directory where you want your workspace to be located, then select **Make New Folder**. Type the name of your new workspace, then click **OK**.

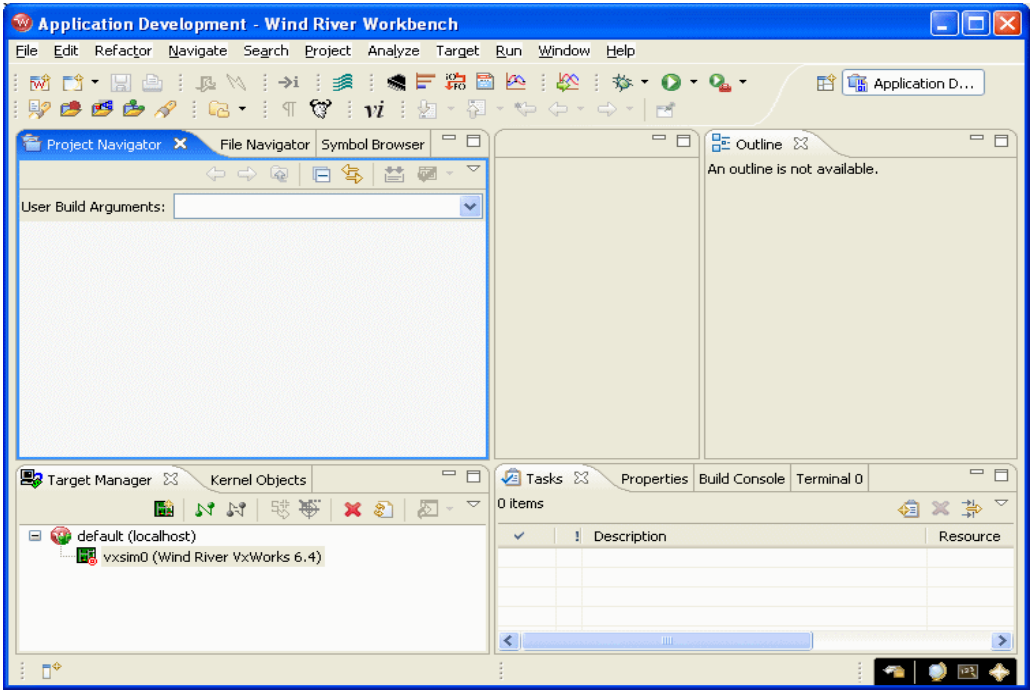
NOTE: The path to each of your workspaces must be unique. If you want a new workspace to be located in the installation directory alongside your original workspace, it must have a unique name (for example, **workspace2** or **newWorkspace**). If it is located in a different directory, it can have the same name as the original: **workspace**.

1.5.3 Perspectives

A *perspective* groups together an editor area and one or more views that are convenient to have available while working on a particular task.

For example, Figure 1-1 shows the Application Development perspective, which is designed to help you create projects, browse files, and edit and build source code.

Figure 1-1 **Application Development Perspective**

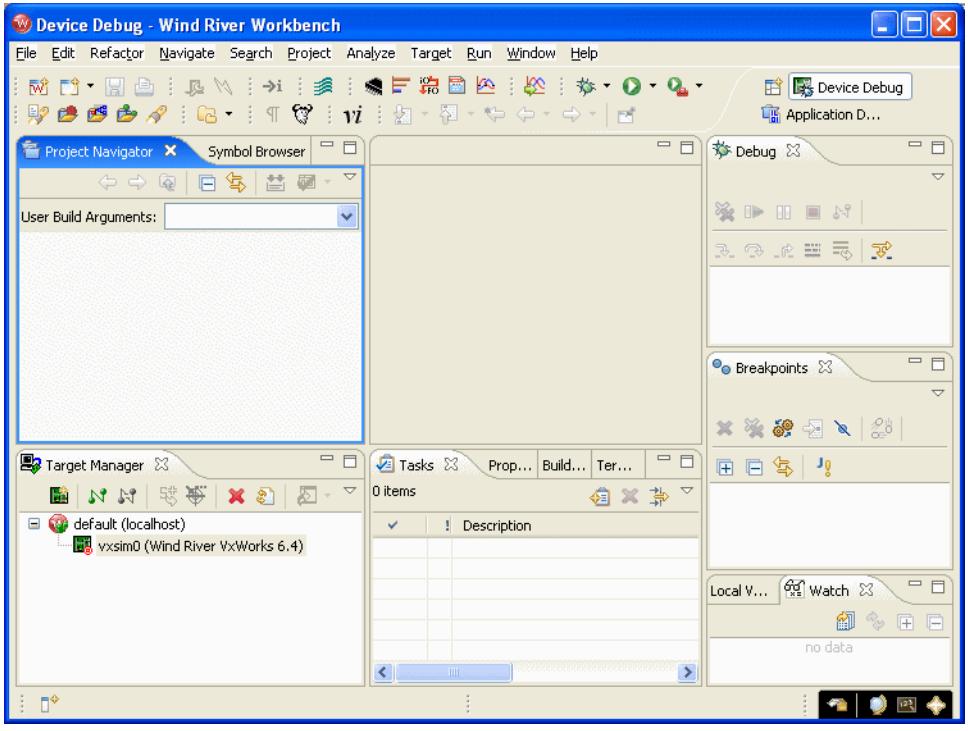


It includes the Project Navigator on the top-left side of the screen, the Outline view on the top-right, the Target Manager on the bottom-left, and the Stacked view (also known as a tabbed notebook) on the bottom-right with the Tasks view visible.

To open a new perspective, select **Window > Open Perspective > Other** and choose a perspective you want to explore, or click the **Open a perspective** icon in the upper right corner of the Workbench window, select **Other**, and choose a perspective.

Figure 1-2 shows the Device Debug perspective, which contains views that are useful when you are running and debugging programs, including the Debug and Breakpoints views, and a tabbed notebook containing the Local Variables, Watch, and Register views. These views replace the Outline view of the Application Development perspective.

Figure 1-2 **Device Debug Perspective**



The Application Development perspective opens by default, but you can switch between perspectives by selecting an icon in the shortcut bar along the top right

edge of the Workbench window. When you start Workbench for the first time, the **Open a perspective** icon and the **Application Development** tab appear as shown in Figure 1-1.

As you open perspectives, their icons appear in the shortcut bar, as seen in Figure 1-2. To see them all side by side, click to the left of the **Open a Perspective** icon and drag to the left until all open perspectives are visible.

To customize a perspective, you can open, close, and move views to create a comfortable work environment, then select **Window > Save Perspective As** and give your perspective a name. That configuration of views will be restored the next time you open your perspective. You can further customize your perspective by selecting **Window > Customize Perspective**.

You can restore a perspective to its default configuration by selecting **Window > Reset Perspective**.

1.5.4 Views

Views reside in perspectives, and allow you to display, manipulate, and navigate the information in Workbench.

Certain views appear in particular perspectives by default, but you can add any view to any perspective by selecting **Window > Show View**, then either selecting the view you want, or selecting **Other**, selecting the perspective containing that view, then choosing the view from the list.

There are two things to remember when using views:

- Only one view (or editor) can be active at a time. The title bar of the active view is highlighted.
- Only one instance of a type of view can be present in a perspective at a time (but multiple editors can be present to view multiple source files).

Many views have their own menus. To open the menu for a view, click the down arrow at the right end of the view's title bar. Some views also have their own tool bars. The actions represented by buttons on view toolbars only affect the items within that view.

Moving and Maximizing Views

Move a view by clicking either its title bar or its tab in a stacked notebook, and dragging it to a new location.

There are several ways to relocate a view:

- Drag the view to the edge of another view and drop it. The area is then split, and both views are tiled in the same area. The cursor changes to an appropriate directional arrow as you approach the edge of a view.
- Drag the view to the title bar of an existing view and drop it. The view will be added to a stacked notebook with tabs. When you drag the view to stack it, the cursor changes to an icon of a set of stacked folders.
- If you drag a view over a tab in an existing view, the view will be stacked in that notebook with its tab at the left of the existing view. You can also drag an existing tab to the right of another tab to arrange tabs to your liking.

To quickly maximize a view to fill the entire perspective area, double-click its title bar. Double-click the title bar again to restore it.

1.5.5 Editors

An editor is a special type of view used to edit files. You can associate different editors with different types of files such as C, C++, Ada, Assembler, and Makefiles. When you open a file, the associated editor opens in the perspective's editor area.

Any number of editors can be open at once, but only one can be active at a time. By default, editors are stacked in the editor area, but you can tile them in order to view source files simultaneously (see *15. Navigating and Editing* for more information about Editors).

Tabs in the editor area indicate the names of files that are currently open for editing. An asterisk (*) indicates that an editor contains unsaved changes.

1.5.6 Projects

Workbench uses projects as logical containers and as building blocks that can be linked together to create a software system. The Project Navigator lets you visually organize projects into structures that reflect their inner dependencies, and therefore the order in which they are compiled and linked.

1.6 Accessing Additional Interface Information

For more information about the Workbench interface, press the help key for your host. On Windows press **F1**, on Linux press **CTRL+F1**, and on Solaris press the **HELP** key to open a help view containing a brief description of the current view and links to sections of Workbench documentation with more information on the same topic. You can also access the help system by selecting **Help > Help Contents > Wind River Documentation**.

For more information on Eclipse functionality, see the *Eclipse Workbench User Guide* under **Help > Help Contents > Wind River Partner Documentation > Eclipse Platform Documentation**, as well as the Eclipse web site at **www.eclipse.org**.

2

Wind River Workbench Tutorials

2.1 Introduction 13

2.2 Starting Wind River Workbench 14

2.3 Tutorial: Creating a Project and Running a Program 15

2.4 Tutorial: Editing and Debugging Source Files 22

2.5 Tutorial: Using the Editor's Code Development Features 24

2.6 Tutorial: Tracking Items of Interest in Your Files 26

2.7 Tutorial: Using Workbench to Debug a VxWorks 5.5.x Target 27

2.1 Introduction

This chapter provides tutorials that are designed to introduce you to Wind River Workbench and to familiarize you with its views and development concepts. The VxWorks Simulator is used to execute the sample programs, and no special hardware or system setup is required.

In the course of these tutorials, you will:

- Create a project.
- Import source files.
- Build a project.
- Connect to a simulator.

- Set breakpoints.
- Step through code.
- Set a Watch variable.
- Run code.
- Edit source files.
- Track build errors.
- Debug a project.
- Rebuild and rerun your code.

These tutorials assume a basic understanding of embedded projects and debugging concepts. They also assume that you have the Workbench software (with VxWorks support) installed correctly on your host, and that the software is installed in the default location and with the default settings.

To run the VxWorks 5.5 debugging tutorial, you must also have VxWorks 5.5 and Tornado 2 installed.

For definitions of unfamiliar terminology, see *D. Glossary*.

2.2 Starting Wind River Workbench

1. Before you can run the tutorials, you must start Workbench.

   On Windows:

   Start > Programs > Wind River > Workbench 2.*x* > Wind River Workbench 2.*x*

   On Linux and Solaris:

   Open a terminal window, then navigate to your Workbench installation directory. From the command line, type:

   ```
   ./startWorkbench.sh
   ```

2. When you start Workbench for the first time, Workbench creates a new registry database[1]. A dialog appears telling you how to migrate your registry settings from a previous registry to the new one[2].

1. A new database will also be created in **/tmp** if the default database is not accessible.
2. If you did not have a previous version of Workbench installed and therefore do not have registry settings to migrate over, you can safely ignore this dialog.

3. Click **OK**. The Wind River Workbench welcome screen appears.
4. Select the arrow to open Workbench to the Application Development perspective.

2.3 Tutorial: Creating a Project and Running a Program

This tutorial uses the **ball** sample program, written in C. This program implements a set of balls bouncing in a two-dimensional grid. As the balls bounce, they collide with each other and with the walls. You see the balls move by setting a breakpoint with the property **Continue on break** at the outer move loop, and watching a global grid array variable in the Memory window.

First, you will create a new project in your workspace, then you will import the ball source files into it from their Workbench installation directory.

2.3.1 Before You Begin

Workbench preserves its configuration when you close it, so that at next launch you can resume where you left off in your development.

If you experimented with opening perspectives and moving views before starting this tutorial, switch back to the Application Development perspective by clicking its icon in the upper right corner of the Workbench window. If its icon is not visible, drag the shortcut bar to the left (your cursor will turn to a double-headed arrow) or click the double-right arrows and select the perspective.



To reset the perspective and its views to their default settings, select **Window > Reset Perspective**.

2.3.2 Creating a Project

1. Select **File > New > Wind River Workbench Project**. The **New Wind River Workbench Project** dialog appears.
2. From the Target operating system drop-down list, select **Wind River VxWorks 6.***x*. Click **Next**.
3. From the **Build type** drop-down list, select **Downloadable Kernel Module**. Click **Next**.
4. In the **Project Name** field, type **ball**. For the purposes of this tutorial, keep **Create project in workspace** selected. Click **Finish**. The **ball** project appears in the Project Navigator.

2.3.3 Importing Source Files Into Your Project

Next, import the ball sample project files.

1. Right-click the **ball** project folder, then select **Import**. The **Import** wizard appears.
2. Select **General**, then **File System**, then click **Next**. The **File System** page of the **Import** wizard appears.
3. Click the **Browse** button next to the **From directory** field. The **Import from directory** page appears.
4. Navigate to the *installDir***workbench-2.***x****samples** directory. Select **ball**, then click **OK**. The **File system** page reappears, with the ball folder in the left pane and the files in that folder in the right pane.
5. Select the check box next to **ball**. This automatically selects all the files in the right pane. Because you are importing into the ball project, **ball** appears in the **Into folder** field. Click **Finish**.
6. To see the contents of the **ball** project folder, click the plus next to it in the Project Navigator. You will see the project files in black, and the build targets in green. Any files that appear in gray are read-only.

2.3.4 Building Your Project

1. Build the ball project by right-clicking the **ball** folder in the Project Navigator and selecting **Build Project** from the context menu.

2. The first time you build a project, a dialog appears asking if you want Workbench to generate include paths. You do not need to do this for the tutorial, so click **Continue**.[3]
3. Build output displays in the Build Console at the bottom of the screen, and the output file **ball.out** appears in **ball/SIMNTdiab/ball/Debug**.



NOTE: The directory name **SIMNTdiab** reflects the active build spec, which is comprised of build settings appropriate for the VxWorks simulator and the Wind River Compiler. The **Debug** directory reflects the fact that debug mode flags are turned on by default.

If you select a different build spec by right-clicking the project and selecting **Build Options > Set Active Build Spec**, or if you clear the debug mode checkbox, the string will be different.

2.3.5 Creating a Connection Definition to the VxWorks simulator

You create and manage connections to a target, including the VxWorks simulator, using the Target Manager.



NOTE: If you installed VxWorks support when you installed Workbench, a VxWorks simulator connection definition named **vxsim0** automatically appears below **default(localhost)**.

This is a valid connection definition, and you can use it. However, to understand how to manually create new target connections, continue with this tutorial.

1. To create a new target connection definition, click the **Create a New Target Connection** icon on the Target Manager toolbar, or right-click in the Target Manager and select **New > Connection**.
2. In the **New Connection** wizard, select **Wind River VxWorks Simulator Connection**, then click **Next.**
3. Continue clicking **Next** to accept all of the default configuration settings, then click **Finish** to create your connection definition. Because the **Immediately connect to target if possible** box is selected by default, Workbench attempts to connect to the simulator.

3. For more information about include paths, open the build properties dialog by right-clicking on your project and selecting **Properties**, then press the help key for your host.

Workbench displays **connecting**, then **connected - target server running** in the Workbench status line at the bottom of the window[4] once the connection is made. A VxWorks simulator window opens[5], and the connection appears under **default(localhost)**, with the type of target displayed under the connection.

You are now ready to run the sample program.

2.3.6 Downloading the Program and Attaching the Debugger

1. In the **Project Navigator**, right-click the build target **ball/SIMNTdiab/ball/Debug/ball.out**, then select **Debug Kernel Task**. The **Debug** launch configuration dialog appears, with **ball.out** already filled in as part of the **Name** of the launch.
2. Type **main** in the **Entry Point** field (or click **Browse** and select Downloads > ball.out > main), then click **Debug**.
3. Several events now occur: Workbench automatically builds the **ball** project, switches to the Device Debug perspective, runs the **ball** program on the simulator, attaches the debugger, executes the program up to **main( )**, and then breaks.

For more information about using the other tabs and fields in the launch configuration dialog, see *22. Launching Programs* or open the launch configuration dialog and press the help key for your host.

2.3.7 Setting Up the Device Debug Perspective

The views in the Device Debug perspective can be repositioned to suit your needs.

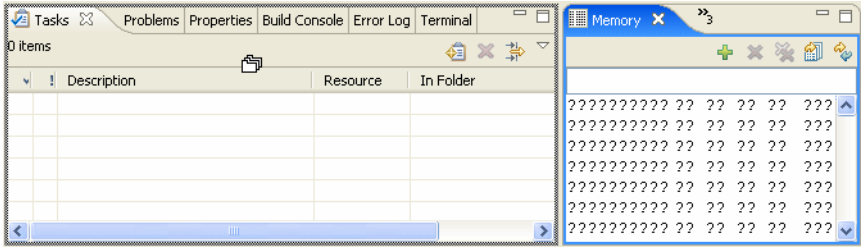
To set up the Device Debug perspective to match this tutorial:

1. The action of the **ball** program is displayed in the Memory view, so select **Window > Show View > Memory**. Click **OK**.

   The Memory view appears in the lower-right corner of the Workbench window, in the tabbed notebook with the Local Variables and Watch views.

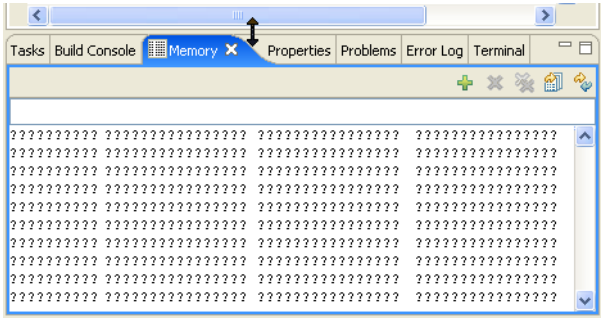
4. To display this and other status information in the Target Manager, select **Window > Preferences > Target Manager > Label Decorations**, then choose what you want to display.
5. You do not need the VxWorks simulator window for this tutorial, so minimize it if you wish, but do not close it. For more information, see *Wind River VxWorks Simulator User's Guide*.

2. Click on the title bar of the Memory view and drag it to the left, over the tabbed notebook containing the Tasks view and the Build Console. Wait for an icon of a set of stacked folders to appear at the cursor, then drop the view.



3. Right-click in the Memory view and select **Display > Items size - 8 bytes**.
4. Resize the Memory view so you see at least 10 rows (place the cursor over the top border of the view, and when it becomes a double-headed arrow, click and drag upwards).
5. Resize the view horizontally so you see one column in the Address pane, two columns in the Binary pane (the central section of the Memory view) and one column in the Text pane (the right-hand section of the view). The view should look similar to Figure 2-1.

Figure 2-1 **Resizing the Memory View**



6. In the Watch view[6], click the **Name** column, then type **grid** and press **ENTER**. The memory address of the **grid** global variable appears in the **value** column. This address can vary from one session to another if something changes, for

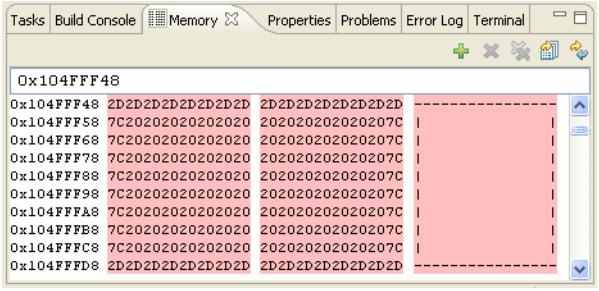
example if you compile with the GNU compiler instead of the Wind River Compiler.

7. Type the memory address of the **grid** global variable into the Memory view address bar and press **ENTER**.
8. To initialize the program, click the **Step Over** icon in the Debug view twice so the Memory view displays an empty box. If necessary, resize the Memory view horizontally or vertically to frame the box correctly, as shown in Figure 2-2.



NOTE: If the box does not appear, make sure the address you entered in the **Memory** window is that of the **grid** global variable.

Figure 2-2 **Initializing the ball Program**



As the program runs, characters representing different types of balls (two zeros, two @ signs, and two asterisks) appear in this empty box, bounce around, and collide with each other and with the walls.

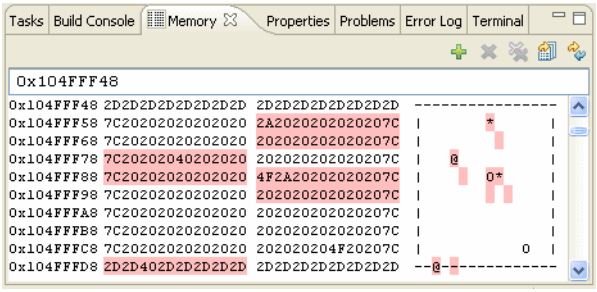
2.3.8 Setting and Running to a Breakpoint.

1. In **main.c** (which should still be open in the Editor) scroll past the three initialization **for** loops and set a breakpoint at the **while** statement by double-clicking in the vertical ruler to the left of it.

   A blue dot appears in the vertical ruler, and the Breakpoints view displays the module and line number of the breakpoint.

6. If it is not visible in the lower right corner, click the >> next to the name tab and select it from the list.

2. With the breakpoint set, run to it by clicking the **Resume** button in the Debug view. Workbench stops when it hits the breakpoint.
3. Examine the Memory view. You should see the six characters of the sample program (representing balls) in the box. Click **Resume** again; colored highlights indicate changes in ball position since the Memory view was last refreshed.



2.3.9 Modifying the Breakpoint

Next, change the behavior of the breakpoint so that at each break, the display will refresh (to show the bouncing balls) without stopping execution.

1. Right-click the breakpoint in the vertical ruler and select **Breakpoint Properties** from the context menu (or right-click the breakpoint in the Breakpoints view and select **Properties**). The **Line Breakpoint Properties** dialog appears.
2. Select **Continue on Break**, change the **Continue Delay** to **50**, then click **OK**.
3. Now click the **Resume** button and watch the balls bounce in the Memory window.
4. To stop the program, open the **Breakpoint Properties** dialog again, clear **Continue on Break**, then click **OK**. The balls may bounce once more after you click **OK**, but they will stop.

2.4 Tutorial: Editing and Debugging Source Files

This tutorial demonstrates how Workbench can help you with some of the most basic activities in development: editing code, building your project and noting where the build fails, and tracking and fixing errors.

2.4.1 Before You Begin

To set up Workbench for this tutorial, switch back to the Application Development perspective by clicking its icon in the upper-right corner of the Workbench window.

2.4.2 Introducing an Error into the Source Code

Because the ball sample program is shipped without errors, you must introduce one into the sources in order to view a failed build.

1. In the Project Navigator, double-click **main.c** to open it in the Editor.
2. Select **main( )** in the Outline view. The Editor switches focus to display it.
3. Delete the semicolon (**;**) after the call to **gridInit( )** so that it reads as follows:

```
gridInit()
```



NOTE: The status bar at the bottom of the Workbench window displays the line number and column (61:16) where your cursor is located in the Editor.

4. Close and save the file.

2.4.3 Tracking Down a Build Failure

1. Build the ball project by right-clicking the **ball** folder in the Project Navigator and selecting **Build Project** from the context menu. Build output appears in the Build Console tab at the bottom of the screen.
2. When the build encounters the error you created in the **main.c** file, the build fails. Workbench displays a red icon containing a white **X** in several places:
   - In the Build Console, which comes to the foreground and displays information about the error, including the general location where the problem is suspected to be.

- In the Project Navigator, which displays red error markers to alert you that the build failure was in the **ball** project, and that **main.c** is the file containing the error.
- In the Problems view, which displays a description of the error, including the filename, folder, and line number.

3. Double-clicking the red icon in any of these locations opens the **main.c** file in the Editor, with the focus at (or near) the line suspected of containing the error.
4. Replace the semicolon after **gridInit**.
5. Save and close the file.

2.4.4 Displaying File History

Workbench tracks all changes that you make to any files in the project. To display the change history of the **main.c** file, right-click the file in the Project Navigator and select **Compare With > Local History**.

The **Compare with Local History** dialog appears. The upper area of the dialog displays a list of the dates and times when the file was changed. When you select one of the list entries in the upper part, the lower part displays the file as of that time on the left and the version before then on the right (that is, the changes associated with that save). Note the changes you just made. When you are finished, click **OK** to close the dialog.



NOTE: You can also use the local history feature to restore deleted files. Right-click the folder the files were in, select **Restore from Local History**, choose the files you want to restore, then click **Restore**.

2.4.5 Rebuilding the Project

Right-click the **ball** folder at the top of the project tree and this time, select **Rebuild Project**. The project compiles with no errors.

2.5 Tutorial: Using the Editor's Code Development Features

The Wind River Workbench editor provides code completion, parameter hints, and bracket matching that can help you develop your code.

2.5.1 Using Code Completion to Add Symbols to Your File

Code completion automatically suggests methods, properties and events as you enter code.

To use code completion, begin typing in the Editor. Right-click in the Editor and select **Source > Content Assist**. You can also use **CTRL+SPACE** to display a pop-up list containing valid choices based on the letters you have typed so far.

For example, in ball's **main.c**:

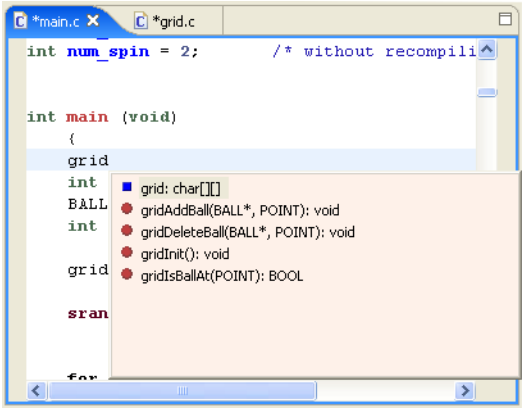
1. Position your cursor inside the function **main( )** to the right of the first **{** character and press **ENTER**. Note that the cursor automatically indents beneath the brace.



NOTE: You can change indentation, brace style, and other code formatting options by selecting **Window > Preferences > General > Editors > Wind River Workbench Editor**.

2. Begin typing **grid** and invoke code completion: **g**, **r**, **CTRL+SPACE**.

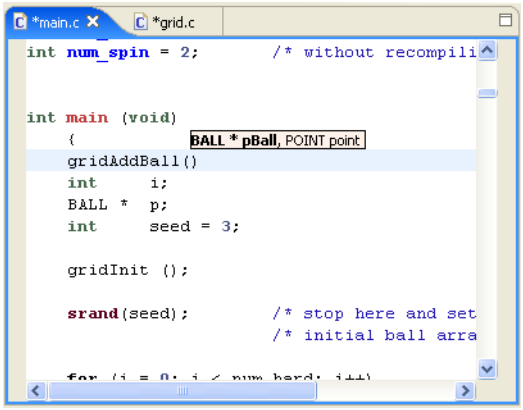
   A dialog appears with suggestions, and as you continue to type the **i** and the **d**, your choices narrow:



Select **gridAddBall( )** and press **ENTER** to add the function to the file.

2.5.2 Using Parameter Hints

Parameter hints describe what data types a function accepts. When you add a function using code completion, or when you enter a function name and an open parenthesis, the Workbench Editor automatically displays any available parameter hints.



You can also request parameter hints as you enter your code by right-clicking in the Editor and selecting **Source > Parameter Hints**, or by using the **CTRL+SHIFT+SPACE** keyboard shortcut.

2.5.3 Using Bracket Matching to Clarify Syntax

Bracket matching helps you read and troubleshoot complex syntax by highlighting related parentheses, square brackets, and braces.

If you position your cursor before an open bracket or after a closing bracket, a rectangle will enclose the corresponding bracket to make it easier to find. You can jump between the opening and closing brackets by pressing **CTRL+SHIFT+P**. Bracket matching operates on the following characters:

( ), [ ], { }, " ", /* */, < > *(C/C++ only)*

2.5.4 Finding Symbols in Source Files

The easiest way to find a symbol in a source file you are working with is to select it in the Outline view, but sometimes that is not possible. So Workbench also provides other ways to find symbols.

1. If it is not already open, double-click the ball project's **main.c** file to open it.
2. Select **main( ): int** in the Outline view. The Editor immediately switches focus and highlights the declaration of **main( )**.
3. Several lines below **main( )** is the symbol **gridInit( )**. This symbol does not appear in the Outline view because that view only displays symbols declared in the file that is open (and has focus) in the Editor.



NOTE: Hovering over **gridInit( )** displays a pop-up showing the comments and declaration for the function.

4. To see the declaration of **gridInit( )**, double-click it and then press **F3**. The **grid.c** file opens automatically in the Editor, positioned at the declaration of **gridInit( )**.

2.6 Tutorial: Tracking Items of Interest in Your Files

Adding a bookmark to a source file is similar to placing a bookmark in a book: it allows you to find an item you are interested in at a later time by looking in the Bookmarks view. Open the Bookmarks view by selecting **Window > Show View > Bookmarks**.

You can create a bookmark on a particular line of code within a file, or you can bookmark the file itself.

2.6.1 Creating a Bookmark on a Source Line in a File

1. To create a bookmark on a line of code in your file, right-click in the Editor gutter to the left of the item you want to keep track of, then select **Add Bookmark**.

2. In the **Add Bookmark** dialog, enter a meaningful comment to help you identify it later, then click **OK**. A small bookmark icon appears in the Editor gutter, and a marker, or annotation, appears in the overview ruler at the right edge of the Editor showing your bookmark relative to its position in the file. An entry also appears in the Bookmarks view.

   Hovering over the bookmark icon shows you the text you entered, and clicking the annotation on the right will return the Editor's focus to this position if you scroll to a different line in the file.

2.6.2 Creating a Bookmark for an Entire File

1. To create a bookmark for a file, right-click it in the Project Navigator and select **Add Bookmark**.
2. In the **Add Bookmark** dialog, enter a meaningful comment to help you identify it later, then click **OK**.
3. The file will not look any different in the Project Navigator, but the comment you typed, the filename, and the folder appear in the Bookmarks view.

2.6.3 Locating and Viewing Your Bookmarks

1. To see the bookmarks in all your projects, open the Bookmarks view by selecting **Window > Show View > Bookmarks**.
2. To open the file that contains a particular bookmark, double-click the bookmark (or right-click it and select **Go To**). The file opens in the Editor with the bookmark location highlighted.
3. To remove a bookmark you no longer need, right-click it and select **Remove Bookmark**.

2.7 Tutorial: Using Workbench to Debug a VxWorks 5.5.x Target

This tutorial explains how to use Workbench to create a Tornado 2 target server connection and debug a VxWorks 5.5.*x* target.

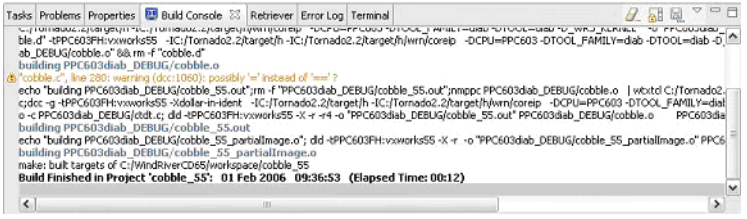
To use these instructions, you must have Workbench **2.***x*, VxWorks 5.5.*x*, and Tornado 2 installed.

2.7.1 Before You Begin

1. To allow Workbench to find your Tornado installation, run the *installDir***/workbench-2.***x***/x86-win32/bin/wrregistert22x.bat** script in a command shell.
2. When the script asks for the location of your Tornado installation, type the path and press **Enter**. The script will update your **install.properties** file.

2.7.2 Creating a Project

1. Select **File > New > Example**, then select **VxWorks 5.5 Downloadable Kernel Module Sample Project** and click **Next**.
2. Select the **Cobble** demo, then click **Finish**. The project appears in the Project Navigator.
3. Right-click the new **cobble_55** project, then select **Build Options > Set Active Build Spec**. From the dialog that appears, change the active build spec to **PPC603diab** and select **Debug mode**. Click **OK**.
4. Right-click the project and select **Build Project**. The Build Console displays a warning about a bug in the code.



5. Double-click the error symbol to open **cobble.c** to line 280, then fix the bug.
6. Edit line 133[7] and change the priority of task **tCrunch** to **210**. If you do not do this, it will appear that breakpoints are not hit.

7. Right-click the project folder and select **Rebuild Project** (Workbench saves your changes before starting the build).

Your project should build cleanly this time.

2.7.3 Creating a VxWorks 5.5.*x* Target Server Connection

Now that you have created your project, you are ready to create a target server connection.

1. From the Target Manager toolbar, click **Create a New Target Connection**. The New Connection wizard opens.
2. From the Connection Type list, select **Wind River VxWorks 5.5.x Target Server Connection**. Click **Next**.
3. Click **Next** through the next few screens, reviewing and customizing the target server options as necessary. Click **Finish** to create your connection definition.



NOTE: If you get a target server connection error, it could be caused by a long delay in checking out a license for the Tornado 2 target server. To lengthen the timeout, select **Window > Preferences > Target Manager** and increase the time in the **Workbench timeout till target server must be connected** field.

Once the connection to the Tornado target server is established, it appears under **default(localhost)** followed by **[Target Server running]**.

4. To connect to the Tornado 2 target, right-click the target server and select **Connect**. The target connection appears under the target server connection[8].

2.7.4 Launching a Kernel Task and Attaching the Debugger

1. In the Target Manager, right-click your target, then select **Target Mode**. Make sure **Task** is selected.
2. Right-click your target again, and this time select **Debug Kernel Task**. The debug launch configuration dialog appears. This dialog allows you to define

7. You can display line numbers by right-clicking in the Editor, selecting **Preferences**, then selecting **Show line numbers**, or you can just scroll up or down and click in the file. The line number and column position of the cursor is displayed at the bottom of the window.
8. If you like, click ⊞ next to Kernel Tasks to display the list of kernel tasks running in the system as well as the name of the core file.

which downloadable module to load, which entry point to call, which debugging options to implement, and what the source lookup path should be.

You will see that the **Name** field displays your project's build target and target name, but as yet no entry point is defined.

3. Click **Browse** next to the Entry Point field, then select **Downloads > cobble_55.out > progStart**. Click **OK**.
4. Click the Download tab to bring it to the foreground, then click **Add**. In the Download dialog, type or browse to the location of your project's build target (*installDir*/**workspace/cobble_55/PPC603diab_DEBUG/cobble_55.out**). Make sure **Load Symbols to Debug Server** is selected, then click **OK**.
5. Click the Debug Options tab to bring it to the foreground. Select **Automatically attach spawned Kernel Tasks**.
6. The launch configuration is now complete. Click **Debug** to launch the task and attach the debugger. Workbench changes to the Device Debug perspective, displays the task in the Debug view, and opens **cobble.c** in the Editor (if it is not already open) with the focus in **progStart**.

2.7.5 Setting and Running to a Breakpoint

The easiest way to find a particular function that you want to place a breakpoint on is to use the Outline view.

1. The Outline view does not appear by default in the Device Debug perspective, so open it by selecting **Window > Show View > Outline**.
2. Select the function **cosmos**; the Editor will shift focus to that section of cobble.c, with **cosmos** highlighted.
3. Scroll down to line 166 (containing **nadaNichtsIdx**) then right-click in the left Editor gutter and select **Breakpoints** to open the Breakpoints submenu.
4. When adding a breakpoint, you can specify the breakpoint's scope: either the task that is selected in the Debug view (here, **tProgStart**) or every task (**Unrestricted**). In this example, the code is run by the task **tCosmos**, not **tProgStart**, so select **Add Breakpoint (Scope = Unrestricted)**. If you had selected **Scope = tProgStart**, the breakpoint would never have been triggered.
5. Workbench also allows you to specify the stop scope: either **Stop Triggering Thread**, or to **Stop All**. However, in VxWorks 5.5, **Stop All** is not supported; it behaves the same as **Stop Triggering Thread**. So in this example you do not need to select either one.

6. With your breakpoint set, select **tProgStart** in the Debug view and click **Resume**.

   The task **tProgStart** disappears from the Debug view; its only purpose was to launch the **tCrunch**, **tCosmos**, and **tMonitor** tasks that now appear in the Debug view and under **Kernel Tasks** in Target Manager.

7. Select the task **tCrunch** in the Debug view to set the scope, then select the function **crunch** in the Outline view. The Editor switches its focus and highlights the function. Several lines below **crunch**, right-click in the gutter beside line 268 (beginning **while**) and select **Breakpoints > Add Breakpoint (Scope = tCrunch)**.

8. Back in the Debug view, select **tCosmos** and click **Resume**. When **tCosmos** hits its breakpoint, click **Resume** again 9 more times. At this point the task **tCrunch** hits its breakpoint and both tasks stop.

2.7.6 System Mode Debugging

In system mode, when a breakpoint is hit, the whole system stops.

1. Before switching to system mode, highlight **tCrunch** and **tCosmos** in the Debug view, click **Resume**, then click **Disconnect** (not **Terminate**).

2. In the Target Manager, right-click your target (which should still be running), then select **Target Mode > System** to switch into system mode.

3. Open the target console by right-clicking your target and selecting **Target Tools > Target Console**. At the prompt, type **i** to display the list of running tasks. If you clicked **Disconnect** (and not **Terminate**), **tCosmos** and **tCrunch** are still running.

4. In the Target Manager, right-click your target and select **Attach to Kernel (System Mode)**.

   Once the target is in system mode, you can right-click various system tasks and select **Attach to Kernel Task (System Mode)**. Then when the system stops, you can get the backtrace of the tasks you have attached.

5. Right-click **tCosmos** and **tCrunch** and select **Attach to Kernel Task**. The tasks appear in the Debug view.

6. Select **monitor** in the Outline view; this will switch the Editor's focus to that part of **cobble.c**.

7. Set a breakpoint by right-clicking in the gutter next to line 302 (beginning **if**).

If you select **Breakpoints > Add Breakpoint (Scope = tCosmos)**, the breakpoint will never be triggered because the code is only run by the **tMonitor** task.

Therefore, set the breakpoint using either **Scope = Unrestricted** or **Scope = tMonitor** (you must select **tMonitor** in the Debug view before you can choose it as the breakpoint scope).

8. When the breakpoint is triggered, the whole system stops, as shown in the Target Manager and in the Debug view. If you try to type something in the target console, nothing appears because the whole target is stopped.
9. Remove the breakpoint by right-clicking it in the Breakpoints view and selecting **Remove**. In the Debug view, select **tMonitor** and click **Resume** to resume the system.

2.7.7 Using Core Dump Files

You can use core dump files to see backtraces of various tasks.

1. On the Target Manager toolbar, click **Create a New Target Connection**.
2. In the New Connection wizard, select **Wind River VxWorks 5.5.x Core Dump Connection**, then click **Next**.
3. On the next screen, type in or navigate to the location of your core dump file and the VxWorks kernel image. Click **Next** through the next few screens and adjust settings if necessary, then click **Finish**. Since **Immediately connect to target if possible** is selected by default, the connection definition will appear in the Target Manager and Workbench will connect.
4. A dialog appears telling you that the core dump was successfully attached, but since Workbench cannot determine the cause for a VxWorks 5.5 core dump the cause is listed as **UNKNOWN**. The dialog also displays the program counter of the current execution context.
5. In the Debug view, the backtrace of the current execution context appears. Note that the run control icons are disabled. You can also attach to other tasks and see their backtraces.

2.7.8 Using Already Available Tornado 2 Projects

You can import existing Tornado 2 projects into Workbench.

1. Create a new user-defined project by selecting **File > New > User-Defined Project**.
2. In the Target Operating System dialog, select **Wind River VxWorks 5.5** (this allows you to use the Tornado 2 compilers). Click **Next**.
3. On the next screen, type a descriptive name into the **Project name** field, then select **Create project at external location** and type in or browse to the location of your existing Tornado 2 project. Click Next.
4. A dialog appears telling you that the directory you pointed to already contains project information. Click **Yes** to overwrite existing project information.
5. Click **Finish**. Your project now appears in the Project Navigator.
6. Right-click the new project and select **Build Project**. Since this is a user-defined project, the build calls the Makefile generated by Tornado 2 tools. If you need to add or remove files, you still need to use the Tornado 2 IDE or edit the Makefile manually.

To launch Tornado from Workbench, select **Target > Launch Wind River Tornado**.

To debug a kernel module, proceed as described in *2.7.4 Launching a Kernel Task and Attaching the Debugger*, p.29. You can also import a kernel project and rebuild it as well.

Workbench provides added value over Tornado in your ability to use the Search view, the Outline view, and the very powerful static analysis tools to manage your projects.

Limitations and Known Issues

To have module synchronization, you must specify the **-s** option to the target server.

When the target loads a module, it appears in the Target Manager. You can select and delete it, and it will disappear from the Target Manager, but the module is still running on the target. This is because the target server cannot remove a module loaded by the target. This is a limitation of Tornado 2, and the Workbench debugger cannot overcome this limitation.

This chapter has been a brief introduction to basic operations with perspectives, views, and editors, and simple debugging capabilities. The rest of this guide provides more in depth information about these and other features of Wind River Workbench.

3 Setting Up Your Development Environment

3.1 Introduction 35

3.2 Configuring Your Cross-Development System 40

3.3 Setting Up a Boot Mechanism 49

3.4 Booting VxWorks 50

3.5 Configuring Host-Target Communication for Workbench 61

3.6 Troubleshooting VxWorks Problems 67

3.1 Introduction

This chapter explains how to configure your host and target, including how to download a VxWorks image and boot your target.

The most common development environment setup uses both a serial and a network connection between the host and target. The serial connection is used to communicate with the boot loader, and the network connection is used to transfer files, including the VxWorks system image. A default VxWorks image is provided for this configuration.

For a discussion of common configuration and setup problems and tips for how to solve them, see *25.5 Troubleshooting VxWorks Configuration Problems*, p.355. For definitions of terminology that may be unfamiliar to you, see *D. Glossary*.

You do not need much of this chapter if all you want to do is connect to a target that is already set up on your network. If this is the case, read *3.2 Configuring Your Cross-Development System*, p.40 and then proceed with *3.4 Booting VxWorks*, p.50.

3.1.1 Overview of Host and Target Configuration Tasks

Host Configuration Tasks

You will need to complete these configuration tasks once per host:

- Install Wind River Workbench.
- Configure TCP/IP for your host.
- Configure a method for transferring a VxWorks image to your target, such as FTP.

Target Configuration Tasks

You will need to complete these configuration tasks once *for each new target*:

- Install the VxWorks boot loader for your target (see the *Wind River Workbench On-Chip Debugging Guide* for details).
- Set up one or more physical connections between your target and host.
- Define a Workbench target server to connect to the new target.

Normal Operation

You will need to repeat these tasks each time you want to re-initialize your target during development:

- Boot VxWorks on the target. VxWorks includes a target agent, the interface between VxWorks and all other Wind River Workbench tools.
- Launch or restart a Workbench target server on the host.



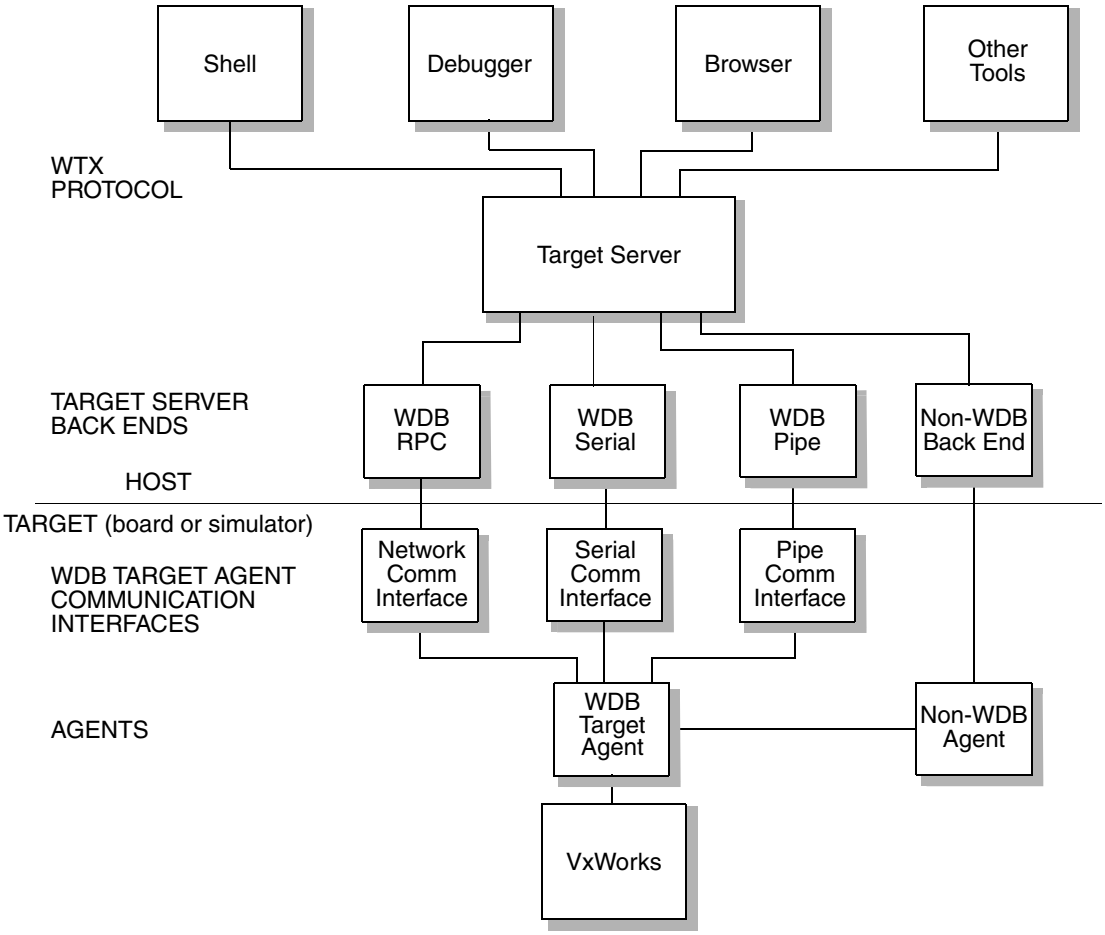
NOTE: Paths to Workbench directories and files are prefixed by *installDir* in this guide. Substitute the actual path to your Workbench installation directory.

3.1.2 Understanding Target Servers and Target Agents

Wind River Workbench host tools such as shells and debuggers communicate with the target system through a *target server* running on the host. A target server can be configured with a variety of *back ends*, which provide for various modes of communication with the *target agent* running on the target. VxWorks can be configured and built with a variety of target agent communication interfaces.

Your choice of target server back end and target agent communication interface is based on the mode of communication that you establish between the host and target (network, serial, JTAG, and so on). The target server *must* be configured with a back end that matches the target agent interface with which VxWorks has been configured and built. See Figure 3-1 for a detailed diagram of host-target communications.

Figure 3-1 **Wind River Workbench Host-Target Communication**



Target Agent Modes

All of the target server back ends included with Workbench connect to the target through the target agent. Thus, in order to understand the features of each back end, you must understand the modes in which the target agent can execute. These modes are called *user mode*, *system mode*, and *dual mode*.

- In user mode, the agent runs as a VxWorks task. Debugging is performed on a per-task basis: you can isolate the task or tasks of interest without affecting the rest of the target system.
- In system mode, the agent runs externally from VxWorks, almost like a ROM monitor. This allows you to debug an application as if it and VxWorks were a single thread of execution. In this mode, when the target encounters a breakpoint, VxWorks and the application are stopped and interrupts are locked. One of the biggest advantages of this mode is that you can single-step through ISRs; on the other hand, it is more difficult to manipulate individual tasks. Another drawback is that this mode is more intrusive; it adds significant interrupt latency to the system, because the agent runs with interrupts locked when it takes control (for example, after a breakpoint).
- To support dual mode debugging, VxWorks images are configured with both agents by default: a user-mode agent (**INCLUDE_WDB_TASK**), and a system-mode agent (**INCLUDE_WDB_SYS**). Only one of these agents is active at a time; switching between the two can be controlled from either the Workbench debugger (see *24.5 Using Debug Modes*, p.320) or the host shell (see *Wind River Workbench Host Shell User's Guide*).

In order to support a system-mode or dual-mode agent, the target communication path must work in polled mode (because the external agent needs to communicate to the host even when the system is suspended). Thus, the choice of communication path can affect what debugging modes are available.

Communication Paths

The most common VxWorks communication path—both for server-agent communications during development, and for applications—is TCP/IP networking over Ethernet. That connection method provides a very high bandwidth, as well as all the advantages of a network connection.

Nevertheless, there are situations where you may wish to use a non-network connection, such as a serial line or a ROM-emulator connection. For example, if you have a memory-constrained application that does not require networking, you may wish to remove the VxWorks network code from the target system during development. Also, if you wish to perform system-mode debugging, you need a communication path that can work in polled mode.

Note that the target server back end connection is not always the same as the connection used to load the VxWorks image into target memory. For example, you

can boot VxWorks over Ethernet, but use a serial line connection to exploit a polled-mode serial driver for system-mode debugging.

You can also use a non-default method of getting the run-time system itself into your target board. For example, you might burn your VxWorks run-time system directly into target ROM, as described in the *VxWorks Programmer's Guide: Configuration and Build*.

Or you can use a ROM emulator to quickly download new VxWorks images to the target's ROM sockets. Another possibility is to boot from a disk locally attached to the target; see the *VxWorks Programmer's Guide: Local File Systems*. Individual Board Support Packages (BSPs) may provide other alternatives, such as flash memory; see the reference information for your BSP.

For a tutorial that explains how to use Wind River ICE or Wind River Probe to load the run-time system onto your target, see *Wind River ICE for Wind River Workbench Hardware Reference* or *Wind River Probe for Wind River Workbench Hardware Reference*.

3.2 Configuring Your Cross-Development System

Before VxWorks can boot an executable image obtained from the host, the network software on the host must be correctly configured (see *Configuring Host Software*, p.40), your target must be connected and powered up (see *Verifying Serial Setup and Power*, p.45), and the boot loader must be loaded onto your target.

3.2.1 Configuring Host Software

For your target to communicate with the Workbench host tools, you need to have a Wind River registry, TCP/IP, and FTP running on your host.

The following sections describe these procedures in more detail.

Establishing the VxWorks Target Name and IP Address

You can configure the server that provides Domain Name Service (DNS) so that your computer uses that server to translate system names to network IP addresses.

Consult your operating system documentation on how to configure your system to take advantage of DNS.

If you do not have a domain name server at your site, you can specify how to map machine names to IP addresses in a file called **hosts** (**C:\Windows\system32\drivers\etc\hosts** on Windows, **/etc/hosts** on Linux and Solaris) which records the names and IP network addresses of systems accessible on the network from the local system (otherwise, you would have to identify targets by IP address).

Each line consists of an IP address and the name (or names) of the system at that address.

For example, suppose your host system is called **mars** and has Internet address 90.0.0.1, and you want to name your VxWorks target **phobos** and assign it address 90.0.0.50. The **hosts** file must then contain the following lines:

```
90.0.0.1       mars
90.0.0.50      phobos
```

This configuration is represented in Figure 3-7 on page 53



CAUTION: If you are in a networked environment, do not pick arbitrary IP addresses for your host and target as they could be assigned to someone else. Contact with your system administrator for available IP addresses.

Configuring FTP on Windows

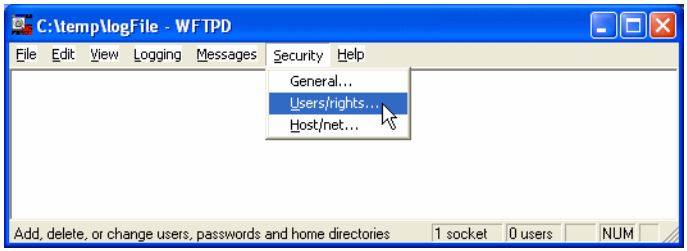
To use the default VxWorks configuration and boot VxWorks over the network, you must have an FTP server running on the host where the VxWorks system image is stored, and the FTP server must have a user ID and password defined that your VxWorks target can use to identify itself.

Workbench includes an FTP server application, **WFTPD**. Start this FTP server from the Windows **Start** menu by selecting **Programs > Wind River > VxWorks 6.*x* and General Purpose Technologies > FTP Server**.

Before an FTP client can connect to WFTPD, you must complete the following steps:

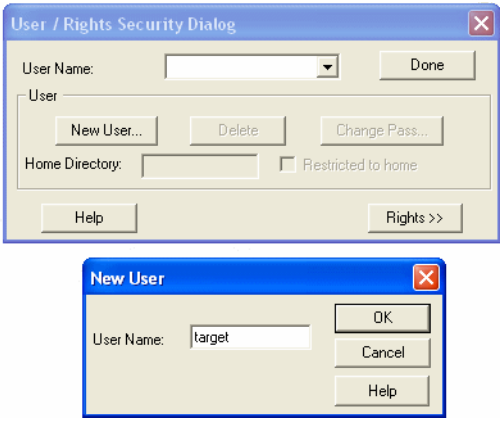
1. Open the WFTPD window and select **Security > Users/rights** (Figure 3-2).

Figure 3-2 **WFTPD Security Menu**



2. WFTPD displays the **User / Rights Security Dialog** box shown in Figure 3-3. Click the **New User** button; another dialog box (also shown in Figure 3-3) appears where you can enter whatever arbitrary name you wish as the user ID for the VxWorks boot ROM. Be sure to use this same name when you assign the **user (u)** VxWorks boot parameter described in *3.4.4 Description of Boot Parameters*, p.55.

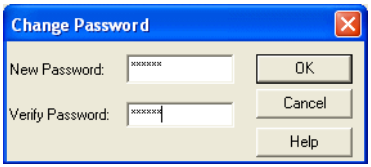
Figure 3-3 **Adding a New User for WFTPD**



3. After you specify the user name and click **OK**, WFTPD displays a third dialog box where you can specify a password (Figure 3-4). Use any memorable arbitrary string, and be sure to use this same string when you assign the **ftp password (pw)** VxWorks boot parameter described in *3.4.4 Description of Boot Parameters*, p.55.

   Because the password does not display as you type it, you must type it twice to be sure the correct password is recorded.

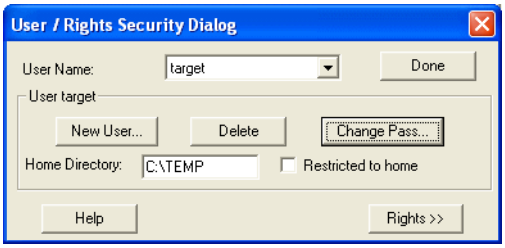
Figure 3-4 **WFTPD Password Dialog Box**



NOTE: Your password must not be an empty string.

4. After defining the new user ID and password, be sure to fill in the **Home Directory** text box (Figure 3-5). The VxWorks boot loader does not require this information, but WFTPD refuses to connect to a client unless you specify a home directory. Any directory will be fine, as long as you permit sufficient disk access for the VxWorks boot loader to read the boot image on your Windows disk.

Figure 3-5 **WFTPD Home Directory**



5. Close the **User / Rights Security Dialog** box by clicking **Done**.

NOTE: You can run the FTP server as a restricted user, but you cannot add new users and passwords if you are a restricted user. A non-restricted user must add the new users and passwords for you.

6. To enable logging of FTP activities, select **Logging > Log Options** and select the types of activities you want to log. The log file will be saved in the home directory you specified above.

When you have finished configuring your FTP settings, leave the FTP server running. It must be running on your host when your target tries to access the VxWorks image.

Configuring FTP on Linux and Solaris

To use the default VxWorks configuration and boot VxWorks over the network, you must have an FTP daemon running on the host that the target is connected to, and the user ID and password that your VxWorks target uses to identify itself must be able to be authenticated by the network.

If desired, you can use **rsh** on Solaris instead of FTP.

Becoming Familiar with the Wind River Registry

The Wind River registry is a service that keeps track of running target servers. The registry must be running for Workbench tools to communicate with VxWorks targets. Workbench development tools communicate with the target server using TCP/IP, which in turn communicates with the VxWorks target over the selected communication method (such as serial, Ethernet, or TMD). The registry is always required, independent of the link between the target server and the VxWorks target.

Workbench starts the default registry automatically, though if required you can connect to a registry running on a networked host instead (see *19.4.2 Connecting to the Target*, p.259 for details about connecting to other registries).

You can tell that the Wind River registry is running on your host system if:

- The registry icon is displayed in the Windows system tray.
- Running the **ps** command on Linux or Solaris shows **wtxregd.ex** in the jobs list.

To shut down the registry:

- Right-click the registry icon in the Windows system tray and select **Shutdown**.
- Type **killall wtxregd.ex** in a Solaris terminal window.
- Type **wrenv.linux -p workbench-2.***x* **wtxregd stop** in a Linux terminal window.

Changing Wind River Registry Daemon Default Behavior

The behavior of the Wind River registry daemon can be changed by updating the registry default options. These options control the location of the registry daemon database, log file locations, levels, and timeouts, and so on.

You can update the registry default options from a terminal window command line, or by modifying the registry daemon default options configuration file (*installDir*/**workbench-2.***x*/**foundation/4.***x*/**resource/wtxregd/wtxregd.conf**).

For available options and other information about the operation of the registry, type *installDir*/**workbench-2.***x*/**foundation/4.***x*/*host_type*/**bin/wtxregd help** at a command line, refer to the **wtxregd.conf** file, or see the online reference entry for **wtxregd** (**Help > Help Contents > Wind River Documentation > References > Host Tools > Wind River Workbench Host Tools API Reference**).

Example Usage

Store the Wind River registry daemon database within a user specific directory.

On Windows:

```
wtxregd -d $(HOME)/registry-db
```

On UNIX:

```
wtxregd start -d $(HOME)/registry-db
```

3.2.2 Verifying Serial Setup and Power

Hardware settings are specific to your target and host. This section describes in general terms the types of hardware connections you must make to follow the instructions in this book, but be aware that you may need to make adjustments to accommodate your specific cross-development system.

Configuring your target hardware may involve the following tasks:

- Protecting your equipment against electrostatic discharge.
- Setting switches and jumpers on the target CPU board.
- Connecting a serial cable and/or an Ethernet cable, if the target supports networking.
- Connecting a power supply.

Perform the following general procedures as appropriate for your particular target hardware. For details, see the target reference for your BSP (such as *installDir*/**vxworks-6.***x*/**target/config/***bspname*/**target.ref**) and the documentation provided by your target system's manufacturer.

NOTE: If you are using a Wind River ICE or Wind River Probe emulator to connect to your target, see the *Wind River ICE for Wind River Workbench Hardware Reference* or *Wind River Probe for Wind River Workbench Hardware Reference* for information about how to connect to your target.

Protecting Equipment from Electrostatic Discharge (ESD)

You should always discharge the static electricity that may have collected on your body before you touch integrated circuit boards, including targets and network interface cards (NICs).

Electrostatic discharge precautions include:

- touching the metal enclosure of a plugged-in piece of electrical equipment (such as a PC or a power supply in a metal case)
- placing your equipment on, or standing on, an anti-static mat
- wearing an ESD wrist strap

CAUTION: Failure to take proper ESD precautions can degrade target hardware over time, leading to intermittent errors or hardware failure.

Setting Board Switches and Jumpers

Many CPU and Ethernet controller boards still have configuration options that are selected by hardware jumpers, although this is less common than in the past. These jumpers must be set correctly before VxWorks can boot successfully.

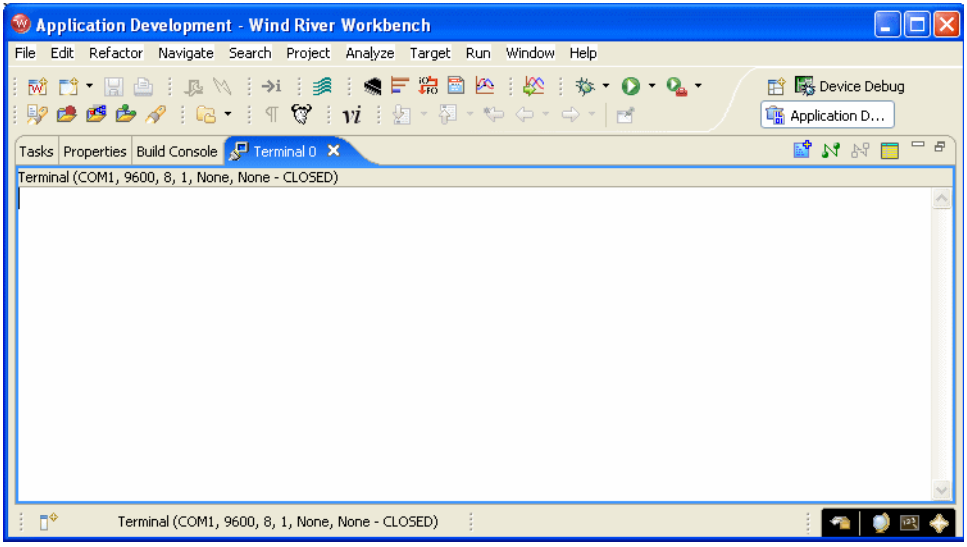
You can determine the correct jumper configuration for your target CPU from the information provided in the target information reference for your BSP, and in the target system's documentation.

Connecting a Serial Cable and Configuring the Terminal View

Most targets include at least one on-board serial port. Wind River Workbench includes a Terminal view that you can use to open a serial connection from within Workbench, just as you would with any other terminal emulation program such as hyperterminal, minicom, or telnet.

To configure the Terminal view:

1. Stop any other program already using the serial port.
2. If it is not already running, start Workbench.
3. If it is not already visible, open the Terminal view (select **Window > Show View > Terminal**).
4. To get a better view of what is happening in the Terminal view, double click on the tab at the top of the view. The view will expand to fill the Workbench window.



5. If the default settings shown at the top of the view are correct for your board, click the green **Connect** icon. You will see **CLOSED** turn to **OPENED**.

   If the settings need to be adjusted, click the square **Settings** button to open the **Terminal Settings** dialog. Configure the terminal settings as appropriate for your system:

   Connection Type: Serial

   Select this for a connection to a local target. On Linux, if you are not running your Linux host as a root user, make sure the permissions are set correctly for you to access the serial port (if you do not have permissions set correctly, only the **NET** option is available under **Connection Type**).

   To set permissions, issue one of the following commands (depending on which port you plan to use):

```
$ chmod 666 /dev/ttyS0
$ chmod 666 /dev/ttyS1
```

Port
Set to the port you are using. Defaults are **COM1** on Windows, **ttyS0** on Linux, and **/dev/cua/a** on Solaris.

Baud Rate
Configure the baud rate to match the speed of your connection.

Data Bits
Default on all platforms is **8**.

Stop Bits
Default on all platforms is **1**.

Parity
Default on all platforms is **None**.

Flow In
Default on all platforms is **None**.

Flow Out
Default on all platforms is **None**.

Connection Type: Network
Select this for a connection to a remote target.

Host
Type in the IP address of the host the target is connected to.

Port
Set to the port you are using. You can select **telnet** or **tgtcons** from the drop-down menu, or you can type in the port number.

6. Click **OK** to open a connection to your target.
7. To disconnect from your target, click **Disconnect**.
   To reopen the connection with the existing settings, click **Connect**.

After initially configuring the boot parameters and getting started with VxWorks, you may wish to configure VxWorks to boot automatically without a terminal. Refer to the target system hardware documentation for proper connection of the RS-232 signals.

Connecting a Cable for the Ethernet Connection

Always make sure you use the correct cable:

- when connecting your board directly to your host, use a crossover cable
- when connecting your board to a LAN, use a non-crossover cable



CAUTION: Be sure to follow ESD precautions (see *Protecting Equipment from Electrostatic Discharge (ESD)*, p.46) whenever working with integrated circuit boards, including targets and NICs.

Connecting A Power Supply

For standalone targets, use the power supply recommended by the board manufacturer.

3.3 Setting Up a Boot Mechanism

Workbench is shipped with the following VxWorks images, compiled both with the Wind River Compiler and with the GNU compiler:

```
vxWorks
vxWorks_rom
vxWorks_romCompress
vxWorks_romResident
```

In every case, you will need to create your own boot medium.Your board will require one of the following media:

ROM

Most boards boot from ROMs.

For cases where boot ROMs are used to boot VxWorks, install the appropriate set of boot ROMs on your target board(s). When installing boot ROMs, be careful to:

- Install each device only in the socket indicated on the label.
- Note the correct orientation of pin 1 for each device.
- Use anti-static precautions whenever working with integrated circuit devices. For more information, see *Protecting Equipment from Electrostatic Discharge (ESD)*, p.46.

Floppy Disk

Some BSPs for systems that include floppy drives use boot diskettes instead of a boot ROM. For example, Pentium systems usually boot from diskette.

Flash Memory

For boards that support flash memory, the BSP may be designed to write the boot program there. In such cases, an auxiliary program to write the boot program into flash memory is supplied by the board vendor.

For specific information on a particular booting method, see **Help** > **Help Contents** > **Wind River Documentation > Guides > Operating System > VxWorks BSP Developer's Guide**. Instructions for making a floppy disk for booting a Pentium target are in the **target.ref** file for the BSP.

You may also wish to replace a boot ROM, even if it is available, with a ROM emulator. This is particularly desirable if your target has no Ethernet capability, because the ROM emulator can be used to provide connectivity at near-Ethernet speeds. Contact Wind River for information about support for ROM emulators.

3.4 Booting VxWorks

Once you have configured your host software and target hardware, you are ready to boot VxWorks.

With your target connected to your host and a serial connection open in the Terminal view, click **Connect** (see *Connecting a Serial Cable and Configuring the Terminal View*, p.46).



NOTE: If you are using a VxWorks image configured for a network connection (the default), you must have an FTP server running on the host where the VxWorks system image is stored. See *Configuring FTP on Windows*, p.41 or *Configuring FTP on Linux and Solaris*, p.44 for more information.

3.4.1 Default Boot Process

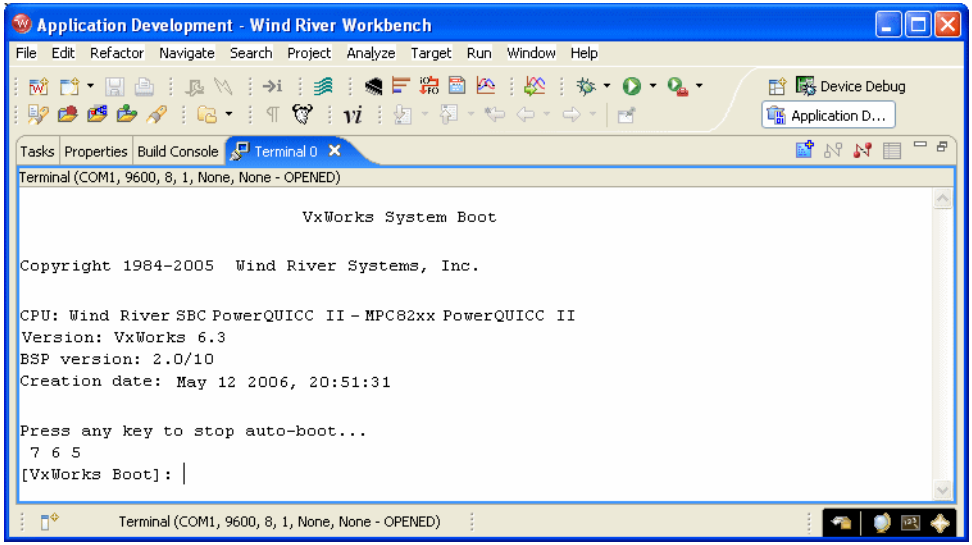
When you boot VxWorks with the default boot program (from a ROM, a diskette, or another medium), you must use the boot loader prompt to provide the boot

program with information that allows it to find the VxWorks image on the host and load it onto the target. The default boot program is designed for a networked target, and needs to have the correct host and target network addresses, the full path and name of the file to be booted, the user name, and so on.

Unless your target CPU has non-volatile RAM (NV-RAM), you will eventually find it useful to build a new version of the boot loader that includes all parameters required for booting a VxWorks image. In the course of developing an application, you will also build bootable applications

When you power on the target hardware (and each time you reset it), the target system executes the boot program from ROM; during the boot process, the target uses its serial port to communicate with your terminal or workstation. The boot program first displays a banner page, and then starts a seven-second countdown, visible on the screen as shown in Figure 3-6.

Figure 3-6 **Boot Program Banner Display**



Unless you press any key on the keyboard within that seven-second period, the boot loader will attempt to proceed with a default configuration, and will not be able to boot the target with VxWorks.

3.4.2 Entering New Boot Parameters

To interrupt the boot process and provide the correct boot parameters, first power on (or reset) the target; then stop the boot sequence by pressing any key during the seven-second countdown.

The boot program displays the VxWorks boot prompt:

```
[VxWorks Boot]:
```

To display the current (default) boot parameters, type **p** at the boot prompt:

```
[VxWorks Boot]: p
```

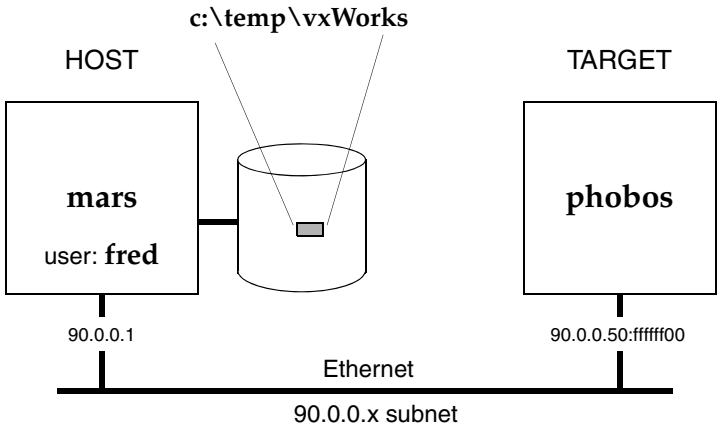
A display similar to the following appears; the meaning of each of these parameters is described in *3.4.4 Description of Boot Parameters*, p.55.

```
boot device             : ln
unit number             : 0
processor number        : 0
host name               : mars
file name               : c:\temp\vxWorks[1]
inet on ethernet (e)    : 90.0.0.50:ffffff00
inet on backplane (b)   :
host inet (h)           : 90.0.0.1
gateway inet (g)        :
user (u)                : fred
ftp password (pw)(blank=use rsh) :secret
flags (f)               : 0x0
target name (tn)        : phobos
startup script (s)      :
other (o)               :
```

This example corresponds to the configuration shown in Figure 3-7. The **p** command does not actually display the lines with blank fields, although this example shows them for completeness.

1. Pre-built VxWorks images are available in *installDir***vxworks-6.***x****target****proj***bsp-compiler***default**, but in this example the **vxWorks** file has been copied to **c:\temp**.

Figure 3-7 **Boot Configuration Example**



To change the boot parameters, type **c** at the boot prompt:

```
[VxWorks Boot]: c
```

In response, the boot program prompts you for each parameter. If a particular field has the correct value already, press **ENTER**. To clear a field, enter a period (**.**), then press **ENTER**. To go back to change the previous parameter, enter a dash (**-**), then press **ENTER**. If you want to quit before completing all parameters, type **CTRL+D**.

Network information *must* be entered to match your particular cross-development system configuration. The Internet addresses must match those in the **hosts** file on your system (or those known to your Domain Name Server), as described in *Establishing the VxWorks Target Name and IP Address*, p.40.

If your target has non-volatile RAM (NV-RAM), the boot parameters are stored there and retained even if power is turned off. For each subsequent power-on or system reset, the boot program uses these stored parameters for the automatic boot configuration.

3.4.3 Boot Program Commands

The VxWorks boot program provides a limited set of commands. To see a list of available commands, type either **h** or **?** at the boot prompt, followed by **ENTER**:

```
[VxWorks Boot]: ?
```

Table 3-1 describes each of the VxWorks boot commands and their arguments.

Table 3-1 **VxWorks Boot Commands**

Command	Description
h	Help command—print a list of available boot commands.
?	Same as **h**.
@	Boot (load and execute file) using the current boot parameters.
p	Print the current boot parameter values.
c	Change the boot parameter values.
l	Load the file using current boot parameters, but without executing.
g *adrs*	Go to (execute at) hex address *adrs*.
d *adrs*[**,** *n*]	Display *n* words of memory starting at hex address *adrs*. If *n* is omitted, the default is 64.
m *adrs*	Modify memory at location *adrs* (hex). The system prompts for modifications to memory, starting at the specified address. It prints each address, and the current 16-bit value at that address, in turn. You can respond in one of several ways: **ENTER**: Do not change that address, but continue prompting at the next address. *number*: Set the 16-bit contents to *number*. **.** (dot): Do not change that address, and quit.
f *adrs*, *nbytes*, *value*	Fill *nbytes* of memory, starting at *adrs* with *value*.
t *adrs1*, *adrs2*, *nbytes*	Copy *nbytes* of memory, starting at *adrs1*, to *adrs2*.
s [**0** | **1**]	Turn the CPU system controller **ON** (**1**) or **OFF** (**0**) (only on boards where the system controller can be enabled by software).
e	Display a synopsis of the last occurring VxWorks exception.

Table 3-1 **VxWorks Boot Commands** (cont'd)

Command	Description
v	Display BSP and boot ROM version.
N	Set Ethernet address.

3.4.4 Description of Boot Parameters

Each of the boot parameters is described below, with reference to the example in *3.4.2 Entering New Boot Parameters*, p.52. The letters in parentheses after some parameters indicate how to specify the parameters in the command line boot procedure described in *3.4.6 Alternate Boot Methods*, p.59.

boot device
The type of device to boot from. This must be one of the drivers included in the boot loader (for example, **enp** for a CMC controller). Due to limited space in the boot media, only a few drivers can be included. A list of included drivers is displayed at the console (type **?** or **h**).

unit number
The unit number of the boot device, starting at zero.

processor number
A unique numerical target identifier for systems with multiple targets on a backplane. The backplane master must have its processor number set to zero. For boards not connected to a backplane, a value of zero is typically used but is not required.

host name
The name of the host machine to boot from. This is the name by which the host is known to VxWorks; it need not be the name used by the host itself. (The host name is **mars** in the example of *3.4.2 Entering New Boot Parameters*, p.52.)

file name
The full pathname of the VxWorks image to be booted (**c:\temp\vxWorks** in the example). This pathname is also reported to the host when you start a target server, so that it can locate the host-resident image of VxWorks. The pathname is limited to a 160 byte string, including the null terminator.[2]

inet on ethernet (e)
The Internet Protocol (IP) address of a target system Ethernet interface, as well as the subnet mask used for that interface. The address consists of the IP address, in dot decimal format, followed by a colon, followed by the mask in

hex format (here, 90.0.0.50:ffffff00). For more information about working with IP addresses, see *Establishing the VxWorks Target Name and IP Address*, p.40.

inet on backplane (b)
The Internet address of a target system with a backplane interface (blank in the example).

host inet (h)
The Internet address of the host to boot from (90.0.0.1 in the example).

gateway inet (g)
The Internet address of a gateway node for the target if the host is not on the same network as the target (blank in the example).

user (u)
The user ID that VxWorks uses to access the host for the purpose of loading the VxWorks image file specified by the **filename** boot parameter (**fred** in the example); use the user name you created in *Configuring FTP on Windows*, p.41. That user must have permission to read the VxWorks boot-image file.

On a Windows host, the user must have FTP access to that host. On a UNIX host, the user must have FTP or **rsh** access. The **ftp password** boot parameter described below controls how the boot loader accesses the host. For **rsh**, the user must be granted access by adding the user ID to the host's **/etc/host.equiv** file, or more typically to the user's **.rhosts** file (**~***userName***/.rhosts**).

ftp password (pw)
The **user** password used by the boot loader to access the host using FTP for the purpose of boot loading the file specified by the **filename** boot parameter. Use the password you created in *Configuring FTP on Windows*, p.41.



NOTE: This field is not required by the boot program, but you must supply it to boot over the network from a Windows host. Without it, the boot loader attempts to load the run-time system image using a protocol based on the UNIX **rsh** utility, which is not available for Windows hosts. So an FTP password is required, but only for host access during boot loading.

2. If the same pathname is not suitable for both host and target—for example, if you boot from a disk attached only to the target—you can specify the host path separately to the target server, using the **-c** *filename* option in the **Advanced Target Server Options** field of the **New Target Server Connection** dialog.

flags (f)

Configuration options specified as a numeric value that is the sum of the values of selected option bits defined below. (This field is zero in the example because no special boot options were selected.)

0x01 = Do not enable the system controller, even if the processor number is 0. (This option is board specific; refer to your target documentation.)
0x02 = Load all VxWorks symbols[a], instead of just globals.
0x04 = Do not auto-boot.
0x08 = Auto-boot fast (short countdown).
0x20 = Disable login security.
0x40 = Use BOOTP to get boot parameters.
0x80 = Use TFTP to get boot image.
0x100 = Use proxy ARP.
0x200 = Use WDB agent.
0x400 = Set system to debug mode for the error detection and reporting facility (depending on whether you are working on kernel modules or user applications, for more information see the *VxWorks Kernel Programmer's Guide: Error Detection and Reporting* or the *VxWorks Application Programmer's Guide: Error Detection and Reporting*).

a. Loading a very large group of symbol can cause delays of up to several minutes while Workbench loads the symbols. For information about how to specify the size of the symbol batch to load, click in the Debug view and press the help key for your host.

target name (tn)

The name of the target system to be added to the host table (here, **phobos**).

startup script (s)

If the kernel shell is included in the downloaded image, this parameter allows you to pass to it the path and filename of a startup script to execute after the system boots. A startup script file can contain only the shell's C interpreter commands. This parameter can also be used to specify process-based applications to run automatically at boot time, if VxWorks has been configured with the appropriate components. See *VxWorks Application Programmer's Guide: Applications and Processes* and *Target Tools*.

other (o)

This parameter is generally unused and available for applications (blank in the example). It can be used when booting from a local SCSI disk to specify a network interface to be included.

3.4.5 Booting With New Parameters

After entering the boot parameters, initiate booting by typing the @ command:

```
[VxWorks Boot]: @
```

Figure 3-8 **VxWorks Booting Display**

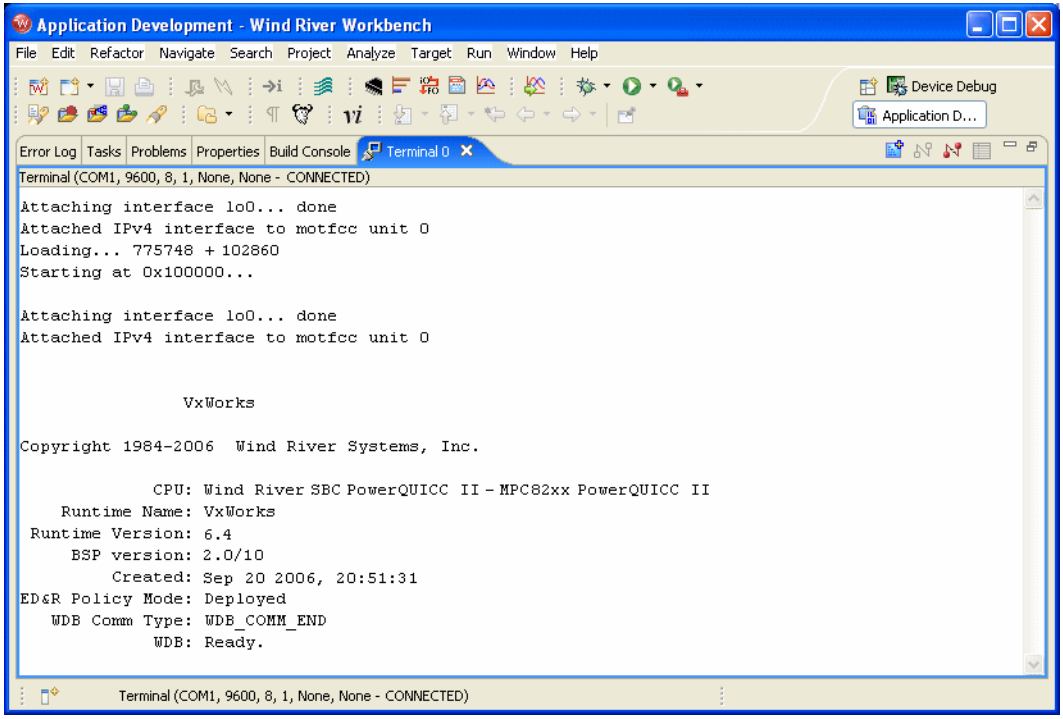


Figure 3-8 shows a typical VxWorks boot display. The VxWorks boot program prints the boot parameters, and the downloading process begins.

The following information is displayed during the boot process:

- The boot program first initializes its network interfaces.
- After the system is completely loaded, the boot program displays the entry address and transfers control to the loaded VxWorks system.
- When VxWorks starts up, it begins just as the boot ROM did, by initializing its network interfaces; the network-initialization messages appear again, sometimes accompanied by other messages about optional VxWorks facilities.

- After this point, VxWorks is up and ready to attach to the Wind River Workbench tools.
- The boot display may be useful for troubleshooting. The following hints refer to Figure 3-8. For more troubleshooting ideas, see *25.5 Troubleshooting VxWorks Configuration Problems*, p.355.
  - If **Attaching network interface** is displayed without the corresponding **done**, verify that the system controller is configured properly and the network interface card is properly jumpered. This error may also indicate a problem with the network driver in the newly loaded VxWorks image.
  - If **Loading...** is displayed for more than 30-45 seconds without the size of the VxWorks image appearing, this may indicate problems with the Ethernet cable or connection, or an error in the network configuration (for example, a bad host or gateway Internet address).
  - If the line **Starting at** is printed and there is no further indication of activity from VxWorks, this generally indicates there is a problem with the boot image.

3.4.6 Alternate Boot Methods

To boot VxWorks, you can also use the command line, take advantage of non-volatile RAM, or create new boot programs for your target.

Command Line Parameters

Instead of being prompted for each of the boot parameters, you can supply the boot program with all the parameters on a single line at the boot prompt (**[VxWorks Boot]:)** beginning with a dollar sign character (“**$**”). For example:

```
$ln(0,0)mars:c:\temp\vxWorks e=90.0.0.50 h=90.0.0.1 u=fred pw=…
```

The order of the assigned fields (those containing equal signs) is not important. Omit any assigned fields that are irrelevant. The codes for the assigned fields correspond to the letter codes shown in parentheses by the **p** command. For a full description of the format, see the reference entry for **bootParseLib**.

This method can be useful if your workstation has programmable function keys. You can program a function key with a command line appropriate to your configuration.

Non-volatile RAM (NV-RAM)

As noted previously, if your target CPU has non-volatile RAM (NV-RAM), all the values you enter in the boot parameters are retained in the NV-RAM. In this case, you can let the boot program auto-boot without having a terminal program connected to the target system.

Customized Boot Programs

See the *VxWorks Kernel Programmer's Guide* for instructions on creating a new boot program for your boot media, with parameters customized for your site. With this method, you no longer need to alter boot parameters before booting.

BSPs Requiring TFTP on the Host

Some Motorola boards that use Bug ROMs and place boot code in flash require the TFTP protocol on the host in order to burn a new VxWorks image into flash. Workbench ships with a version of TFTP. See your target system documentation on how to burn flash for these boards.

3.4.7 Rebooting VxWorks

When VxWorks is running, there are several ways you can reboot it. Rebooting by any of these means restarts the attached target server on the host as well:

- Entering **CTRL+X** in the Terminal view (other Windows terminal emulators do not pass **CTRL+X** to the target, because of its standard Windows meaning.)
- Invoking **reboot( )** from the host shell.
- Pressing the reset button on the target system.
- Turning the target's power off and on.

When you reboot VxWorks in any of these ways, the auto-boot sequence begins again from the countdown.



CAUTION: Be sure to follow ESD precautions (see *Protecting Equipment from Electrostatic Discharge (ESD)*, p.46) whenever working with integrated circuit boards, including targets and NICs.

3.5 Configuring Host-Target Communication for Workbench

If you are developing applications, an Ethernet connection is the easiest to set up and use, since most VxWorks targets already use the network (for example, to boot), so no additional target set-up is required. Furthermore, a network interface is typically a board's fastest physical communication channel.

If you need a JTAG or other emulator connection, see the *Wind River ICE for Wind River Workbench Hardware Reference* or the *Wind River Probe for Wind River Workbench Hardware Reference* for information about making emulator connections to your target.

The next few sections describe the setup of Ethernet and serial line connections within Workbench.

3.5.1 Ethernet Connections

When VxWorks is configured and built with a network interface for the target agent (the default configuration), the target server can connect to the target agent using the default *wdbrpc* back end.



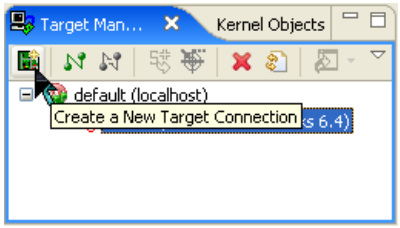
NOTE: If you experience problems when using Workbench tools with a hardware platform with a new Ethernet driver/chipset, it is highly recommended that you use the WDB agent over a different communications link (such as serial or the JTAG Transparent Mode Driver) to isolate if the driver is the source of the problem.

The target agent can receive requests over any device for which a VxWorks network interface driver is installed. The typical case is to use the device from which the target was booted; however, any device can be used by specifying its IP address to the target server.

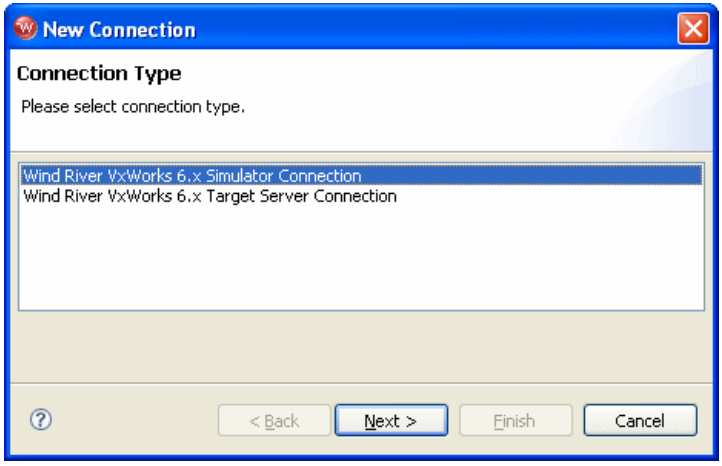
Connecting to the Target Server

You can connect the target server to the agent by following these steps:

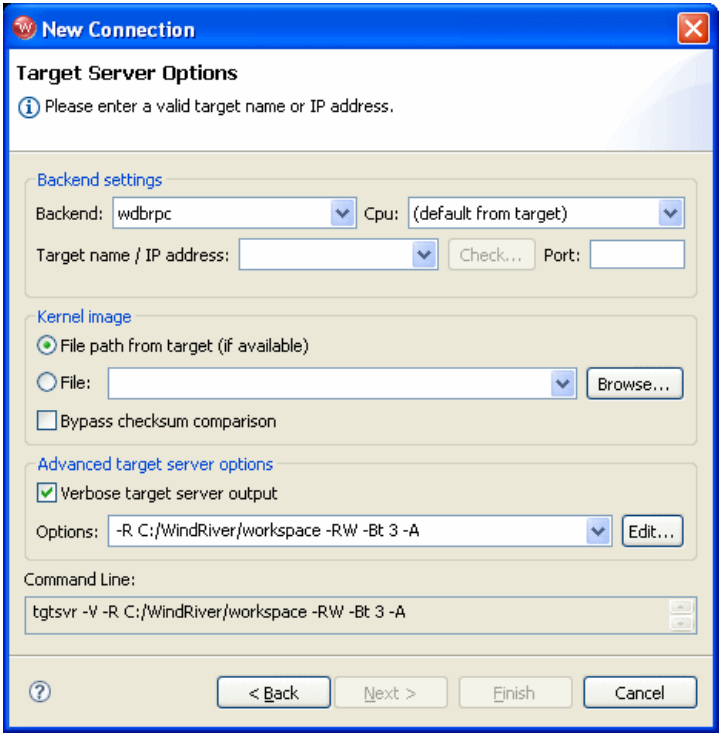
1. Click the **Create a new target connection** icon in the Target Manager toolbar.



The **Connection Type** dialog appears.



2. Select **Wind River VxWorks 6.*x* Target Server Connection** then click **Next.** The **Target Server Options** dialog appears.



3. Select the **wdbrpc** back end, and type in the name or IP address of the target (you may specify a name only if you added it to your hosts file in *Establishing the VxWorks Target Name and IP Address*, p.40).

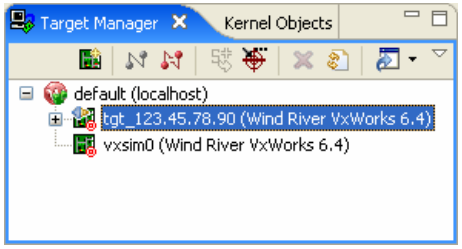
4. In the **Advanced Target Server Options** section, select the **Verbose target server output**.

   Your command line should look like this:

   `tgtsvr -V -R C:/`*`installDir`***`/workspace -RW`** *ipaddress*

5. Click **Next** through the next few screens, then click **Finish**. Your new target server connection definition will appear in the Target Manager connection list, along with the simulator connection definition you created in *2.3.5 Creating a Connection Definition to the VxWorks simulator*, p.17.

   The **Immediately connect to target if possible** box is selected by default, so if your target booted successfully in *Booting With New Parameters*, p.58, the Target Manager will attempt to connect to your target.



6. If everything is set up properly, you will see **connected - target server running** at the bottom of the Workbench window. If you have problems connecting, see *Troubleshooting VxWorks Configuration Problems*, p.355.

3.5.2 Serial-Line Connections

A minimal cross-development configuration is one in which the standalone target is connected to the host development system by a single serial line. For a configuration of this sort, use a combination of a boot mechanism that does not require a network and an alternative Workbench communications back end.

Workbench can operate over a raw serial connection between the host and target systems, and can operate on non-networked systems, but this type of connection is very slow and may not be practical for real-world debugging.

When you connect the host and target exclusively over serial lines, you must:

- Configure and build a boot program to download over the serial connection, or build an image that boots directly from on-board Flash/ROM memory.
- Reconfigure and rebuild VxWorks with a target agent configuration for a serial connection.
- Configure and start a target server for a serial connection.

A raw serial connection has some advantages over an IP connection. The raw serial connection allows you to scale down the VxWorks system (even during development) for memory-constrained applications that do not require networking: you can remove the VxWorks network code from the target system.

When working over a serial link, use the fastest possible line speed. The Workbench tools—especially the debugger—make it easy to set up system snapshots that are periodically refreshed. Refreshing such snapshots requires continuing traffic between host and target. On a serial connection, the line speed

can be a bottleneck in this situation. If your Workbench tools seem unresponsive over a serial connection, try turning off periodic updates in the browser, or else closing any debugger displays you can spare.

Configuring the Target Agent For Serial Connection

To configure the target agent for a raw serial communication connection, reconfigure and rebuild VxWorks with a serial communication interface for the target agent (see the *VxWorks Programmer's Guide* for details).

Configuring the Boot Program for Serial Connection

When you connect the host and target exclusively over serial lines, you must configure and build a boot program for the serial connection because the default boot configuration uses an FTP download from the host.

Testing the Connection

Be sure to use the correct kind of cable to connect your host and target. Use a simple Tx/Rx/GND serial cable because the host serial port is configured not to use handshaking. Many targets require a null-modem cable; consult the target system's documentation. Configure your host system serial port for a full-duplex (no local echo), 8-bit connection with one stop bit and no parity bit. The line speed must match whatever is configured into your target agent.

Before trying to attach the target server for the first time, test that the serial connection to the target is good. To help verify the connection, the target agent sends the following message over the serial line when it boots (with **WDB_COMM_SERIAL**):

```
WDB READY
```

To test the connection, attach a terminal emulator to the target-agent serial port, then reset the target (see *Connecting a Serial Cable and Configuring the Terminal View*, p.46). If the **WDB READY** message does not appear, or if it is garbled, check the configuration of the serial port you are using on your host.

As a further debugging aid, you can also configure the serial-mode target agent to echo all characters it receives over the serial line. This is not the default configuration, because as a side effect it stops the boot process until a target server

is attached. If you need this configuration in order to set up your host serial port, edit *installDir***vxworks-6.***x****target\config\comps\src\wdbSerial.c**.

Look for the following lines:

```
#ifdef  INCLUDE_WDB_TTY_TEST
    {
#if WDB_TTY_ECHO == TRUE
    int waitChar = 0333;
#else   /* WDB_TTY_ECHO == FALSE */
    int waitChar = 0;
#endif  /* WDB_TTY_ECHO == TRUE */

#ifdef INCLUDE_KERNEL
    /* test in polled mode if the kernel hasn't started */

    if (taskIdCurrent == 0)
    wdbSioTest (pSioChan, SIO_MODE_POLL, waitChar);
    else
    wdbSioTest (pSioChan, SIO_MODE_INT, waitChar);
#else   /* INCLUDE_KERNEL */
    wdbSioTest (pSioChan, SIO_MODE_POLL, waitChar);
#endif  /* INCLUDE_KERNEL */
    }
#endif  /* INCLUDE_WDB_TTY_TEST */
```

In each call to **wdbSioTest( )**, change **waitChar** to **0300**.

With this configuration, attach any terminal emulator on the host to the COM port connected to the target to verify the serial connection. When the serial-line settings are correct, whatever you type to the target is echoed as you type it.



NOTE: This configuration change also prevents the target from completing its boot process until a target server attaches to it. Thus, it is best to change the **wdbSioTest( )** calls back to the default as soon as you verify that the serial line is set up correctly.

Connecting to the Target Server

After successfully testing the serial connection, you can connect the target server to the agent by following steps similar to those in *Connecting to the Target Server*, p.61:

1. Close the serial port that you opened for testing (if you do not close the port, it will be busy when the target server tries to use it).
2. Click the **Create a new target connection** icon in the **Target Manager** toolbar. The **Connection Type** dialog appears.

3. Select **Wind River VxWorks 6.*x* Target Server Connection** then click **Next.** The **Target Server Connection** dialog appears.
4. Select the **wdbserial** back end, and type in the name or IP address of the target (you may specify a name only if you added it to your hosts file in *Establishing the VxWorks Target Name and IP Address*, p.40).
5. In the **Advanced Target Server Options** section, select **Verbose target server output**, then specify the communications port with **-d**, and also specify the line speed to match the speed configured into your target. Your command line should look like this:

   `tgtsvr -V -d` *comport* **`-bps`** *speed* **`-B wdbserial`** *ipaddress*
6. Click **Next** through the next few screens, then click **Finish**. Your new target server connection definition will appear in the Target Manager connection list.
7. Select the target server definition you just created, then click the **Connect** icon. If everything is set up properly, you will see **connected - target server running** after the target server connection. If you have problems connecting, see *Troubleshooting VxWorks Configuration Problems*, p.355.

3.6 Troubleshooting VxWorks Problems

If you encountered problems booting or exercising VxWorks, there are many possible causes. Read *25.5 Troubleshooting VxWorks Configuration Problems*, p.355 before contacting Wind River customer support. Often, you can locate the problem just by rechecking the installation steps and your hardware configuration.

PART II

Projects

4 Projects Overview 71

5 Creating VxWorks Image Projects 87

6 Creating Boot Loader Projects 103

7 Creating VxWorks ROMFS File System Projects 109

8 Creating VxWorks Real-time Process Projects 113

9 Creating VxWorks Shared Library Projects 123

10 Creating VxWorks Downloadable Kernel Module Projects 133

11 Creating User-Defined Projects 143

12 Creating Native Application Projects 151

13 Working in the Project Navigator 161

14 Advanced Project Scenarios 171

4

Projects Overview

4.1 Introduction 71

4.2 Workspace/Project Location 72

4.3 Creating New Projects 73

4.4 Overview of Preconfigured Project Types 74

4.5 Projects and Project Structures 80

4.6 Project-Specific Execution Environments 83

4.1 Introduction

Workbench uses projects as logical containers and as building blocks that can be linked together to create a software system. The Project Navigator lets you, among other things, visually organize projects into structures that reflect their inner dependencies, and therefore the order in which they are compiled and linked.

Pre-configured templates for various project types allow you to create or import projects using simple wizards that need only minimal input.

4.2 Workspace/Project Location

Wind River Workbench cannot know where your source files are located, so it initially suggests a default workspace directory within the installation directory. However, this is not a requirement, or even necessarily desirable. If you use a workspace directory outside of the Workbench installation tree this ensures that the integrity of your projects is preserved after product upgrades or installation modifications.

Normally, you would set your workspace directory at the root of your existing source code tree and create your Workbench projects there. For multiple, unrelated source code trees, you can use multiple workspaces.

Some considerations when deciding where to create your project:

Create project in workspace
Leave this selected if you want to create the project under the current workspace directory. This is typical for:

- Projects created from scratch with no existing sources.
- Projects where existing sources will be imported into them later on (for details, see *Adding Application Code to Projects*, p.162).
- Projects where you do not have write permission to the location of your source files.

Create project at external location
Select this option, click **Browse**, then navigate to a different location if you want to create the project outside the workspace. This is typical for:

- Projects being set up for already existing sources, removing the need to import or link to them later on.
- Projects being version-controlled, where sources are located outside the workspace.

Create project in workspace with content at external location
Select this option, click **Browse**, then navigate to your source location if you do not want to mix project files with your sources, or copy sources into your workspace. This is useful for:

- Projects where you do not have write permission to the location of your source files.
- Projects where team members have their own projects, but share common (sometimes read-only) source files. This option eliminates the need to

create symbolic links to your external files before you can work with them in Workbench.

4.3 Creating New Projects

Although you can create projects anywhere, you would generally create them in your workspace directory (see *4.2 Workspace/Project Location*, p.72). If you follow this recommendation, there will generally be no need to navigate out of the workspace when you create projects. Note that if you do create projects outside the workspace, you must have write permission at the external location because Workbench project administration files are written to this location.

To create a new project, click the  toolbar icon or select **File > New > Wind River Workbench Project** to open the New Wind River Workbench Project wizard. It will help you create one of the pre-configured project types. You can also select the specific type of project you want to create by clicking the  toolbar icon or by selecting **File > New >** *ProjectType*. For more information about these projects, see *Overview of Preconfigured Project Types*, p.74.

To create one of the demonstration sample projects, select **File > New > Example** to open the New Example wizard. Each comes with instructions explaining the behavior of the program.

Whichever menu command you choose, a wizard will guide you through the process of creating the specific type of project you select. For step-by-step descriptions of how to create projects of each type, see the following chapters:

- *5. Creating VxWorks Image Projects*
- *6. Creating Boot Loader Projects*
- *7. Creating VxWorks ROMFS File System Projects*
- *8. Creating VxWorks Real-time Process Projects*
- *9. Creating VxWorks Shared Library Projects*
- *10. Creating VxWorks Downloadable Kernel Module Projects*
- *11. Creating User-Defined Projects*
- *12. Creating Native Application Projects*

4.3.1 Subsequent Modification of Project Creation Wizard Settings

All project creation wizard settings can be modified in the Project Properties once the project exists. To access the Project Properties from the Project Navigator, right-click the icon of the project you want to modify and select **Properties**. For more information about project properties, see *16.4 Accessing Build Properties*, p.214.

Project structural settings (sub- and superproject context of the project you are creating) can be most easily modified in the Project Navigator by dragging-and-dropping project folders into or outside other folders.

4.3.2 Projects and Application Code

All application code is managed by projects of one type or another. You can import an existing Workbench-compatible project as a whole, or you can add new or existing source code files to your projects. For more information, select **File > Import** to open the Import File dialog and press the help key for your host.

4.4 Overview of Preconfigured Project Types

Different types of projects are used for different purposes. Workbench supports a number of such project types, each of which will be discussed in more detail in later chapters. This section contains a brief overview of the available project types.

In the Project Navigator, you can identify the project type by its icon.

Table 4-1 **Project Type Icons**

Icon	Project Type
	VxWorks Image Project
	VxWorks Boot Loader Project
	VxWorks Downloadable Kernel Module Project
	VxWorks Real-time Process Project

Table 4-1 **Project Type Icons** (cont'd)

Icon	Project Type
	VxWorks Shared Library Project
	VxWorks ROMFS File System Project
	User-Defined Project
	Native Application Project

NOTE: This manual does not discuss Middleware Component projects, as they are only functional if you have licensed the **Wind River VxWorks Platforms** product. For more information about these projects, see the documentation for your run-time technologies products.

This manual also does not discuss Standalone Application Projects, as they are only functional if you have licensed the **Wind River On-Chip Debugging** product. For more information about these projects, see *Wind River Workbench On-Chip Debugging Guide: Using the OCD Standalone Project Wizard*.

4.4.1 Workbench Sample Projects

A good place to start exploring the sample projects is *2. Wind River Workbench Tutorials*. The tutorials use sample projects to walk you through many aspects of Workbench and shows you some of the project types introduced below.

4.4.2 VxWorks Image Project

Use a VxWorks Image project to configure (customize/scale) and build a VxWorks kernel image to boot your target. By adding a VxWorks ROMFS File System project and kernel modules, applications, libraries, and data files, you can link a complete system into a single image.

A new VxWorks Image project can be based either on an existing project of the same type, or on a *Board Support Package*. For more information, please see *5.7 Notes on Board Support Packages (BSPs)*, p.100.

Refer to *5. Creating VxWorks Image Projects* for more information on working with this type of project.

4.4.3 VxWorks Boot Loader Project

Use a *VxWorks Boot Loader* project to create a VxWorks *boot loader* (also referred to as the VxWorks *boot ROM*) to boot load a target with the VxWorks kernel.

Boot loaders are used in a development environment to load a VxWorks image that is stored on a host system, where VxWorks can be quickly modified and rebuilt. Boot loaders are also used in production systems where both the boot loader and operating system image are stored on a disk.

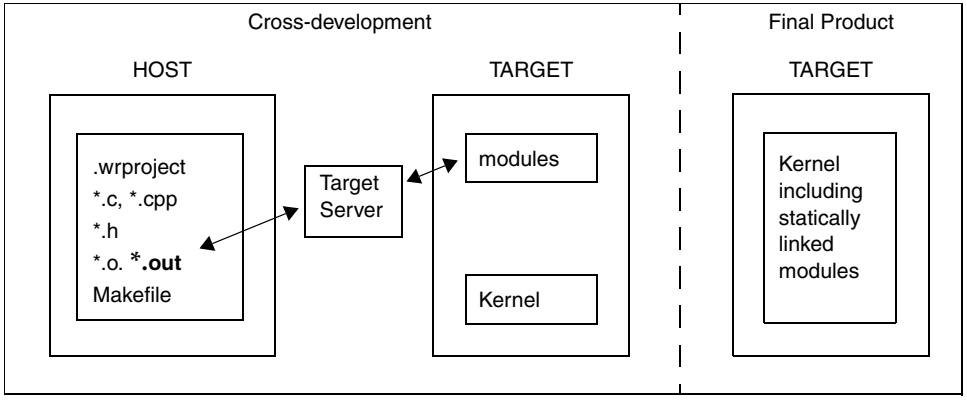
Boot loaders are not required for standalone VxWorks systems stored in ROM.

Refer to *6. Creating Boot Loader Projects* for more information on working with this type of project.

4.4.4 VxWorks Downloadable Kernel Module Project

Use *Downloadable Kernel Module* projects to manage and build modules that will exist in the kernel space. You can separately build the modules, run, and debug them on a target running VxWorks, loading, unloading, and reloading on the fly. Once your development work is complete, the modules can be statically linked into the kernel, or they can use a file system if one is present (see *4.4.7 VxWorks ROMFS File System Project*, p.78). Figure 4-1 illustrates a situation without a file system on the target.

Figure 4-1 **Downloadable Kernel Modules: Overview**



Kernel-mode development is the traditional VxWorks method of development; all the tasks you spawn run in an unprotected environment, and all have full access to the hardware in the system.

A Downloadable Kernel Module that is linked into the kernel is a bootable application that starts when the target is booted.

Refer to *10. Creating VxWorks Downloadable Kernel Module Projects* for more information on working with this type of project.

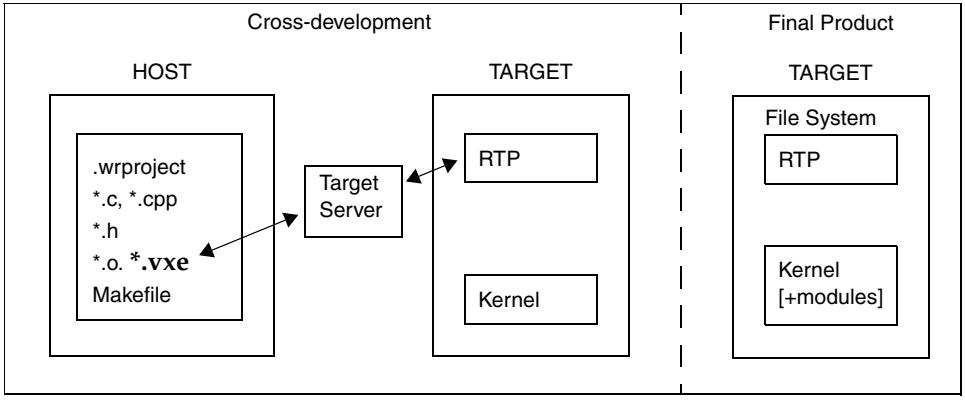
4.4.5 VxWorks Real-time Process Project

Use *VxWorks Real-time Process* projects to manage and build executables that will exist outside the kernel space. You can separately build, run, and debug the executable.

At run-time, the executable file is downloaded to a separate process address space to run as an independent process. A Real-time Process binary can be stored on a target-side file system such as ROMFS, see *7. Creating VxWorks ROMFS File System Projects*.

Figure 4-2 shows how executables, when loaded into a Real-time Process, run as a separate entity from the kernel.

Figure 4-2 **Real-time Processes: Overview**



Refer to *8. Creating VxWorks Real-time Process Projects*, *17.6 Executables that Dynamically Link to Shared Libraries*, p.229, and *18. RTPs and Shared Libraries from Host to Target* for more information on working with this type of project.

4.4.6 VxWorks Shared Library Project

Use *VxWorks Shared Library* projects for libraries that are dynamically linked to VxWorks Real-time Process projects at run-time. Like the Real-time Process project, you will need to store the shared library on a target-side file system, which you can create using *4.4.7 VxWorks ROMFS File System Project*, p.78. You can also use VxWorks Shared Library projects to create subprojects that are statically linked into other project types at build time.

Refer to *9. Creating VxWorks Shared Library Projects*, *17.6 Executables that Dynamically Link to Shared Libraries*, p.229, and *18. RTPs and Shared Libraries from Host to Target* for more information on working with this type of project.

4.4.7 VxWorks ROMFS File System Project

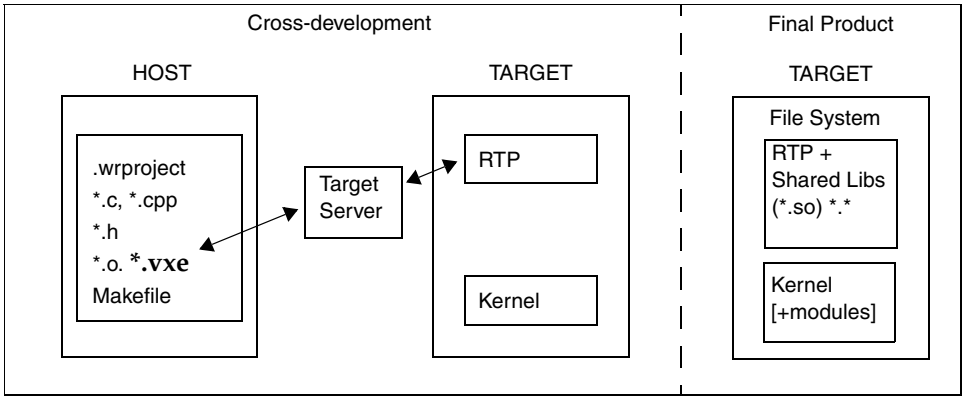
Use a *VxWorks ROMFS File System* project as a subproject of any other project type that requires target-side file system functionality.

So, for example, you may not need a file system project for Downloadable Kernel Module projects (which can be linked to the VxWorks kernel directly, see *10. Creating VxWorks Downloadable Kernel Module Projects* for details), but you will need to create one for other project types.

This project type is designed for bundling applications and other files, of any type, with a VxWorks system image in a ROMFS file system. No storage media is required beyond that used for the VxWorks boot image. Therefore, no other file system is required to store files; systems can be fully functional without recourse to local or NFS drives, RSH or FTP protocols, and so on. Note that the name ROMFS has nothing to do with ROM media. It stands for *Read Only Memory File System*.

Refer to *7. Creating VxWorks ROMFS File System Projects* for more information on working with this type of project.

Figure 4-3 **VxWorks ROMFS File System: Overview**



4.4.8 User-Defined Projects

User-Defined projects assume that you are responsible for setting up and maintaining your own build system, file system population, and so on. The user interface nevertheless provides support for the following:

- You can configure the build command used to launch your build utility; this allows you to start builds from the Workbench GUI.
- You can create build targets in the Project Navigator that reflect rules in your makefiles; this allows you to select and build any of your make rules directly from the Project Navigator.
- You can view build output in the Build Console.

Refer to *11. Creating User-Defined Projects* for more information on working with this type of project.

4.4.9 Native Application Project

Use a *Native Application project* for C/C++ applications developed for your host environment. Wind River Workbench provides build and static analysis support for native GNU 2.9*x*, GNU 3.*x*, and Microsoft development utilities (assembler, compiler, linker, archiver). There is no debugger integration for such projects in Workbench, so you have to use the appropriate native tools for debugging.

4.5 Projects and Project Structures

All individual projects of whatever type are self-contained units that have no inherent relationship with any other projects. The system is initially flat and unstructured. You can, however, construct hierarchies of project references within Workbench. These hierarchies will reflect inter-project dependencies and therefore also the build order.

When you attempt to create such hierarchies of references, this is validated by Workbench; that is, if a certain project type does not make sense as a subproject of some other project type, or even the same project type, such a reference will not be permitted.

4.5.1 Adding Subprojects to a Project

Workbench provides several ways to create a subproject/superproject structure:

- You can drag-and-drop project nodes in the Project Navigator. This is the easiest way to set up a structure among existing projects. Select the project that you want to make into a subproject and drag it onto its new parent (superproject).
- You can use the **Add as Project Reference** dialog. In the Project Navigator, right-click the project that you want to make into a subproject and choose **References > Add as Project Reference**, or select the project and choose **Project > Add as Project Reference**. In the dialog, you will see a list of valid superprojects; you can select more than one.
- You can use the **Project References** page in the **Properties** dialog. In the Project Navigator, right-click the project that you want to make into a *super*project and choose **Properties**, or select the project and choose **Project > Properties**. Then select **Project References**. In the dialog, you will see a list of projects; select the ones that you want to make into subprojects.

Subprojects appear as a subnodes of their parents (superprojects); see Figure 4-4 and Figure 4-5.

Workbench validates subproject/superproject relationships based on project type and target operating system. It does not allow you to create certain combinations. For example, a Real-time Process project cannot be a direct subproject of a VxWorks Image project (but it can be added to a ROMFS File System project). In general, a user-defined project can be a subproject or superproject of any other project with a compatible target operating system.

For additional information about project structure, see *14.4 Complex Project Structures*, p.174.

Removing Subprojects

To undo a subproject/superproject relationship, use one of these methods:

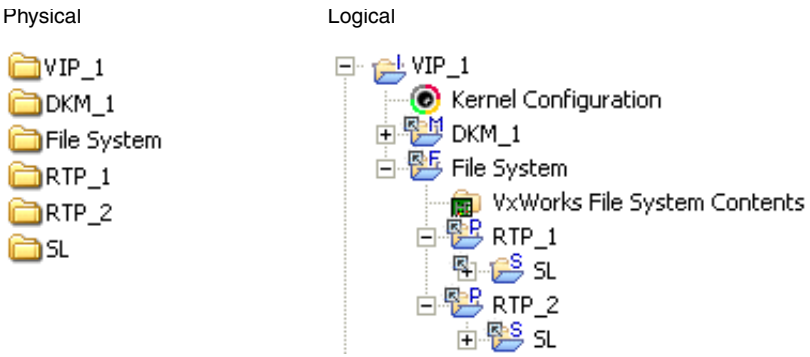
- In the Project Navigator, right-click the subproject and choose **References > Remove Project Reference**, or select the subproject and choose **Project > Remove Project Reference**.
- In the Project Navigator, right-click the superproject and choose **Properties**, or select the superproject and choose **Project > Properties**. Then select **Project References** and uncheck the subprojects that you want to disassociate from their current parent.

4.5.2 Project Structures and Host File System Directory Structure

A tree of directories has only one Workbench project at the top, all subdirectories will automatically be included in this project. Do *not* attempt to create project hierarchies by creating projects for subdirectories in a tree. This will result in overlapping projects, which is not permissible.

Figure 4-4 illustrates an ideal host file system directory structure.

Figure 4-4 **Workspace/Directory Structure and Project Structure**



This flat system, on the left, maps to the project structure displayed on the right, which also represents the ideal (recommended) basic project structure (you may not need all the project types displayed).

The illustrated project structure is achieved as follows:

1. Create a project for each of the directories on the left.
2. In the Project Navigator, drag and drop the individual projects into place.

4.5.3 Project Structures and the Build System

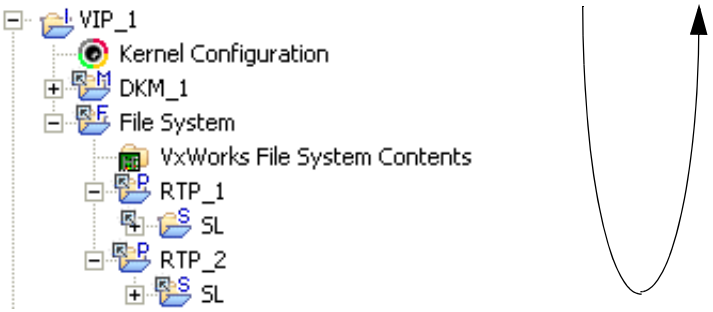
As you can see in Figure 4-4, project structures are logical, not physical, hierarchies. These hierarchies define and reflect the inner dependencies between projects, and therefore also the order in which they have to be built.



NOTE: All references in this section to *build* and *the build system* assume that your projects use Workbench build support. Your user-defined projects are not automatically included in these descriptions, though it is possible to integrate custom projects into such a system.

Figure 4-5 illustrates the build order in this project structure.

Figure 4-5 **Build Order in Project Structures**



The build starts at the top of the structure, recursively checks dependencies in each branch, and builds all out-of-date objects and targets at each leaf, until it finishes at the top of the tree.

Assuming that everything in Figure 4-5 needs to be built, the build order will be:

1. **DKM _1**

2. **SL**
3. **RTP_1**
4. (**SL** already built in 2 above.)
5. **RTP_2**
6. **FS**
7. **VIP_1**

4.5.4 Project Structures and Sharing Subprojects

Project structures can share subprojects. That is, a single physical project can be referenced by (dragged and dropped into) any number of logical project structures.

The products of any update or build of a subproject, or element thereof, will be available to project structures that reference that subproject.

4.5.5 Customizing Build Settings for Shared Subprojects

A single file system folder can be imported into multiple logical project structures, appearing as a subproject of each one. In each case, you can assign a different build specification (known as a *build spec*) depending on what is required by each project.

A folder can also be assigned several different build specs within the same project.

Later, when you set a particular active build spec for the project as a whole, the sub folder that is assigned the same build spec will be included in the build, while others assigned different build specs will be excluded. See *17.5 Architecture-Specific Implementation of Functions*, p.228 for an example.

4.6 Project-Specific Execution Environments

If your development process requires you to maintain different build and external tool execution environments for each of your projects, Workbench allows you to

create a **project.properties** file within each project that define which tools, tool versions, and environment variable settings should be used for each one.

You can share the **project.properties** file with your team to maintain consistency, and you should add it to source control along with your other project files.

1. In the Project Navigator, right-click your project, then select **New > File**.
2. In the New File dialog, create or link to a **project.properties** file:
   - To create a new file, type **project.properties** in the **File name** field, then click **Finish**.
   - To link to an existing **project.properties** file, click **Advanced**, then select **Link to file in the file system**. Type in the path or navigate to the file, then click **Finish**.



NOTE: When sharing files with a team, or accessing them from a common location, it is advisable to use a path variable instead of an absolute path since each team member's path to the location may be different.

To define a path variable, click **Variables**, then click **New**, then type a **Name** for the path variable and the location it represents (or click **File** or **Folder** to navigate to it). Click **OK** twice to return to the New File dialog; your path variable and its resolved location appear at the bottom of the dialog. Click **Finish**.

3. The new **project.properties** file appears under your project in the Project Navigator, and opens in the Editor so you can add or edit its content.
4. The **project.properties** file uses the same syntax as other properties files used by **wrenv** (such as **install.properties** and **package.properties**). For more information about wrenv syntax and options, see *VxWorks Command Line Tools User's Guide: Creating a Development Shell with wrenv*.

   As an example of what you can specify, the following lines define an extension to the **workbench** package, adding the variable **PROJECT_CONTEXT** to the environment with the value of **set**:

```
projectprops.name=projectprops
        projectprops.type=extension
        projectprops.subtype=projectprops
        projectprops.version=0.0.1
        projectprops.compatible=[workbench,,2.6]
        projectprops.eval.01=export PROJECT_CONTEXT=set
```

5. To find the information you will need to create your own extension, find the project's platform by looking to the right of your project's name in the Project Navigator (for example, it might say Wind River VxWorks 6.4).
6. Open your *installDir*/**install.properties** file and look for the section listing the platform information. This is the type, subtype, and other information you must include to identify the package you want to extend.
7. Workbench uses the project properties specified in this file whenever you build a target in the project. To apply the project properties from the command line, include the **-i** option for both the **project.properties** and **install.properties** files when invoking **wrenv**.

```
-i installDir/install.properties -i installDir/workspace/myproject/project.properties
```

In both cases, the environment for **make** is altered to include the environment and properties specified in the file.

4.6.1 Using a project.properties file with a Shell

The **Project > Open Shell** menu item also takes advantage of the settings you specified in the **project.properties** file. This action is context sensitive, so the opened shell sets the environment of the selected project's platform, plus the extension from the properties file if one exists. If you did not have a project selected before opening the shell, a dialog appears with the environments you can choose.

4.6.2 Limitations When Using project.properties Files

A **project.properties** file creates an *extension* to a project, meaning it can include new tools, define variables, and specify versions. But it cannot exclude things that are already included, or overwrite existing variables, or undo PATH settings that are set within the properties you are trying to extend.

You cannot use a **project.properties** file with Native Application projects because they do not have a package associated with them and so cannot be extended.

5 Creating VxWorks Image Projects

5.1 Introduction 87

5.2 Creating a VxWorks Image Project 88

5.3 Importing a VxWorks Image Project 92

5.4 Configuring Kernel Components 93

5.5 VxWorks Image Projects in the Project Navigator 94

5.6 Adding Application Projects to the VxWorks Image Project 99

5.7 Notes on Board Support Packages (BSPs) 100

5.1 Introduction

Use a *VxWorks Image* project (VIP) to configure, customize, scale, and build a VxWorks kernel image to boot your target. A VIP can be a complete application and can also contain projects of other types. For example, you can add Downloadable Kernel Modules or, through an intermediary VxWorks ROMFS File System, you can add Shared Libraries and Real-time Processes to your VIP.

A new VxWorks Image project can be based on an existing VIP (which can be imported into your workspace; see *Importing a VxWorks Image Project*, p.92) or on a *Board Support Package* (see *5.7 Notes on Board Support Packages (BSPs)*, p.100).

5.2 Creating a VxWorks Image Project

Before creating the project, see the general comments on projects and project creation in *4. Projects Overview* and the comments on specifying drivers in *5.2.1 Specifying a Non-Default Driver*, p.91.

1. Create a VxWorks Image Project by selecting **File > New > Wind River Workbench Project**. The New Wind River Workbench Project wizard appears.
2. Select a target operating system, then click **Next**.
3. From the **Build type** drop-down list, select **System Image**. Click **Next**.
4. Type a name for your project.
5. Decide where to create your project:

   Create project in workspace
   Leave this selected if you want to create the project under the current workspace directory.

   Create project at external location
   Select this option, click **Browse**, then navigate to a different location if you want to create the project outside the workspace.

   Create project in workspace with content at external location
   Select this option, click **Browse**, then navigate to your source location if you do not want to mix project files with your sources, or copy sources into your workspace.
6. The next page of the wizard asks what the project is based on. You are asked whether you would like to base your project on **An existing VxWorks Image project**, or on **A board support package** (BSP).
   - If you have already configured a **VxWorks Image project** that closely matches your current needs, you can base your project on that.

     Project creation will be faster using an existing VxWorks Image project since the project facility does not have to regenerate configuration information from BSP configuration files. The files are simply copied.
   - You can select a supplied BSP from the drop-down list, or navigate to a third party or other custom BSP (see also *5.7 Notes on Board Support Packages (BSPs)*, p.100). The list of known BSPs will depend on the BSPs you have installed (including the simulator).



NOTE: Once the VxWorks Image project is created, you cannot change the BSP that it is based on. You must create a new project with the correct BSP.

– If you select **A board support package**, you are asked to select a **Tool chain**. A tool chain is the suite of tools (compiler, linker, and so on) that will be used to build projects. This is part of the build spec that configures how things are built. The available list of tool chains depends on what you have installed.



NOTE: If you intend to select one of the VxWorks scalability profiles, your toolchain must be based on the Wind River Compiler (**diab**).

When you are ready, click **Next**.

7. You are asked to select networking options for the kernel.

   – Select **Use IPv6 enabled kernel libraries** to include IPv6 support.

   – Select **Use System Viewer free kernel libraries** to exclude Wind River System Viewer support. This option builds the project without System Viewer instrumentation, provided the kernel has previously been compiled with **OPT=-fr** or **OPT=-inet6_fr** specified. (Instrumentation-free kernel libraries are not supplied with the product.)

     For information on building the VxWorks kernel, see the source-code installation and build instructions in your getting started guide. For information about System Viewer, see the *Wind River System Viewer User's Guide*.

   – Select **Use source mode build** to build from source, rather than from libraries, whenever possible. This compiles only those parts of the system that are needed by that specific project configuration, greatly increasing its ability to scale VxWorks down to smaller sizes. Source builds also enable the system to perform better, because only the needed source is compiled.

     If the component configuration does not allow a build from source, then the project facility will build from libraries as usual.



NOTE: In this release, only the following BSPs allow configurations that are buildable from source: **integrator1136jfs**, **wrSbcPowerQuiccII**, **Bcm1250_cpu0_mips64**, **Bcm1250_cpu1_mips64**, **Bcm1250eCpu0_mips64**, and **Bcm1250eCpu1_mips64**.

When you are ready, click **Next**.

8. You are asked if you want to select a kernel configuration **Profile**. A *Profile* is a preconfigured collection of kernel components that attempts to match given needs. Selecting a profile can save you quite a bit of manual configuration, but it is not required.

   PROFILE_BASIC_OS—Basic VxWorks OS Profile
   This profile provides a small operating system on which higher level constructs and facilities can be built. It supports an I/O system, file descriptors, and related ANSI routines. It also supports task and environment variables, signals, pipes, coprocessor management, and a VxWorks ROMFS file system. (See the *VxWorks Profiles for a Scaling the Operating System* section in *VxWorks Kernel Programmer's Guide: Kernel*)

   PROFILE_ENHANCED_NET—Enhanced Network Profile
   This profile adds to the default profile certain components appropriate for typical managed network client host devices. The primary components added are the DHCP client and DNS resolver, the Telnet server (shell not included), and several command-line-style configuration utilities.

   PROFILE_BOOTAPP—BootApp Configuration Profile
   This profile provides a VxWorks boot loader. The default boot-loader parameters can be changed with the **DEFAULT_BOOT_LINE** parameter to the **INCLUDE_BSP_MACROS** component. For information about selecting non-default drivers, see *5.2.1 Specifying a Non-Default Driver*, p.91. Note that this profile is not available for all BSPs for this release. If your BSP is not supported for this profile, you must use a boot loader project to create a boot loader (for information about boot loader projects, see *6. Creating Boot Loader Projects*.)

   PROFILE_BASIC_KERNEL—Basic VxWorks Kernel Profile
   This profile builds on the minimal kernel profile, adding support for message queues, task hooks, and memory allocation and deallocation. Applications based on this profile can be more dynamic and feature rich than the minimal kernel. (See the *VxWorks Profiles for a Scaling the Operating System* section in *VxWorks Kernel Programmer's Guide: Kernel*.)

   PROFILE_MINIMAL_KERNEL—Minimal VxWorks Kernel Profile
   This profile provides the lowest level at which a VxWorks system can operate. It consists of the micro-kernel, and basic CPU and BSP support. This profile is meant to provide a very small VxWorks systems that can support multitasking and interrupt management at a very minimum, but semaphores and watchdogs are also supported by default. (See the *VxWorks Profiles for a Scaling the Operating System* section in *VxWorks Kernel Programmer's Guide: Kernel*.)

PROFILE_COMPATIBLE—VxWorks 5.5 Compatible Profile
This profile provides the minimal configuration that is compatible with VxWorks 5.5.

PROFILE_CERT—VxWorks DO-178 Certification Profile
This profile provides a DO-178 Level B-certifiable API subset of the VxWorks operating system.

PROFILE_DEVELOPMENT—VxWorks Kernel Development Profile
This profile provides a VxWorks kernel that includes development and debugging components.

Please refer to the *VxWorks Kernel Programmer's Guide: Kernel* and the help page for **vxprj::profile** for more information about profiles.



CAUTION: The OS scale profiles (**PROFILE_MINIMAL_KERNEL**, **PROFILE_BASIC_KERNEL**, and **PROFILE_BASIC_OS**) are built from source code, so you must install VxWorks source to use them. For this release, the profiles can only be built with the Wind River Compiler. In addition, the profiles are only available for the BSPs listed in the Note on page 89.

9. When you are done configuring your project, click **Finish**. The new VxWorks Image project will appear at the root level in the Project Navigator.



NOTE: If Workbench encounters a problem during project configuration, it will display an error and ask you if you want to delete the project. If you click **OK**, the New Project wizard will reappear with all the settings you chose for the project that failed. This gives you the opportunity to fix just the setting causing the problem, rather than having to re-enter all the selections in the wizard.

If you do not want to fix the problem and recreate the project, click **Cancel**.

5.2.1 Specifying a Non-Default Driver

If your system requires a (supported) driver that is not provided as the default, how you select the appropriate driver and de-select the default depends on whether or not the driver is VxBus-compliant or not.

Drivers that are compatible with the VxBus facility can be added or removed as standard configuration components (for example, **INCLUDE_BCM52XXPHY**) using **vxprj** or Workbench. Drivers that are not compatible with VxBus must be added or

removed by defining or undefining the respective macros in *installDir*/**vxworks-6.***x*/**target/config/***bspName*/**config.h**.



NOTE: Changes to **config.h** must be made before you create a VxWorks image project (using either **vxprj** or Workbench). Any changes made to **config.h** after a VIP is created are not picked up by the project.

For information about the VxBus drivers available for your system, see *installDir*/**vxworks-6.***x*/**target/src/hwif/util/cmdLineBuild.c**. For information about non-VxBus drivers supported for a given BSP, see the VxWorks BSP References.

5.3 Importing a VxWorks Image Project

One situation where you would want to import a VxWorks Image project is if you are using the **vxprj** command-line project facility to build a custom board support package (see the *VxWorks Command-Line Tools User's Guide: Working with Projects and Components* and the **vxprj** API reference entry for more information).

You can then import the custom-built BSP into Workbench as follows:

1. Right-click the Project Navigator and choose **Import**.
2. Select **VxWorks 6.x > Existing VxWorks 6.x Image Project into Workspace**, then click **Next**.
3. Browse to the location of the custom-built BSP (a ***.wpj** file), then click **Finish**.

5.3.1 Migrating a VxWorks Image Project

In Workbench 2.3, the **.wrmakefile** template used to generate the Makefiles for the VIP project contained all functionality on how to build VIP projects.

In Workbench 2.4, the VIP-specific instructions moved to a dedicated **vxWorks.makefile**, which now contains the necessary functionality to build the VIP. The **.wrmakefile** now only covers generic managed build process instructions such as recursion.

So when migrating existing VIP projects created with versions older than Workbench 2.4 to Workbench 2.6, you must update the makefile template manually.

- If you updated your Workbench installation to version 2.5 or 2.6 and want to continue using VxWorks 6.3, copy over the newly installed (version 2.5 or 2.6) **.wrmakefile** and **vxWorks.makefile** to your existing VIP project to cause the project to work properly with the new build system. The simplest way to get these files is to create a new VIP (using the defaults), copy over the two files, and delete the VIP again.
- If you also updated your VxWorks installation to version 6.3 or 6.4, then you must not only copy over the above two files but also run **tcMigrate** to migrate your VIP project from VxWorks 6.1 to 6.2, 6.3, or 6.4.

  For more information about migrating to a new version of VxWorks, see the **tcMigrate** help entry (by typing **tcMigrate** into the help system **Search** field) and the *Wind River Workbench Migration Guide: Workbench Projects*.



NOTE: If you made any manual modifications to your previous **.wrmakefile** file, you must manually migrate those to the new version of the file. If your modifications affected VxWorks image-specific instructions, migrate them to the new **vxWorks.makefile**.

5.4 Configuring Kernel Components

The Wind River Workbench distribution includes VxWorks kernel images located in *installDir*/**vxworks-***version*/**target/config**. A *kernel image* is a binary module that can be booted and run on a target system. The kernel image consists of system object modules linked into a single non-locatable object module with no unresolved external references. In most cases, you will find the supplied kernel image adequate for initial development. However, later in the cycle you may want to create a custom VxWorks kernel image.

The VxWorks kernel is a flexible, scalable operating system with numerous facilities that can be tuned, and included or excluded, depending on the requirements of your application and the stage of the development cycle. For example, various networking and file system components may be required for one application and not another; the Kernel Configuration Editor provides a simple

means for including or excluding such components. In addition, it may be useful to build VxWorks with various target tools (such as the target-resident shell) during development, and then exclude them from the production application.

For more information about kernel components, please refer to the *VxWorks Kernel Programmer's Guide: Kernel*.

5.4.1 The Kernel Configuration Editor

To configure the kernel of a VxWorks Image project, in the Project Navigator, double-click the **Kernel Configuration** node immediately under the VxWorks Image project root node. This opens the **Kernel Configuration Editor**.

The **Kernel Configuration Editor** consists of three tabs (select at the bottom edge of the view).

- The **Overview** tab provides a read-only summary of the configuration that is updated by changes you make on the other two tabs.
- The **Bundles** tab allows you to add or remove entire bundles of components that you can fine-tune to your needs in the **Components** tab.
- The **Components** tab displays a tree of bundles and, at the leaf nodes of expanded bundles, individual components and their parameters.

For more information about the Kernel Configuration Editor, open it and press the help key for your host.

5.5 VxWorks Image Projects in the Project Navigator

After a VxWorks Image project has been created (see *5.2 Creating a VxWorks Image Project*, p.88), a number of nodes appear in the Project Navigator. This section describes these nodes as they appear immediately after project creation, as well as some that only appear after the projects are built using a specific build specification (referred to here, and in the user interface, as a build spec).

For general notes about manipulating nodes, for example, moving, copying, filtering, and so forth, please see *13. Working in the Project Navigator*.

5.5.1 Global Project Nodes



ProjectName

The icon at the root of the project tree identifies the type of project; the icon's label is the name you gave the project when you created it.



Kernel Configuration

Immediately below the project node of a VxWorks Image project, there is the **Kernel Configuration** node. Double-click the **Kernel Configuration** node to open the **Kernel Configuration Editor**. Please refer to *5.4 Configuring Kernel Components*, p.93, for information on using this editor.

5.5.2 Project Build Specs and Target Nodes

Each target node is associated with a predefined build spec.

The default VIP target is a RAM-based image. If you want to create an image of another type, select a different target node when you build the project. See *Creating New Build Targets*, p.96 for more information.



NOTE: What follows is a typical list of build specs. The build specs initially available for a project are determined by the board support package. The VxWorks simulator BSP (see *5.7.1 Using the Simulator BSP*, p.100) does not supply ROM build specs.

default

This represents the target built using the **default** build spec and appears immediately after the project is created. It is a RAM-based image, usually loaded into memory by a VxWorks boot loader. This is the default development image and the only one that is available if you specify a simulator as your target "board". It is also available in formats such as **vxWorks.bin** and **vxWorks.hex**. The **.hex** options are variants of the main options with Motorola S-Record output. The **.bin** options are variants of the main options with binary output.

default_rom

This is a ROM-based image that copies itself to RAM before executing. This image generally has a slower startup time, but a faster execution time than **default_romResident**. It is also available in **.bin** and **.hex** formats.

default_romCompress
A compressed ROM image that copies itself to RAM and decompresses before executing. It takes longer to boot than **default_rom** but takes up less space than other ROM-based images (nearly half). The run-time execution is the same speed as **default_rom**. It is also available in **.bin** and **.hex** formats.

default_romResident
A ROM-resident image. Only the data segment is copied to RAM on startup. It has the fastest startup time and uses the smallest amount of RAM. Typically, however, it runs slower than the other ROM images because ROM access is slower. It is also available in **.bin** and **.hex** formats.

Creating New Build Targets

To add a build target to a project, right-click the project and choose **New > Build Target** (or select the project and choose **File > New > Build Target**). Type a name for the new build target and click **Finish**.

For VxWorks Image projects, build-target names should have the form **vxWorks**[*type*][*format*], where *type* can be empty (the default RAM-based image), **_rom**, **_romCompress**, or **_romResident**, and *format* can be empty (the default ELF image), **.bin**, or **.hex**. Examples:

vxWorks
vxWorks.hex
vxWorks_rom
vxWorks_romResident.hex
vxWorks_romCompress.bin

Each target name corresponds to one of the build specs described above. Target names are case-sensitive and must be spelled correctly to invoke the intended predefined build specs.

5.5.3 Build Output Folders

When you create the project, a node called **vxWorks (default)** is added to the project tree. It will hold the build output of the **default** target (created by setting the active build spec to **default**). Nodes are created for each target as you build

them. The names of the nodes match those of the targets and will, once built, hold the corresponding target's build output.

Other build output folders will be created if you use other build specs. These will have the same names as the build spec used (see *5.5.2 Project Build Specs and Target Nodes*, p.95).

5.5.4 Makefile Nodes

Three Makefiles are created in the project folder. One is a template that can also be used for entering custom make rules. The others are dynamically regenerated from build spec data at each build.



.wrmakefile
A template used by Wind River Workbench to generate the project's **Makefile**. Add user-specific build-targets and make rules in this file. These will then be automatically dumped into the Makefile.



Makefile.mk
Called from **Makefile**. Connects the Workbench project to the VxWorks build system. Includes a list of components and build parameters. Do not edit.



Makefile
Do *not* add custom code to this file. This **Makefile** is regenerated every time the project is built. The information used to generate the file is taken from the build specification on which the target node is based.

5.5.5 Project File Nodes

The project creation facility generates, or includes copies of, a variety of files when a VxWorks Image project is created.

Application Initialization Stubs

Two of the files that are copied to the project at creation time are stubs for entering calls to your application code:



usrAppInit.c
A stub for adding DKM application initialization routines.



usrRtpAppInit.c
A stub for adding RTP application initialization routines.

Other Project Files

Normally, you need not be concerned with the remaining project files. However, here a brief summary of the remaining VxWorks Image project files displayed in the Project Navigator:



projectName.**wpj**
Contains information about the project used for generating the project makefile, as well as project source files such as **prjConfig.c**.



.project
Eclipse platform project file containing builder information and project nature.



.wrproject
Workbench project file containing common project properties such as project type, etc.



linkSyms.c
A dynamically generated configuration file (therefore not to be checked in to your version control system) that includes code from the VxWorks archive by creating references to the appropriate symbols. It contains symbols for components that do not have initialization routines.



prjConfig.c
A dynamically generated configuration file (therefore not to be checked in to your version control system) that contains initialization code for components included in the current configuration of VxWorks.



prjComps.h
A dynamically generated configuration file (therefore not to be checked in) that contains the preprocessor definitions (macros) used to include VxWorks components.



prjParams.h
A dynamically generated configuration file (therefore not to be checked in) that contains component parameters.

5.6 Adding Application Projects to the VxWorks Image Project

Once you have created application projects, populated these with code, and successfully built them, you will want to add these to the VxWorks Image project. You may also want to add a VxWorks ROMFS file system (see *7. Creating VxWorks ROMFS File System Projects*).

Step 1: Link the application projects to the VxWorks Image project.

Some projects, including downloadable kernel modules and user-defined projects, can be managed as subprojects of a VxWorks Image project. If your application projects are not already set up as subprojects of a VIP, see *4.5.1 Adding Subprojects to a Project*, p.80 for information on how to do this. Building VIPs with application subprojects helps assure correct linking and dependency-checking.

RTP and shared-library projects cannot be direct subprojects of a VIP, but they can be subprojects of a File System project that is in turn a subproject of a VIP.

Step 2: Add the application initialization routines to the VxWorks Image project.

When VxWorks boots, it initializes all operating system components (as needed), and then passes control to the user's application for initialization. To add application initialization calls to VxWorks, do the following:

- For DKM projects, double-click **userAppInit.c** to open the file for editing, and add the necessary calls to the **usrAppInit( )** function.
- For RTP projects, double-click **userRtpAppInit.c** to open the file for editing, and add the necessary calls to the **usrRtpAppInit( )** function.

Step 3: Configure the VxWorks Image project VxWorks kernel.

VxWorks must be configured to support the calls your application makes to it, or you will not be able to link your image. If your BSP provides a "bare-bones" VxWorks configuration, you may wish to use the Kernel Configuration Editor's **Auto Scale** facility to detect and add most of the VxWorks functionality you require. **Auto Scale** will compile your code, analyze the symbols in your object modules, map them to components, and offer to include those components. There may be some components that **Auto Scale** does not detect. If you Auto Scale, build, and still get link errors, you will need to add the additional components from the Kernel Configuration Editor (for more information about auto scale and the kernel configuration editor, open the editor and press the help key for your host).

5.7 Notes on Board Support Packages (BSPs)

A *Board Support Package* (*BSP*) consists primarily of the hardware-specific VxWorks code for a particular target board. A BSP includes facilities for hardware initialization, interrupt handling and generation, hardware clock and timer management, mapping of local and bus memory space, and so on.

You can base a VxWorks Image project on the VxWorks simulator BSP, a Wind River BSP supplied with Workbench, or a third-party BSP; or you can create your own custom BSP.

5.7.1 Using the Simulator BSP

You can base your VxWorks Image project on the VxWorks simulator BSP if you want to develop a custom BSP and application code for your product in parallel, or if your target hardware is not yet ready. The simulator BSP contains default VxWorks functionality sufficient for supporting most applications.

5.7.2 Using a Wind River BSP

If your BSP was installed with Workbench 2.6, you can create a VxWorks Image project from it directly (see *5.2 Creating a VxWorks Image Project*, p.88).

For information on migrating a Tornado 2.x-compliant BSP or a SNiFF+ 4.1 (or newer) BSP to Workbench, see the *Wind River Workbench Migration Guide*.

5.7.3 Using a Custom BSP for Custom Hardware

Creating a BSP

If you need to create your own BSP, please refer to the *VxWorks BSP Developer's Guide*. If you wish to develop the BSP and the application code in parallel, you may want to begin application development on the VxWorks Simulator. See *5.7.1 Using the Simulator BSP*, p.100.

Using a Pre-Existing BSP with the Wind River Workbench Project Facility

If you already have a custom BSP that is Tornado 2.x compliant, please see the *VxWorks Migration Guide* for information on migrating to Workbench.

If you already have a custom, non-compliant BSP, you will need to modify it to conform to the guidelines outlined in the *VxWorks BSP Developer's Guide* in order to use it with the Workbench project facility. Once you have modified it, verify that it builds properly before creating a project for it.



NOTE: If you do not make your BSP Workbench compliant, Workbench will not be able to provide project-based support for customizing, configuring, or building it.

Using a BSP Outside of Wind River Workbench

You may use a non-compliant BSP by managing its configuration manually. For information on using manual methods, see the *VxWorks Command-Line Tools User's Guide*. You can still create downloadable projects to hold your application code and download them to a target booted with a non-compliant BSP.

6 Creating Boot Loader Projects

6.1 Introduction 103

6.2 Creating a Boot Loader Project 104

6.3 Creating a Customized Boot Loader 105

6.4 Boot Loader Projects in the Project Navigator 106

6.1 Introduction

Use a *VxWorks Boot Loader* project to create a customized VxWorks *boot loader* (also referred to as the VxWorks *bootrom*) to boot a target with a VxWorks image.

Boot loaders are used in a development environment to load a VxWorks image that is stored on a host system, where VxWorks can be quickly modified and rebuilt. Boot loaders are also used in production systems where both the boot loader and operating system image are stored on a disk.

Because Boot Loader projects provide rudimentary board support (boot loading), they can also be used for loading standalone Downloadable Kernel Module Applications without a full-blown VxWorks kernel.

Boot loaders are not required for standalone VxWorks images, nor is it possible to create a boot loader for an image meant to be run on the VxWorks simulator.

For more information about boot loaders, see the *VxWorks Kernel Programmer's Guide: Kernel* and *3.3 Setting Up a Boot Mechanism*, p.49.

6.2 Creating a Boot Loader Project

Before creating the project, see the general comments on projects and project creation in *4. Projects Overview*.



NOTE: The **PROFILE_BOOTAPP** configuration profile provides a simpler method of creating a boot loader (based on a VxWorks Image Project) than the one described here. It is not, however, available for all BSPs with this release. For more information, see the description of *PROFILE_BOOTAPP—BootApp Configuration Profile*, p.90.

1. Create a VxWorks Boot Loader Project by selecting **File > New > Wind River Workbench Project**. The New Wind River Workbench Project wizard appears.
2. Select a target operating system, then click **Next**.
3. From the **Build type** drop-down list, select **Boot Loader**. Click **Next**.
4. Type a name for your project.
5. Decide where to create your project:

   Create project in workspace
   Leave this selected if you want to create the project under the current workspace directory.

   Create project at external location
   Select this option, click **Browse**, then navigate to a different location if you want to create the project outside the workspace.

   Create project in workspace with content at external location
   Select this option, click **Browse**, then navigate to your source location if you do not want to mix project files with your sources, or copy sources into your workspace.
6. The next page of the wizard asks you to set:
   - The **Board support package** for which you want to create a boot loader.
   - The **Tool chain** that you will use with the project.
   - The **Boot loader image Style** and **Format**.

     Boot loader images come in the following styles: **Compressed**, **Uncompressed**, (ROM-)**Resident**, and (ROM-)**Resident At High Address**. These are functionally the same but have different memory requirements and execution times. After the project has been created, you

can change the **Style** by right-clicking the project and selecting **Set Active Build Spec**.

The *VxWorks Kernel Programmer's Guide: Kernel* chapter provides detailed information on **Style** and **Format**. BSP documentation specifies which types are available for a specific target.

7. When you are ready, click **Finish**. The new project appears at the root level in the Project Navigator.



NOTE: Once the Boot Loader project is created, you cannot change the BSP that it is based on. You must create a new project with the correct BSP.

6.3 Creating a Customized Boot Loader

By default, a Boot Loader project merely creates a default boot loader. You may find it necessary or desirable to customize various features of the boot loader, by doing one or more the following:

- Adding or removing VxWorks components. For example, you can exclude networking components if you are not going to use the network to boot your system.
- Selecting non-default drivers. If the boot loader's default drivers are not appropriate for your target, you need to change the driver selection for the boot loader. For more information, see *6.3.1 Selecting Boot Loader Drivers*, p.106.
- Setting boot parameters that are appropriate for your development environment, or for deployed systems. Boot parameters specify the IP addresses of the host and target systems, FTP user names and passwords, the location of the VxWorks image to boot, and so on. For information about boot parameters, see *3.4.4 Description of Boot Parameters*, p.55.

In order to change the default configuration of a boot loader you must edit the *installDir*/**vxworks-6.x/target/config/***bspName*/**config.h** file. You can open the file in the editor by clicking on **config.h** in the Project Navigator.

For more information about these topics, see the *VxWorks Kernel Programmer's Guide: Kernel*, *3.3 Setting Up a Boot Mechanism*, p.49, and the VxWorks BSP References entry for your BSP.

6.3.1 Selecting Boot Loader Drivers

If your boot loader requires a (supported) driver that is not provided as the default, you must edit *installDir*/**vxworks-6.***x*/**target/config/***bspName*/**config.h** to define the macro for the correct driver, and undefine the macro for the one you do not need.



NOTE: Changes to **config.h** must be made before you create a VxWorks image project (using either **vxprj** or Workbench). Any changes made to **config.h** after a VIP is created are not picked up by the project.

For information about the VxBus drivers available for your system (and the macro names to use in **config.h**), see *installDir*/**target/src/hwif/util/cmdLineBuild.c**. For information about non-VxBus drivers supported for a given BSP, see the VxWorks BSP References entry for the BSP in question. Note that the macro names for VxBus drivers do not have the leading **INCLUDE_** element (for example, **DRV_SIO_NS16550**), whereas the names for non-VxBus drivers do (for example, **INCLUDE_ELT_3C509_END**).

6.4 Boot Loader Projects in the Project Navigator

After a Boot Loader has been created, a number of nodes appear in the Project Navigator. This section describes these nodes as they appear immediately after project creation.

For general notes about manipulating nodes, for example, moving, copying, filtering, etc., please see *13. Working in the Project Navigator*.

6.4.1 Global Project Nodes



ProjectName

The icon at the root of the project tree identifies the type of project; the icon's label is the name you gave the project when you created it.

6.4.2 Project Build Specs and Target Nodes

Each Boot Loader project has a single IDE-managed build target whose name has the form *bsp* **(***buildSpec***)**—for example, **simpc (bootloader_res)**. To switch build specs, right-click and choose **Set Active Build Spec**.

Build-spec names have the form **bootloader***[style][format]*, where *style* can be empty (the default compressed image), **_uncmp** (uncompressed), **_res** (ROM-resident), or **_res_high** (ROM-resident at high address), and *format* can be empty (the default ELF image), **.bin** (binary output), or **.hex** (Motorola S-Record). Examples:

bootloader
bootloader.bin
bootloader_res_high
bootloader_uncmp.hex

You can create new build targets with user-defined make rules by right-clicking on the project and choosing **New > Build Target** or by choosing **File > New > Build Target**.

6.4.3 Makefile Nodes



Makefile
Do *not* add custom code to this file. This **Makefile** is regenerated every time the project is built. The information used to generate the file is taken from the build specification on which the target node is based.

6.4.4 Other Project Files

Normally, you need not be concerned with the remaining project files. However, here a brief summary of the remaining VxWorks Boot Loader project files displayed in the Project Navigator:



.project
Eclipse platform project file containing builder information and project nature.



.wrproject
Workbench project file containing common project properties such as project type, etc.

7 Creating VxWorks ROMFS File System Projects

7.1 Introduction 109

7.2 Creating a VxWorks ROMFS File System Project 110

7.3 Configuring the VxWorks ROMFS File System 110

7.4 VxWorks ROMFS File System Projects in the Project Navigator 111

7.1 Introduction

Use a *VxWorks ROMFS File System* project as a subproject of a VxWorks Image project that requires ROMFS. The VxWorks ROMFS file system provides a means for bundling RTP applications and shared libraries with the VxWorks system image. At runtime, these files can be accessed in the VxWorks **/romfs** directory (and any subdirectories you create).

To use other file systems—such as dosFs—in your applications, configure VxWorks with the appropriate components.

For more information about ROMFS and other file systems, see the *VxWorks Kernel Programmer's Guide: Local File Systems* or the *VxWorks Application Programmer's Guide: Local File Systems*; and the *VxWorks Application Programmer's Guide: Applications and Processes*.

7.2 Creating a VxWorks ROMFS File System Project

Before creating the project, see the general comments on projects and project creation in *4. Projects Overview*.

1. Create a VxWorks ROMFS File System project by selecting **File > New > Wind River Workbench Project**. The New Wind River Workbench Project wizard appears.
2. Select a target operating system, then click **Next**.
3. From the **Build type** drop-down list, select **ROMFS File System**. Click **Next**.
4. Type a name for your project.
5. Decide where to create your project:

   Create project in workspace
   Leave this selected if you want to create the project under the current workspace directory.

   Create project at external location
   Select this option, click **Browse**, then navigate to a different location if you want to create the project outside the workspace.

   Create project in workspace with content at external location
   Select this option, click **Browse**, then navigate to your source location if you do not want to mix project files with your sources, or copy sources into your workspace.
6. When you are ready, click **Finish**. The VxWorks ROMFS File System is created at the root level in the Project Navigator, and the ROMFS File System Contents Editor opens (for more information, see *7.3 Configuring the VxWorks ROMFS File System*, p.110).

7.3 Configuring the VxWorks ROMFS File System

1. If it is not already open, double-click the **VxWorks ROMFS File System Contents** node under the VxWorks ROMFS File System project. This opens the File System Contents Editor so you can add files or create subdirectories.

2. Two panels display the contents of the host and the target. Select files then click **Add** and **Remove** to move files between the two panels. Click **Add External** to add a file from outside your workspace to the target contents.
3. To create a subdirectory, right-click in the **Target Contents** panel and select **Add New Folder to File System**. To remove it, right-click it and select **Remove From File System**.
4. When you are finished, save and close the editor.

Make sure that you add the correct binary or data files. Click the file names in the **Target Contents** pane and verify the path in the **Host path** field in the bottom panel. This can be useful, for example, to check that:

- You have used the correct version of a versioned shared library.
- You have taken files from the correct build-spec output folder.

7.4 VxWorks ROMFS File System Projects in the Project Navigator

After you have created a VxWorks ROMFS file system project, a number of nodes appear in the Project Navigator. This section describes these nodes as they appear immediately after project creation. For general notes about manipulating nodes, for example, moving, copying, filtering, etc., please see *13. Working in the Project Navigator*.

7.4.1 Global Project Nodes



ProjectName

The icon at the root of the VxWorks ROMFS File System project tree identifies the type of project; the icon's label is the name you gave the project when you created it.



VxWorks ROMFS File System Contents

Below the project node is the **VxWorks ROMFS File System Contents** node. Double-click the **VxWorks ROMFS File System Contents** to open the File System Contents Editor. Please refer to *7.3 Configuring the VxWorks ROMFS File System*, p.110, for information on using this editor.

7.4.2 Project File Nodes

The project creation facility generates, or includes copies of, a variety of files when a VxWorks ROMFS File System project is created. Normally, you need not be concerned with these files. However, here is a brief summary of the VxWorks ROMFS File System project files displayed in the Project Navigator:



.project

Eclipse platform project file containing builder information and project nature.



.wrproject

Workbench project file containing common project properties such as project type, etc.

8 Creating VxWorks Real-time Process Projects

8.1 Introduction 113

8.2 Creating a VxWorks Real-time Process Project 114

8.3 Configuring VxWorks Real-time Process Projects 115

8.4 VxWorks Real-time Process Projects in the Project Navigator 120

8.5 Application Code for a VxWorks Real-time Process Project 122

8.6 Linking to VxWorks and Using Shared Libraries 122

8.1 Introduction

Using *VxWorks Real-time Process* (RTP) projects to manage and build modules that will exist outside of the kernel space, you can separately build, run, and debug the VxWorks Real-time Process executable.

At run-time, the executable file is downloaded to a separate address space to run as an independent process. The binary produced from a VxWorks Real-time Process project must be stored on a target-side file system, see *7. Creating VxWorks ROMFS File System Projects*.

VxWorks Real-time Process projects provide a protected, process-based, user-mode environment for developing applications. In this mode, applications are developed as VxWorks executables. An application has a well-defined start address. When the executable is loaded, memory is allocated by the system for the

executable, execution begins at the known start address, and all tasks in the process run within the same memory-protected address space. When the application terminates, all the resources associated with it are freed back to the system.

8.2 Creating a VxWorks Real-time Process Project

Before creating the project, see the general comments on projects and project creation in *4. Projects Overview*.

1. Create a VxWorks Real-Time Process project by selecting **File > New > Wind River Workbench Project**. The New Wind River Workbench Project wizard appears.
2. Select a target operating system, then click **Next**.
3. From the **Build type** drop-down list, select **Real-time Process Application**. Click **Next**.
4. Type a name for your project.
5. Decide where to create your project:

   Create project in workspace
   Leave this selected if you want to create the project under the current workspace directory.

   Create project at external location
   Select this option, click **Browse**, then navigate to a different location if you want to create the project outside the workspace.

   Create project in workspace with content at external location
   Select this option, click **Browse**, then navigate to your source location if you do not want to mix project files with your sources, or copy sources into your workspace.
6. When you are ready, click **Finish**. The VxWorks Real-time Process project is created and appears at the root level in the Project Navigator.

8.3 Configuring VxWorks Real-time Process Projects

Once you have created your project, you can configure its build support and specs, build tools, build macros, and build paths.

For general details about build properties, see *16.4 Accessing Build Properties*, p.214 or press the help key for your host.

1. To access build properties for your project, right-click it in the Project Navigator and select **Properties**.
2. From the **Properties** dialog, click **Build Properties**.

8.3.1 Configuring Build Support and Specs

Use this tab to configure build support and build specs for your project.

1. A VxWorks Real-time Process project is a predefined project type that uses Workbench build support, so build support is enabled by default. If you are creating a project because you want to browse symbol information and you are not interested in building it, click **Disabled** to disable build support (you can click **Managed build** to re-enable it later, if you want).
2. If necessary, edit the default **Build command**.
3. All available build specs are selected (and therefore enabled) by default. To restrict the list of enabled build specs to only those you need for your project, click **Disable All** and then select the checkbox next to the build spec(s) you want to enable for this project.



NOTE: Highlighting the name of the build spec is not sufficient to enable it; there must be a check in the checkbox to enable the build spec.

4. To reset build properties to their default settings or import build settings from another project, click **Import** and select the source of the build settings.
5. If you enabled one build spec for this project, it appears in the **Default build spec** and **Active build spec** fields.

   If you enabled more than one, the build spec in the **Default build spec** field is used for builds, though you can select a different **Active build spec** for a particular build. The build spec in the **Active build spec** field is also propagated to the appropriate fields on the Build Tools, Build Macros, and Build Paths tabs.

6. Select or clear the **Debug Mode** checkbox, depending on whether you want the build output to include debug information or not.



NOTE: You can also change the active build spec and the debug mode by right-clicking the project in the Project Navigator and selecting **Build Options > Set Active Build Spec**.

7. For more information about the build settings on this tab, press the help key for your host. You may continue configuring your project by selecting another build tab, or if you are finished, click **OK**.

8.3.2 Configuring Build Tools

Use this tab to configure build tools, build output generation, and build flags for your project.

1. The build tools you can select for RTP projects are **C-Compiler**, **C++-Compiler**, **Linker**, **Librarian**, or **Assembler**. In addition, you can define your own build tool.

   - **C-Compiler**, **C++-Compiler**, **Assembler**: These tools produce a *BuildTargetName***.obj** file on Windows or a *BuildTargetName***.o** file on UNIX.
   - **Linker**: The linker produces a *BuildTargetName***.vxe** file. This single, partially linked and *munched* (integrated with code to call C++ static constructors and destructors) object is intended for downloading.

     The **Linker** output product cannot be passed up to superprojects, although the current project's own, unlinked object files can, as can any output products received from projects further down in the hierarchy.
   - **Librarian**: The Librarian produces an archive *BuildTargetName***.a file.**

     The **Librarian** output product can be passed up to superprojects, as can the current project's own, unlinked object files, as well as any output products received from projects further down in the hierarchy.
   - To define your own build tool, click **New** and enter a build tool name, then click **OK**. Your build tools appears in the drop-down list, and you can configure all build tool settings to fit your needs.

2. For more information about the build settings on this tab, press the help key for your host. You may continue configuring your project by selecting another build tab, or if you are finished, click **OK**.



NOTE: If your project uses the standard managed build, and you select a different build tool on the Build Tools tab, this choice does not carry over to the Build Targets tab. You must manually assign the new build tool to your build target.

8.3.3 Configuring Build Macros

Use this tab to define global and build spec-specific macros that are added to the build command when executing builds.

1. To change the value of an existing global build macro, select the value in the **Build macro definitions** table and type in a new one, or click **Edit** and type in the new value, then click **OK**.
2. To define a new global build macro, click **New** next to the table, then enter a **Name** and **Value** for the macro. Click **OK**.



NOTE: You can define and use global build macros even if you do not select or define any build specs for your project.

3. To change the value of an existing build spec-specific macro, select the **Active build spec** for which the value should be applied, select the value in the **Build spec-specific settings** table, then type in a new one. Or click **Edit** and type in the new value, then click **OK**.
4. To define a new build spec-specific macro, click **New** next to the table, enter a **Name** for the macro, leave the **Value** blank, then click **OK**.

   To define the value, select the macro, select the **Active build spec** for which the value should be applied, click **Edit** and enter the **New value**, then click **OK**.

   The macro will always be appended to the build command when a build is launched, and the value will be set according to the active build spec, including empty values.

   For example, if the build command is **make --no-print-directory** and the macro is **TEST_SPEC**, you can define values to be used with different build specs:

   spec 1: Value = **spec1Val**
   spec 2: Value = **spec2Val**
   spec 3: Value =

   The resulting build commands are as follows:

build command for spec 1: **make --no-print-directory TEST_SPEC=spec1Val**
build command for spec 2: **make --no-print-directory TEST_SPEC=spec2Val**
build command for spec 3: **make --no-print-directory TEST_SPEC=**

5. For more information about the build settings on this tab, press the help key for your host. You may continue configuring your project by selecting another build tab, or if you are finished, click **OK**.

8.3.4 Configuring Build Paths

Use this tab to specify a redirection root directory for your build output, and add, delete, or reorder the directories searched by the build tools.

1. By default, build output is directed to a subdirectory in your workspace. However, if you want to redirect it somewhere else in your file system, specify a location in the **Redirection root directory** field.
2. The **Redirection directory** is a subdirectory of the **Redirection root directory**. By default this directory has the same name as the **Active build spec**, though you can change it by typing a new directory name in the field.
3. The **Include paths** table shows the paths used by the compiler to resolve include directives. To analyze your project and update the displayed include paths, click **Generate** to open the **Generate Include Search Paths** dialog.
   a. On the **Analyze include directives** page, specify which include directives you want Workbench to ignore when generating include paths.
      – Select the first box to ignore inactive include directives, or clear it to analyze them. Inactive include directives are those directives ignored by the build system because they are surrounded by compiler options and preprocessor directives such as **#ifndef**.
      – Select the second box to ignore system includes, or clear it to analyze them.

      When you are ready, click **Next** to analyze include directives. This may take some time for a large project.
   b. The **Resolve include directives** page displays results from the analysis. The upper field displays **unresolved include directives** in three groups: resolvable by one include search path, resolvable by multiple search paths, and not resolvable. The lower field displays **resolved directives**: predefined search paths, as well as the search paths Workbench was able to resolve.

- To automatically resolve all include directives that can be resolved, click **Resolve All**.
- To automatically resolve an individual include directive or one of the groups of unresolved directives, click **Resolve**. If Workbench found one matching header, it will resolve the include directive. If it found multiple matching headers, a dialog displays the headers and asks you to select one.
- To open the file so you can see the context of an unresolved include directive, click **Show in Editor**.



NOTE: When automatically resolving include directives, Workbench uses heuristics to determine the best matches, but the results may be incorrect. So you should examine and if necessary **Remove** undesired search paths in the lower field. The newly-generated search paths are marked with a yellow plus on the folder icon ().

- To manually resolve include directives for which no matching headers were found, click **Add** and navigate to the location of the appropriate headers. Click **OK** to add the path to the list of resolved directives; any directives resolved by that path are removed from the unresolved list.
- To view, enable, or disable variables for paths and path segments, click **Substitution**. Click a variable or group of variables to enable or disable them.

  Variables are grouped into four groups, and are applied to all generated paths in the resolved directives field:

  Wind River environment: includes Wind River platform and Workbench variables.

  Build macros: includes global and local build macros, as defined in project properties.

  Project locations: includes variables referring to projects in the workspace.

  System environment: includes all environment variables that do not appear in the build macro or Wind River environment groups.

  Click **Apply** to see your changes but leave the dialog open for further editing, or click **OK** if you are finished.
- To copy search paths to the clipboard so you can paste them into a make file, select the search path then click **Copy**.

c. When you are ready, click **Finish** to return to the Build Paths tab.

4. To manually add an include directory for an **Active build spec**, select the build spec from the drop-down list, click **Add**, and browse to or type in the path (be sure not to erase the **-I** at the beginning of the path). Click **OK**.

   To add an include path that applies to all your build specs, click **Add to all** and then browse to or type in the path, then click **OK**.

5. When you are finished configuring your project, click **OK**.

8.4 VxWorks Real-time Process Projects in the Project Navigator

After you have created a VxWorks Real-time Process project, a number of nodes appear in the Project Navigator. This section describes these nodes as they appear immediately after project creation. For general notes about manipulating nodes, for example, moving, copying, filtering, etc., please see *13. Working in the Project Navigator*.

8.4.1 Global Project Nodes



ProjectName

The icon at the root of the project tree identifies the type of project; the icon's label is the name you gave the project when you created it.

8.4.2 Project Build Specs and Target Nodes

Each target node is associated with a predefined build specification.

The RTP build targets depend on the options you selected during project creation. Specifically, you will not have both an archive (*TargetName***.a**) target and a *TargetName***.out** target immediately after project creation. Which of these will be visible depends on the build tool you selected. Also, the presence or absence of the green upward arrow on the target icon (to indicate whether the target is passed up the hierarchy) will be determined by your project settings.



TargetName.**vxe (***BuildSpecName***[_DEBUG])**
This single, partially linked and munched (integrated with code to call C++ static constructors and destructors) object, produced by the **Linker** build tool is intended for downloading.



TargetName.**a (***BuildSpecName***[_DEBUG])**
An archive produced by the **Librarian** build tool that has to be statically linked into an executable.

8.4.3 Makefile Nodes

At project generation time two Makefiles are copied to the project. One is a template that can also be used for entering custom make rules. The other is dynamically regenerated from build spec data at each build.



.wrmakefile
A template used by Workbench to generate the project's **Makefile**. Add user-specific build-targets and make-rules in this file. These will then be automatically dumped into the Makefile.



Makefile
Do *not* add custom code to this file. This **Makefile** is regenerated every time the project is built. The information used to generate the file is taken from the build specification that on which the target node is based.

8.4.4 Project File Nodes

The project creation facility generates, or includes copies of, a variety of files when a project is created. Normally, you need not be concerned with these files. However, here a brief summary of the DKM project files displayed in the Project Navigator:



.project
Eclipse platform project file containing builder information and project nature.



.wrproject
Workbench project file containing common project properties such as project type, etc.

8.5 Application Code for a VxWorks Real-time Process Project

After project creation you have the infrastructure for a VxWorks Real-time Process project, but often no actual application code. If you are writing code from the beginning, you can add new files to a project. If you already have source code files, you will want to import these to the project. For more information please refer to *13.3.1 Importing Resources*, p.162, and *13.3.2 Adding New Files to Projects*, p.163.

8.6 Linking to VxWorks and Using Shared Libraries

In order to have your VxWorks Real-time Process project binary initialized once the kernel has booted, you will need to:

- Create a VxWorks Image project, see *5.2 Creating a VxWorks Image Project*, p.88.
- Configure the VxWorks Image project as described under *5.6 Adding Application Projects to the VxWorks Image Project*, p.99 and *5.4 Configuring Kernel Components*, p.93.
- Create a ROMFS target file system before the target is disconnected from the host system, see *7.2 Creating a VxWorks ROMFS File System Project*, p.110.
- If you want to dynamically link to shared libraries, the VxWorks Real-time Process project needs to be appropriately configured, see *17.6 Executables that Dynamically Link to Shared Libraries*, p.229, and *18. RTPs and Shared Libraries from Host to Target*.

9 Creating VxWorks Shared Library Projects

9.1 Introduction 123

9.2 Creating a VxWorks Shared Library Project 124

9.3 Configuring VxWorks Shared Library Projects 124

9.4 Shared Libraries in the Project Navigator 130

9.5 Source Code for the Shared Library 131

9.6 Making Shared Libraries Available to Applications 131

9.1 Introduction

Use *VxWorks Shared Library* projects for libraries that are dynamically linked to Real-time Process applications at run-time. Such a shared library can be stored on a host file system, a network file system, or a local file system on the target (including ROMFS). You can also use VxWorks Shared Library projects to create subprojects that are statically linked into other project types at build time.

Please refer to *17.6 Executables that Dynamically Link to Shared Libraries*, p.229, and *18. RTPs and Shared Libraries from Host to Target* for more information on working with this type of project. Also refer to the *VxWorks Application Programmer's Guide: Applications and Processes* for more information about shared libraries.

9.2 Creating a VxWorks Shared Library Project

Before creating the project, see the general comments on projects and project creation in *4. Projects Overview*.

1. Create a VxWorks Shared Library project by selecting **File > New > Wind River Workbench Project**. The New Wind River Workbench Project wizard appears.
2. Select a target operating system, then click **Next**.
3. From the **Build type** drop-down list, select **Shared User Library**. Click **Next**.
4. Type a name for your project.
5. Decide where to create your project:

   Create project in workspace
   Leave this selected if you want to create the project under the current workspace directory.

   Create project at external location
   Select this option, click **Browse**, then navigate to a different location if you want to create the project outside the workspace.

   Create project in workspace with content at external location
   Select this option, click **Browse**, then navigate to your source location if you do not want to mix project files with your sources, or copy sources into your workspace.
6. When you are ready, click **Finish**. The VxWorks Shared Library project is created and appears at the root level in the Project Navigator.

9.3 Configuring VxWorks Shared Library Projects

Once you have created your project, you can configure its build support and specs, build tools, build macros, and build paths.

For general details about build properties, see *16.4 Accessing Build Properties*, p.214 or press the help key for your host.

1. To access build properties for your project, right-click it in the Project Navigator and select **Properties**.

2. From the **Properties** dialog, click **Build Properties**.

9.3.1 Configuring Build Support and Specs

Use this tab to configure build support and build specs for your project.

1. A VxWorks Shared Library project is a predefined project type that uses Workbench build support, so build support is enabled by default. If you are creating a project because you want to browse symbol information and you are not interested in building it, click **Disabled** to disable build support (you can click **Managed build** to re-enable it later, if you want).
2. If necessary, edit the default **Build command**.
3. All available build specs are selected (and therefore enabled) by default. To restrict the list of enabled build specs to only those you need for your project, click **Disable All** and then select the checkbox next to the build spec(s) you want to enable for this project.



NOTE: Highlighting the name of the build spec is not sufficient to enable it; there must be a check in the checkbox to enable the build spec.

4. To reset build properties to their default settings or import build settings from another project, click **Import** and select the source of the build settings.
5. If you enabled one build spec for this project, it appears in the **Default build spec** and **Active build spec** fields.

   If you enabled more than one, the build spec in the **Default build spec** field is used for builds, though you can select a different **Active build spec** for a particular build. The build spec in the **Active build spec** field is also propagated to the appropriate fields on the Build Tools, Build Macros, and Build Paths tabs.
6. Select or clear the **Debug Mode** checkbox, depending on whether you want the build output to include debug information or not.



NOTE: You can also change the active build spec and the debug mode by right-clicking the project in the Project Navigator and selecting **Build Options > Set Active Build Spec**.

7. For more information about the build settings on this tab, press the help key for your host. You may continue configuring your project by selecting another build tab, or if you are finished, click **OK**.

9.3.2 Configuring Build Tools

Use this tab to configure build tools, build output generation, and build flags for your project.

1. The build tools you can select for Shared Library projects are **C-Compiler**, **C++-Compiler**, **Shared Library Linker**, **Static Librarian**, or **Assembler**. In addition, you can define your own build tool.
   - **C-Compiler**, **C++-Compiler**, **Assembler**: These tools produce a *BuildTargetName***.obj** file on Windows or a *BuildTargetName***.o** file on UNIX.
   - **Shared Library Linker**: The shared library linker produces a *BuildTargetName***.so** target that is dynamically linked to at run-time.

     The output product of the shared library linker will normally be passed up to superprojects. If you do not pass the library target up to its superprojects, references in the superprojects' application code cannot be resolved at compile time.
   - **Static Librarian**: The static librarian produces an archive *BuildTargetName***.a file.**

     The static librarian output product can be passed up to superprojects, as can the current project's own, unlinked object files, as well as any output products received from projects further down in the hierarchy.
   - To define your own build tool, click **New** and enter a build tool name, then click **OK**. Your build tools appears in the drop-down list, and you can configure all build tool settings to fit your needs.
2. For more information about the build settings on this tab, press the help key for your host. You may continue configuring your project by selecting another build tab, or if you are finished, click **OK**.



NOTE: If your project uses the standard managed build, and you select a different build tool on the Build Tools tab, this choice does not carry over to the Build Targets tab. You must manually assign the new build tool to your build target.

9.3.3 Configuring Build Macros

Use this tab to define global and build spec-specific macros that are added to the build command when executing builds.

1. To change the value of an existing global build macro, select the value in the **Build macro definitions** table and type in a new one, or click **Edit** and type in the new value, then click **OK**.
2. To define a new global build macro, click **New** next to the table, then enter a **Name** and **Value** for the macro. Click **OK**.



NOTE: You can define and use global build macros even if you do not select or define any build specs for your project.

3. To change the value of an existing build spec-specific macro, select the **Active build spec** for which the value should be applied, select the value in the **Build spec-specific settings** table, then type in a new one. Or click **Edit** and type in the new value, then click **OK**.
4. To define a new build spec-specific macro, click **New** next to the table, enter a **Name** for the macro, leave the **Value** blank, then click **OK**.

   To define the value, select the macro, select the **Active build spec** for which the value should be applied, click **Edit** and enter the **New value**, then click **OK**.

   The macro will always be appended to the build command when a build is launched, and the value will be set according to the active build spec, including empty values.

   For example, if the build command is **make --no-print-directory** and the macro is **TEST_SPEC**, you can define values to be used with different build specs:

   spec 1: Value = **spec1Val**
   spec 2: Value = **spec2Val**
   spec 3: Value =

   The resulting build commands are as follows:

   build command for spec 1: **make --no-print-directory TEST_SPEC=spec1Val**
   build command for spec 2: **make --no-print-directory TEST_SPEC=spec2Val**
   build command for spec 3: **make --no-print-directory TEST_SPEC=**
5. For more information about the build settings on this tab, press the help key for your host. You may continue configuring your project by selecting another build tab, or if you are finished, click **OK**.

9.3.4 Configuring Build Paths

Use this tab to specify a redirection root directory for your build output, and add, delete, or reorder the directories searched by the build tools.

1. By default, build output is directed to a subdirectory in your workspace. However, if you want to redirect it somewhere else in your file system, specify a location in the **Redirection root directory** field.
2. The **Redirection directory** is a subdirectory of the **Redirection root directory**. By default this directory has the same name as the **Active build spec**, though you can change it by typing a new directory name in the field.
3. The **Include paths** table shows the paths used by the compiler to resolve include directives. To analyze your project and update the displayed include paths, click **Generate** to open the **Generate Include Search Paths** dialog.
    a. On the **Analyze include directives** page, specify which include directives you want Workbench to ignore when generating include paths.
        – Select the first box to ignore inactive include directives, or clear it to analyze them. Inactive include directives are those directives ignored by the build system because they are surrounded by compiler options and preprocessor directives such as **#ifndef**.
        – Select the second box to ignore system includes, or clear it to analyze them.

        When you are ready, click **Next** to analyze include directives. This may take some time for a large project.
    b. The **Resolve include directives** page displays results from the analysis. The upper field displays **unresolved include directives** in three groups: resolvable by one include search path, resolvable by multiple search paths, and not resolvable. The lower field displays **resolved directives**: predefined search paths, as well as the search paths Workbench was able to resolve.
        – To automatically resolve all include directives that can be resolved, click **Resolve All**.
        – To automatically resolve an individual include directive or one of the groups of unresolved directives, click **Resolve**. If Workbench found one matching header, it will resolve the include directive. If it found multiple matching headers, a dialog displays the headers and asks you to select one.

– To open the file so you can see the context of an unresolved include directive, click **Show in Editor**.



NOTE: When automatically resolving include directives, Workbench uses heuristics to determine the best matches, but the results may be incorrect. So you should examine and if necessary **Remove** undesired search paths in the lower field. The newly-generated search paths are marked with a yellow plus on the folder icon ().

– To manually resolve include directives for which no matching headers were found, click **Add** and navigate to the location of the appropriate headers. Click **OK** to add the path to the list of resolved directives; any directives resolved by that path are removed from the unresolved list.

– To view, enable, or disable variables for paths and path segments, click **Substitution**. Click a variable or group of variables to enable or disable them.

  Variables are grouped into four groups, and are applied to all generated paths in the resolved directives field:

  Wind River environment: includes Wind River platform and Workbench variables.

  Build macros: includes global and local build macros, as defined in project properties.

  Project locations: includes variables referring to projects in the workspace.

  System environment: includes all environment variables that do not appear in the build macro or Wind River environment groups.

  Click **Apply** to see your changes but leave the dialog open for further editing, or click **OK** if you are finished.

– To copy search paths to the clipboard so you can paste them into a make file, select the search path then click **Copy**.

c. When you are ready, click **Finish** to return to the Build Paths tab.

4. To manually add an include directory for an **Active build spec**, select the build spec from the drop-down list, click **Add**, and browse to or type in the path (be sure not to erase the **-I** at the beginning of the path). Click **OK**.

   To add an include path that applies to all your build specs, click **Add to all** and then browse to or type in the path, then click **OK**.

5. When you are finished configuring your project, click **OK**.

9.4 Shared Libraries in the Project Navigator

After a VxWorks Shared Library project has been created, a number of nodes appear in the Project Navigator. This section describes these nodes as they appear immediately after project creation. For general notes about manipulating nodes, for example, moving, copying, filtering, and so on, please see *13. Working in the Project Navigator*.

9.4.1 Global Project Nodes



ProjectName
The icon at the root of the project tree identifies the type of project; the icon's label is the name you gave the project when you created it.

9.4.2 Target Node



TargetName.so (*BuildSpecName*[_DEBUG])
A VxWorks Shared Library produced by the **Shared Library Linker** that is dynamically linked at run-time.

9.4.3 Makefile Nodes

At project generation time a template that can also be used for entering custom make rules is copied to the project.



.wrmakefile
A template used by Workbench to generate the project's **Makefile**. Add user-specific build-targets and make-rules in this file.

9.4.4 Project File Nodes

The project creation facility generates, or includes copies of, a variety of files when a project is created. Normally, you need not be concerned with these files. However, here a brief summary of the Shared Library project files displayed in the Project Navigator:



.project

Eclipse platform project file containing builder information and project nature.



.wrproject

Workbench project file containing common project properties such as project type, etc.

9.5 Source Code for the Shared Library

After project creation you have the infrastructure for a Shared Library project, but often no actual library source code. If you are writing code from the beginning, you can add new files to a project. If you already have source code files, you will want to import these to the project. For more information refer to *13.3.1 Importing Resources*, p.162, and *13.3.2 Adding New Files to Projects*, p.163.

9.6 Making Shared Libraries Available to Applications

To make shared libraries accessible to your applications at run-time, you have to make sure of a few configuration details, both on the library side and on the application side. You also need a file system project to store the library on the target (see *7. Creating VxWorks ROMFS File System Projects*).

1. Make sure the shared library is a subproject of all applications that need to access it. If the library is used by many applications, create projects for each application and make the library a subproject of each (see *13. Working in the Project Navigator* for information on how to do this).

2. Make sure the library target is passed to superprojects. You can do this in the Project Properties as follows:
   - In the Project Navigator, right-click the shared library project folder you are interested in and select **Properties**. (If the project folder is a subnode under several different superprojects, it does not matter which you choose because these nodes are only logical representations of the same project.)
   - In **Project Properties**, select **Build Properties** node, then the **Build Tools** tab. On the **Build Tools** tab, be sure the **Generated build target can be passed** check box is selected. If the output of the library build is not passed up to superprojects, references from the superproject to the library subproject cannot be resolved at build-time.
3. Click **OK** to close the Project Properties.

9.6.1 Configuring the Application Projects

Most shared library projects are created as subprojects of one or more application projects. Although a superproject knows the location of its subprojects, it does not know that a particular subproject is a shared library, so the application project's linker has to be configured to accommodate dynamic access to shared libraries. For more information, please see *17.6 Executables that Dynamically Link to Shared Libraries*, p.229, and *18. RTPs and Shared Libraries from Host to Target*.

10

Creating VxWorks Downloadable Kernel Module Projects

10.1 Introduction 133

10.2 Creating a VxWorks Downloadable Kernel Module Project 134

10.3 Configuring VxWorks Downloadable Kernel Module Projects 134

10.4 Downloadable Kernel Modules in the Project Navigator 140

10.5 Application Code for a VxWorks DKM Project 142

10.1 Introduction

Use *VxWorks Downloadable Kernel Module* (DKM) projects to manage and build modules that will exist in the kernel space. You can separately build the modules, then run and debug them on a target running VxWorks, loading, unloading, and reloading on the fly.

Once your development work is complete, the modules can be statically linked into the kernel or added to a file system if one is present.

Kernel-mode development is the traditional VxWorks method of development. All the tasks you spawn run in an unprotected environment and all have full access to the hardware in the system.

10.2 Creating a VxWorks Downloadable Kernel Module Project

Before creating the project, see the general comments on projects and project creation in *4. Projects Overview*.

1. Create a VxWorks Downloadable Kernel Module Project by selecting **File > New > Wind River Workbench Project**. The New Wind River Workbench Project wizard appears.
2. Select a target operating system, then click **Next**.
3. From the **Build type** drop-down list, select **Downloadable Kernel Module**. Click **Next**.
4. Type a name for your project.
5. Decide where to create your project:

   Create project in workspace
   Leave this selected if you want to create the project under the current workspace directory.

   Create project at external location
   Select this option, click **Browse**, then navigate to a different location if you want to create the project outside the workspace.

   Create project in workspace with content at external location
   Select this option, click **Browse**, then navigate to your source location if you do not want to mix project files with your sources, or copy sources into your workspace.
6. When you are ready, click **Finish**. The Downloadable Kernel Module project is created and appears at the root level in the Project Navigator.

10.3 Configuring VxWorks Downloadable Kernel Module Projects

Once you have created your project, you can configure its build support and specs, build tools, build macros, and build paths.

For general details about build properties, see *16.4 Accessing Build Properties*, p.214 or press the help key for your host.

1. To access build properties for your project, right-click it in the Project Navigator and select **Properties**.
2. From the **Properties** dialog, click **Build Properties**.

10.3.1 Configuring Build Support and Specs

Use this tab to configure build support and build specs for your project.

1. A VxWorks Downloadable Kernel Module project is a predefined project type that uses Workbench build support, so build support is enabled by default. If you are creating a project because you want to browse symbol information and you are not interested in building it, click **Disabled** to disable build support (you can click **Managed build** to re-enable it later, if you want).
2. If necessary, edit the default **Build command**.
3. All available build specs are selected (and therefore enabled) by default. To restrict the list of enabled build specs to only those you need for your project, click **Disable All** and then select the checkbox next to the build spec(s) you want to enable for this project.



NOTE: Highlighting the name of the build spec is not sufficient to enable it; there must be a check in the checkbox to enable the build spec.

4. To reset build properties to their default settings or import build settings from another project, click **Import** and select the source of the build settings.
5. If you enabled one build spec for this project, it appears in the **Default build spec** and **Active build spec** fields.

   If you enabled more than one, the build spec in the **Default build spec** field is used for builds, though you can select a different **Active build spec** for a particular build. The build spec in the **Active build spec** field is also propagated to the appropriate fields on the Build Tools, Build Macros, and Build Paths tabs.
6. Select or clear the **Debug Mode** checkbox, depending on whether you want the build output to include debug information or not.



NOTE: You can also change the active build spec and the debug mode by right-clicking the project in the Project Navigator and selecting **Build Options > Set Active Build Spec**.

7. For more information about the build settings on this tab, press the help key for your host. You may continue configuring your project by selecting another build tab, or if you are finished, click **OK**.

10.3.2 Configuring Build Tools

Use this tab to configure build tools, build output generation, and build flags for your project.

1. The build tools you can select for DKM projects are **C-Compiler**, **C++-Compiler**, **Linker**, **Partial Image Linker**, **Librarian**, or **Assembler**. In addition, you can define your own build tool.
   - **C-Compiler**, **C++-Compiler**, **Assembler**: These tools produce a *BuildTargetName***.obj** file on Windows or a *BuildTargetName***.o** file on UNIX.
   - **Linker**: The linker produces a *BuildTargetName***.out** file. This single, partially linked and *munched* (integrated with code to call C++ static constructors and destructors) object is intended for downloading. The **Linker** output product cannot be passed up to superprojects, although the current project's own, unlinked object files can, as can any output products received from projects further down in the hierarchy.
   - **Partial Image Linker**: The Partial Image Linker produces a *BuildTargetName***.o** file. This single, partially linked, but not munched (not integrated with code to call C++ static constructors and destructors) object is for subproject support only; it is *not* intended for download. The **Partial Image Linker** output product can be passed up to superprojects, as can the current project's own, unlinked object files, as well as any output products received from projects further down in the hierarchy
   - **Librarian**: The Librarian produces an archive *BuildTargetName***.a file.** The **Librarian** output product can be passed up to superprojects, as can the current project's own, unlinked object files, as well as any output products received from projects further down in the hierarchy.
   - To define your own build tool, click **New** and enter a build tool name, then click **OK**. Your build tools appears in the drop-down list, and you can configure all build tool settings to fit your needs.
2. For more information about the build settings on this tab, press the help key for your host. You may continue configuring your project by selecting another build tab, or if you are finished, click **OK**.



NOTE: If your project uses the standard managed build, and you select a different build tool on the Build Tools tab, this choice does not carry over to the Build Targets tab. You must manually assign the new build tool to your build target.

10.3.3 Configuring Build Macros

Use this tab to define global and build spec-specific macros that are added to the build command when executing builds.

1. To change the value of an existing global build macro, select the value in the **Build macro definitions** table and type in a new one, or click **Edit** and type in the new value, then click **OK**.
2. To define a new global build macro, click **New** next to the table, then enter a **Name** and **Value** for the macro. Click **OK**.



NOTE: You can define and use global build macros even if you do not select or define any build specs for your project.

3. To change the value of an existing build spec-specific macro, select the **Active build spec** for which the value should be applied, select the value in the **Build spec-specific settings** table, then type in a new one. Or click **Edit** and type in the new value, then click **OK**.
4. To define a new build spec-specific macro, click **New** next to the table, enter a **Name** for the macro, leave the **Value** blank, then click **OK**.

   To define the value, select the macro, select the **Active build spec** for which the value should be applied, click **Edit** and enter the **New value**, then click **OK**.

   The macro will always be appended to the build command when a build is launched, and the value will be set according to the active build spec, including empty values.

   For example, if the build command is **make --no-print-directory** and the macro is **TEST_SPEC**, you can define values to be used with different build specs:

   spec 1: Value = **spec1Val**
   spec 2: Value = **spec2Val**
   spec 3: Value =

   The resulting build commands are as follows:

build command for spec 1: **make --no-print-directory TEST_SPEC=spec1Val**
build command for spec 2: **make --no-print-directory TEST_SPEC=spec2Val**
build command for spec 3: **make --no-print-directory TEST_SPEC=**

5. For more information about the build settings on this tab, press the help key for your host. You may continue configuring your project by selecting another build tab, or if you are finished, click **OK**.

10.3.4 Configuring Build Paths

Use this tab to specify a redirection root directory for your build output, and add, delete, or reorder the directories searched by the build tools.

1. By default, build output is directed to a subdirectory in your workspace. However, if you want to redirect it somewhere else in your file system, specify a location in the **Redirection root directory** field.
2. The **Redirection directory** is a subdirectory of the **Redirection root directory**. By default this directory has the same name as the **Active build spec**, though you can change it by typing a new directory name in the field.
3. The **Include paths** table shows the paths used by the compiler to resolve include directives. To analyze your project and update the displayed include paths, click **Generate** to open the **Generate Include Search Paths** dialog.
   a. On the **Analyze include directives** page, specify which include directives you want Workbench to ignore when generating include paths.
      – Select the first box to ignore inactive include directives, or clear it to analyze them. Inactive include directives are those directives ignored by the build system because they are surrounded by compiler options and preprocessor directives such as **#ifndef**.
      – Select the second box to ignore system includes, or clear it to analyze them.

      When you are ready, click **Next** to analyze include directives. This may take some time for a large project.
   b. The **Resolve include directives** page displays results from the analysis. The upper field displays **unresolved include directives** in three groups: resolvable by one include search path, resolvable by multiple search paths, and not resolvable. The lower field displays **resolved directives**: predefined search paths, as well as the search paths Workbench was able to resolve.

- To automatically resolve all include directives that can be resolved, click **Resolve All**.
- To automatically resolve an individual include directive or one of the groups of unresolved directives, click **Resolve**. If Workbench found one matching header, it will resolve the include directive. If it found multiple matching headers, a dialog displays the headers and asks you to select one.
- To open the file so you can see the context of an unresolved include directive, click **Show in Editor**.



NOTE: When automatically resolving include directives, Workbench uses heuristics to determine the best matches, but the results may be incorrect. So you should examine and if necessary **Remove** undesired search paths in the lower field. The newly-generated search paths are marked with a yellow plus on the folder icon ().

- To manually resolve include directives for which no matching headers were found, click **Add** and navigate to the location of the appropriate headers. Click **OK** to add the path to the list of resolved directives; any directives resolved by that path are removed from the unresolved list.
- To view, enable, or disable variables for paths and path segments, click **Substitution**. Click a variable or group of variables to enable or disable them.

  Variables are grouped into four groups, and are applied to all generated paths in the resolved directives field:

  Wind River environment: includes Wind River platform and Workbench variables.

  Build macros: includes global and local build macros, as defined in project properties.

  Project locations: includes variables referring to projects in the workspace.

  System environment: includes all environment variables that do not appear in the build macro or Wind River environment groups.

  Click **Apply** to see your changes but leave the dialog open for further editing, or click **OK** if you are finished.
- To copy search paths to the clipboard so you can paste them into a make file, select the search path then click **Copy**.

c. When you are ready, click **Finish** to return to the Build Paths tab.

4. To manually add an include directory for an **Active build spec**, select the build spec from the drop-down list, click **Add**, and browse to or type in the path (be sure not to erase the **-I** at the beginning of the path). Click **OK**.

   To add an include path that applies to all your build specs, click **Add to all** and then browse to or type in the path, then click **OK**.

5. When you are finished configuring your project, click **OK**.

10.4 Downloadable Kernel Modules in the Project Navigator

After a VxWorks Downloadable Kernel Module has been created, a number of nodes appear in the Project Navigator. This section describes these nodes as they appear immediately after project creation. For general notes about manipulating nodes, for example, moving, copying, filtering, and so forth. Please see *13. Working in the Project Navigator*.

10.4.1 Global Project Nodes



ProjectName

The icon at the root of the project tree identifies the type of project; the icon's label is the name you gave the project when you created it.

10.4.2 Project Build Specs and Target Nodes

Each target node is associated with a predefined build specification.

The VxWorks Downloadable Kernel Module project software targets depend on the options you selected during project creation. Specifically, you will not have both an archive (*TargetName***.a**) target and a *TargetName***.out** target immediately after project creating. Which, if any, of these will be visible depends on the build tool you selected. Also, the presence or absence of the green upward arrow on the target icon (to indicate whether the target is passed up the hierarchy) will be determined by your creation settings.



PartialImage.pl
This default target is always built for VxWorks Downloadable Kernel Module project. This single, partially linked, but not munched object is for subproject support only; it is *not* intended for download. By default, the build target is passed to the next level (hence the green upward arrow on the icon).



TargetName.**out (***BuildSpecName***[_DEBUG])**
This single, partially linked and munched object, produced by the **Linker** build tool is intended for downloading.



TargetName.**a (***BuildSpecName***[_DEBUG])**
An archive produced by the **Librarian** build tool that has to be statically linked into an executable.

10.4.3 Makefile Nodes

At project generation time two Makefiles are copied to the project. One is a template that can also be used for entering custom make rules. The other is dynamically regenerated from build spec data at each build.



.wrmakefile
A template used by Workbench to generate the project's **Makefile**. Add user-specific build-targets and make-rules in this file. These will then be automatically dumped into the Makefile.



Makefile
Do *not* add custom code to this file. This **Makefile** is regenerated every time the project is built. The information used to generate the file is taken from the build specification on which the target node is based.

10.4.4 Project File Nodes

The project creation facility generates, or includes copies of, a variety of files when a project is created. Normally, you need not be concerned with these files. However, here a brief summary of the VxWorks Downloadable Kernel Module project files displayed in the Project Navigator:



.project

Eclipse platform project file containing builder information and project nature.



.wrproject

Workbench project file containing common project properties such as project type, and so on.

10.5 Application Code for a VxWorks DKM Project

After project creation, you have the infrastructure for a VxWorks Downloadable Kernel Module project, but often no actual application code. If you are writing code from the beginning, you can add new files to a project. If you already have source code files, you will want to import these to the project. For more information please refer to *13.3.1 Importing Resources*, p.162, and *13.3.2 Adding New Files to Projects*, p.163.

You can link your VxWorks Downloadable Kernel Module with the operating system and have it start automatically at boot time. To do this:

1. Create a VxWorks Image project. See *5.2 Creating a VxWorks Image Project*, p.88.
2. Configure the VxWorks Image project as described under *5.6 Adding Application Projects to the VxWorks Image Project*, p.99 and *5.4 Configuring Kernel Components*, p.93.

11 Creating User-Defined Projects

11.1 Introduction 143

11.2 Creating and Maintaining Makefiles 144

11.3 Creating a User-Defined Project 144

11.4 Configuring a User-Defined Project 145

11.5 Creating an Application for VxWorks 148

11.1 Introduction

User-Defined Projects assume that you are responsible for setting up and maintaining your own build system, file system population, and so on. The user interface provides support for the following:

- You can configure the build command used to launch your build utility; this allows you to start builds from the Workbench GUI. You can also configure different rules for building, rebuilding and cleaning the project.
- You can create build targets in the Project Navigator that reflect rules in your makefiles; this allows you to select and build any of your make rules directly from the Project Navigator.
- Build output is captured to the Build Console.

11.2 Creating and Maintaining Makefiles

When you create a User-Defined project, Workbench checks the root location of the project's resources for the existence of a file named **Makefile**[1]. If it does not exist, Workbench creates a skeleton makefile with a default **all** rule and a **clean**. This allows you to use the **Build Project**, **Rebuild Project**, and **Clean Project** menu commands, as well as preventing the generation of build errors. You are responsible for maintaining this Makefile, and you can write any other rules into this file at any time.

If you base your User-Defined project on an existing project, the makefile of that project will be copied to the new project and will overwrite a makefile in the new project's location. If necessary, you can change the name of the new project's makefile using the **-f** make option to avoid overwriting an existing makefile.

11.3 Creating a User-Defined Project

Before creating the project, see the general comments on projects and project creation in *4. Projects Overview*.

1. Create a User-Defined project by selecting **File > New > Wind River Workbench Project**. The New Wind River Workbench Project wizard appears.
2. Select a target operating system, then click **Next**.
3. From the **Build type** drop-down list, select **User-Defined**. Click **Next**.
4. Type a name for your project.
5. Decide where to create your project:

   Create project in workspace
   Leave this selected if you want to create the project under the current workspace directory.

1. If you specified a different filename in the New Project wizard's **Build Command** field using the **-f** make option, which can include a relative or absolute path to a subdirectory, Workbench checks for the file you specified.

Create project at external location
Select this option, click **Browse**, then navigate to a different location if you want to create the project outside the workspace.

Create project in workspace with content at external location
Select this option, click **Browse**, then navigate to your source location if you do not want to mix project files with your sources, or copy sources into your workspace.

6. When you are ready, click **Finish**. Your project appears in the Project Navigator.

11.4 Configuring a User-Defined Project

Once you have created your project, you can configure its build targets, build specs, and build macros.

For general details about build properties, see *16.4 Accessing Build Properties*, p.214 or press the help key for your host.

1. To access build properties for your project, right-click it in the Project Navigator and select **Properties**.
2. From the **Properties** dialog, click **Build Properties**.



NOTE: Build tools and build paths cannot be configured for User-defined projects.

11.4.1 Configuring Build Support

Use this tab to configure build support for your project.

1. Build support is enabled by default. Click **Disabled** to disable it, and click **User-defined build** to re-enable it.
2. If necessary, edit the default build command.
3. Specify whether received build targets should be passed to the next level.
4. Specify when Workbench should refresh the project after a build.

Because a refresh of the entire project can take some time (depending on its size) Workbench does not do it by default. You may choose to refresh the current project, the current folder, the current folder and its subfolders, or nothing at all. This option applies to all build runs of the project.

5. You may continue configuring your project by selecting another build tab, or if you are finished, click **OK** to close the Build Properties.

11.4.2 Configuring Build Targets

Use this tab to configure make rules and define custom build targets for your project.

1. Type the desired make rules into the fields in the **Make rules** section. These rules are run when you select the corresponding options from the **Project** menu or when you right-click your project in the Project Navigator and select them from the context menu.

   The **Build Folder** and **Clean Folder** options are available when you select a folder in the Project Navigator.

2. To define a custom build target, click **New**. The **New Custom Build Target** dialog opens.

3. Type in a name for your build target, then type in the make rule or external command that Workbench should execute. You can also click **Variables** and add a context-sensitive variable to the make rule or command.

   The variables represented in the **Select Variable** dialog are context-sensitive, and depend on the current selection in the Project Navigator. For variables that contain a file-specific component, the corresponding target is only enabled when a file is selected and the variable can be evaluated. Build targets without file-specific components are always enabled.

4. Choose the type, whether it is a **Rule** or a **Command**.

5. Choose a refresh option for the build target, specifying whether Workbench should use the project setting, refresh the current folder or project, or do nothing. Click **OK** to close the dialog.

6. Edit a build target's options by clicking **Edit** or **Rename**. You can also edit the options (except name) by clicking in the column itself.

7. You may continue configuring your project by selecting another build tab, or if you are finished, click **OK** to close the Build Properties.

Once you have defined a build target, it is available when you right-click a project and select **Build Options**. The build targets are inherited by each folder within the project, eliminating the need to define the same build targets in each individual folder.

11.4.3 Configuring Build Specs

Use this tab to define and import build specs.

1. To define a new build spec for your project, click **New** and enter a build spec name. Click **OK**. If this is the first build spec for this project, it automatically appears in the **Default build spec** and **Active build spec** fields. Once you have defined more than one, you can choose a different default and active spec from the drop-down list.
2. To reset build properties to their default settings or import build settings from another project, click **Import** and select the source of the build settings.
3. Decide whether to clear build setting overrides, then click **Finish** to return to the Build Specs tab.



NOTE: The Debug mode option is not available for User-defined builds, as this has an effect only on build tool-specific fields, which are not available for User-defined projects.

4. You may continue configuring your project by selecting another build tab, or if you are finished, click **OK** to close the Build Properties.

11.4.4 Configuring Build Macros

Use this tab to define global and build spec-specific macros that are added to the build command when executing builds.

1. To change the value of an existing global build macro, select the value in the **Build macro definitions** table and type in a new one, or click **Edit** and type in the new value, then click **OK**.
2. To define a new global build macro, click **New** next to the table, then enter a **Name** and **Value** for the macro. Click **OK**.



NOTE: You can define and use global build macros even if you do not select or define any build specs for your project.

3. To change the value of an existing build spec-specific macro, select the **Active build spec** for which the value should be applied, select the value in the **Build spec-specific settings** table, then type in a new one. Or click **Edit** and type in the new value, then click **OK**.

4. To define a new build spec-specific macro, click **New** next to the table, enter a **Name** for the macro, leave the **Value** blank, then click **OK**.

   To define the value, select the macro, select the **Active build spec** for which the value should be applied, click **Edit** and enter the **New value**, then click **OK**.

   The macro will always be appended to the build command when a build is launched, and the value will be set according to the active build spec, including empty values.

   For example, if the build command is **make --no-print-directory** and the macro is **TEST_SPEC**, you can define values to be used with different build specs:

   spec 1: Value = **spec1Val**
   spec 2: Value = **spec2Val**
   spec 3: Value =

   The resulting build commands are as follows:

   build command for spec 1: **make --no-print-directory TEST_SPEC=spec1Val**
   build command for spec 2: **make --no-print-directory TEST_SPEC=spec2Val**
   build command for spec 3: **make --no-print-directory TEST_SPEC=**

5. For more information about the build settings on this tab, press the help key for your host. You may continue configuring your project by selecting another build tab, or if you are finished, click **OK**.

11.5 Creating an Application for VxWorks

In order to have your application initialized once the kernel has booted, you will need to:

- Create a VxWorks Image project, see *5.2 Creating a VxWorks Image Project*, p.88.

- Configure the VxWorks Image project as described under *5.6 Adding Application Projects to the VxWorks Image Project*, p.99 and *5.4 Configuring Kernel Components*, p.93.
- Before the target is disconnected from the host system, create a target-side file system, see *7.2 Creating a VxWorks ROMFS File System Project*, p.110.

12 Creating Native Application Projects

12.1 Introduction 151

12.2 Creating a Native Application Project 152

12.3 Configuring Native Application Projects 152

12.4 Native Applications in the Project Navigator 158

12.5 Application Code for a Native Application Project 160

12.1 Introduction

Use a *Native Application project* for C/C++ applications developed for your host environment.

Workbench provides build and static analysis support for native GNU 2.9*x*, GNU 3.*x*, and Microsoft development utilities (assembler, compiler[1], linker, archiver) though you must acquire and install these utilities as they are not distributed with Workbench.

There is no debugger integration for native application projects in Workbench, so you must acquire and use the appropriate native tools for debugging as well.

1. Workbench supports the MinGW, Cygnus, and MS DevStudio compilers. Compilers for native development are distributed with Wind River VxWorks Platforms, but not with Workbench.

12.2 Creating a Native Application Project

Before creating the project, see the general comments on projects and project creation in *4. Projects Overview*.

1. Create a Native Application project by selecting **File > New > Wind River Workbench Project**. The New Wind River Workbench Project wizard appears.
2. Select **Host Operating System (Native Development)**, then click **Next**.
3. From the **Build type** drop-down list, select the type of application you want to create. Click **Next**.
4. Type a name for your project.
5. Decide where to create your project:

   Create project in workspace
   Leave this selected if you want to create the project under the current workspace directory.

   Create project at external location
   Select this option, click **Browse**, then navigate to a different location if you want to create the project outside the workspace.

   Create project in workspace with content at external location
   Select this option, click **Browse**, then navigate to your source location if you do not want to mix project files with your sources, or copy sources into your workspace.
6. When you are ready, click **Finish**. The Native Application project is created and appears at the root level in the Project Navigator.

12.3 Configuring Native Application Projects

Once you have created your project, you can configure its build support and specs, build tools, build macros, and build paths.

For general details about build properties, see *16.4 Accessing Build Properties*, p.214 or press the help key for your host.

1. To access build properties for your project, right-click it in the Project Navigator and select **Properties**.

2. From the **Properties** dialog, click **Build Properties**.

12.3.1 Configuring Build Support and Specs

Use this tab to configure build support and build specs for your project.

1. A Native Application project is a predefined project type that uses Workbench build support, so build support is enabled by default. If you are creating a project because you want to browse symbol information and you are not interested in building it, click **Disabled** to disable build support (you can click **Managed build** to re-enable it later, if you want).
2. If necessary, edit the default **Build command**.
3. All available build specs are selected (and therefore enabled) by default. To restrict the list of enabled build specs to only those you need for your project, click **Disable All** and then select the checkbox next to the build spec(s) you want to enable for this project.

   If you are working on a Windows application, you would normally enable the **msvc_native** build spec, and disable the **gnu-native** build specs. If you are working on a Linux or Solaris native application, you would normally enable the GNU tool version you are using, and disable all others.



NOTE: Highlighting the name of the build spec is not sufficient to enable it; there must be a check in the checkbox to enable the build spec.

4. To reset build properties to their default settings or import build settings from another project, click **Import** and select the source of the build settings.
5. If you enabled one build spec for this project, it appears in the **Default build spec** and **Active build spec** fields.

   If you enabled more than one, the build spec in the **Default build spec** field is used for builds, though you can select a different **Active build spec** for a particular build. The build spec in the **Active build spec** field is also propagated to the appropriate fields on the Build Tools, Build Macros, and Build Paths tabs.
6. Select or clear the **Debug Mode** checkbox, depending on whether you want the build output to include debug information or not.



NOTE: You can also change the active build spec and the debug mode by right-clicking the project in the Project Navigator and selecting **Build Options > Set Active Build Spec**.

7. For more information about the build settings on this tab, press the help key for your host. You may continue configuring your project by selecting another build tab, or if you are finished, click **OK**.

12.3.2 Configuring Build Tools

Use this tab to configure build tools, build output generation, and build flags for your project.

1. The build tools you can select for Native Application projects are **C-Compiler**, **C++-Compiler**, **C-Linker**, **C++-Linker**, **Librarian**, or **Assembler**. In addition, you can define your own build tool.
   - **C-Compiler**, **C++-Compiler**, **Assembler**: These tools produce a *BuildTargetName***.obj** file on Windows or a *BuildTargetName***.o** file on UNIX.
   - **C-Linker**: The linker produces a *BuildTargetName***.exe** file on Windows and a *BuildTargetName* file on UNIX. This partially linked and *munched* (integrated with code to call C++ static constructors and destructors) object is intended for downloading. The C-Linker output product cannot be passed up to superprojects, although the current project's own, unlinked object files can be passed, as can any output products received from projects further down in the hierarchy.
   - **C++-Linker**: This linker produces a *BuildTargetName***.exe** file on Windows and a *BuildTargetName* file on UNIX.
   - **Librarian**: The Librarian produces a *BuildTargetName***.lib** file on Windows and a *BuildTargetName***.a** file on UNIX. The Librarian output product can be passed up to superprojects, as can the current project's own, unlinked object files, as well as any output products received from projects further down in the hierarchy.
   - To define your own build tool, click **New** and enter a build tool name, then click **OK**. Your build tools appears in the drop-down list, and you can configure all build tool settings to fit your needs.
2. For more information about the build settings on this tab, press the help key for your host. You may continue configuring your project by selecting another build tab, or if you are finished, click **OK**.



NOTE: If your project uses the standard managed build, and you select a different build tool on the Build Tools tab, this choice does not carry over to the Build Targets tab. You must manually assign the new build tool to your build target.

12.3.3 Configuring Build Macros

Use this tab to define global and build spec-specific macros that are added to the build command when executing builds.

1. To change the value of an existing global build macro, select the value in the **Build macro definitions** table and type in a new one, or click **Edit** and type in the new value, then click **OK**.
2. To define a new global build macro, click **New** next to the table, then enter a **Name** and **Value** for the macro. Click **OK**.



NOTE: You can define and use global build macros even if you do not select or define any build specs for your project.

3. To change the value of an existing build spec-specific macro, select the **Active build spec** for which the value should be applied, select the value in the **Build spec-specific settings** table, then type in a new one. Or click **Edit** and type in the new value, then click **OK**.
4. To define a new build spec-specific macro, click **New** next to the table, enter a **Name** for the macro, leave the **Value** blank, then click **OK**.

   To define the value, select the macro, select the **Active build spec** for which the value should be applied, click **Edit** and enter the **New value**, then click **OK**.

   The macro will always be appended to the build command when a build is launched, and the value will be set according to the active build spec, including empty values.

   For example, if the build command is **make --no-print-directory** and the macro is **TEST_SPEC**, you can define values to be used with different build specs:

   spec 1: Value = **spec1Val**
   spec 2: Value = **spec2Val**
   spec 3: Value =

   The resulting build commands are as follows:

build command for spec 1: **make --no-print-directory TEST_SPEC=spec1Val**
build command for spec 2: **make --no-print-directory TEST_SPEC=spec2Val**
build command for spec 3: **make --no-print-directory TEST_SPEC=**

5. For more information about the build settings on this tab, press the help key for your host. You may continue configuring your project by selecting another build tab, or if you are finished, click **OK**.

12.3.4 Configuring Build Paths

Use this tab to specify a redirection root directory for your build output, and add, delete, or reorder the directories searched by the build tools.

1. By default, build output is directed to a subdirectory in your workspace. However, if you want to redirect it somewhere else in your file system, specify a location in the **Redirection root directory** field.
2. The **Redirection directory** is a subdirectory of the **Redirection root directory**. By default this directory has the same name as the **Active build spec**, though you can change it by typing a new directory name in the field.
3. The **Include paths** table shows the paths used by the compiler to resolve include directives. To analyze your project and update the displayed include paths, click **Generate** to open the **Generate Include Search Paths** dialog.
   a. On the **Analyze include directives** page, specify which include directives you want Workbench to ignore when generating include paths.
      – Select the first box to ignore inactive include directives, or clear it to analyze them. Inactive include directives are those directives ignored by the build system because they are surrounded by compiler options and preprocessor directives such as **#ifndef**.
      – Select the second box to ignore system includes, or clear it to analyze them.

      When you are ready, click **Next** to analyze include directives. This may take some time for a large project.
   b. The **Resolve include directives** page displays results from the analysis. The upper field displays **unresolved include directives** in three groups: resolvable by one include search path, resolvable by multiple search paths, and not resolvable. The lower field displays **resolved directives**: predefined search paths, as well as the search paths Workbench was able to resolve.

- To automatically resolve all include directives that can be resolved, click **Resolve All**.
- To automatically resolve an individual include directive or one of the groups of unresolved directives, click **Resolve**. If Workbench found one matching header, it will resolve the include directive. If it found multiple matching headers, a dialog displays the headers and asks you to select one.
- To open the file so you can see the context of an unresolved include directive, click **Show in Editor**.



NOTE: When automatically resolving include directives, Workbench uses heuristics to determine the best matches, but the results may be incorrect. So you should examine and if necessary **Remove** undesired search paths in the lower field. The newly-generated search paths are marked with a yellow plus on the folder icon ().

- To manually resolve include directives for which no matching headers were found, click **Add** and navigate to the location of the appropriate headers. Click **OK** to add the path to the list of resolved directives; any directives resolved by that path are removed from the unresolved list.
- To view, enable, or disable variables for paths and path segments, click **Substitution**. Click a variable or group of variables to enable or disable them.

  Variables are grouped into four groups, and are applied to all generated paths in the resolved directives field:

  Wind River environment: includes Wind River platform and Workbench variables.

  Build macros: includes global and local build macros, as defined in project properties.

  Project locations: includes variables referring to projects in the workspace.

  System environment: includes all environment variables that do not appear in the build macro or Wind River environment groups.

  Click **Apply** to see your changes but leave the dialog open for further editing, or click **OK** if you are finished.
- To copy search paths to the clipboard so you can paste them into a make file, select the search path then click **Copy**.

c. When you are ready, click **Finish** to return to the Build Paths tab.

4. To manually add an include directory for an **Active build spec**, select the build spec from the drop-down list, click **Add**, and browse to or type in the path (be sure not to erase the **-I** at the beginning of the path). Click **OK**.

   To add an include path that applies to all your build specs, click **Add to all** and then browse to or type in the path, then click **OK**.

5. When you are finished configuring your project, click **OK**.

12.4 Native Applications in the Project Navigator

After a Native Application project has been created, a number of nodes appear in the Project Navigator. This section describes these nodes as they appear immediately after project creation. For general notes about manipulating nodes, for example, moving, copying, filtering, etc., please see *13. Working in the Project Navigator*.

12.4.1 Global Project Nodes



ProjectName

The icon at the root of the project tree identifies the type of project; the icon's label is the name you gave the project when you created it.

12.4.2 Project Build Specs and Target Nodes

Each target node is associated with a predefined build specification.

The build target depends on the options you selected during project creation. Specifically, you will not have both an archive (*TargetName*.**a** for a **gnu** build spec, or *TargetName*.**lib** for a **msvc** build spec) target and a *TargetName*(**.exe** for a **msvc** build spec) target immediately after project creation. Which of these will be visible depends on the build tool you selected. Also, the presence or absence of the green upward arrow on the target icon (to indicate whether the target is passed up the hierarchy) will be determined by your project settings.


TargetName[**.exe**] (*BuildSpecName*[**_DEBUG**])
An executable.


TargetName.**a** | **.lib** (*BuildSpecName*[**_DEBUG**])
An archive produced by the **Librarian** build tool.

12.4.3 Makefile Nodes

At project generation time two Makefiles are copied to the project. One is a template that can also be used for entering custom make rules. The other is dynamically regenerated from build spec data at each build.


.wrmakefile
A template used by Workbench to generate the project's **Makefile**. Add user-specific build-targets and make-rules in this file. These will then be automatically dumped into the Makefile.


Makefile
Do *not* add custom code to this file. This **Makefile** is regenerated every time the project is built. The information used to generate the file is taken from the build specification that on which the target node is based.

12.4.4 Project File Nodes

The project creation facility generates, or includes copies of, a variety of files when a project is created. Normally, you need not be concerned with these files. However, here a brief summary of the DKM project files displayed in the Project Navigator:


.project
Eclipse platform project file containing builder information and project nature.


.wrproject
Workbench project file containing common project properties such as project type, etc.

12.5 Application Code for a Native Application Project

After project creation you have the infrastructure for a Native Application project, but often no actual application code. If you are writing code from the beginning, you can add new files to a project. If you already have source code files, you will want to import these to the project. For more information please refer to *13.3.1 Importing Resources*, p.162, and *13.3.2 Adding New Files to Projects*, p.163.

13 Working in the Project Navigator

13.1 Introduction 161

13.2 Creating Projects 162

13.3 Adding Application Code to Projects 162

13.4 Opening and Closing Projects 163

13.5 Scoping and Navigation 164

13.6 Moving, Copying, and Deleting Resources and Nodes 166

13.1 Introduction

The Project Navigator is your main graphical interface for working with projects. You use the Project Navigator to create, open, close, modify, and build projects. You also use it to add or import application code, to import, or customize build specifications, and to access your version control system.

Various filters, sorting mechanisms, and viewing options help to make project management and navigation more efficient. Use the arrow at the top-right of the Project Navigator to open a drop-down menu of these options.

13.2 Creating Projects

Creating projects is discussed in general under *4. Projects Overview*. Specific descriptions for creating individual project types are provided in the other chapters in *Part II. Projects*.

13.3 Adding Application Code to Projects

After creating a project, you have the infrastructure for a given project type, but no actual application code. If you already have source code files, you will want to import these to the project.

13.3.1 Importing Resources

You can import various types of existing resources to projects by choosing **File > Import**.

For details about the entries in the Import File dialog, open it and press the help key for your host.



NOTE: If Workbench encounters a problem while importing resources, it will display an error (such as the project already contains a file with the same name). If you click **OK**, the Import wizard reappears with all the original settings. This gives you the opportunity to fix just the item causing the problem, rather than having to re-enter all the selections in the wizard.

If you do not want to fix the problem and import the resources now, click **Cancel**.



NOTE: Importing resources creates a link to the location of those resources; it does not copy them into your workspace.

Later, if you want to delete a project, check the path in the **Confirm Project Delete** dialog very carefully when deciding whether to choose **Also delete contents under** '*path*' or **Do not delete contents**—choosing to delete the project contents may delete your original sources or the contents of a project in a different workspace, rather than the project in your current workspace.

If this happens, right-click the folder that originally contained the files, then select Restore from Local History. Workbench will show you a list of files you can choose to restore.

13.3.2 Adding New Files to Projects

To add a new file to a project, choose **File > New > File**.

You are asked to **Enter or select the parent folder**, and to supply **Filename**.

For a description of the **Advanced** button, and what it reveals, open the **New File** dialog and press the help key for your host.

13.4 Opening and Closing Projects

You can open or close a project by right-clicking it in the tree and choosing **Open Project** (if it is currently closed), or **Close Project** (if it is currently open). You can also select **Project > Open Project** or **Project > Close Project**.

13.4.1 Closing a Project

- The icon changes to its closed state (by default grayed) and the tree collapses.
- All project member files that are open in the editor are closed.
- All subprojects that are linked exclusively to the closed project are closed as well. However, subprojects that are shared among multiple projects remain open as long as a parent project is still open, but can be closed explicitly at any time.

- In general, closed projects are excluded from all actions such as symbol information queries, and from workspace or project structure builds (that is, if a parent project of a closed subproject gets built).
- It is not possible to manipulate closed projects. You cannot add, delete, move, or rename resources, nor can you modify properties. The only possible modification is to delete the project itself.
- Closed projects require less memory.

13.5 Scoping and Navigation

There are a number of strategies and Workbench features that can help you manage the projects in your workspace, whether you are working with multiple projects related to a single software system, or multiple unrelated software systems.

- **Close projects**

  If you expect to be working in a different context (under a different root project) for a while, you can right-click the project you are leaving and select **Close Project**.

  If you close your root projects when you stop working on them, you will see just the symbols and resources for the project on which you are currently working (see also *13.4.1 Closing a Project*, p.163).

- **Go into a project**

  If you want to see, for example, the contents of only one software system in the Project Navigator, select its root project node and right-click **Go Into**. You can then use the navigation arrows at the top of the Project Navigator to go back out of the project you are in, or to navigate history views.

- **Open a project in a new window**

  If you expect to be switching back and forth between software systems (or other contexts) at short intervals, and you do not want to change your current configuration of open editors and layout of other views, you can open the other software system's root project in a new window (right-click **Open in New Window**). This essentially does the same as **Go Into** (see

Go Into a Project), except that a new window is opened, thereby leaving your current Workbench layout intact.

- **Open a new window**

  You can open a new window by choosing **Window > New Window**. This opens a new window to the same workspace, leaving your current Workbench window layout intact while you work on some other context in the new window.

- **Use Working Sets**

  Using working sets lets you set the scope for all sorts of queries. You can, for example, create working sets for each of your different software systems, or any constellation of projects, and then scope the displayed Project Navigator content (and other query requests) using the pull-down at the top-right of the Project Navigator.

  To create a Working Set, from the drop-down menu, choose **Select Working Set**. In the dialog that appears, click **New**, then, in the next dialog, specify the **Resource** type.

  In the next dialog select, for example, a software-system root project and give the working set a name. When you click **Finish**, your new working set will appear in the **Select Working Set** dialog's list of available working sets.

  After the first time you select a working set in the **Select Working Set** dialog, the working set is inserted into the Project Navigator's drop-down menu, so that you can directly access it from there.

- **Use the Navigate Menu**

  For day-to-day work, there is generally no absolute need to see the contents of your software systems as presented in the Project Navigator.

  Using the **Navigate > Open Resource** (to navigate files) and **Navigate > Open Symbol** (to jump straight to a symbol definition) may often prove to be the most convenient and efficient way to navigate within, or among, systems.

13.6 Moving, Copying, and Deleting Resources and Nodes

The resources you see in the Project Navigator are normally displayed in their logical, as opposed to physical, configuration (see *4.5 Projects and Project Structures*, p.80). Depending on the type of resource (file, project folder) or purely logical element (target node) you are manipulating, different things will happen. The following section briefly summarizes what is meant by resource types and logical nodes.

13.6.1 Resources and Logical Nodes

Resources is a collective term for the *projects*, *folders*, and *files* that exist in Workbench.

There are three basic types of resources:

- *Files*

  Equivalent to files as you see them in the file system.

- *Folders*

  Equivalent to directories on a file system. In Workbench, folders are contained in projects or other folders. Folders can contain files and other folders.

- *Projects*

  Contain folders and files. Projects are used for builds, version management, sharing, and resource organization. Like folders, projects map to directories in the file system. When you create a project, you specify a location for it in the file system.

  When a project is open, the structure of the project can be changed and you will see the contents. A discussion of closed projects is provided under *13.4.1 Closing a Project*, p.163.

Logical nodes is a collective term for nodes in the Project Navigator that provide structural information or access points for project-specific tools.

- *Subprojects*

  A project is a resource in the root position. A project that references a superproject is, however, a logical entity; it is a reference only, not necessarily (or even normally) a physical subdirectory of the superproject's directory in the file system.

- *Build Target Nodes*

  These are purely logical nodes to associate the project's build output with the project.

- *Tool Access Nodes*

  These allow access to project-specific configuration tools. VxWorks ROMFS File System Projects have a node that opens a tool for mapping host-side project contents to target file system contents. VxWorks Image Projects have a node that opens the Kernel Configuration Editor for configuring the VxWorks kernel.

13.6.2 Manipulating Files

Individual files, for example source code files, can be copied, moved, or deleted. These are physical manipulations. For example, if you hold down **CTRL** while you drag-and-drop a source file from one project to another, you will create a physical copy, and editing one copy will have no effect on the other.

13.6.3 Manipulating Project Nodes

Although copying, moving, or deleting project nodes are undertaken with the same commands you would use for normal files, the results are somewhat different because a project folder is a semi-logical entity. That is, a project is a normal resource in the root position. A project that is referenced as a subnode is, however, a logical entity; it is a reference only, not a physical instance.

If you copy/paste (or hold down **CTRL** while you drag-and-drop) a project folder node to a new location in the project editor (for example, under some other project node to be used as a subproject there) all that happens is that a reference to one and the same project is inserted. This means that if you modify the properties of one instance of the subproject node, all other instances (which are really only references) are also modified. One such property would be, for example, the project name. If you rename the project node in one context, it will also be renamed in all other contexts.

Moving and (Un-)Referencing Project Nodes

If you drag-and-drop a project folder, you are making a logical, structural change. However, if you select a project folder node and right-click **Move**, you will be

asked to enter (browse for) a new file system location. All the files associated with the current project will then be physically moved to the location you select, without any visible change in the Project Navigator (you can verify the new location in the **Project Properties**).

When you drag-and-drop a project node, you are actually performing the equivalent of right-click **Add as Reference** or, if you have selected a subproject, also right-click **Remove Reference**. These commands open a dialog allowing you to either have the currently selected project reference other projects as a subproject, or, in the **Remove Reference** dialog, to remove the currently selected project from its structural (logical) context as a subproject, in which case it will be moved to the root level as a standalone project in the Project Navigator.

Deleting Project Nodes

To delete a subproject, which might potentially be linked into any number of other project structures, you first have to either unlink (right-click it and press **Delete**) all instances of the subproject, or get a *flat* view of your workspace. To do this, open the drop-down list at the top-right of the Project Navigator's toolbar and choose **Hide > Project Structure**. This hides the logical project organization and provides a flat view with a single instance of the (sub)project that you can then delete by pressing **Delete** again.

When you delete a project you are asked whether or not you want to delete the contents. If you choose not to delete the contents, the only thing that happens is that the project (and all its files) are no longer visible in the workspace; there are no file system changes.

13.6.4 Manipulating Target Nodes

Target nodes cannot be copied or moved. These are purely logical nodes that make no sense anywhere except in the projects for which they were created. If you copy or move entire projects, however, the target nodes and generated build-targets beneath them are also copied.

Deleting Target Nodes

Deleting a target node also removes the convenience node that represents the generated, physically existing build-target. However, the physically existing build-target (if built) is not deleted from the disk.

The convenience node referred to above, lets you see at a glance whether the target has been built or not, even if you have uncluttered your view in the Project Navigator by hiding build resources (in the drop-down menu at the top-right choose **Hide > Build Resources**) and/or collapsing the actual target node. If you have collapsed the node, the **+** sign will indicate that the build-target exists).

14

Advanced Project Scenarios

14.1 Introduction 171

14.2 Resource Locations 172

14.3 Multiple, Unrelated Software Systems 173

14.4 Complex Project Structures 174

14.1 Introduction

The scenarios developed in this chapter suggest how you could use the Wind River Workbench to manage various constellations of projects and project types. Because Workbench provides a variety of possibilities for achieving different ends, the scenarios are neither prescriptive, nor comprehensive. All we can do here is offer some suggestions.

The scenarios do not look at the edit/compile/debug cycle; the emphasis is on project organization and handling. The discussion looks at:

- resource locations
- strategies for working with multiple, unrelated software systems
- complexities within a single software system, including project structure design, development, and finalization steps

14.2 Resource Locations

One complexity that you might be faced with, especially in team development situations, is that you might have to use file system resources (files and directories) that are outside your workspace.

As long as file system resources are located in the default location (your own workspace), for example because you have checked them out from your version control system, there is nothing to discuss.

When you create projects in Workbench, project-specific administrative files are stored at the file system location of the resources used by the project. This means that, if these resources are outside your workspace, you may not have write permission there and that the necessary files therefore cannot be created.

This may be an issue, for example, also with respect to centrally maintained header files and third party libraries. In such cases you have the following options:

- Have your administrator, who does have write permission, create the project (see *Creating Projects for External Headers*, p.185) and import the project as follows:
  - In the Project Navigator, right-click **Import**.
  - In the **Import** wizard, select **Existing Project into Workspace** and click **Next**.
  - **Browse** to the directory where the project was created and click **OK**, then **Finish**.

  This is the recommended way to proceed in cases where not everyone is allowed to write to resource directories. This way all team members always access both the *same*, most up to date source files and the *same* project, thereby ensuring consistency across the entire team without any synchronization overhead. Note that, if you have multiple workspaces, you would have to import the project to each workspace.

  Furthermore, if the external resources are not just header files, that is, if they are buildable, build support must be either disabled for the imported project (if existing build output is externally available), or build output of the imported projects must be redirected somewhere that users have write permission (open the build properties dialog, click the build paths tab, and press the help key for your host). Write permission will also be required for the **.wrproject** file in the project directory and the **.wrfolder** files in each folder, for modifications (added/removed resources) and for maintaining changes in build properties.

- The other option is to copy the resources to somewhere that you do have write permission.

  This option is *not* recommended because of the synchronizations problems that are bound to arise sooner or later. Consider this a last resort.

14.3 Multiple, Unrelated Software Systems

The assumption is that you work on multiple, unrelated software systems in parallel. Each of these systems will normally (but not necessarily) consist of any number of subprojects organized into project structures; that is, each system will normally be arranged as a tree under a single superproject. However, ignoring the internal organization of your software systems for the moment (this is discussed under *Complex Project Structures*, p.174), first look at the software systems as a whole.

During the course of any working day you might spend time working in different software systems that have nothing to do with each other (other than the fact that you happen to be working in them). You will presumably want to be able to focus as fully as possible, with as little distraction as possible, on the software system you are working on at any given time. If you have to switch from one system to the other fairly frequently, the switch should be easy and rapid.

14.3.1 Using Different Workspaces for Different Systems

Using different workspaces for unrelated software systems lets you keep these systems completely separate, without seeing any sign of the currently non-relevant context anywhere.

However, when you switch from one workspace to another (choose **File > Switch Workspace**), you are actually closing your current Workbench instance and reopening a new instance that uses the selected new workspace. This takes time, but offers the advantage that the new workspace opens exactly as you left it when you last closed it.

This option, because of the time overhead involved in switching, is probably most feasible if you have only a few separate software systems, and if you spend extended periods of time in one or other context without interruption.

However, if you have, a system that you work on most of the time, and several other systems where you have to frequently do relatively minor maintenance work, you might find it more convenient to use a single workspace for all, or many of, your projects.

Naturally, there is no reason why you should not have both multiple workspaces as outlined here, and, within one or more of these, also maintain multiple, unrelated software systems in the same workspace as discussed below.

14.3.2 Using the Same Workspace for Different Software Systems

Using the same workspace for any number of unrelated software systems does not stop you from keeping these systems completely separate. The only sign of each currently non-relevant system can be a single icon (or not even that if you **Go Into** a project - see *13.5 Scoping and Navigation*, p.164). This means that all software systems are immediately visible and accessible, without being unduly obtrusive. Furthermore, switching from one software system to another is much faster than using different workspaces as described above. On the other hand, if you are working on multiple, very large software systems, general performance might become an issue that would suggest using separate workspaces.

Some of the ways that will help you handle multiple software systems in the same workspace are introduced under *13.5 Scoping and Navigation*, p.164

14.4 Complex Project Structures

This section develops a simple infrastructure as a possible approach to a high-level, internal organization of an individual software system.

14.4.1 Project Assumptions

The following discussion attempts to align how Workbench project structures and project types can support a software system that includes the following requirements.

- **There is a kernel**

  In the design phase, you need not think too much about the kernel. It is sufficient to know that there will be one at some point.

  Use a simulator for initial development and testing.

- **The output product must be a single flashable image**

  This image will contain the kernel as well as all the run-time components (binaries from Real-time Process Projects, libraries, data files, and so on). A target-side file system is therefore required; this will be implemented using Wind River ROMFS technology by setting up a VxWorks ROMFS File System project.

  However, in the design phase, you do not need not worry about this; it is sufficient to know that there will be a VxWorks ROMFS File System project at some point.

- **The software system will have to be ported to different boards**

  Although the kernel as such is not initially of primary importance, the assumption that you will have to port the system at some stage may be a design consideration. If you are developing and testing on a simulator (see above), there will be porting to do anyway.

- **There are run-time products.**

- **One or more modules are needed as abstraction layers that wrap around the kernel**

  Use Downloadable Kernel Module Projects for these.

- **There are application modules**

  These have to be process-based and they have to run in their own memory-protected address space.

  Use Real-time Process Projects for these.

- **There are shared libraries**

  These are potentially used by any or all of the application modules.

Use Shared Library Projects for these.

- **There is legacy code**

  Use User-Defined Projects and/or Real-time Process Projects and/or Downloadable Kernel Module Projects.

  User-Defined Projects are appropriate in situations where you would rather not tamper with how the application is built. In other situations, you can wrap your legacy projects in one of the standard project types supported by Workbench.

- **There are external headers**

  These are centrally maintained and are potentially used by any or all of the software system's modules.

  Use a User-Defined project (without build support) for these.

- **Building a complete product image must be simple**

  14.4.2 Infrastructure Design, p.176, tries to meet all the above requirements and provide a push-button build of the full product image, including all its components, for multiple architectures.

14.4.2 Infrastructure Design

Based on the *Project Assumptions*, p.175, the following describes how you could go about building an infrastructure for such a software system.



NOTE: The screenshots in the following have been filtered in various ways to hide everything that is not related to project structure. If you follow the procedures described, you will see this same structure, as well as a number of additional files, folders, target nodes, and so on.

The infrastructure described here is not a requirement for project management in Workbench. It can however be convenient to create such an infrastructure to facilitate porting a software system to other boards, as well as to allow building an entire product image, even for multiple boards, all at once. Furthermore, such an infrastructure does not need to be in place from the start; it can be folded over a project system at any stage of development.

Create Container Projects

This infrastructure uses empty container projects at the superproject level as well as at subproject levels. The type of container used in each case will depend on the type of content the container will later accommodate.

In the current context, the term *container project* is therefore used to denote a project of any type that does not, however, itself contain any source code files; all application source files will be in subprojects referenced by the empty container project.

Step 1: Create a container project.

Creating a container project as the topmost superproject the software system is an organizational artifact to provide a convenient way of keeping everything together, and thereby also cleanly separating the software system from other software systems you might work on in the same workspace.

The only other real functionality the superproject container project needs to provide is that it has to be buildable. Although the project itself contains no source code files, you will want to able to start the build at the top of your future project tree to recursively build the whole structure.

The default User-Defined project provided by Workbench is exactly what you need for a topmost container project.

To create a new User-Defined project, in the Project Navigator, right-click **New > User-Defined Project**.

In the wizard that appears, in the **Project name** field, enter: **playpen_sim** (this is an arbitrary name for a fictitious software system; the suffix **_sim** reflects that this system will be built for the simulator) and click **Finish**. (You can ignore the **Next** button and the other Wizard pages because the defaults are fine.)

This creates a default User-Defined project; that is, one that supports a user-defined build based on existing makefiles. Since this is a just a container project without any (user-defined) makefiles, Workbench will create a **Makefile** with a default **all** rule and a **clean**. This allows you to use the **Build Project**, **Rebuild Project**, and **Clean Project** menu commands, as well as preventing the generation of irritating build errors. If you want, you can write any other rules into this file at any time. See also *11. Creating User-Defined Projects*.

Step 2: Create container projects for each project type and for external headers.

Recall that *Project Assumptions*, p.175, stated requirements for Downloadable Kernel Modules, Real-time Process Projects, Shared Library Projects, and User-Defined projects.

Creating empty projects for each of these project types facilitates porting from the simulator to a board, and from one board to another. This is because, in a tree of projects of the same type, all subprojects are built using the same build spec as that used by the topmost project. This applies to all project types except User-Defined projects (there is no way to predict how these are built).

So, for example, by creating an empty Real-time Process project type container project and later populating this container with real Real-time Process project type subprojects, then you only need to use a different build spec for the container when it comes to porting the system to a different board (more about this later).

Note that Real-time Process projects and Shared Library projects actually use the same build specs, so, technically speaking, you could lump these two project types together under one container and save yourself a couple of steps. However, the orderly separation of project types appears a little cleaner and is therefore adopted here.

The naming convention used for these containers indicates the project type that will be stored within (actually only reference) these containers, plus a suffix that indicates the software system they belong to and the board they will be built for (**_playpen_sim**).

To create the empty container project types, proceed as follows:



NOTE: You can ignore the **Next** button and click **Finish** on the first page in each of the wizards because the defaults are fine for the moment.



NOTE: Project references can only be created if the projects are based on the same Platform. *Platform* here refers to the settings in **Window > Preferences > General > Target Operating Systems**.

- To create a new container Downloadable Kernel Module project, in the Project Navigator, right-click **New > Downloadable Kernel Module Project**.

  In the wizard that appears, in the **Project name** field, enter: **DKMs_playpen_sim** and click **Finish**.

- To create a new container Real-time Process project, in the Project Navigator, right-click **New > Real Time Process Project**.

In the wizard that appears, in the **Project name** field, enter: **RTPs_playpen_sim** and click **Finish**.

- To create a new container Shared Library project, in the Project Navigator, right-click **New > Shared Library Project**.

  In the wizard that appears, in the **Project name** field, enter: **LIBs_playpen_sim** and click **Finish**.

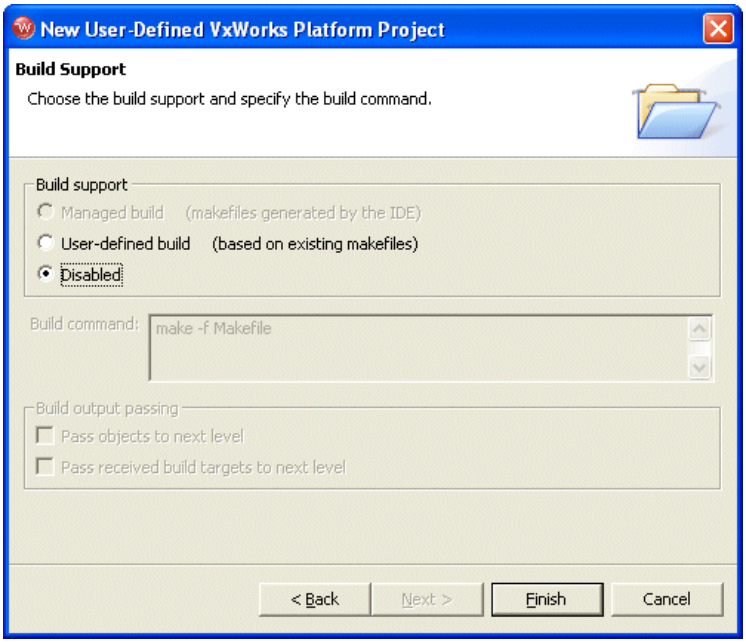
- To create a new container User-Defined project, in the Project Navigator, right-click **New > User Defined Project**.

  In the wizard that appears, in the **Project name** field, enter: **UDPs_playpen_sim** and click **Finish**.

- To create a new container User-Defined project (without build support) to accommodate the external, centrally maintained header files mentioned in *Project Assumptions*, p.175, in the Project Navigator, right-click **New > User-Defined Project**.

  In the wizard that appears, in the **Project name** field, enter: **headers_playpen** (notice that we have not appended the suffix **_sim**; this is because this project does not use a build spec, see below) and keep clicking **Next** until you get to the wizard's **Build Support** page.

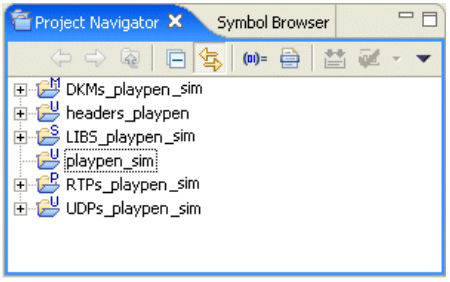
Figure 14-1 **Disable Build Support for Header Projects**



In the wizard's **Build Support** page, select the **Disabled** option and click **Finish**.

In the Project Navigator you should now see the flat list of container projects (collapsed) shown in Figure 14-2.

Figure 14-2 **Container Projects**



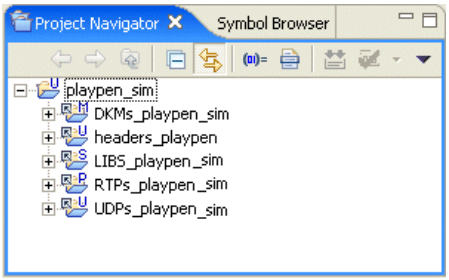
Step 3: Drop all container projects onto the topmost container project.

The topmost container project must be referenced by all other container projects; in other words, all other container projects must by subprojects of **playpen_sim**.

In the Project Navigator, select all projects except **playpen_sim** and drag-and-drop them into **playpen_sim**.

Figure 14-3 illustrates the infrastructure created in the above steps. Notice the referencing arrows at the left of the subproject icons.

Figure 14-3 **Container Projects Referenced by the Topmost Container**



14.4.3 Development

Once you set up the infrastructure for your first board (or simulator), you will populate the container projects with real projects that actually contain source files.

In order to later facilitate porting the software system to other boards you would, organize these, at least initially, so that:

- All Real-time Process projects are subprojects of **RTPs_playpen_sim**.
- All Downloadable Kernel Module projects are subprojects of **DKMs_playpen_sim**.
- All Shared Library projects are subprojects of **LIBs_playpen_sim**.
- All projects for external headers are in **headers_playpen**.
- All User-Defined projects (except the ones in **headers_playpen**, where build support is disabled) are subprojects of **UDPs_playpen_sim**.

Referencing Containers

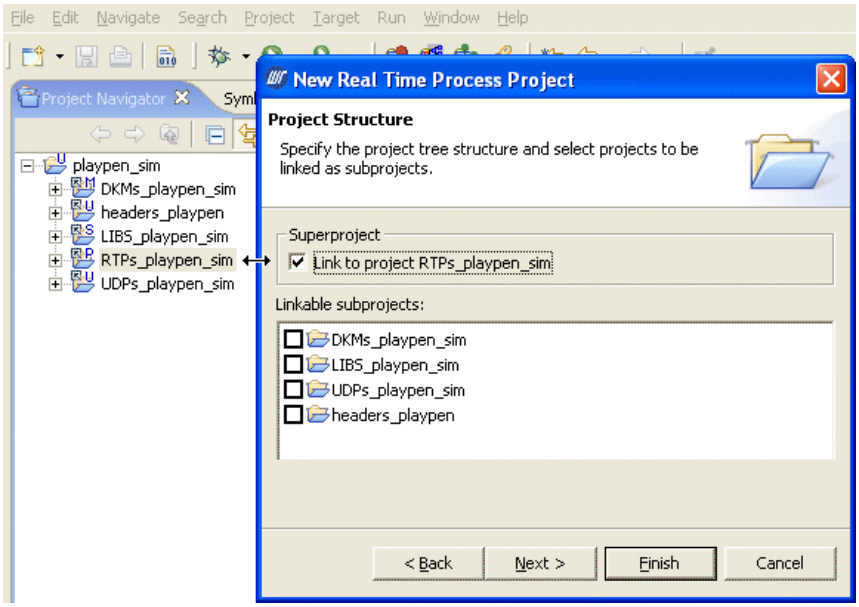
There are a number of ways you can associate projects with other projects as subprojects. Above, during the creation of the container project infrastructure, drag-and-drop was used. Another method is to right-click **Add Reference**. You can also create the reference during project creation as demonstrated in the example below.

Example 14-1 **Creating and a Project and Referencing its Container**

Assumption: you are creating a Real-time Process project. This, according to the conventions outlined above, will be a subproject of **RTPs_playpen_sim**. The quickest way to achieve this is:

1. In the Project Navigator, select **RTPs_playpen_sim**. This is the Real-time Process project you are currently creating should reference as a subproject.
2. Right-click **New > Real Time Process Project**.
3. In the wizard, enter a **Project name** (use **rtp_1** in this example) and click **Next**.
4. In the wizard's **Project Structure** page there is a **Superproject** check box labelled **Link to project RTPs_playpen_sim**. This check box appears because you selected the **RTPs_playpen_sim** project in step 1, above.
5. Select **Reference RTPs_playpen_sim** and continue to create the project.

Figure 14-4 **Linking as Subproject during Project Creation**



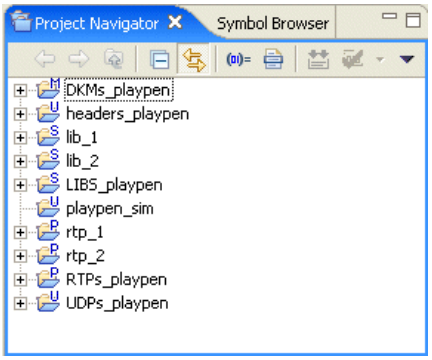
14

Shared Libraries

The recommended convention, above that "All Shared Library projects are subprojects of LIBs_playpen_sim." might seem strange. Shared libraries are normally subprojects of the projects that use them, so why put shared libraries in this seemingly disconnected location (**LIBs_playpen_sim**)?

The libraries are actually even more disconnected than they appear. Remember that, physically speaking, all the projects in any project structure, no matter how deep, are topographically flat as shown in Figure 14-5. This figure shows exactly the same system as Figure 14-6, which displays the logical view you normally see (you can switch from one representation to the other using the drop-down menu at the top-right of the Project Navigator **Hide > Project Structure**).

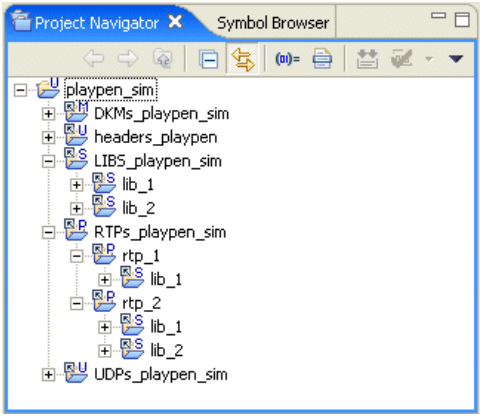
Figure 14-5 **Physical View of the System**



While it is true that you will normally only have libraries as subprojects of applications that use them (even if you are developing a library you will probably have a test application project above the library), it does not matter how often a library node occurs in a given tree, or even in the entire workspace, it is physically only one library and will therefore only be built once (see Figure 14-5). In this sense, it does not matter that the libraries will appear in one extra place, **LIBs_playpen_sim**.

Figure 14-6 shows exactly the same system as Figure 14-5. Notice that the Shared Library project, **lib_1**, occurs three times: once each under **rtp_1** and **rtp_2**, and once, seemingly unnecessarily, under **LIBs_playpen_sim**.

Figure 14-6 **Logical View of the System**



If you adhere to the convention recommended above, that "All Shared Library projects are subprojects of LIBs_playpen_sim.", you will have to copy (for example, using hold down **CTRL** while you drag-and-drop) library nodes to subproject locations under applications that use them. Note again that when you do this, you are not really copying anything; you are creating references—if anything, you are copying *links* (again note the reference arrows on subproject icons). However, on the upside, whenever you need to add your library projects to applications, you will know exactly where to find them because they are neatly collected in their container project, in our example, **LIBs_playpen_sim**.

The other advantage of adhering to this convention will, as already mentioned, become apparent when it is time to port the software system to different boards.

External Headers and Projects that Use Them

This section starts by describing how to create projects for external headers on the assumption that you follow the convention of having projects of the same type referencing their respective container projects, in our example, **headers_playpen**. The discussion continues with an outline of the steps you need to apply to the projects that use these header projects.

Creating Projects for External Headers

Headers, or any other resources that are external to your workspace, might be a problem if you do not have write permission. If you do not have write permission, proceed as described under *14.2 Resource Locations*, p.172.

If you have write permission, and it is up to you to create projects for external headers, you would create User-Defined projects for these. These projects, like their container project, **headers_playpen**, will have build support disabled.

To create projects for the external headers:

1. In the Project Navigator, right-click **headers_playpen** and select **New > User-Defined Project**.
2. On the first wizard page, give the project a name (**headers_1** in the example), clear the **Default** check box and browse to the root directory that contains the files you need. Click **Next**.

   On the **Project Structure** page, select **Link to project headers_playpen**. If you do not see a check box, or if the label is different, you did not select **headers_playpen** in step 1, above. In this case you can manually move the project when you are finished. Click **Next** twice.

3. In the wizard's **Build Support** page, select **Disabled**, click **Finish**.

Generating Include Search Paths for Projects

Once you have created the header project(s), others can import them (see *14.2 Resource Locations*, p.172). Whether you create the header projects yourself, or whether you import them, the include paths of the projects that use the headers have to be updated. If you are able to import the header projects, the chances are that you will also be able to import (or use your version control system to synchronize) the projects that use the headers.

On the other hand, if you are the one who creates the headers project(s), you will probably also be the one who updates the projects that use them and then makes these available to others. In this case, or if you create a new project that uses the headers project from the start, you will generally proceed as follows.

Once your workspace knows the headers (because there is a project for them), include search paths can be generated.

For each topmost project that uses the headers proceed as follows:

In the Project Navigator, select the project that uses the headers and choose **Project > Generate Include Search Paths**.

In the wizard that appears you can configure and generate include search paths for the project, its subprojects and folders, as well as for multiple build specs.

Note that in the **Project Properties** dialog, **Build Properties** node, **Build Paths** tab, and the **Generate** button (for include paths) invoke a similar wizard. This wizard, only lets you configure include paths for one build spec at a time.

Testing and Debugging

A simulator connection should be sufficient for initial testing and debugging of your applications. Please refer to *21. New VxWorks Simulator Connections* for information about simulator connections, and to *24. Debugging Projects* for information on debugging.

14.4.4 Finalization

Once things are working on the simulator, and your board and hardware connections are up and running, it is time to port the software system from the simulator to the board(s).

The steps below, especially step 2, where you create four new container (sub) projects might initially seem tedious. However, you cannot just copy the existing ones because as you remember, no physical copies are created, only references (that look like copies) are created.

Creating four empty container projects per architecture does not take long, and you only do it once. After that, the advantages include:

- Your projects are clearly and systematically organized.
- You never need to worry about changing build specs for individual projects.
- You can build your whole workspace (all the boards you support) at one time, again without manipulating the build specs.
- Any resource modifications, adding, removing, editing, at source project level will be reflected in all the project structures (=boards) simultaneously, regardless of where you make the modification since these are references, not copies.

Repeat the following steps for each board you will be supporting.

Step 1: Create VxWorks Image project and VxWorks ROMFS File System projects.

1. First, create a VxWorks Image project using the BSP appropriate to your board, see *5. Creating VxWorks Image Projects*.

   This will be a top-level project. If you follow the naming conventions used in this chapter, the project might be named something like **playpen_ppc**.

2. Then, if you are using Real-time Process projects and/or Shared Library projects, you will also need to create a VxWorks ROMFS File System project, see *7. Creating VxWorks ROMFS File System Projects*.

   This will be a subproject of the VxWorks Image project (**playpen_ppc**). The file system will be linked with the VxWorks system image created from the VxWorks Image project, and will hold the binary and data files of the system's run-time components. These are associated with the file system in Step 3 below.

   When you build the VxWorks Image project, the VxWorks ROMFS File System subproject and the other associated subprojects will be compiled to binaries and linked to the kernel. If you update files in the file system, rebuilding it creates a new file system image, which is then re-linked to the kernel.

Step 2: Create container subprojects for each project type (except headers).

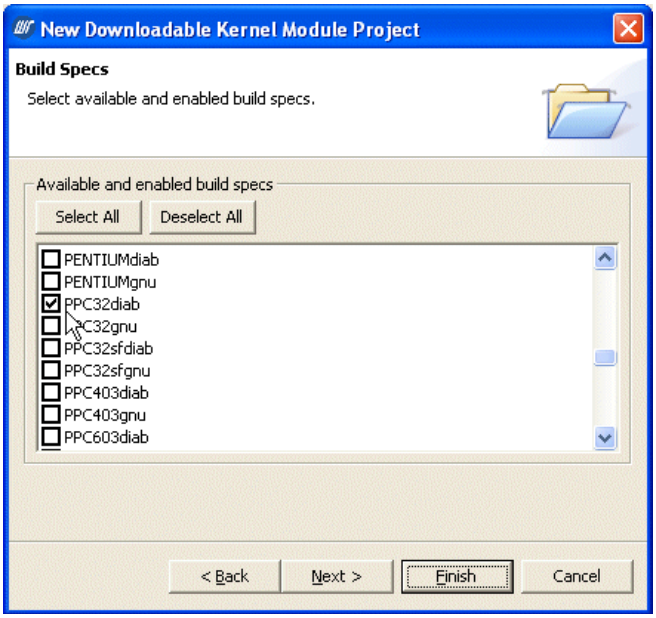
Essentially, you repeat the procedure outlined under *Step 2:Create container projects for each project type and for external headers.*, p.178, *except* that:

- You do not need to create another project for the headers as they do not use a build spec.
- Instead of appending the suffix **_sim** to the project names, you would, in our example, append **_ppc**.
- You have to set the build spec for each container (except the one for User-Defined projects, which cannot have pre-defined build specs) because the wizard default (simulator) will no longer apply.

Step-by-step, the procedure is as follows:

1. To create a new container Downloadable Kernel Module project:
   a. Right-click in the Project Navigator and select **New > VxWorks Downloadable Kernel Module Project**.
   b. In the wizard that appears, in the **Project name** field, enter: **DKMs_playpen_ppc** and click **Next** until you reach the wizard's **Build Specs** page.
   c. In the **Build Specs** page, select **Deselect All**, then select the check box next to the appropriate build spec (only one) from the list, for example, **PPC32diab** and click **Finish**.

Figure 14-7 **Select the Build Spec**



2. To create a new container Real-time Process project, right-click in the Project Navigator and select **New > VxWorks Real Time Process Project**.

   a. In the wizard that appears, in the **Project name** field, enter: **RTPs_playpen_ppc** and click **Next** until you reach the wizard's **Build Specs** page.

   b. In the **Build Specs** page, select **Deselect All**, then select the check box next to the appropriate build spec (only one) from the list, for example, **PPC32diab_RTP** and click **Finish**.

3. To create a new container Shared Library project, right-click in the Project Navigator and select **New > VxWorks Shared Library Project**.

   a. In the wizard that appears, in the **Project name** field, enter: **LIBs_playpen_ppc** and click **Next** until you reach the wizard's **Build Specs** page.

   b. In the **Build Specs** page, select **Deselect All**, then select the check box next to the appropriate build spec (only one) from the list, for example, **PPCdiab and** click **Finish**.

4. To create a new container User-Defined project, right-click in the Project Navigator and select **New > User Defined Project**.

   a. In the wizard that appears, in the **Project name** field, enter: **UDPs_playpen_ppc** and click **Finish**.

      By definition, there can be no predefined build specs for User-Defined projects. Workbench does not manage the build; it is up to you to know what needs to be done with them to complete the porting.

Step 3: Drop all new container projects onto the VxWorks ROMFS File System project.

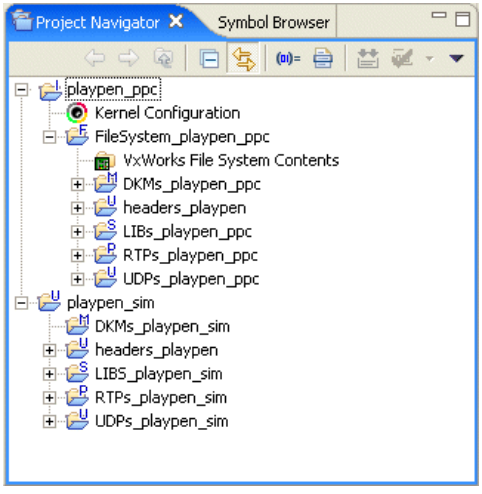
The VxWorks ROMFS File System project is a subproject of the VxWorks Image project (see Step 1). The new containers you have just created, as well as the **headers_playpen** project, should, in turn, be subprojects of this VxWorks ROMFS File System project.

- Select all the container projects you have just created and drop them onto the **FileSystem_playpen_ppc** project you created under Step 1.
- Select the **headers_playpen** subproject under **playpen_sim** and while holding down **CTRL,** drag-and-drop it onto the **FileSystem_playpen_ppc** project. It should now appear under both **playpen_sim** and **FileSystem_playpen_ppc**.

The infrastructure for the new board is now complete (see Figure 14-8).

Next, you have to create references to the source code projects.

Figure 14-8 **Project Organization for Two Boards**



Step 4: Referencing source code subprojects.

Insert references from the source code subprojects from each per-type container subproject under **playpen_sim** to the corresponding container under **playpen_ppc**.

That is, while holding down **CTRL,** drag-and-drop it to create the references from all source code subprojects under:

- **DKMs_playpen_sim** to **DKMs_playpen_ppc**
- **LIBs_playpen_sim** to **LIBs_playpen_ppc**
- **RTPs_playpen_sim** to **RTPs_playpen_ppc**
- **UDPs_playpen_sim** to **UDPs_playpen_ppc**

Step 5: Configure the VxWorks Image project and VxWorks ROMFS File System projects.

You will need to configure the VxWorks Image project (add initialization routines and configure components) and the VxWorks ROMFS File System project.

For more information on this subject, see *5. Creating VxWorks Image Projects*, *7. Creating VxWorks ROMFS File System Projects*, and the *VxWorks Kernel Programmer's Guide*.

PART III

Development

15 Navigating and Editing 195

16 Building Projects 205

17 Building: Use Cases 221

18 RTPs and Shared Libraries from Host to Target 241

15 Navigating and Editing

15.1 Introduction 195
15.2 Wind River Workbench Context Navigation 196
15.3 The Editor 198
15.4 Search and Replace 201
15.5 Static Analysis 201

15.1 Introduction

Workbench navigation views allow seamless cross-file navigation based on symbol information. For example, if you know the name of a function, you can navigate to that function without worrying about which file it is in. You can do this either from an editing context, or starting from the Symbol Browser. On the other hand, if you prefer navigating within and between files, you can use the File Navigator. For more information about these views, open them then press the help key for your host.

Static analysis is the parsing and analysis of source code symbol information. This information is used to provide code editing assistance features such as multi-language syntax highlighting, code completion, parameter hints, definition/declaration navigation for files within your projects.

Apart from the things you see directly in the Editor, static analysis also provides the data for code comprehension and navigation features such as include browsing, call trees, as well as resolving includes to provide the compiler with include search paths.



NOTE: Syntax highlighting is provided for filesystem files that you open in the Editor, but no other static analysis features are available for files that are outside your projects.

15.2 Wind River Workbench Context Navigation

Various filters are available on each tool's local toolbar. Hover the mouse over the buttons to see a tooltip describing what these buttons do. At the top-right, a pull-down menu provides additional filters, including *working sets* (if you have defined any). An active working set is marked by a bullet next to its name in the pull-down menu.

Generally, you will want to navigate to symbols, or analyze symbol-related information, from an Editor context. The entry points are:

- The right-click context menu of a symbol
- Keyboard shortcuts that act on the current selection in the Editor:

  F3 — Jump between associated code, for example, between definition/declaration or function definition/call. There is no navigation from workspace files to external files, i.e. files outside your projects.

  F4 — Open the type hierarchy of the current selection (for details, open the view and press the help key for your host).

  CTRL+ALT+H — Open the call tree of the current selection (for details, open the call tree and press the help key for your host).

  CTRL+I — Open the include browser to view the includes of the current selection (for details, open the browser and press the help key for your host).

- Keyboard shortcuts that open dialogs from which you can access symbols in any of your projects:

  SHIFT+F3 — Display the **Open Symbol** dialog.

SHIFT+F4 — Display the **Open Type Hierarchy** dialog.

ALT+SHIFT+H — Display the **Open Call Tree** dialog.

CTRL+SHIFT+R — Displays the **Open Resource** dialog.

These options are also available from the **Navigate** toolbar menu.

15.2.1 The Symbol Browser

By default, the Symbol Browser is a tab in the left pane of the main window, together with the Project Navigator.

Use the Symbol Browser for global navigation. Because the Symbol Browser presents a flat list of all the symbols in all the open projects in your workspace, you might want to constrain the list by using *Working Sets*. You can configure and select working sets using the Project Navigator's local pull-down menu.

In addition, very large symbol loads can cause delays of up to several minutes while Workbench loads the symbols. Loading smaller batches of symbols can decrease this delay. Specify the size of the symbol batch using the Preferences dialog. For more information, click in the Debug view and press the help key for your host.

Text Filtering

The **Name Filter** field at the top of the view provides match-as-you-type filtering. The field also supports wild cards: type a question mark (**?**) to match any single letter; type an asterisk (*****) to match any number of arbitrary letters. Selecting **Hide Matching** next to the **Name Filter** field inverts the filter you entered in the field, so you see only those entries that do not match your search criteria.

For a guide to the icons in the Symbol Browser, open the browser and press the help key for your host.

15.2.2 The Outline View

The Outline view is to the right of the currently active Editor, and shows symbols in the currently active file.

Use the Outline view to sort, filter, and navigate the symbols in the context of the file in the currently active Editor, as well as to navigate out of the current file context by following call and reference relationships.

For a guide to the icons in the Outline view, open the view and press the help key for your host.

15.2.3 The File Navigator

If you have never used the File Navigator, you can open it by choosing **Window > Show View > Other**. In the dialog that opens, select **Wind River Workbench > File Navigator** and click **OK**. After the first time you open the File Navigator, a shortcut appears directly under the **Window > Show View** menu. By default, the File Navigator appears as a tab at the left of the Wind River Workbench window, along with the Project Navigator and the Symbol Browser.

The File Navigator presents a flat list of all the files in the open projects in your workspace, so you can constrain the list by using *Working Sets*. You can configure and select working sets using the File Navigator's local pull-down menu.

The left column of the File Navigator shows the file name, and is active; double-clicking on a file name opens the file in the Editor, and right-clicking on a file allows you to compile the file and build the project, among other tasks. The right column displays the project path location of the file.

The **File Filter** field at the top of the view works in the same way as the **Name Filter** field in the Symbol Browser, see *15.2.1 The Symbol Browser*, p.197.

15.3 The Editor

The Editor is your primary view for editing and debugging source code. The Editor is language-aware, can parse C, C++, Ada, and Assembler files. Many Editor features are configurable in the Preferences (for details, click in the editor and press the help key for your host).



NOTE: You can specify that the Workbench editor emulate the **vi** or **emacs** editors by clicking the appropriate icon on the title bar (*vi* or respectively). Refer to additional editor preferences in **Window > Preferences > General > Editor** and additional online information at **http://help.eclipse.org**.

15.3.1 Code Templates

The Editor uses templates to extend code assist (shortcut **CTRL+SPACE**) by inserting recurring patterns of text.

In the case of source code, common patterns are **for** loops, **if** statements and comment blocks. Those patterns can be parameterized with variable placeholders that are resolved and substituted when the template is inserted into the text. Unresolved variables can be link-edited after inserting the template, which means that the first unresolved variable is selected, and all occurrences of this variable are edited simultaneously when you enter the correct text.

An example template might look like the following:

```
for (int ${var} = 0; ${var} < ${max}; ++${var}) {
    ${cursor}
}
```


Provided Templates

Workbench provides the following templates. Auto-insert is turned on by default.

Name	Description
author	author name
catch	catch block
class	class declaration
comment	default multiline comment
do	do while statement
else	else block
elseif	else if block
for	for loop
for	for loop with temporary variable

Name	Description
if	if statement
ifelse	if else statement
main	main method
namespace	namespace declaration
new	create new object
stderr	print to standard error
stdout	print to standard output
switch	switch case statement
try	try catch block
using	using a namespace

Many template options are configurable in the Preferences (for details, click in the editor and press the help key for your host).

15.3.2 Configuring a Custom Editor

Workbench has a single global mapping between file types and associated editors. This mapping dictates which editor will be opened when you double-click a file in the Project Navigator, or when the debugger stops in a given file.

Configuring the custom editor through file associations will cause the correct editor to be opened, and the instruction pointer to be painted in the editor gutter. To view and modify the mappings, go to **Window > Preferences > General > Editors > File Associations**.



NOTE: Some debugger features require additional configuration; for details, see *24.5.4 Configuring Debug Settings for a Custom Editor*, p.324.

15.4 Search and Replace

The Workbench search tool is a fast, index-based global text search/replace tool. The scope of a search can be anything from a single file to all open projects in the workspace. You can query for normal text strings, or regular expressions. Matches can be filtered according to location context (for example, show only matches occurring in comments). Text can be globally or individually replaced, and restored if necessary. You can create working sets from matched files, and you can save and reload existing queries.

15.4.1 Initiating Text Retrieval

Text retrieval is context sensitive to text selected in the Editor. If no text is selected in the Editor, an empty search dialog opens. If text is selected in the Editor, the retrieval is immediately initiated according to the criteria currently defined in the dialog.

To open the search dialog, or to initiate a context sensitive search, use:

- the keyboard shortcut **CTRL+2**.
- from the global **Search** menu, choose one of the scoping options.

For more information, open the search dialog and press the help key for your host.

15.5 Static Analysis

Editing, navigating, and code comprehension rely on static analysis parsing of source code.

You can enable static analysis in two ways: automatically, by leaving **Enable Static Analysis** selected when you create your project, or manually, by right-clicking your project in the Project Navigator and selecting **Static Analysis > Enable**.



NOTE: If this is your first use of static analysis, you may need to select **Static Analysis > Activate Plug-in** before you can access other static analysis features.

For information about global and project-specific preferences, open the preferences dialog and press the help key for your host.

15.5.1 Sharing Static Analysis Data with a Team

Static analysis of a large project can take quite a bit of time, so once you have parsed the source code of your project, you can share the generated data with your team members using your group's source control tool.

To share generated data with your team:

1. In the Project Navigator, right-click a project then select **Static Analysis > Export Shared Team Data**[1]. The **Team Data Export Options** dialog appears, where you can set export options and specify how to resolve file version differences in other workspaces (for details about this dialog, open it and press the help key for your host). Click **Finish**. Workbench exports the data to the file system location you specified.
2. Using your team's source control tool, make the generated data available to other team members (for example, by checking it into ClearCase). After that, when the project is imported into another workspace, Workbench will use the shared data instead of parsing the project.
3. Changes to the source-code are not propagated to the shared data automatically, they are stored local to the workspace. You must export the data again to make these changes available to team-members.
4. Once you have made local changes to a project in your workspace, Workbench uses that local data in preference to the shared data. To abandon your local changes and go back to using shared data, right-click a project and select **Import Shared Team Data**. Workbench launches a wizard that removes your local data in favor of the shared data.

Comparing Local Data with Shared Team Data

The tricky part about working with shared team data is to figure out which of the resources in your workspace have been changed relative to the shared data. In other words, you need a technique to compare the version of the file in your workspace with the one used to generate the shared data. Workbench comes with several mechanisms to do that:

- **Compare Timestamps**: In some setups (e.g. ClearCase dynamic views) a file of a certain version will have the same timestamp no matter in which workspace it appears. This makes it easy to check the version of a file against the shared

1. If this is your first use of Static Analysis, select **Static Analysis > Activate Plug-in**, then select **Static Analysis > Export Shared Team Data**.

data. This technique is preferable because it is fast, but many setups do not allow it (such as CVS and ClearCase snapshot views).

- **Use team data for all read-only files**: In other setups, all files in a workspace that are in sync with the repository of the source control tool are read-only. As a heuristic, we can use the shared data for all files that are read-only. The fact that a file of the workspace is in sync with the repository does not actually guarantee that the file version is the same as the one used to generate the shared data. If you update the shared data regularly though, the heuristic will be good.
- **Use checksums as a fallback**: If accessing the source code is reasonably fast and doesn't cause too much drag on the system (unlike in a ClearCase dynamic view) computing and comparing checksums for each file is a reasonable approach.

When you export shared data, you can specify in the dialog which of the techniques described above can be used by team members using the shared data. **Compare time stamps** and **Use team data for all files** will always be available to them in the import wizard, but you can disable **Use team data for all files** and **Compare checksums** by unchecking the corresponding checkboxes in the export wizard.

When you import a project with shared data into your workspace, Workbench will choose the best available comparison method.

Team-Shared Exclusion Filter

The project property page, accessible by right-clicking a project and selecting **Static Analysis > Edit Exclusion Filter** now allows you to share filters with your team.

The filters are organized into a tree with two root nodes: one for shared filters and one for workspace private ones. You can convert shared filters into workspace-private ones if you would like to edit them, or you can share a private filter with your team if you find it particularly useful. For details about exclusion filters, open the dialog and press the help key for your host.

16 Building Projects

16.1 Introduction 205
16.2 Configuring Workbench Managed Builds 208
16.3 Configuring User-Defined Builds 214
16.4 Accessing Build Properties 214
16.5 Build Specs 216
16.6 Makefiles 217

16.1 Introduction

The process of building in Workbench starts during project creation, when you select a build type for your projects, folders, and build targets. Individual build settings can be changed later, and in some cases you can switch from a managed build to a user-defined or disabled build, but if you want Workbench to manage your builds, you must select **Managed Build** in the New Project wizard.

Workbench offers several levels of build support:

Managed Build
Workbench provides two types of managed build support—Standard and Flexible—for all project types except VxWorks Image, VxWorks Boot Loader, VxWorks ROMFS File System, and User-Defined projects.

Workbench provides default build settings (that you can change as necessary), creates makefiles, and controls all phases of the build.

There are advantages to each type of managed build, depending on many things including how much control you need over your build output and what your source tree looks like.

Table 16-1 shows a comparison of standard and flexible managed build features.

User-Defined build

With **User-Defined** builds, you are responsible for setting up and maintaining your own build system and Makefiles, but Workbench does provide minimal build support.

- It allows you to configure the build command used to launch your build utility, so you can start builds from the Workbench GUI.
- You can create build targets in the Project Navigator that reflect rules in your makefiles, so you can select and build any of your make rules directly from the Project Navigator.
- Workbench displays build output in the Build Console.

Disabled build

If you select Disabled build for a project or folder, Workbench provides no build support at all. This is useful for projects or folders that contain, for example, only header or documentation files that do not need to be built.

Disabling the build for such folders or projects improves performance both during makefile generation as well as during the build run itself.



NOTE: You cannot change from a lower level of build support to a managed build once the project is created. If you later want Workbench to manage your build, create a new project with the desired type of managed build support, either on top of the existing sources, or import your sources into it.

Table 16-1 **Comparison of Standard and Flexible Managed Build Features**

Standard Managed Build	Flexible Managed Build
Build structure parallels the file system structure.	Build structure can be defined independently from the file system structure.
Build order is determined by the project and folder hierarchies as displayed in the Project Navigator.	Build order is flexible, and you can customize settings per build tool and build target.
Project contains folders and files. Project metadata and build information is stored in the source code location.	Project contains folders, files, and information about how the build targets of the project are built (stored in the **.wrproject** file). No information is stored in **.wrfolder** files in the source code location.
Project can contain multiple build targets, but the build options are the same regardless of the build target the file is built into.	Project can contain multiple build targets. You can add the same file to multiple build targets, and set specific options depending on which build target the file is built into.
Build target can contain only files within the project.	Build targets can contain any file, folder, project, or other build target in the workspace, including linked resources. Virtual folders allow you to group objects from different sources and apply build settings to them.
Build target contains all contents of included folders.	Folders and files can be excluded from the build target using regular expressions.
Workbench creates one Makefile per folder with all build specs. Makefiles are based on data you enter at project creation time, or later in the Build Properties dialog.	Workbench creates one Makefile per build spec for the whole project.
Leveling chain is **project > folder > file**.	Leveling chain is **project > build target > folder > file**.

Table 16-1 **Comparison of Standard and Flexible Managed Build Features** (cont'd)

Standard Managed Build	Flexible Managed Build
Workbench generates include search paths for header files that are visible in the workspace.	Same
Build output is displayed in the Build Console.	Same

16.2 Configuring Workbench Managed Builds

The process of configuring Workbench managed builds differs significantly depending on whether you selected a standard or a flexible managed build.

16.2.1 Configuring Standard Managed Builds

Standard managed builds have not changed from previous versions of Workbench. When you select **Standard** in the New Project wizard, your project is created and contains a preliminary build target in addition to the usual project files.

To create the build target, right-click your project in the Project Navigator and select **Build Project**.

16.2.2 Configuring Flexible Managed Builds

When you select **Flexible**, your projects are created in the same way and also contain the usual project files, but you must create your build targets manually.

Adding Build Targets to Flexible Managed Builds

Once your project is created, you will see a **Build Targets** node inside it.

1. To add a build target to your project, right-click the **Build Targets** node and select **New Build Target**. The **New Build Target** dialog appears.
2. By default the **Build target name** and **Binary output name**[1] are the same as the project name, but if you are going to create multiple build targets you will want to type in more descriptive names. Choose the appropriate **Build tool** for your project, then click **Next**. The **Edit Content** dialog appears.
3. To display files, folders, and other build targets from outside your current project, select **Show all projects**. If you have created a **Working Set**, you can restrict the display by selecting it from the pull-down list.
4. You can add contents to your build target in several ways:
   a. You can select specific files, folders, projects, or other build targets in the left column and click **Add**. What you can add depends on the build tool you use; for example, you cannot add an executable build target to another build target.

      When choosing folders or projects, they can be added "flat" or with recursive content.

      – Clicking **Add** creates a "flat" structure, meaning that Workbench adds the exact items you choose and skips any subfolders and files.

      – Clicking **Add Recursive** creates a structure that includes subfolders and files.



NOTE: Adding linked resources to a build target may cause problems within a team if the linked resources are added using an absolute path instead of a variable.

To define a path variable, select **Window > Preferences > General > Workspace > Linked Resources**, click **New**, then enter a variable name and location.

   b. You can create a virtual folder within your build target by clicking **Add Virtual Folder**, typing a descriptive name in the dialog, and clicking **OK**. Virtual folders allow you to group objects within the build target so you can apply the same build settings to them; they also provide a way to add files with the same name from different locations.

1. Your build targets must have unique names, but you can use the same **Binary output name** for each one. This allows you to deliver an output file with the same name in multiple configurations. Workbench adds a build tool-appropriate file extension to the name you type, so do not include the file extension in this field.

i. To add contents to your virtual folder, right-click it in the Project Navigator and select **Edit Content**.

ii. Select content as described in step a above, and click **Finish**.

5. To adjust the order of the build target contents, select items in the right column and click **Up**, **Down**, or **Remove**.



NOTE: Folders appear in the specified place in the list, but the files within them are added alphabetically.

6. When you have configured your build target, click **Finish**. It appears in the Project Navigator under the **Build Targets** node of your project.

Modifying Build Targets

There are several ways to modify your build target once it has been created.

Editing Content

To add additional items, adjust the order, or make any other changes to your build target, right-click it in the Project Navigator and select **Edit Content**. The **Edit Content** dialog appears, with the build target content displayed in the right column. Adjust the contents as necessary, then click **Finish**.

Renaming Build Targets and Virtual Folders

To rename your build target or virtual folder, select it in the Project Navigator, press **F2**, and type a new name.

Copying Build Targets

To copy a build target, right-click the build target and select **Copy**, then right-click the destination project's **Build Targets** node and select **Paste** (if you are pasting back into the original project, type a unique name for the new build target).

This is useful for setting up the same build targets in multiple projects with different project types (for example, a library for a native application and a downloadable kernel module will have the same contents but different flags).



NOTE: The build target and its contents are copied, but any overridden attributes are not.

Removing Content

To remove an item from the build target, right-click it in the Project Navigator and select **Remove from Build Target**, or just select it and press **Delete**.

Depending on the item you selected, the menu item may change to **Exclude from Build Target** if the item cannot be deleted (for example, recursive content cannot be deleted). Pressing **Delete** also reinstates an item by removing the exclusion.

Excluding Content

To exclude a specific item from the build target that was included recursively, right-click it in the Project Navigator and select **Exclude from Build Target**.

You can also use regular expressions to exclude groups of items.

1. To add a pattern to the excludes list, right-click a folder in the build target, then select **Properties**, then select the **Excludes** tab.
2. Click **Add File** to define a pattern to exclude specific files or file types. For example, type ***_test.c** to exclude any file named *filename*_**test.c**.

   You can include additional parts of the path to better define the file you want to exclude; for example, type **lib/standard_test.c** to exclude that specific file.
3. Click **Add Folder** to define a pattern to exclude folders within specific folders. For example, type ***/lib/*_test.c** to exclude any file located in a folder named **lib** and named *filename*_**test.c**.

Dragging and Dropping Content

To modify build target contents without opening the **Specify Content** dialog, you can drag and drop items in the Project Navigator.

- You can drop resources onto build target nodes or virtual folders to add them to a build. Workbench checks the validity of the action and reports errors if the move is not allowed. Workbench also asks you if the resource should be added “flat” or recursively.
- You can reorder build target contents by dragging and dropping an item on the same level.
- You can drop a build target node onto other build targets to add the first build target as a reference (for example, dropping a library onto an executable, or dropping an executable onto an executable). Workbench checks to make sure it is a valid operation before allowing you to complete the action.



NOTE: If your build target contains projects or folders, any files you add to them later will be automatically added to the build target as well. So you do not need to manually update your build target in this case.

Leveling Attributes

The leveling of build-specific settings in flexible managed builds is significantly different from the leveling of standard managed build projects. The leveling chain for flexible managed build projects is shown below.

Project > Target > Folder > File
Project > Target > Folder > Subfolder > File
Project > Target > Virtual folder > File
Project > Target > Virtual folder > Folder >
Project > Target > File

The folder level here is related to folders underneath a build target, as described in *Adding Build Targets to Flexible Managed Builds*, p.208.

The information that can be leveled is equivalent to the current implementation of standard managed build projects, plus additional information so that you can enable files to be built on a per build-spec basis (standard managed build allows this only on folder level).

You can now configure the build target with specific settings for all build tools on a build target level (for example, you can set compiler options for the source files related to that build target).

Target Passing and Project Structure

Passing build targets is only supported when passing to VxWorks kernel image superprojects; it is not possible to pass flexible managed build targets to standard managed build or other flexible managed build superprojects.

To reference other flexible managed build targets, add them to the contents of a build target as described in *Adding Build Targets to Flexible Managed Builds*, p.208.

Understanding Flexible Managed Build Output

The output of a flexible managed build is significantly different from the output of a standard managed build.

Workbench does not create build redirection directories for each folder, as the objects might be built differently when building them for specific targets. Instead, Workbench creates a build-specific redirection directory, which you can configure on the **Build Properties** > **Build Paths** tab, underneath the project root directory.

The redirection directory contains a directory for each build target; inside those are directories named **Debug** or **NonDebug** depending on the debug mode you chose for the build. Workbench generates the output files according to the structure you defined in the build target, storing them in the debug mode directory.

In general, the build output is structured like this:

Project directory
Project dir/build specific redirection dir
Project dir/build specific redirection dir/target dir
Project dir/build specific redirection dir/target dir/debug mode dir
Project dir/build specific redirection dir/target dir/debug mode dir/binary output file of the build target

All objects belonging to the build target are stored in an additional **Objects** subfolder:

Project dir/build specific redirection dir/target dir/debug mode dir/Objects/structure of object files

Example Build Target and Build Output Structure

To understand how the build target structure influences the build output, below is an example of a project source tree.

```
proj1/
proj1/a.c
proj1/b.c
proj1/folder1/c.c
proj1/folder1/d.c
```

Target1 contains these two items:

```
a.c
folder1/*.c
```

Target2 contains these two items:

```
b.c
d.c
```

Configuring the project to use **spec1** as the active build spec, naming the redirection directory **spec1**, and turning debug-mode **on** produces the output structure seen below.

```
proj1/spec1/Target1/Debug/Target1.out
proj1/spec1/Target1/Debug/Objects/a.o
proj1/spec1/Target1/Debug/Objects/folder1/c.o
proj1/spec1/Target1/Debug/Objects/folder1/d.o

proj1/spec1/Target2/Debug/Target2.out
proj1/spec1/Target2/Debug/Objects/b.o
proj1/spec1/Target2/Debug/Objects/d.o
```

16.3 Configuring User-Defined Builds

When you create a User-Defined project, you can configure the build command, make rules, build target name, and build tool (for more information, see *11. Creating User-Defined Projects*). To create the build target, right-click your project in the Project Navigator and select **Build Project**.

To update the build settings, right-click your project in the Project Navigator and select **Properties**, then select **Build Properties**.

For more information about the settings described on the build properties tabs, open the build properties dialog and press the help key for your host.

16.4 Accessing Build Properties

There are two ways to set build properties: in the Workbench preferences, to be automatically applied to all new projects of a specific type, and manually, on an individual project, folder, or file basis. The properties displayed will differ depending on the type of node and the type of project you selected, as well as the type of build associated with the project.

For details, open the build properties dialog and press the help key for your host.

16.4.1 Workbench Global Build Properties

To access global build properties, select **Window > Preferences** and choose the **Build Properties** node.

This node allows you to select a project type, then set default build properties to be applied to all new projects of that type.

16.4.2 Project-specific Build Properties

To access build properties from the Project Navigator, right-click a project and select **Properties**. In the Properties dialog, select the **Build Properties** node.

The project-specific Build Properties node has tabs that are practically identical to the ones in the Workbench preferences, but these settings apply to an existing project that is selected in the Project Navigator.



NOTE: Build properties for VxWorks Image Projects (VIPs) can differ substantially from the general properties of other project types.

For details, open the build properties dialog and press the help key for your host, and consult the *VxWorks Kernel Programmer's Guide* for general information about VIPs.

16.4.3 Folder, File, and Build Target Properties

Folders, files, and build-targets inherit (reference) project build properties where these are appropriate and applicable. However, these properties can be overridden at the folder/file level. Inherited properties are displayed in blue typeface, overridden properties are displayed in black typeface.

Overridden settings are maintained in the **.wrproject** file (and also in **.wrfolder** files in standard managed builds). These files should therefore also be version controlled. Note that you can revert to the inherited settings by clicking the eraser icon next to a field.

16.4.4 Multiple Target Operating Systems and Versions

If you installed Workbench for multiple target operating systems and/or versions, you can set a default target operating system/version for new projects in the Workbench Preferences, under **General > Target Operating Systems**.

For existing projects, you can verify the target operating system (version) by right-clicking the project in the Project Navigator, then selecting **Properties**, then **Project Info**.



NOTE: In most cases, it will *not* be possible to successfully migrate a project from one target operating system or version to another simply by switching the selected **Target Operating System and Version**.

In the Project Navigator (and elsewhere), the target operating system and version are displayed next to the project name by default. You can toggle the display of this information in the Preferences, **General > Appearance > Label Decorations**, using the **Project Target Operating Systems** checkbox.

If you have multiple versions of the same operating system installed, the New Project wizard allows you to select which version to use when creating a new project.

16.5 Build Specs

A build spec is a group of build-related settings that lets you build the same project for different target architectures and/or different tool chains by simply switching from one build spec to another. Note that the architecture/tool chain associations are preconfigured examples; you can also create your own build specs (usually from copies of existing ones, using the **Copy** button) for any constellation of the many configurable properties that make up a spec (see also *17.8 A Build Spec for New Compilers and Other Tools*, p.233).

It is important to remember that the build spec used when you build must match the target board; that is, it must match the VxWorks Image project that the application project will be associated with.

16.6 Makefiles

The build system uses the build property settings to generate a self-contained makefile named **Makefile**.

- For standard managed builds, a Makefile is generated in each project and folder at each build run. This allows you to build individual folders, projects, and subtrees in a project structure.
- For flexible managed builds, only one Makefile is created per build spec.

By default makefiles are stored in project directories; if you specified an absolute **Redirection Root Directory** (for details, open the build paths tab and press the help key for your host), they are stored there, in subdirectories that match the project directory names.

The generated makefile is based on a template makefile named **.wrmakefile** that is copied over at project creation time. If you want to use custom make rules, enter these in **.wrmakefile**, *not* in **Makefile**, because this is regenerated for each build. The template makefile, **.wrmakefile**, references the generated macros in the placeholder **%IDE_GENERATED%**, so you can add custom rules either before or after this placeholder. You can also add ***.makefile** files to the project directory.

For other ways of setting custom rules, see *17.7 User-Defined Build-Targets in the Project Navigator*, p.232.



NOTE: If you configure your project for a remote build, the generated Makefile contains paths for remote locations rather than local ones. For more information about remote builds, see *17.9 Developing on Remote Hosts*, p.235.

16.6.1 Derived File Build Support

The Yacc Example

Workbench provides a sample project, **yacc_example**, that includes a makefile extension showing how you can implement derived file build support. It is based on **yacc** (Yet Another Compiler Compiler) which is not contained in the Workbench or VxWorks installation. To actually do a build of the example you need to have **yacc** or a compatible tool (like GNU's **bison**) installed on your system, and you should have extensive knowledge about make.

The makefile, **yacc.makefile**, demonstrates how a yacc compiler can be integrated with the managed build and contains information on how this works.

1. Create the example project by selecting **New > Project > Example > Native Sample Project > Yacc Demonstration Program**.
2. Right-click the **yacc_example** project folder, then select **New > Build Target**. The New Build Target dialog appears.
3. In the **Build target name** field, type **pre_build**.
4. From the **Build tool** drop-down list, select **(User-defined)**, then click **Finish** to create the build target.
5. In the Project Navigator, right-click **pre_build** and select **Build Target**. This will use the makefile extension **yacc.makefile** to compile the yacc source file to the corresponding C and header files. The build output appears in the Build Console.



NOTE: It is necessary to execute this build step prior to the project build, because the files generated by **yacc** will not be used by the managed build otherwise. This is due to the fact that the managed build generates the corresponding makefile before the build is started and all files that are part of the project at this time are taken into account.

6. When the build is finished, right-click the **yacc_example** folder and select **Build Project**.

Additional information on how you can extend the managed build is located in **yacc.makefile**. It makes use of the extensions provided in the makefile template .wrmakefile, which can also be adapted to specific needs.

General Approach

To implement derived file support for your own project, create a project-specific makefile called *name_of_your_choice***.makefile**. This file will automatically be used by the managed build and its make-rules will be executed on builds.

It is possible to include multiple ***.makefile** files in the project, but they are included in alphabetical order. So if multiple build steps must be done in a specific order, it is suggested that you use one ***.makefile** and specify the order of the tools to be called using appropriate make rules.

For example:

1. Execute a lex compiler.
2. Execute a yacc compiler (depending on lex output).
3. Execute a SQL C tool (depending on the yacc output).

Solution: (using the **generate_sources** make rule)

```
generate_sources :: do_lex do_yacc do_sql
do_lex:
    @...

do_yacc:
    @...

do_sql:
    @...
```

or

```
generate_sources :: $(LEX_GENERATED_SOURCES) $(YACC_GENERATED_SOURCES)
$(SQL_GENERATED_SOURCES)
```

Add appropriate rules like those shown in the file **yacc.makefile**.

17 Building: Use Cases

17.1 Introduction 221

17.2 Adding Compiler Flags 222

17.3 Building Applications for Different Boards 224

17.4 Creating Library Build-Targets for Testing and Release 225

17.5 Architecture-Specific Implementation of Functions 228

17.6 Executables that Dynamically Link to Shared Libraries 229

17.7 User-Defined Build-Targets in the Project Navigator 232

17.8 A Build Spec for New Compilers and Other Tools 233

17.9 Developing on Remote Hosts 235

17.1 Introduction

This chapter suggests some of the ways you can go about completing various build-specific tasks in Wind River Workbench.

17.2 Adding Compiler Flags

Let us assume:

1. You are working on a Real-time Process project (referred to in the following as **MyRTP**), and you are using the build spec, **SIMPENTIUMgnu_RTP**.
2. You want to suppress compiler warnings, and you are familiar with the GNU compiler (used by the given build spec) command line.
3. You later have to change the build spec to **SIMPENTIUMdiab_RTP**; that is you have to use the Wind River Compiler tools, with which you are not familiar, but you still want to suppress compiler warnings.

17.2.1 Add a Compiler Flag by Hand

According to assumption 2, above, you are familiar with the GNU compiler command line, so you just want to know *where* to enter the **-w** option.

1. In the Project Navigator, right-click on the **MyRTP** project and select **Properties**.
2. In the Properties dialog, select the **Build Properties** node.
3. In the **Build Properties** node, select the **Build Tools** tab.
4. In the **Build Tools** tab:
   - Set the **Build tool** to **C-compiler**
   - The **Active build spec** will, according to assumption 1, above, already be set to **SIMPENTIUMgnu_RTP**.
   - In the field next to the **Tool Flags** button, append a space and **-w**

     The contents of this, the **Tool Flags** field you have just modified, is expanded to the **%ToolFlags%** macro you see in the **Command** field above it. Because you entered the **-w** in the **Tool Flags** field, rather than the **Debug** or **Non Debug** mode fields, warnings will always be suppressed, rather than only in either **Debug** or **Non Debug** mode**.**

17.2.2 Add a Compiler Flag with GUI Assistance

According to assumption 3, above, you have to change to the Wind River Compiler tool chain used by the **SIMPENTIUMdiab_RTP** build spec, and you are not familiar with the new command line tool options.

Step 1: Change the Active Build Spec

1. In the Project Navigator, right-click on the **MyRTP** project, and select **Set Active Build Spec**.

   If the **SIMPENTIUMdiab_RTP** build spec is enabled, you will see it listed in the dialog that appears. In this case, all you would have to do is select **SIMPENTIUMdiab_RTP** from the list and click **OK**.

   However, we shall assume **SIMPENTIUMdiab_RTP** is not enabled, and therefore not available in the list.

2. In the Project Navigator, right-click on the **MyRTP** project, and select **Properties**.

3. In the Properties dialog, select the **Build Specs** node.

4. In the **Build Specs** node, select the **SIMPENTIUMdiab_RTP** build spec and set both the **Default build spec** and the **Active build spec** to **SIMPENTIUMdiab_RTP**.

   Leave the Properties dialog open to complete Step 2, below.

Step 2: Use the GUI to Add a Compiler Flag

1. Select the **Build Tools** tab.

2. In the **Build Tools** tab:

   – Set the **Build tool** to **C-compiler**

   – The **Active build spec** will already be set to **SIMPENTIUMdiab_RTP** (see 4 above).

   – We assumed you are unfamiliar with the Wind River compiler options so, to open the Wind River Compiler Options dialog, click the **Tool Flags** button.

– In the Wind River Compiler Options dialog, click your way down the navigation tree at the left of the dialog and take a look at the available options.

  When you get to the **Compilation > Diagnostics** node, select the check box labelled **Suppress all compiler warnings**.

  Notice that **-Xsuppress-warnings** now appears in the list of command line options at the right of the dialog.

  Click **OK**.

3. Back in the **Build Tools** node of the Properties dialog, you will see that the option you selected now appears in the field next to the **Tool Flags** button.

   The contents of this, the **Tool Flags** field, is expanded to the **%ToolFlags%** macro you see in the **Command** field above it.

17.3 Building Applications for Different Boards

Generally, but not necessarily, you would have a VxWorks Image project (VIP) for each architecture you support. If, however, you are developing applications and/or libraries only, you might not have VIPs.

If you do have VIPs, you will probably only set the build spec once for the application subprojects to match the VIP they are under. On the other hand, if you do not have VIPs, you might switch the build spec to build projects for different architectures.

The target nodes under projects in the Project Navigator display, in blue, the name of the currently active build spec.

If, for example, you want to build an application for testing on a simulator, and then build the same project to run on a real board, you would simply switch build specs as follows:

1. Right-click the project or the target node and, from the context menu, select **Set Active Build Spec**.

2. In the dialog that appears, select the build spec you want to change to and specify whether or not you want debug information.

   When you close the dialog, you will notice that the label of the target node has changed. If you selected debug mode in the dialog, the build spec name is suffixed with **_DEBUG**.

3. Build the project for the new architecture.

17.4 Creating Library Build-Targets for Testing and Release

Assume you have a library that consists of the files **source1.c**, **source2.c**, and **test.c**. The file **test.c** implements a **main( )** function and is required exclusively for testing, and is not to be included in the release version of the library.

One way to handle this is to use different targets that are built with different tools as described below.

1. Create a Real-time Process project to hold all the files mentioned above. Use this project type, because you will need to use both the **Linker** and the **Librarian** build tools later.

   In the first page of the project creation wizard, name the project, for example, **LIB** and click **Finish**. You will need to do some tweaking in the **Project's Properties** dialog anyway, so you might as well do everything there.

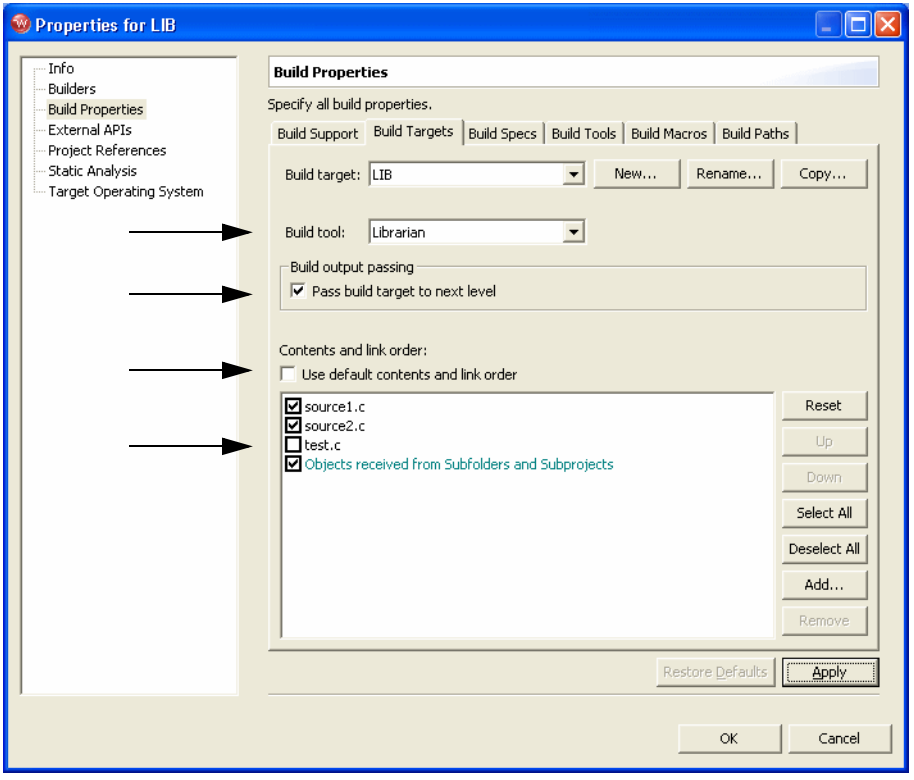
2. Right-click the newly created **LIB** project, and select **Properties**. In the **Properties** dialog, select the **Build Properties** node, then the **Build Targets** tab.

   First create a build-target for the release version of your library.

   - Change the **Build tool** to **Librarian**.
   - Select **Pass build target to next level**.
   - Clear the **Use default contents and link order** check box.
   - Clear the check box next to **test.c**.
   - Click **Apply**.

   Figure 17-1 shows the results.

Figure 17-1 **Release Version of the Library**

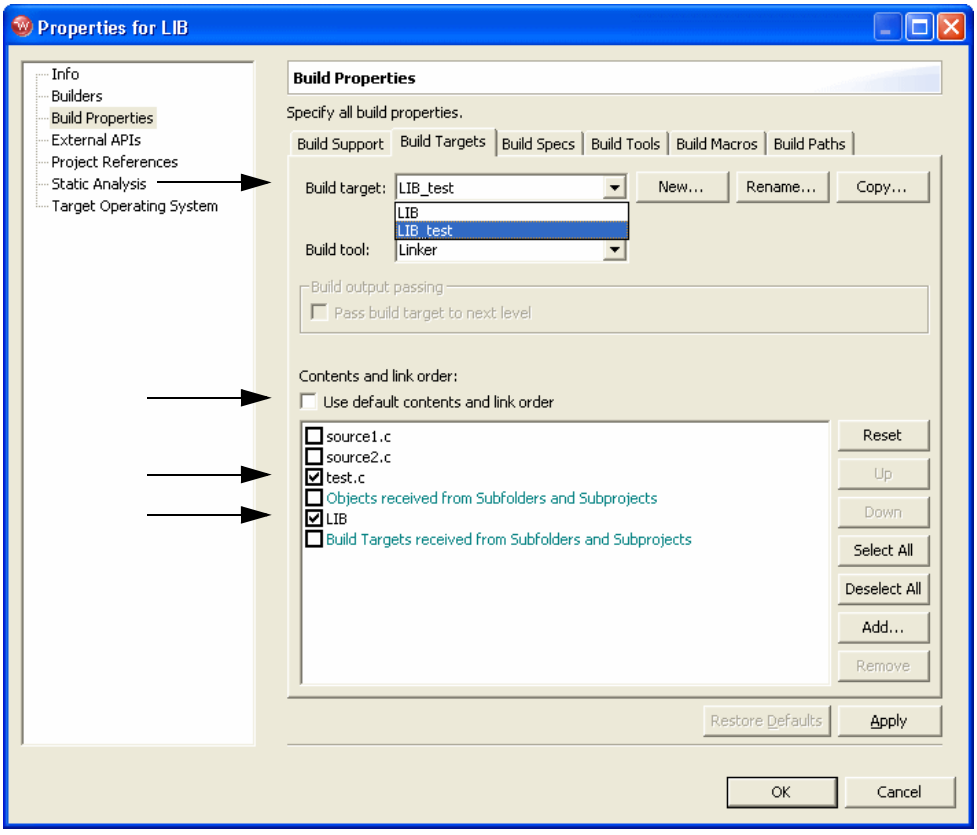


3. Next, create a target for the test version of the library.
   - Click **New** then enter, for example, **LIB_test** in the dialog that appears.

     Notice that the **Build Tool** is set to **Linker**, this is because the Linker is the default tool for Real-time Process Projects, and that the **LIB** (your previous build-target) has been added to the **Contents and link order** list.
   - Clear the **Use default contents and link order** check box.
   - In the **Contents and link order** list, select only the check boxes next to **LIB** and **test.c**; clear all other check boxes.

   Figure 17-2 shows the results.

Figure 17-2 **Test Version of the Library**



After you close the **Properties** dialog, there will be two new build-target nodes in the **LIB** project. If you build **LIB_test**, then **LIB** will automatically also be built if it is out of date.

17.5 Architecture-Specific Implementation of Functions

You can enable/disable build specs at the project as well as at the folder level. This allows architecture-specific implementation of functions within same project.

Figure 17-3 shows a simplified project tree with two subprojects, **arch 1** and **arch2**, that each use code that is specific to different target architectures. This is how projects could be set up to build a software target that requires the implementation of a function that is specific to different target boards, where only the active build spec in the topmost project has to be changed. The inner build spec relationships are outlined in Table 17-1.

Figure 17-3 **Simple Project Structure for Architecture-Specific Functions**

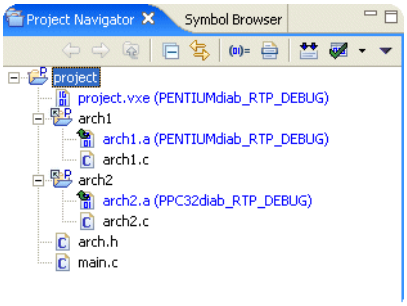


Table 17-1 **Project Content and Build Spec Configuration of the Structure in Figure 17-3**

Directories/Folders	Files	Enabled Build Specs
/project	**main.c, arch.h**	**PENTIUMdiab_RTP** and **PPC32diab_RTP**
/project/arch1	**arch1.c**	**PENTIUMdiab_RTP** only
/project/arch2	**arch2.c**	**PPC32diab_RTP** only

The function **int arch_specific (void)** is declared in **arch.h** and the file **arch1.c** implements **int arch_specific (void)** for **PENTIUM** (the only build spec enabled for the **arch1** project), while the file **arch2.c** implements **int arch_specific (void)** for **PPC32** (the only build spec enabled for the **arch2** project).

If the active build spec for **project** is set to **PENTIUMdiab_RTP**, the subproject **arch1** will be built, and its objects will be passed up to be linked into the **project**

build-target. The **arch2** subproject will not be built, and its objects will not be passed up to be linked into the **project** build target because the **PENTIUMdiab_RTP** build spec is not enabled for **arch2**.

The same applies if the **PPC32diab_RTP** is the active build spec for **project**: the **arch2** subproject will be built, but the **arch1** subproject will not.

17.6 Executables that Dynamically Link to Shared Libraries

Only executables produced from RTP projects can dynamically link to shared libraries. Note that you will need a VxWorks ROMFS File System project to hold the library binary on the target. The compiled library must be located in the host and target side directories you specify as described below.

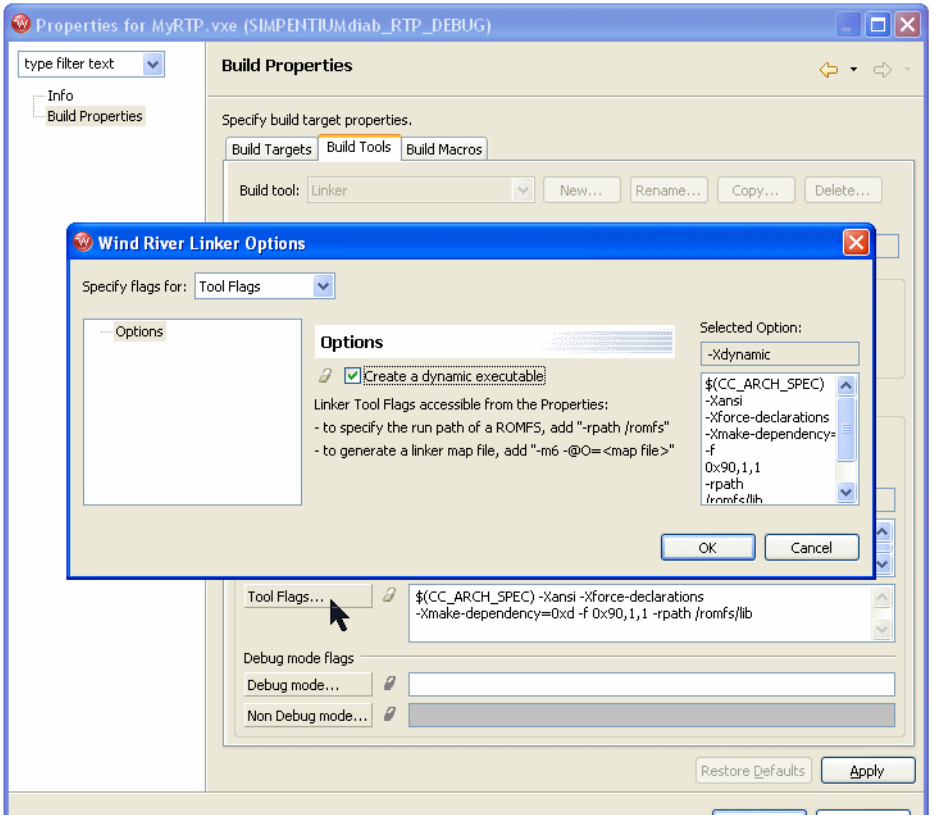
Step 1: Modify the Real-time Process build-target build properties.

1. Right-click the RTP's target node and select **Properties**.
2. In the **Properties** dialog, select the **Build Properties** node.

Step 2: Set up the Linker Build Tool for a dynamic executable and target-side run path.

1. Select the **Build Tools** tab.
2. In the field next to the **Tool Flags** button, enter the *run path* (**-rpath**) to the directory that will hold the shared library on your target, for example, **-rpath /romfs/lib** (**romfs** is the default root directory of the ROMFS created by VxWorks ROMFS File System projects).
3. Click **Tool Flags**.

Figure 17-4 **Build Options for Dynamic Executables**



4. In the **Linker Options** dialog that appears, select **Create a dynamic executable**.

   You will notice that the option, as used on the command line, appears in the **Selected Options** list on the right. After you click **OK** to close the **Linker Options** dialog, you will see the option again in the field next to the **Tool Flags** button.

Step 3: Define Build Macros for the host-side location and library binary.

1. Select the **Build Macros** tab.
2. In the list of **Build spec specific settings**, select the **LIBS** macro and click **Edit**.

   In the dialog that appears, add a space after the existing value (**-lstlstd**), followed by the basename of the shared library binary you want to link to, for example, **MySharedLibrary**:

   `-l:MySharedLibrary.so`

   When you close the dialog you should see:

   ```
   LIBATH     -lstlstd -l:MySharedLibrary.so
   ```

3. In the list of **Build spec specific settings**, select for the **LIBPATH** macro and click **Edit**.

   In the dialog that appears, enter the host-side directory location of the library binary you want to dynamically link to, for example:

   `-L../MySharedLibrary/$(OBJ_DIR)`

   Note that **$(OBJ_DIR)** expands to wherever the build output for **MySharedLibrary** is generated to. Using **$(OBJ_DIR)** is generic and therefore offers the advantage of not having to change the **LIBPATH** macro if you change build specs**.**

   Note further that the relative reference assumes the Shared Library project is located in the same workspace as the Real-time Process project.

   Click **OK** to close the project's build-target **Properties** dialog.

   The next time you build the project structure, a dynamic executable capable of run-time linking to the shared library with the file basename and the directories (host and target side) you specified above will be produced.



NOTE: If your application is *not* built as described in this section (*17.6 Executables that Dynamically Link to Shared Libraries*, p.229), you must set the **LD_LIBRARY_PATH** environment variable.

Click **Edit** beside the **Environment** field, then click **Add** in the **Edit Environment** dialog, then type **LD_LIBRARY_PATH** in the **Name** field and the full path to your shared library file (using forward slashes and excluding the filename itself) in the **Value** field. The path must be defined in terms of the file system as seen on the target.

Click **OK**. The **Edit Environment** dialog should contain the new environment variable; click **OK**.

17.7 User-Defined Build-Targets in the Project Navigator

In the Project Navigator you can create custom build-targets that reflect rules in makefiles. This is especially useful if you have User-Defined projects, which are projects where the build is not managed by Workbench. However, you might also find this feature useful in other projects as well.

17.7.1 Custom Build-Targets in User-Defined Projects

Assuming you have two rules in a makefile, **clean** and **all**, you can define a custom build-target for either or both of these rules. To do so:

1. Right-click a project or folder and select **New > Build Target**.
2. In the dialog that appears, enter the rule(s) you want to create a target for. If you want to execute multiple rules, separate each one with a space.

   In our example, enter **clean all** to have both these rules, which must exist in your makefile(s), executed when you build your new user-defined target.

   Click **Finish**. The new build-target node appears under the project or folder you selected. The node icon has a superimposed **M** to identify it as a user-defined make rule.

   To execute the rule(s), right-click the new target node and select **Build Target**.

17.7.2 Custom Build-Targets in Workbench Managed Projects

First write the make rules you need into the **.wrmakefile** file in the project directory.

1. Right-click a project or folder and select **New > Build Target**.
2. In the dialog that appears, enter the rule name(s) you created in **.wrmakefile**. If you want to execute multiple rules, separate each one with a space.

   Set the **Build tool** to **User-defined,** click **Finish**.

   The new build target node appears under the project or folder you selected. The node icon has a superimposed **M** to identify it as a user-defined rule.

   To execute the rule(s), right-click the new target node and select **Build Target**.

17.7.3 User Build Arguments

You can use the User Build Arguments view to execute any existing make rule, or overwrite any macro, or anything else that is understood by make, at every build, regardless of what is being built. The view is toggled by choosing User Build Arguments view from the drop-down menu at the top right of the Project Navigator, or by clicking the button in the Project Navigator's toolbar.

If the **User Build Arguments** check box is selected, the rule, or rules separated by a space, or macro re-definitions, and so on will override the makefile entries on the fly at every build, regardless of context.

17.8 A Build Spec for New Compilers and Other Tools

The easiest way to define a build spec for a new compiler and other associated tools (known as a *tool chain*) is to copy one of the pre-configured build specs of an existing tool chain and architecture, and modify the copy.

Step 1: Copy an Existing Build Spec.

1. Open an application project's, for example an RTP, build properties as described under *16.4 Accessing Build Properties*, p.214.

Using an application project has the advantage that these have a fuller range of generic build tools (**Assembler**, language-specific **Compiler**, **Librarian**, and **Linker**).

2. Select the **Build Specs** tab and look at the existing specs. The pre-configured build spec names follow an *ArchitectureToolChain_ProjectType* convention, for example, **PENTIUMgnu_RTP**. This spec is configured for a Pentium target board, using GNU tools to create an Real-time Process (RTP).

   In the **Build Specs** tab, select the build spec that comes closest to your needs, at least in terms of target architecture, or a tool chain that you are familiar with, and click **Copy**.

   You will be warned that build properties need to be saved before proceeding. Click **OK**, then enter a name for the copy you are creating in the next dialog and click **OK** again.

3. Still in the **Build Specs** tab, set the **Active build spec** to your newly created copy (this is initially right at the bottom of the list of **Available and enabled build specs**. Whatever you set here is also propagated to the Build Tools, Build Macros, and Build Paths tabs (for details open the build properties dialog and press the help key for your host).

   Each of these tabs has a generic section at the top with **Build spec specific settings** below. The generic section will normally be correct, which is one advantage of copying an existing spec, rather than creating a new spec from the beginning.

Step 2: Configure the Build Tool.

The build system uses generic build tools, for example, a **C-Compiler**. So if you are adding a new, unsupported C compiler, you will have to configure a build spec that understands this specific instance of the generic **C-Compiler** build tool. Using the compiler as an example, proceed as follows:

1. Select the **Build Tools** tab and set the **Build tool** drop-down list to **C-Compiler**.

   The generic settings regarding **Suffixes** and **Build output generation** should be correct, if not modify accordingly. (If you were adding a compiler for a new language, **foolanguage**, you could first create a **Copy** of a generic **C-Compiler Build tool** and name that, for example, **Foo-Compiler**, and then configure the generic settings as required.)

2. In the **Build spec specific settings** you would configure the options that are specific to your particular compiler.

- The **Active build spec** should already be set to your newly created build spec.
- The **Derived suffix** refers to the file suffix of the compiler's output.
- The **Command** is the command line call to your compiler with all the options you want to pass.

  In theory, you could simply type a hard command in this field. However, using the predefined macros of the form **%***MacroName***%** and macros (your own and/or pre-defined) that are defined on the **Macros** tab and referenced using **$(***MacroName***)** generally makes more sense, as does separating common **Tool Flags** and **Debug mode** and **Non Debug mode** flags. For more detailed information, open the build properties dialog, press the help key for your host, and see the *Build Tools* section.

3. If you are using your own and/or pre-defined using macros in the **Command** field, set these in the **Build Macros** tab.

   For more detailed information, open the build properties dialog, press the help key for your host, and see the *Build Macros* section.

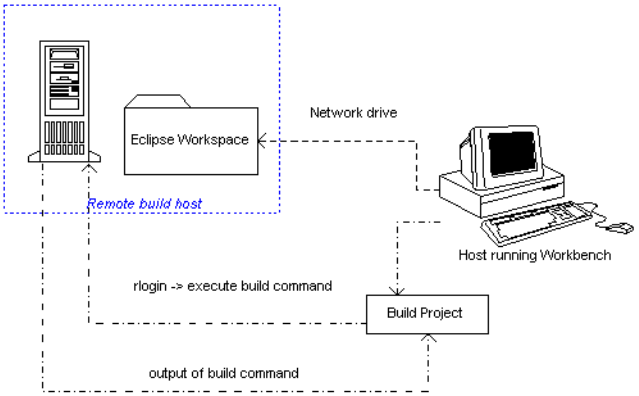
4. In the **Build Paths** tab, configure the redirection directories for build output and set the include search paths (if applicable; that is, if you are configuring a build spec for a C/C++ compiler) using the **Generate** and **Add** buttons.

   For more detailed information, open the build properties dialog, press the help key for your host, and see the *Build Paths* section.

After you have configured the build spec for the first tool in the chain, for example, the compiler, go back to the **Build Tools** tab (see Step 2, above) to configure any additional tools, such as the **Linker** or **Librarian**.

17.9 Developing on Remote Hosts

The Workbench *remote build* feature allows you to develop, build and run your applications on a local host running Workbench, using a workspace that is located on a remote host as if it were on a local disk.



In the case of a managed build (standard or flexible), Workbench generates the makefiles on the local machine running Workbench using a path mapping of the workspace root location, so that the generated makefiles will be correctly dumped for a build that is executed on the remote machine. When launching the build, a network connection (**rlogin** or **SSH**) is established to the build host, and the actual build command is executed there by using an intermediate script to allow you to set up the needed environment for the build process.

17.9.1 General Requirements

- The workspace root directory has to be accessible from both machines.
- Only Eclipse projects located underneath the workspace root can be remotely built. In other words, linked resources are not supported for files outside the workspace.
- A **rlogin** or **SSH** remote connection to the build machine must be possible.

17.9.2 Remote Build Scenarios

Local Windows, Remote UNIX:

The workspace root directory should be located on the remote UNIX host and mapped to a specific network drive on Windows. It may also be possible to locate the root directory on the Windows machine, but then there is the need to mount the Windows disk on the build host. This may lead to problems regarding

permissions and performance, so a mapping of the workspace root-directory is definitely needed.

Local UNIX, Remote UNIX:

As it is possible to access the workspace root directory on both machines with the equivalent path (automount) it may be possible to skip the path mapping.

Local UNIX, Remote Windows:

This scenario is not supported, as you would need to execute the build command on Windows from a UNIX host.

17.9.3 Setting Up a Remote Environment

To set up your environment on the remote machine prior to a build or run, use the **Edit remote command script** button to include additional commands. It will open the file *workspaceDir*/**.metadata/.plugins/com.windriver.ide.core/remote_cmd.sh**.

For example, to extend the path before a build, add the highlighted lines to the default file:

```
#!/bin/sh

WORKSPACE_ROOT="%WorkspaceRoot%"
export WORKSPACE_ROOT
DISPLAY=%Display%
export DISPLAY

PATH=/MyTools/gmake_special/bin:$PATH
export PATH

cd $WORKSPACE_ROOT

cd "$1"
shift 1

exec "$@"
```

You can add any commands you need, but all commands must be in **sh** shell style.

17.9.4 Building Projects Remotely

1. Switch to a workspace that contains existing projects by selecting **File > Switch Workspace**. Type the path to the appropriate workspace, or click **Browse** and navigate to it.
2. In the Project Navigator, right-click a project and select **Build Options > Remote Connection**. The **Remote Connections** dialog appears.
3. Click **Add** and type a descriptive name for this remote connection. Click **OK**.
4. In the **Connection Settings** fields, add the following information to create a remote connection:

   Connection Type
   Select **Rlogin** or **SSH**.

   Hostname
   The name of the build host (can also be an IP address).

   Username
   The username used to establish the connection (the remote user may differ from the local user).

   Remote Workspace Location
   The root directory of the workspace as seen on the remote host.

   NOTE: This field must contain the absolute path to the directory; environment variables are not supported.

   Display (XServer)
   IP address of the machine where the output should be displayed.

   By clicking the **Advanced** button you can also access these fields:

   Password request string
   A string that will be recognized as a password request to prompt you for the password. If you selected **SSH**, this field is not available.

   Remember Password during Workbench sessions
   A switch to specify whether the password entered should be remembered during the current session. This is useful during a lengthy build/run session.
5. Click **Connect** to connect immediately. Remote connection settings are stored, and are specific to this workspace. They are not accessible from any other workspace.

6. The build is executed on the remote host, with the build output listed in the standard Workbench Build Console. The XServer (IP address listed in the Display field) is used whenever any type of X application is started, either during builds or runs.
7. To return to local development, select **Local Host** from the list of connections, then click **Connect**.

17.9.5 Running Applications Remotely

This section provides information about running native applications only, as running VxWorks projects remotely is handled differently.

Running native applications remotely is quite similar to running applications locally: a **Native Application** launch configuration must be created that defines the executable to be run, as well as remote execution settings for the launch. On the **Remote settings** tab are:

Remote Program
Enter the command that is used to launch the application. This may be useful for command-line applications that could then be launched within an xterm, for instance.

Remote Working Directory
This setting is optional, but if a remote working directory is given, it overrides the entry in the **Working Directory** field of the **Arguments** tab.

For remote runs, a new connection similar to the active connection will be established to allow control of Eclipse process handling, as the new remote process will be shown in the Debug view. The Remember password during Workbench sessions feature is very useful here.

Command-line application's output and input is redirected to the standard Eclipse console unless the application is started within an external process that creates a new window (such as **xterm**). The default for remote execution is a remote command like **xterm -e %Application%**, therefore a local XServer (like Exceed or Cygwin X) must be set up and running.

For more information about creating launch configurations, see *22. Launching Programs*.

17.9.6 RIogin Connection Description

The **rlogin** connection used in the Workbench remote build makes use of the standard **rlogin** protocol and ports. It establishes a connection on port 513 on the remote host, and the local port used must be in the range of 512 to 1023 per **rlogin** protocol convention.

On Windows the **rlogin** connection is implemented directly from within Workbench, so you do not need an existing **rlogin** client. The UNIX implementation is different, because for security reasons the local port (range: 512 to 1023) is restricted to root access, which cannot be granted from within Workbench. Therefore an external **rlogin** process is spawned using the command-line:

`rlogin -l` *username hostname*

rlogin on UNIX platforms makes use of **setUID root** to ensure that the needed root privileges are available.

The standard **rlogin** protocol doesn't support access to **stderr** of the remote connection, to all output is treated as **stdout**. Coloring in the Build Console of Workbench for **stderr** is therefore not available.



NOTE: On Linux the **rlogin** client and server daemon can be switched off per default. So if the machine is used as a Workbench (remote build client) host, the **rlogin** executable must be enabled (or built) and if the machine is acting as build server (remote build host) the **rlogin** daemon must be enabled. Details may be found in the system documentation of the host.

17.9.7 SSH Connection Description

The supported protocol is **SSH2**, and it establishes a connection on port 22 (the default **SSH** port).

Strict host key checking is disabled. Workbench does not use a known hosts file, so host key information is stored in memory, and you are not prompted if the host key changes.

Only password authentication is supported.

18

RTPs and Shared Libraries from Host to Target

18.1 Introduction 241

18.2 A VxWorks Real-time Process from Host to Target 242

18.3 A VxWorks Shared Library from Real-time Process to Target 247

18.1 Introduction

This chapter uses hands-on examples to illustrate one of the ways you might set up, build, and run VxWorks Real-time Processes and VxWorks Shared Libraries from host to target. The target used will be a VxWorks Simulator.



NOTE: Descriptions, screenshots, default values, and settings reflect working with Workbench on a Windows host. If you are working on another host OS, defaults might change but usually should work in a similar way. The default build spec for Windows is **SIMPENTIUMdiab_RTP_DEBUG**.

The Shared Library example (*18.3 A VxWorks Shared Library from Real-time Process to Target*, p.247) follows from the Real-time Process example, and therefore assumes that you have completed the steps outlined under *18.2 A VxWorks Real-time Process from Host to Target*, p.242.

18.2 A VxWorks Real-time Process from Host to Target

This section describes how to set up a VxWorks Real-time Process. Although the section is self-contained, you will also need to set up a VxWorks Real-time Process with the infrastructure described here if you want to use a VxWorks Shared Library.

18.2.1 Set Up the Project Structure for Real-time Processes

You will need a VxWorks Image project, a VxWorks ROMFS File System project, and a VxWorks Real-time Process project to start with. When this is done, you will need to add some code and to create a target connection to test the system.

Step 1: Set up a VxWorks image project.

1. In the Project Navigator, right-click and select **New > VxWorks Image Project**.
2. In the **Project name** field, type **VxWorksSim**. Click **Next**.
3. On the next wizard page, set **A board support package** to **simpc** and **Tool chain** to **diab**. Click **Finish**.

Step 2: Set up a VxWorks ROMFS File System project.

1. In the Project Navigator, be sure the project you just created, **VxWorksSim**, is selected so that the new VxWorks ROMFS File System project you are about to create will automatically become a subproject of **VxWorksSim**.
2. In the Project Navigator, right-click and select **New > VxWorks ROMFS File System Project**.
3. In the **Project name** field, type **VxWorksSimFS**, then click **Finish**.
4. In the Project Navigator, you will see that the **VxWorksSimFS** project you just created is a subproject of **VxWorksSim**.

Step 3: Set up a VxWorks Real-time Process project.

1. Right-click in the Project Navigator and select **New > VxWorks Real Time Process Project**.
2. In the **Project name** field, type **MyRTP**. Click **Next** a few times, then on the Build Target page choose **Linker** from the **Build Tool** dropdown list. Click **Finish**.

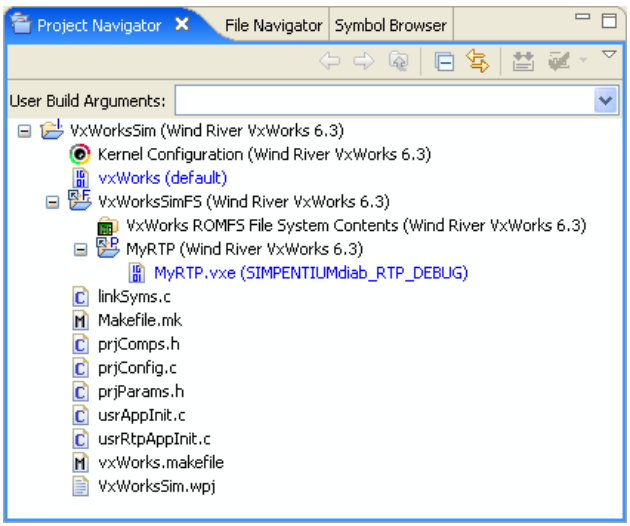
By default, the build spec **SIMPENTIUMdiab_RTP_DEBUG** is activated for the project. This matches the board support package you set for the VxWorks Image project in Step 1 above.

Step 4: Add the Real-time Process project to the VxWorks ROMFS File System project.

In the Project Navigator, drag-and-drop the **MyRTP** project onto the **VxWorksSimFS** project (the **VxWorksSim** project must be expanded).

Figure 18-1 shows the project setup so far (note that **.*** files have been filtered out by clicking the downward arrow on the Project Navigator toolbar, selecting **Filters**, selecting the **.*** checkbox, then clicking **OK**).

Figure 18-1 **Project Hierarchy for a VxWorks Real-time Process**



18.2.2 Add Code to the Real-time Process Project

You need some code that will let you verify that everything works.

1. In the Project Navigator, right-click **MyRTP** and select **New > File from Template**. In the **File Name** field, type **MyRTP.c**.
2. Click **Finish**; the file **MyRTP.c** opens in the Editor view.
3. In **MyRTP.c**, change the following lines (the results are also shown in Figure 18-2):

Change:

```
#include "MyRTP.h"
```

to read:

```
#include <stdio.h>
```

4. Change the function:

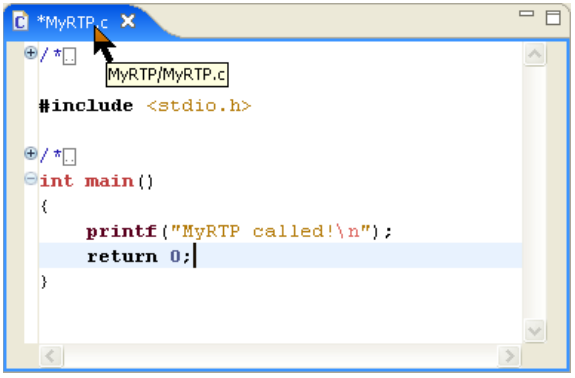
```
void MyRTP()
{
}
```

to read:

```
int main()
{
    printf("MyRTP called!\n");
    return 0;
}
```

Figure 18-2 shows the modified source file, **MyRTP.c**.

Figure 18-2 **The MyRTP.c Source**



5. Save the file but leave it open. You will need it again if you intend to follow *18.3 A VxWorks Shared Library from Real-time Process to Target*, p.247.

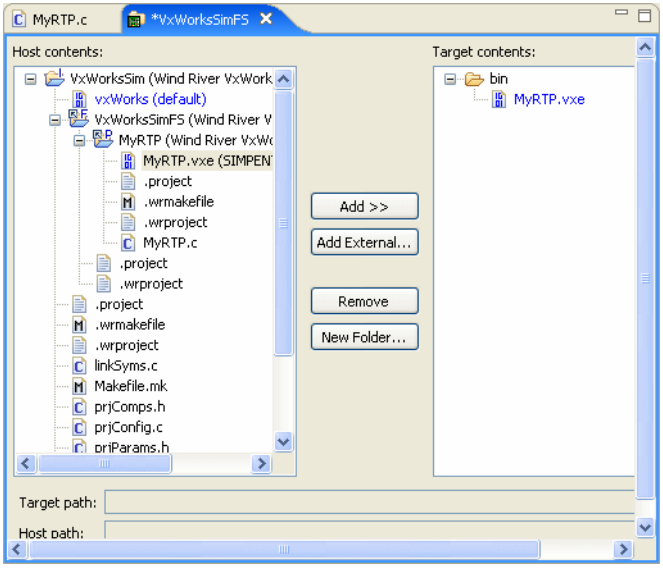
18.2.3 Add the Real-time Process to the VxWorks ROMFS Target File System

Although you do not have a Real-time Process binary yet, you can make provision for the binary to be on the target file system once the system is built.

1. In the Project Manager, under the **VxWorksSimFS** File System project you created earlier, double-click the **VxWorks ROMFS File System Contents** node to open the **File System Contents Editor**.
2. In the **File System Contents Editor**, click **New Folder** and type **bin** in the **Name** field. Click **OK**. The **bin** folder appears in the **Target Contents** panel.
3. In the **Target Contents** panel select the **bin** node, then in the **Host Contents** panel, expand the project nodes until you can see the target node that represents the (future) Real-time Process binary, namely: **MyRTP.vxe (SIMPENTIUMdiab_RTP_DEBUG)**.
4. Select this node and click **Add**. The **MyRTP.vxe** binary (which does not yet exist) appears in the **Target Contents** panel under the **bin** folder.

   Figure 18-3 shows the results.

Figure 18-3 **The Executable on the VxWorks ROMFS Target File System**



5. Leave the **File System Contents Editor** open. You will need it again if you intend to follow *18.3 A VxWorks Shared Library from Real-time Process to Target*, p.247.

18.2.4 Build the System

Now build the system and locate your VxWorks image.

1. In the Project Navigator, right-click the VxWorks Image project, **VxWorksSim**, and select **Build Project**.

   The structure is recursively built, starting in **MyRTP**, as you can see in the Build Console. Next the VxWorks ROMFS File System project, **VxWorksSimFS**, is built. The build finishes at the top, in **VxWorksSim**.

2. In the next step (*Set up the Target Connection*, p.246) you need to know the location of the VxWorks kernel image you just built. Under **VxWorksSim** expand **vxWorks (default)**, right-click **vxWorks**, then select **Properties**. Highlight the path next to **Location**, then right-click it and select **Copy**. Click **OK** to close the dialog.

18.2.5 Set up the Target Connection

To test the system you have just built, you need to create a connection to the target.

1. In the **Target Manager**, right-click **default(localhost)** and select **New > Connection**.
2. On the first wizard page, select **Wind River VxWorks 6.*x* Simulator Connection** and click **Next**.
3. On the next wizard page, select **Custom Simulator**. You have to select **Custom Simulator** because your image, unlike the **Standard Simulator**, has a VxWorks ROMFS File System that holds a Real-time Process binary linked to it.

   - If you copied the path to the **VxWorks kernel image** you just built (under *Build the System*, **p.246**), paste it in this field.
   - Otherwise, click **Browse** and navigate to the image, then click **Open**.

     In general terms, the location of the image is *WorkspaceDirectory/ProjectName/BuildSpecName*/**vxWorks**.

     In this example, the image is *installDir*/**workspace/VxWorksSim/default/vxWorks** (the *BuildSpecName* is **default** because you are using the simulator rather than a target board).

4. Click **Next** until you reach the last wizard page (when the **Finish** button is enabled). Type **VxWorksSim** in the **Connection name** field. Leave the **Immediately connect to target if possible** check box selected. Click **Finish**.

   The connection is immediately established and the kernel shell opens.

18.2.6 Run the Real-time Process on the Simulator

To test the system, run your RTP in the kernel shell.

1. At the kernel shell prompt, type:

   ```
   cd "/romfs/bin"
   ```

2. Press **ENTER** and type:

   ```
   rtpSpawn "MyRTP.vxe"
   ```

3. Press **ENTER** again. The output **MyRTP called!** should appear. Close the kernel shell.

18.3 A VxWorks Shared Library from Real-time Process to Target

Much of the initial work required for using a VxWorks Shared Library project is done under *18.2 A VxWorks Real-time Process from Host to Target*, p.242. Complete all the steps in that section; the procedures outlined below follow on from there.

18.3.1 Set Up the VxWorks Shared Library Project

First, create a new VxWorks Shared Library project to add to the project structure you created under *18.2 A VxWorks Real-time Process from Host to Target*, p.242.

1. Right-click in the Project Navigator and select **New > VxWorks Shared Library Project**.
2. On the first wizard page, type **MySharedLibrary** in the **Project name** field and click **Next**.
3. Click **Next** until you get to the **Build Target** screen. Select **Pass build target to next level**, then click **Finish**.

Note that if you add the Shared Library as a subproject beneath a Real-time Process project, the Shared Library will automatically inherit the active build spec of the Real-time Process during build, no matter which build spec is active in the Shared Library. The active build spec of the Shared Library will be taken only if you build the Shared Library directly.

18.3.2 Add Code to the Shared Library Project

You need some code that will let you verify that everything works.

1. In the Project Navigator, right-click **MySharedLibrary**, select **New > File from Template**, and type **MySharedLibrary.h** in the **File Name** field. Click **Finish**; the file **MySharedLibrary.h** opens in the Editor view.
2. In the Project Navigator, right-click **MySharedLibrary**, select **New > File from Template**, and under **File Name** type **MySharedLibrary.c**. Click **Finish**; the file **MySharedLibrary.c** opens in the Editor view.
3. In **MySharedLibrary.c** add the following line:

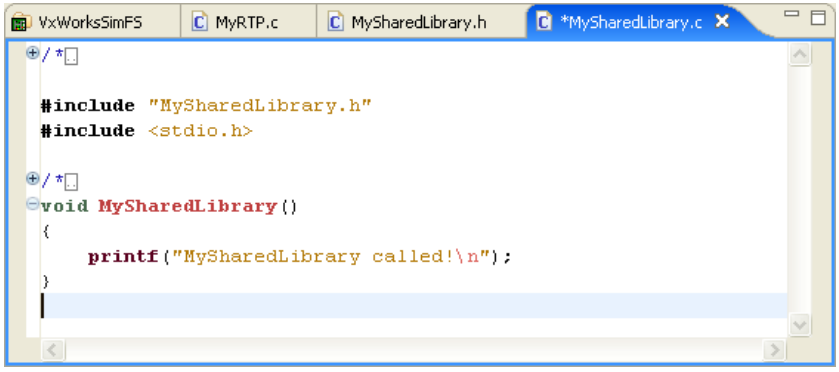
   ```
   #include <stdio.h>
   ```

4. Insert the following line into the **MySharedLibrary** function:

   ```
   printf("MySharedLibrary called!\n");
   ```

   Figure 18-4 shows the modified source file, **MySharedLibrary.c**.

Figure 18-4 **The MySharedLibrary.c Source**



5. Save **MySharedLibrary.c**.

18.3.3 Add the Shared Library to the Run-Time Process

The Shared Library must be built before the Real-time Process, so it is added as a subproject. Furthermore, the library binary must be located in the host and target side directories you specify as described below.

Step 1: Add the library as a subproject.

In the Project Navigator, drag-and-drop the **MySharedLibrary** project onto the **MyRTP** project.

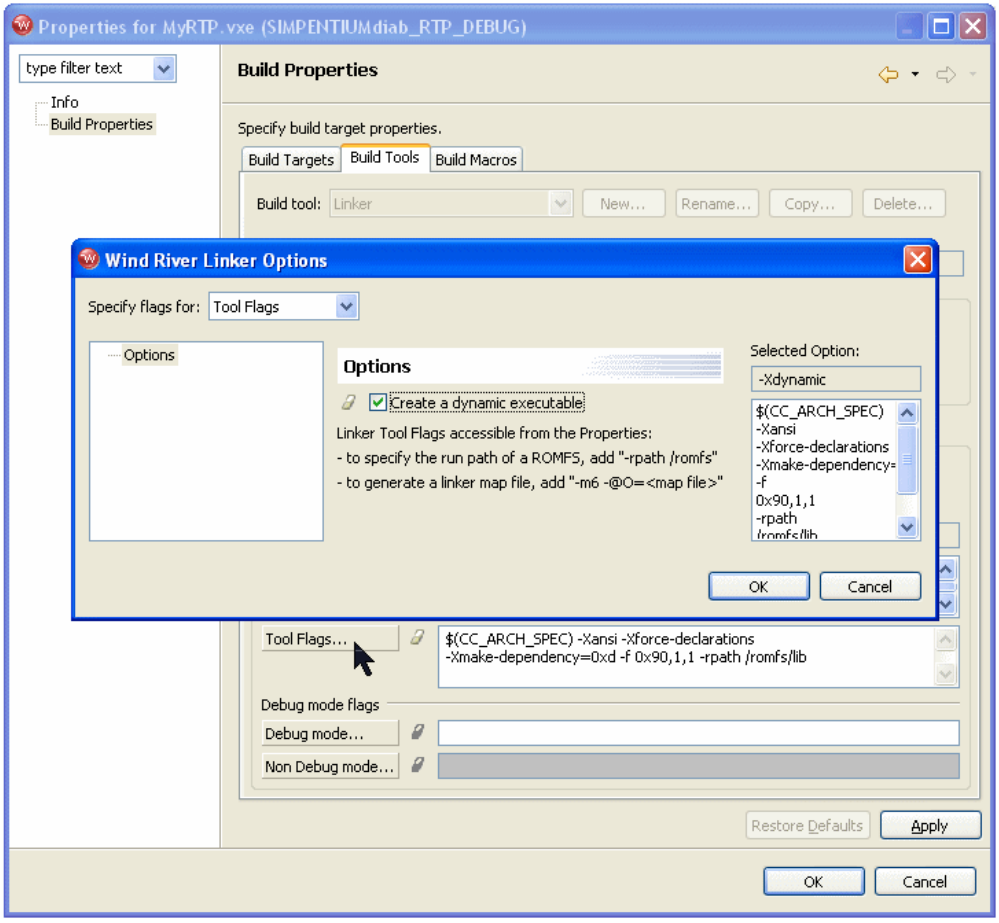
Step 2: Modify the Real-time Process build-target build properties.

1. Right-click the **MyRTP** project's build target node, **MyRTP.vxe(SIMPENTIUMdiab_RTP_DEBUG)**, and select **Properties**.
2. In the project's **Properties** dialog, select the **Build Properties** node.

Step 3: Set up the Linker Build Tool for a dynamic executable and target-side run path.

1. Select the **Build Tools** tab.
2. In the field next to the **Tool Flags** button, append the *run path*, in other words the directory that will hold the shared library on your target.
   - If you are using the Wind River Compiler, type **-rpath /romfs/lib** (**romfs** is the default root directory of the ROMFS created by the File System project you set up earlier).
   - If you are using the Wind River GNU Compiler, type **-Wl,-rpath /romfs/lib** instead.
3. Click the **Tool Flags** button. In the **Linker Options** dialog that appears, select **Create a dynamic executable**.

Figure 18-5 **Tool Flags for Dynamic Executables**



You will notice that the option as used on the command line (**-Xdynamic**) appears in the **Selected Options** list on the right.

4. Click **OK** to close the **Linker Options** dialog. You will see the option again in the field next to the **Tool Flags** button. Click **OK** to close the Build Properties dialog.

18.3.4 Modify the Code in the Real-time Process Project

You need some code that will let you verify that everything works.

1. Open **MyRTP.c** in the editor, if it is not already open.
2. In **MyRTP.c**, insert the line:

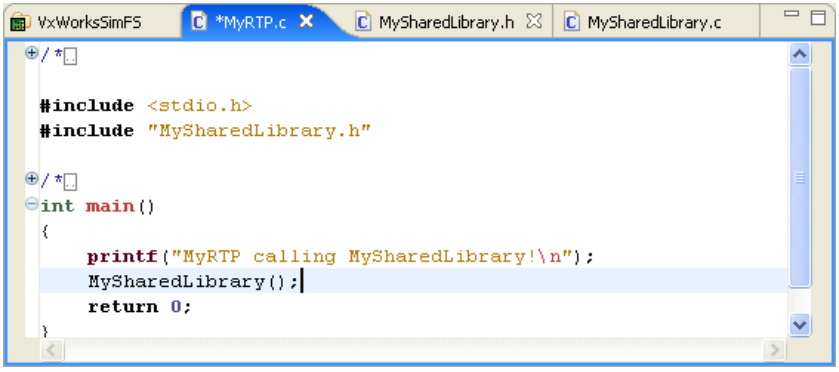
```
#include "MySharedLibrary.h"
```

3. Modify **main( )** as follows:

```
int main()
{
    printf("MyRTP calling MySharedLibrary!\n");
    MySharedLibrary();
    return 0;
}
```

Figure 18-6 shows the modified source file, **MyRTP.c**.

Figure 18-6 **The Modified MyRTP.c Source**



4. Save **MyRTP.c**.

18.3.5 Generate Include Search Paths

The file you included in **MyRTP.c**, **MySharedLibrary.h**, is not in the same project (directory). So you need to resolve the include path for successful compilation.

1. In the Project Navigator, right-click the **MyRTP** project and select **Build Options > Generate Include Search Paths**.

2. In the dialog that appears, click **Next**, and on the next page click **Resolve All**.

   Notice that the entry **%Proj-MySharedLibrary%** appears in the lower **Include Search Paths** panel. Click **Next**, then **Finish**.

18.3.6 Add Libraries to the VxWorks ROMFS Target File System

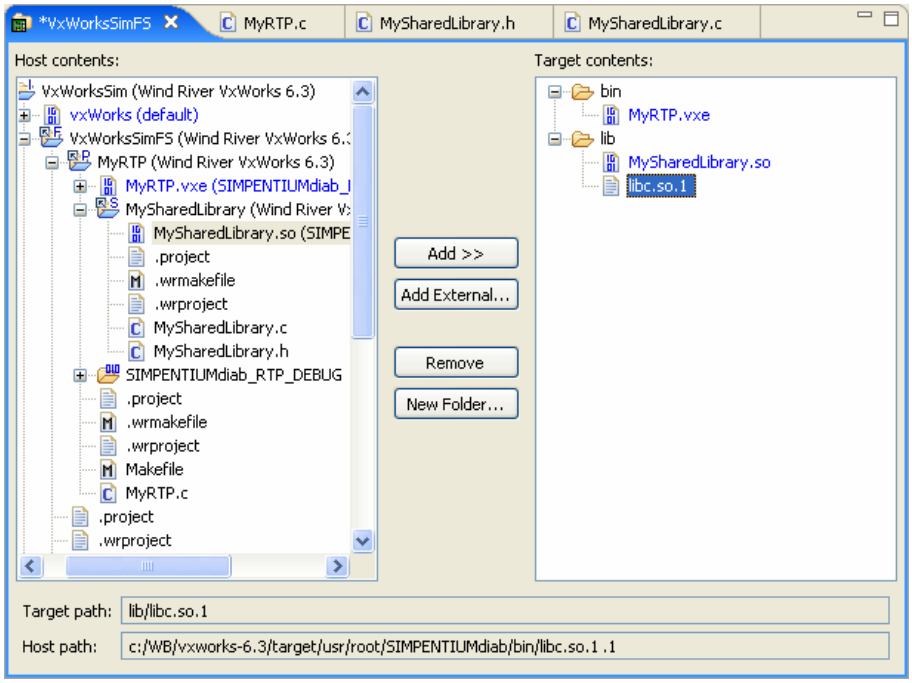
Although you do not have a Shared Library binary yet, you can make provision for the binary to be on the target file system once the system is built. You must also include **libc.so.1** (the RTP run-time shared library version of **libc**) so it is accessible to the executable at run-time.

1. The File System Contents Editor should still be open. If not, in the **Project Navigator**, under the **VxWorksSimFS** File System project you created earlier, double-click the **VxWorks File System Contents** node to open it.
2. In the File System Contents Editor, make sure **bin** is not selected. Click **New Folder** and type **lib** in the **Name** field, then click **OK**. The **lib** folder appears at the same level as **bin** in the **Target Contents** panel[1].
3. In the **Target Contents** panel select the **lib** node, and in the **Host Contents** panel, expand the project nodes until you can see the target node that represents the (future) Shared Library binary, namely: **MySharedLibrary.so (SIMPENTIUMdiab_RTP_DEBUG)**.
4. Select this node and click **Add**. The **MySharedLibrary.so** binary (which does not yet exist) appears in the **Target Contents** panel under the **lib** folder.
5. To add **libc.so.1** to the target contents, select **lib**, then click **Add External** and navigate to *installDir*/**vxworks-6.***x*/**target/usr/root/***cpuTool*/**bin.** Select **libc.so.1** and click **Open**.

   Figure 18-3 shows the results. Notice that when you click on items in the **Target contents** pane, their **Target path** and **Host path** information appears in the fields at the bottom of the view.

1. If the **lib** folder is created under the **bin** folder, drag it to the left and away from **bin**, then drop it.

Figure 18-7 **The Library Binary on the Target File System**



18.3.7 Build the System Again

In the Project Navigator, right-click the VxWorks Image project, **VxWorksSim**, and select **Build Project**.

The structure is recursively built, starting, as you can see in the Build Console, in **MySharedLibrary**, followed by **MyRTP**, followed by the VxWorks ROMFS File System project, **VxWorksSimFS**, and finishing at the top in **VxWorksSim**.

18.3.8 Run the RTP with the Shared Library on the Simulator

1. In the Target Manager, right-click **VxWorksSim**, the connection you created under *18.2.5 Set up the Target Connection*, p.246, and select **Connect**.

   Once the connection has been established, the kernel shell appears.

2. At the kernel shell prompt, type:

   ```
   cd "/romfs/bin"
   ```

3. Press **ENTER** and type:

   ```
   rtpSpawn "MyRTP.vxe"
   ```

4. Press **ENTER** again.

   The following output should appear:

   ```
   MyRTP calling MySharedLibrary!
   MySharedLibrary called!
   ```

5. Close the kernel shell.

For more information about using shared libraries and **romfs** subdirectories, see *VxWorks Application Programmer's Guide: Applications and Processes*, specifically the *Developing Shared Libraries* section.

PART IV

Target Management

19 Connecting to Targets 257

20 New Target Server Connections 265

21 New VxWorks Simulator Connections 277

19

Connecting to Targets

19.1 Introduction 257

19.2 The Target Manager View 258

19.3 Defining a New Connection 258

19.4 Establishing a Connection 259

19.5 The Registry 262

19.1 Introduction

A target connection manages communication between the Workbench host tools and the target system. A connection must be configured and established before host tools can interact with the target.

All host-side connection configuration work and connection-related activity is done in the Target Manager view. Connections are registered and made accessible to users by the Wind River Registry.

This chapter describes ways to configure, start, and manage target connections in the Target Manager view. For detailed information about the Target Server and Registry, see the **tgtsvr** and **wtxregd** API reference entries (see **Help > Help Contents > Wind River Documentation > References > Host API and Command References > Wind River Host Tools API Reference**).

19.2 The Target Manager View

A connection to a Target Server or a VxWorks Simulator must be defined and established before tools can communicate with a target system.

All host-side connection configuration work and connection-related activity is done in the Target Manager view. The target side (required for Target Server and VxWorks Simulator connections) is configured in the Kernel Configuration Editor (see *5.4.1 The Kernel Configuration Editor*, p.94).

By default, the Target Manager view is on a tab at the bottom-left of Workbench. It is available in the Application Development perspective and in the Device Debug perspective. If the view is not visible, choose **Window > Show View > Target Manager** (or, if it is not listed there, **Window > Show View > Other**).

The most import tasks in the Target Manager view are:

- defining new connections
- connecting to targets
- disconnecting from targets

Once you have connected to a target, more commands are enabled on the right-click context menu (see also *22. Launching Programs*).

19.3 Defining a New Connection

All connections types are defined from the Target Manager view (see *19.2 The Target Manager View*, p.258).

To open the New Connection wizard, use the appropriate toolbar icon or right-click in the **Target Manager** and select **New > Connection**.

The first thing the New Connection wizard asks you to do is to select one of the following connection types:

- **Wind River Target Server Connection**

  See *20.2 Defining a New Target Server Connection*, p.265.

- **Wind River VxWorks Simulator Connection**

  See *21.2 Defining a New Wind River VxWorks Simulator Connection*, p.277.

Properties you set using the New Connection wizard can be modified later by right-clicking the connection in the Target Manager and choosing **Properties**. In most cases, you have to disconnect and reconnect for the changes to take effect.

19.4 Establishing a Connection

Once you have created your application projects and defined connections, you will want to run, test, and debug the projects on your target or simulator. To do this, you first need to connect to the target.

19.4.1 Assumptions

- You are using a simulator (VxWorks Simulator or the on-chip debugging simulator, ISS), or you are using a target board and your hardware connections are set up and running.
- If you are using a target board (not a simulator), you have correctly configured your FTP service as described in *3. Setting Up Your Development Environment* and in the *Wind River ICE for Wind River Workbench Hardware Reference* and *Wind River Probe for Wind River Workbench Hardware Reference*.
- You have defined one or more host-target connections as described in *20. New Target Server Connections* and *21. New VxWorks Simulator Connections*.

19.4.2 Connecting to the Target

The first step in running an application on the target is to establish a connection to that target.

Connect to and disconnect from targets in the Target Manager by selecting a connection node and then using the appropriate toolbar icon, or by right-clicking and selecting **Connect**.

Once the connection has been established:

- If the connection is to a simulator, the Kernel Shell appears (see *19.4.4 The Kernel Shell*, p.261, for more information).

- On Windows, a registry icon appears (if it is not there already) in the Windows system tray (the area at the right of the Windows taskbar) to indicate the registry is running (see also *19.5 The Registry*, p.262).
- In the Target Manager:
  - A blue check mark is superimposed on the top-left corner of the connection node, the node is labeled *ConnectionName* **[connected]**, and new nodes appear under the connection node.
  - A subnode appears under the connection node. This node's label identifies the connection type and the kernel. The node's right-click context menu offers a subset of the connection node's context menu (restricted to the most commonly used commands) as well as the **Kernel Objects** command. The **Kernel Objects** command populates and opens the **Kernel Objects** tab (by default located behind the Target Manager).
  - A number of additional subnodes appear. These are described in Table 19-1. For a full list of the icons that you might see in the Target Manager, click in the Target Manager and press the help key for your host.

Table 19-1 **VxWorks Connections**

Node	Description
Real Time Processes	When you run RTPs, they will appear as subnodes under this node.
Kernel Tasks	When the connection is initially established, you see the VxWorks tasks. When you download and run DKMs, they will appear as additional subnodes under this node.
VxWorks *location*	The kernel node and its host location. A superimposed red **S** at the top-right of the icon indicates that symbol information has been downloaded.

19.4.3 Specifying an Object File

If you are loading object code on the target using a custom loader, or associating symbols with already loaded modules, you can specify the object file that you want the debugger to use.

1. Right-click a container in the Target Manager, then select **Load/Add Symbols to Debug Server**. A dialog appears with your connection and core already filled in.
2. To add a new object file to the **Symbol Files and Order** list, click **Add**. Navigate to the file, then click **Open**.
3. In the **Symbol Load Options** section, select **Specify base start address** or **Specify start address for each section**.
4. When you are finished, click **OK**.

For more information about the fields in this dialog, click in the Target Manager, then press the help key for your host.

19.4.4 The Kernel Shell

The Kernel Shell[1] that appears when you establish a connection displays output generated by applications running on the kernel.

If you are using a VxWorks Simulator connection, shell components are included in the kernel by default and the Kernel Shell also provides a prompt and accepts input like the Host Shell (see the *VxWorks Command-Line Tools User's Guide*). If you are using a real board connection, the kernel shell does not provide an input prompt by default; you can, however, include the necessary components in the VxWorks kernel (see *5.4 Configuring Kernel Components*, p.93 as well as the *VxWorks Kernel Programmer's Guide* and the *VxWorks Application Programmer's Guide*).

For the most part, the Kernel Shell works the same as the Host Shell. For detailed information about the Host Shell see the *VxWorks Command-Line Tools User's Guide*. For information about the differences between the Host and Kernel shells, see the *VxWorks API Reference* entries for **dbgLib**, **shellLib**, and **usrLib**.

1. In versions of VxWorks prior to 6.0, the Kernel Shell was called the Target Shell. The new name reflects the fact that the target-resident shell runs in the kernel and not in a process.

19.5 The Registry

The Wind River Registry is a database of target servers, boards, ports, and other items used by Workbench to communicate with targets. For details about the registry, see the **wtxregd** and **wtxreg** reference entries.

If Workbench finds an installed VxWorks platform on start-up, it creates a default VxWorks Simulator connection. Before any target connections have been defined, the default registry—which runs on the local host—appears as a single node in the Target Manager. (Under Linux, the default registry is a target-server connection for Linux user mode.) Additional registries can be established on remote hosts.

Registries serve a number of purposes:

- The registry stores target connection configuration data. Once you have defined a connection, this information is persistently stored across sessions and is accessible from other computers.

  You can also share connection configuration data that is stored in the registry. This allows easy access to targets that have already been defined by other team members.



NOTE: Having connection configuration data does not yet mean that the target is actually connected.

- The registry keeps track of the currently running target servers and administrates access to them.
- Workbench needs the registry to detect and launch target servers.

  If Workbench does not detect a running default registry at start-up, it launches one. After quitting, the registry is kept running in case it is needed by other tools.

19.5.1 Launching the Registry

To launch the default registry, open the **Target** menu or right-click in the Target Manager and select **Launch Default Registry**.



NOTE: These menu items are only available if the registry is not running, and the default registry host is identical to the local host.

The registry stores its internal data in the file *installDir*/**.wind/wtxregd.***hostname*. If this file is not writable on launch, the registry attempts to write to

/var/tmp/wtxregd.*hostname* instead. If this file is also not writable, the registry cannot start and an error message appears.

19.5.2 Remote Registries

If you have multiple target boards being used by multiple users, it makes sense to maintain connection data in a central place (the remote registry) that is accessible to everybody on the team. This saves everyone from having to remember communications parameters such as IP addresses for every board that they might need to use.

Creating a Remote Registry

You might want to create a new *master registry* on a networked remote host that is accessible to everybody. To do so:

1. Workbench needs to be installed and the registry needs to be running on the remote host. The easiest way to launch the registry is to start and quit Workbench. However, you can also launch the **wtxregd** program from the command line. (For more information about **wtxregd**, see **Help > Help Contents > Wind River Documentation > References > Host API and Command References > Wind River Host Tools API Reference**.)
2. Right-click in the Target Manager, (see *19.2 The Target Manager View*, p.258), then select **New > Registry** from the context menu.
3. In the dialog that appears, enter either the host name or the IP address of the remote host.

   Workbench immediately attempts to connect to the remote registry. If the host is invalid, or if no registry is identified on the remote host, this information is displayed in the Target Manager.

19.5.3 Shutting Down the Registry

Because other tools use the registry, it is not automatically shut down when you quit Workbench. Before updating or uninstalling Workbench (or other products that use the registry), it is advisable to shut down the registry so that the new one starts with a fresh database. To shut down the registry:

- On Windows, right-click the registry icon in the system tray, and choose **Shutdown**.
- On Linux and UNIX, execute **wtxregd stop**, or manually kill the **wtxregd** process.

If you want to migrate your existing registry database and all of your existing connection configurations to the new version, make a backup of the registry data file *installDir*/**.wind/wtxregd.***hostname* and copy it to the corresponding new product installation location.

19.5.4 Changing the Default Registry

Normally, the default registry runs on the local computer. You can change this if you want to force a default remote registry (see *19.5.2 Remote Registries*, p.263). To do this on Linux and UNIX, modify the **WIND_REGISTRY** environment variable. To do this on Windows, under the Windows Registry **HKEY_LOCAL_MACHINE** node, modify the field **Software\Wind River Systems\Wtx***N.N***\WIND_REGISTRY**.

20

New Target Server Connections

20.1 Introduction 265

20.2 Defining a New Target Server Connection 265

20.3 Kernel Configuration 275

20.1 Introduction

Target Server connections are defined in the Target Manager view (see *19. Connecting to Targets*).

20.2 Defining a New Target Server Connection

To open the **New Connection** wizard, right-click in the Target Manager, then select **New > Connection**.

20.2.1 Wind River Target Server

On the initial page of the New Connection wizard, select **Wind River Target Server Connection for VxWorks** and click **Next**.

20.2.2 Target Server Connection Page

Back End Settings

Back end

The **Back end** settings specify how a target server will communicate with a target. Table 20-1 provides descriptions of the available options in the **Back end** drop-down list.

Table 20-1 **Communications Back Ends for Target Server**

Back End	Description
wdbrpc	WDB RPC. This is the default. It supports any kind of IP connection (for example, Ethernet). Polled-mode Ethernet drivers are available for most BSPs to support system-mode debugging for this type of connection.
wdbpipe	WDB Pipe. The back end for VxWorks target simulators.
wdbserial	WDB Serial. For serial hardware connections; does not require **SLIP** on the host system. If you select this option, also choose a **Host serial device** (port) and **Serial device speed** (bits per second).
wdbproxy	WDB Proxy. The backend for UDP, TCP, and TIPC connections.

⚠ **CAUTION:** The target server *must* be configured with the same communication back end as the one built into the kernel image and used by the target agent. The standard back end options are described in Table 20-1; the compatible kernel components are listed in Table 20-4.



CAUTION: Do *not* choose the TIPC WDB Proxy connection type unless you have included the **TIPC network stack** (**INCLUDE_TIPC_ONLY**) component in your VxWorks Image Project.

For more information about finding components to include in your VxWorks Image Project, open the Kernel Configuration Editor and press the help key for your host.

For more information about TIPC, see *Wind River TIPC for VxWorks 6 Programmer's Guide: Building VxWorks to Include Wind River TIPC*.

CPU

Workbench can correctly identify the target CPU. In rare cases, a close variant might be misidentified, so you can manually set the CPU here.

Name/IP address

The **Name/IP Address** field specifies the network name or the IP address of the target hardware for networked targets. If you are using a serial port, enter either **COM1** or **COM2**.

Kernel Image and Symbols

The **Kernel Image and Symbols** properties relate to a copy of the target kernel that resides on the host.

File path from target (if available)

Select this option to search for an image of the software running on the target using the target path.

File

If the run-time image file is not in the same location on the host that is configured into the target (or if host and target have different views of the file system), select this option and use the adjacent text box to specify the host location of the kernel image.

For example, if you are using a target programmed with a **vxWorks_rom.hex**, **vxWorks_romCompressed.hex**, or any other on-board VxWorks image, you must use this option to identify the kernel file location; otherwise the target server will not be able to identify the target symbols.

Advanced Target Server Options

Please see the **tgtsvr** reference entry in the online API reference and the *VxWorks Programmer's Guide* for more detailed information about target server options in the Target Manager, as well as on additional available options.

Options

These options are passed to the **tgtsvr** program on the command line. Enter these options manually, or use the **Edit** button for GUI-assisted editing.

Advanced Target Server Options Dialog

The properties in the **Advanced Target Server Options** dialog that you open with **Edit** on the main wizard page are subdivided into three tabbed groups: **Common**, **Memory**, and **Logging**.

The Common Tab

Target Server File System

The Target Server File System (TSFS) is a full-featured VxWorks file system that provides target access to files located on the host system. It is used by the Wind River System Viewer. It also provides the most convenient way to boot a target over a serial connection. Although somewhat slow, it is simple and easy to use.

A target can access files on the host it is booted from, if booted via **FTP** or **rsh**. However, if the target is booted from a remote host, you can use the TSFS as a simple method to access files on the local host.

The TSFS is also the default method used by the System Viewer for uploading event data from the target. The TSFS should therefore be enabled and writable (default) when using the System Viewer.



CAUTION: To use the TSFS, you must include the **WDB target server file system** component when you build the kernel image. See *20.3 Kernel Configuration*, p.275, below, and the *VxWorks Kernel Application Programmer's Guide* for more details.

Root

If the **Enable File System** check box is selected, you have to identify the root of the host file system that will be made visible to target processes using the TSFS. By default, this is the Workspace root directory. If you use the TSFS for

booting a target, it is recommended that you use the default root directory. If you do not use TSFS, you must use the **Kernel Image and Symbols** configuration options to specify the location of the kernel image (see *Kernel Image and Symbols*, p.267).

Make Target Server File System writable

To use the Wind River System Viewer, you must select this check box to allow uploading of event data from the target. Because this also allows other users to access your host file system, you may wish to set the TSFS option for your target server to read-only when you are not using the System Viewer.

Timeout Options

Specify allowable spawn time (in seconds) for kernel tasks and RTPs, time (in seconds) to wait for a response from the agent running on the target system, how often to retry, and at what intervals.

The Memory Tab

Memory Cache Size

To avoid excessive data-transfer transactions with the target, the target server maintains a cache on the host system. By default, this cache can grow up to a size of 1 MB.

A larger maximum cache size may be desirable if the memory pool used by host tools on the target is large, because transactions on memory outside the cache are far slower.

The Logging Tab

Options on the **Logging** tab are used mainly for troubleshooting by Customer Support.

A maximum size can be specified for each enabled log file. Files are rewritten from the beginning when the maximum size is reached. If a file exists, it is deleted when the target server restarts (for example, after a reboot).

For the WTX (Wind River Tool Exchange) log file, you can specify a *filter*, a regular expression that limits the type of information logged. In the absence of a filter, the log captures all WTX communication between host and target. Use this option in consultation with Customer Support.

20.2.3 Object Path Mappings Page

Object Path Mappings have two functions:

- To allow the debugger to find symbol files for processes created on the target by creating a correspondence between a path on the target and the appropriate path on the host.
- To calculate target paths for processes that you want to launch by browsing to them with a host file-system browser.

By default, the debug server attempts to load all of a module's symbols each time a module is loaded. In the rare cases where you want to download a module or start a process without loading the symbol file, uncheck **Load module symbols to debug server automatically if possible**.

The simplest way to create Object Path Mappings for a module that does not have symbols yet is to download the output file (or run the executable) manually. In the Target Manager, right-click the file or executable and select **Load/Add Symbols to Debug Server**. From the Load Symbols dialog, select **Create path mappings for the module based on the selected symbol file** and click **OK**. Object path mappings are created automatically, so that after the next disconnect/reconnect sequence the symbols will be found.

Pathname Prefix Mappings

This maps target path prefixes to host paths. Always use full host paths, not relative paths.

For example, mapping **/tgtsvr/** to **C:\workspace** tells the debugger that files accessible under **/tgtsvr/** on the target can be found under **C:\workspace** on the host.

If you launch the process **host:/usr/hello.vxe** on your target, Workbench needs to know what **host:/** corresponds to; in other words, where it can find the **hello.vxe** ELF file in the host file system. With an object path mapping of **host:/** to **C:\WindRiver**, Workbench knows that the host path to the file is **C:\WindRiver\usr\hello.vxe**.

In most cases Workbench provides correct defaults. If necessary, click **Add** to add new mappings, or select existing mappings and click **Edit** to modify existing mappings. The supplied default mappings are not editable.

To disable any listed object path mapping, including default mappings, unselect the checkbox to the left of that mapping. To re-enable it, select the checkbox again.

You can export your object path mappings to XML by clicking **Export** and providing a descriptive filename. Likewise you can import mappings by clicking **Import** and selecting an appropriate XML file.

Reverse Mapping

Sometimes host paths must be mapped to target paths. For example, if you want to browse to the process **C:\WindRiver\usr\hello.vxe** and launch it on the target, Workbench needs to know that the correct target path for this process is **host:/usr/hello.vxe**.

Path Mappings for Working with Remote Hosts

You may need to edit object path mappings if your target boots from a remote host or if your target server runs on a remote host.

Running the target server on a remote host (using a remote registry; see *19.5.2 Remote Registries*, p.263 for details) allows you to:

- Access targets using a serial line **wdb** connection even if the targets are physically connected to a remote host.
- Have different IP subnets for the targets in a lab and the client running Workbench, with the target server being the intermediary to translate between the separate subnets.

In this discussion, the **target** is the VxWorks target, the **host** is the remote registry host that the target server is running on, and the **client** is the system on which Workbench is running.

Prerequisites

To allow Workbench to access targets attached to a remote host, two prerequisites must be met:

1. The VxWorks image must be visible to the target (for booting), the host (for the target server), and the client (for the debugger and the host shell).
2. A file system must be shared between the target and client for running RTPs.

Example: Adding New Path Mappings

When the target server is running on a host that can see the same (networked) file system that the client can, you do not need to adjust your object path mappings. The remote target server connections can be used exactly like local connections.

However, when the remote host and the client see different file systems, you need to create new path mappings to tell the debugger where it can find the files seen by the target server. In this case, the path to the kernel image is entered as seen on the remote host; path mappings must be added to tell the debugger where these paths are on the client.

When there are multiple clients with different file systems, you must add path mappings for each client. The debugger tries them in the order in which they appear.

For example, consider a scenario with two clients (one on Windows, one on UNIX) accessing a common target server host. Table 20-2 shows how each client is set up; this is the information you would have to work with when figuring out the object path mappings for this scenario.

Table 20-2 **Clients Connected to a Common Target Server Host**

Station	Setup Description
target **t100**	Booted using **rsh** from **moon:/export1/images/t100/vxWorks** TSFS enabled
host **moon**	Kernel path from target, on **/export1/images/t100/vxWorks** TSFS enabled, with rootdir **/Net/shares/tsfs/t100** **Tgtsvr** command line: **tgtsvr -R /Net/shares/tsfs/t100 -RW t100**
client **c-unix**	File system shared with host **moon** Kernel seen on **/Net/moon/export1/images/t100/vxWorks** TSFS path same as on **moon**
client **c-win**	Kernel seen on **\\moon\export1\images\t100\vxWorks** TSFS seen on **L:\tsfs\t100**

Based on this information, the host and target path mappings you would enter into the **Pathname Prefix Mappings** fields are shown in Table 20-3.

Table 20-3 **Host and Target Paths Converted to Object Path Mappings**

Target Path	Host Path	Comment
moon:/export1	**/Net/moon/export1**	Access to the boot file system for UNIX clients. Allows Workbench to reverse-map for running RTPs, so when running the RTP **/Net/moon/export1/myrtp.vxe**, the target path will be computed as **moon:/export1/myrtp.vxe**
moon:/export1	**\\moon\export1**	Now the same for Windows clients.

Table 20-3 **Host and Target Paths Converted to Object Path Mappings** (cont'd)

/export1	**/Net/moon/export1**	Allows Workbench to find the kernel path: sent by the target server as **/export1/...**, this can be forward-mapped to the common UNIX file system for clients.[a]
/export1	**\\moon\export1**	Now the same for Windows clients.
/tgtsvr	**/Net/shares/tsfs/t100**	Allow reverse-mapping of the **tgtsvr** file system for UNIX hosts.
/tgtsvr	**L:\tsfs\t100**	Now the same for Windows hosts.

a. This mapping may be used only for forward-mapping the kernel image, so it must be listed *after* the previous mappings, which are used for reverse-mapping as well.

If you do not run RTPs, only the mappings for the kernel image are required (shown in the third and fourth rows of Table 20-3). None of the other mappings are necessary, since a file system is not needed for debugging kernel modules.

Basename Mappings

Use square brackets to enclose each mapping of target file basenames (left element) to host file basenames (right element), separated by a semi-colon (**;**). Mapping pairs (in square brackets) are separated by commas. You can use an asterisk (*****) as a wildcard.

For example, if debug versions of files are identified by the extension ***.unstripped**, the mapping **[*;*.unstripped]** will ensure that the debugger loads *yourApp***.vxe.unstripped** when *yourApp***.vxe** is launched on the target.

20.2.4 Target State Refresh Page

Since retrieving status information from the target leads to considerable target traffic, this page allows you to configure how often and under what conditions the information displayed in the Target Manager is refreshed.

These settings can be changed later by right-clicking the target connection and selecting **Refresh Properties**.

Available CPU(s) on Target Board

Workbench can correctly identify the target CPU. In rare cases, a close variant might be misidentified, so you can manually set the CPU here.

Initial Target State Query and Settings

Specify whether Workbench should query the target on connect, on stopped events, and/or on running events. You can select all options if you like.

Target State Refresh Settings

Specify whether Workbench should refresh the target state only when you manually choose to do so, or if (and how often) the display should be refreshed automatically.

Listen to execution context life-cycle events

Specify whether Workbench should listen for life-cycle events or not.



NOTE: To prevent excessive delay in the update of the Target Manager display, do not use this option when there are more than 100 contexts on the target.

20.2.5 Connection Summary Page

This page proposes a unique **Connection name**, which you can modify, and displays a **Summary** of name and path mappings for review. To modify these mappings, click **Back**.

Shared

This option, which is available only for certain connection types, serves a dual purpose:

- When you define a target connection configuration, this connection is normally visible only for your user ID. If you define it as **Shared**, other users can also see the configuration in your registry, provided that they connect to your registry (by adding it as a remote registry on their computer; see *19.5.2 Remote Registries*, p.263).

- Normally, when you terminate a target connection, the target server (and simulator) are killed because they are no longer needed. In a connection that is flagged as **Shared**, however, they are left running so that other users can connect to them. In other words, you can flag a connection as shared if you want to keep the target server (and simulator) running after you disconnect or exit Workbench.

Immediately connect to target if possible
If you do not want to connect to the target immediately, you can connect to the target later using one of the ways described in *24. Debugging Projects*. If you have applications ready to run using the connection(s) you just created, please see *22. Launching Programs*.

20.3 Kernel Configuration

Once you have defined a Target Server (or VxWorks Simulator) connection, you may have to configure the kernel communication. The default configuration, however, will normally work fine for getting started.

The target server and the simulator communicate with the target system through the *target agent*. To communicate with the target agent, the target server uses a communication back end that has to be configured for the same communication protocol and transport layer as the target agent on the kernel.

When you create Target Server or VxWorks Simulator connections, you define host back end communication in the Kernel Configuration Editor. For more information about this topic, see *5.4.1 The Kernel Configuration Editor*, p.94.

Table 20-4 shows an overview of target server back ends and the kernel components that provide the required target-agent communication interface.

Table 20-4 **Communications Back Ends for Target Server and Compatible Kernel Components**

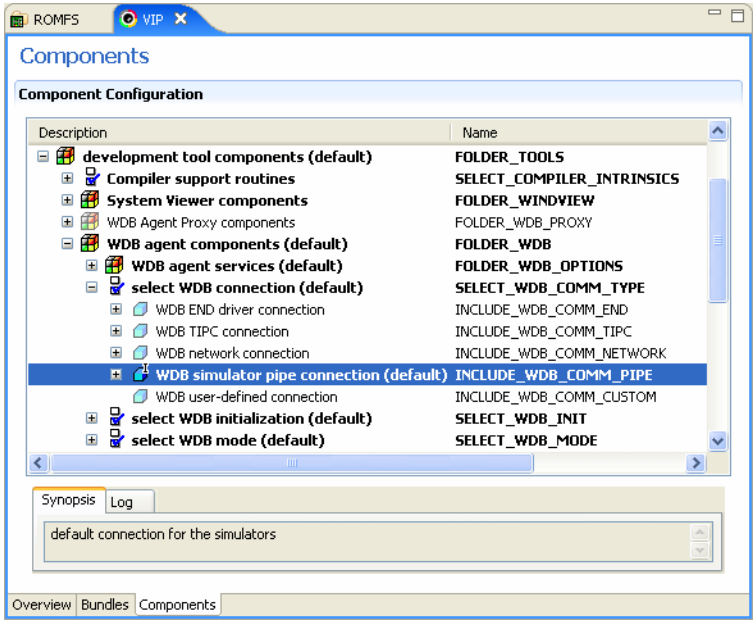
Back End	Compatible Kernel Component
wdbrpc	**WDB END driver connection** or **WDB network connection**
wdbpipe	**WDB simulator pipe connection**
wdbserial	**WDB serial connection**

Table 20-4 **Communications Back Ends for Target Server and Compatible Kernel Components** (cont'd)

Back End	Compatible Kernel Component
wdbproxy	**WDB network connection** (for UDP/TCP) or **TIPC network stack** (for TIPC)

Figure 20-1 shows where to find these kernel components in the Kernel Configuration Editor.

Figure 20-1 **Kernel Configuration Editor Showing WDB Connection Components**



These and other communication-related kernel components are described in detail in the *VxWorks Programmer's Guide: Kernel Images, Components, and Configuration*.

21 New VxWorks Simulator Connections

21.1 Introduction 277

21.2 Defining a New Wind River VxWorks Simulator Connection 277

21.1 Introduction

The Wind River VxWorks Simulator allows you to simulate a connection to a standard or customized version of a VxWorks 6 kernel.

21.2 Defining a New Wind River VxWorks Simulator Connection

For VxWorks Simulator-specific information going beyond this description, please see the *Wind River VxWorks Simulator User's Guide*.

Target Server connections are defined in the Target Manager view (see *19.2 The Target Manager View*, p.258).

To open the New Connection wizard, right-click in the Target Manager and choose **New > Connection**.

On the initial page of the New Connection wizard, select **Wind River VxWorks Simulator Connection** and click **Next**.

21.2.1 VxWorks Boot Parameters Page

Standard Simulator (Default)
Select this option to create a simulated connection to a standard VxWorks kernel.

Custom Simulator
Select this option if you are using a customized VxWorks kernel.

VxWorks Kernel Image
This field is enabled only if you select **Custom Simulator**. Navigate to the location of your customized kernel image.

Processor Number
Your system is automatically configured to run multiple simulators. Workbench assigns each simulator a unique positive number, known as the **Processor number**.

Advanced Boot Parameters
Please see the *Wind River VxWorks Simulator User's Guide* for information on the **vxsim** command-line options that can be set in this dialog.

21.2.2 VxSim Memory Options Page

These options allow you to manage your memory resources. Please see the *Wind River VxWorks Simulator User's Guide* for details.

21.2.3 VxWorks Simulator Miscellaneous Options Page

This page offers file-system location options (see the *Wind River VxWorks Simulator User's Guide* for details), the ability to influence the process priority of the simulator, and a field for entering additional command-line options that are passed as-is to **vxsim**.

21.2.4 Target Server Options Page

WDB back end type
This corresponds to the **Back end**, as described for the Target Server connection; see *Back End Settings*, p.266. The VxWorks Simulator uses the **wdbpipe** back end by default.

Name/IP Address
Available only if the **wdbrpc** back end is selected. Specifies the network name or IP address of the target. If you are using a serial port, enter either **COM1** or **COM2**.

The remaining options in the wizard are the same as those outlined for the Target Server connection settings. These are described starting from *Advanced Target Server Options Dialog*, p.268.

If you have created a connection for a standard simulator, the default settings should work. However, if you have defined a custom simulator connection, you may have to configure the kernel-side communication, see *20.3 Kernel Configuration*, p.275.

If you have applications ready to run using the connection(s) you have just created, please see *22. Launching Programs*.

PART V

Debugging

22 Launching Programs 283

23 Managing Breakpoints 305

24 Debugging Projects 313

25 Troubleshooting 337

22

Launching Programs

22.1 Introduction 283

22.2 Launching a Kernel Task or a Process 284

22.3 Reset & Download: Hardware Debugging Launches 289

22.4 Launching a Native Application 290

22.5 Relaunching Recently Run Programs 292

22.6 Controlling Multiple Launches 293

22.7 Launches and the Console View 298

22.8 Using Attach-to-Target Launches 300

22.9 Suggested Workflow 302

22.1 Introduction

A launch configuration is like a named script that captures the whole process of building, connecting a target, downloading, running, and possibly attaching a debugger. Whenever you run a process, task, or program from the Project Navigator or the Target Manager, a **Launch Configuration** is automatically created for you. Launch configurations are stored persistently, so you can rerun your previous launches by clicking a single button, and you can share them with your team.

The same launch configuration can be executed in *Run-mode* and *Debug-mode*:

- Run-mode connects to your target, then launches a task or process.
- Debug-mode is like run-mode, but in addition to connecting to your target and launching your process, it also attaches the debugger.

This chapter explains how to create, edit, and fine-tune your launch configurations to provide a tight edit-compile-debug cycle, as well as how to manually attach the debugger to tasks and processes.

For descriptions of the tabs in this dialog as well as a guide to the icons you will see in the launch configuration wizard, open the launch configuration dialog and press the help key for your host.

22.2 Launching a Kernel Task or a Process

Launch configurations that run kernel tasks, RTPs, and Linux processes are very similar. Only a few options and settings differ between them.

To create a new launch configuration, select a build target in the Project Navigator then select **Run > Run** or **Run > Debug**[1]. The **Create, manage, and run configurations** dialog appears.

1. From the **Configurations** list, select the type of launch you want to create, then click **New**.
2. The **Name** field will display a default name based on the type of configuration you selected.
   - A new kernel task launch configuration is called **noEntryPoint -** *moduleName - connectionName*[2]. As soon as you select an entry point for the configuration, the name changes to *entryPoint - moduleName - connectionName*. If you prefer, you can type a completely new name in the **Name** field.
   - A new process or RTP configuration is called **noExecPath -** *connectionName*. As soon as you select an Exec Path for the configuration

1. You can also create a launch configuration by right-clicking on the build target in the Project Navigator and selecting the appropriate **Run** or **Debug** command from the context menu.
2. If no target is connected, the default name is **noEntryPoint** - *moduleName* - **noDownload.**

(when you specify the executable to run), the name changes to *executable - connectionName*. Or, if you prefer, you can type a completely new name in the **Name** field.

22.2.1 Defining the Target Connection

The default **Connection to use** is the target that is currently connected. If you have more than one connection defined in the Target Manager, you can select a different one from the drop-down list.

1. To change the properties of the target connection, including target server options and object path mappings, click **Properties**.
2. To create a new connection definition, click **Add**.
3. To retrieve the connection-specific properties from the target, and adjust them if necessary, click **Connect**.

For more information about target connections, click in the Target Manager and press the help key for your host, and see *20. New Target Server Connections*.

22.2.2 Defining the Kernel Task or Process to Run

The settings in this section can be changed only when you are connected to a target.

Once your target is connected, you can select the **Entry Point** of your program from the drop-down list, click **Browse** next to the **Exec Path on Target** field and navigate to the executable to run[3] (if it does not already appear), or change any of the other settings in this section.



NOTE: If your application is *not* built as described in *17.6 Executables that Dynamically Link to Shared Libraries*, p.229, you must set the **LD_LIBRARY_PATH** environment variable. See that section for details.

For more information on the fields on the Main tab, open the dialog and press the help key for your host.

3. Workbench automatically maps the pathname from your host file system into a pathname that is valid on the target file system. To change the mappings, click **Properties**, scroll right to the **Object Path Mappings** tab, highlight the mapping you want to change, click **Edit**, then update and save your new settings.

22.2.3 Specifying a Build Target to Download

If you want Workbench to download a particular build target each time this launch is used, specify it on the **Downloads** tab (this is necessary only for kernel task launches). If you highlighted a build target in the Project Navigator before opening the launch dialog, the file appears in the **Downloads** list automatically.

1. To modify any of the settings of the output file that appears, click **Edit**.

   To add a file or to specify additional files to be downloaded, click **Add**.

   In both cases, the **Download** dialog appears. For details about the fields in this dialog, open the dialog and press the help key for your host.

2. When you are finished adjusting the settings, click **OK**. The new information appears in the **Downloads** list.



NOTE: You can also create launches for kernel tasks that are already downloaded, or are resident in Flash or are part of the kernel image. Those tasks do not require an entry in the **Downloads** list since they do not need to be downloaded each time the configuration is run.

22.2.4 Specifying The Projects to Build

If you want Workbench to build a particular project or projects prior to launching this configuration, specify them on the **Projects to Build** tab. If you selected a build target in the Project Navigator, its project appears in the **Projects to Build** list automatically.

1. To add another project to the list, click **Add Project**, select one or more projects, then click **OK**.
2. To rearrange the build order in the list, select a project then click **Up** or **Down**.
3. If you do not want Workbench to build for this particular launch configuration, such as when you are working with very large projects, select all projects and select **Remove** to clear the list[4].

4. To prevent Workbench from building prior to launching any of your programs, unselect **Window > Preferences > Run/Debug > Launching > Build (if required) before launching**.



NOTE: Workbench is aware of relationships between projects and subprojects. So if **myLib** is a subproject of **myProj** and you choose to add **myProj** to the list, you cannot add **myLib** to the list as well because it will be built automatically when you build **myProj**. Adding **myLib** as well would be redundant and so is disabled.

When you change the list of downloaded files for kernel task launches (see *Specifying a Build Target to Download*, p.286) the projects containing those files are automatically added to the **Projects to Build** list. You should always review this list when you change the list of downloaded files.

22.2.5 Defining Debug Behavior

Break on Entry

When creating debug-mode launches, **Break on entry** is selected by default. Uncheck it if you want this program to run to the first breakpoint you set, rather than breaking immediately after startup.

If **Break on entry** is selected when the launch is run, four things happen:

- Workbench automatically switches to the Device Debug perspective (if it is not already open).[5]
- The task or process is displayed in the Debug view.
- A temporary breakpoint is planted and appears in the Breakpoints view.
- The program executes up to **Entry Point** and breaks.

Automatically Attach Spawned Kernel Tasks

For kernel task launches, select this option if you want Workbench to automatically attach spawned kernel tasks.

5. From the View Management Preferences screen (**Window > Preferences > Run/Debug > View Management**) you can control under what circumstances Workbench switches views based on your selection.

22.2.6 Specifying Where Workbench Should Look for Source Files

If your build target was compiled on the same host where you want to debug it, you do not need to change anything on the **Source** tab.

However, if the build target was compiled on a different host, and Workbench needs to find source files during debugging, it searches the locations listed on this tab in the specified order.



NOTE: If you do not specify a source lookup path, the debugger will ask for the correct source path as soon as it encounters a source it cannot find. So if you prefer, you can configure the source lookup manually as you go, rather than configuring it when creating the launch.

1. On the **Sources** tab, click **Add** to configure the source lookup path.
2. Select the type of source to add, then click **OK**.
3. Most choices require that you select a specific project, folder, or path. Make your selection, then click **OK**.
4. Click **Up** or **Down** to adjust the search order.
5. Check **Search for duplicate source files on the path** to have Workbench search the entire source lookup path and offer you a choice of all the files it finds that have the same filename, rather than automatically using the first file of that name it encounters.

For more information about the source locator, see *24.6 Understanding Source Lookup Path Settings*, p.326, and open the dialog and press the help key for your host.

22.2.7 Configuring Access Methods

Use the **Common** tab to specify whether this launch is local or shared, to add the launch to the Workbench toolbar favorites menus, and to indicate whether the program should be launched in the background or not.

1. By default this launch configuration is a local file available only to you. If you want to share it with others on your team, click **Shared**, then type or browse to the directory where you want to save the shared file.
2. If you want to be able to launch this program from the **Run** or **Debug** favorites menus (the drop-down menus on the Workbench toolbar), select **Run** or **Debug** in the **Display in favorites menu** box.

22.2.8 Using Your Launch Configuration

When you are finished configuring the launch configuration for your program, click **Apply** to save your settings but leave the dialog open, click **Close** to save your launch configuration for later use, or click **Run** or **Debug** to launch it now.

Running Your Program

If you select **Run** to launch your program, the output file or executable is loaded into target memory and its name and host location appear below your target connection in the Target Manager (RTPs appear under Real-time Processes). A red **S** over the output file icon indicates that symbol information has been downloaded to the debugger.



NOTE: If no symbol information was found, right-click the module and select **Load Symbols** to load the symbols for your module from an alternate location.

You can also match module paths with symbol information by selecting the **Create path mappings based on selection** checkbox in the Load Symbols dialog.

Debugging Your Program

If you select **Debug** to launch your program, in addition to loading the output file or executable into target memory and downloading symbol information, the debugger attaches to the task or process that then appears in the Debug view. For more information about debugging your programs, see *24. Debugging Projects* and open the Debug view and press the help key for your host.

22.3 Reset & Download: Hardware Debugging Launches

For information about creating a Reset and Download launch configuration, see *Wind River ICE SX for Wind River Workbench Hardware Reference: Establishing Communications* or *Wind River Probe for Wind River Workbench Hardware Reference: Establishing Communications*, depending on whether you are using a Wind River ICE SX or Wind River Probe for your OCD connection.

22.4 Launching a Native Application

1. To create a new launch configuration that will run a native application on your local host or remote host, select your application's executable in the Project Navigator then select **Run > Run**. The **Create, manage, and run configurations** dialog appears.
2. From the **Configurations** list, select **Native Application**, then click **New**.
3. The default name of the new configuration is **New_configuration**. Type a descriptive name in the **Name** field.

22.4.1 Specifying the Location and Arguments for Your Application

1. To specify the location of your application's executable file, click **Browse Workspace** near the **Location** field. The **Select an application** dialog opens.
2. Select the executable and click **OK**. The executable appears in the **Location** field.
3. To specify the working directory for your application, click **Browse Workspace** to open the **Select a working directory** dialog, or **Browse File System** to open the **Browse for Folder** dialog.
4. Select a working directory, then click **OK**. The directory appears in the **Working Directory** field.
5. Type the arguments your application requires into the **Arguments** field, or click **Variables** to open the **Select Variable** dialog. Double-click the variable you want to use, or select it and click **OK** to add it to the **Arguments** field.

22.4.2 Specifying Remote Settings

These settings are optional, and are required only if you are running your application on a remote host. For more information about working with remote hosts, see *17.9.5 Running Applications Remotely*, p.239.

Command-line application's output and input will be redirected to the standard Eclipse console unless the application is started within an external process that creates a new window, such as **xterm**.

1. If your application requires an interactive shell, type the program and arguments in the **Remote Program** field. The default for remote execution is a

remote command like **xterm -e %Application%**, so a local X-server like Exceed or Cygwin X must be running.

2. If you want to use a different working directory than the one specified on the Arguments tab, type the path to the desired directory (as seen on the remote host).

22.4.3 Setting Environment Variables

These settings define the environment variable values to use when running a Java application. By default, the environment is inherited from the Eclipse run time. You may override or append to the inherited environment.



NOTE: These settings apply to applications that run locally, not to remote applications.

1. To set a new environment variable, or to change or extend variables from the existing environment, click **New**. The **New Environment Variabl**e dialog opens.
2. Type a descriptive name for the variable.
3. Type the value for the variable, or click **Variables** and select the desired variable, add any required arguments, then click **OK**.
4. To include an existing environment variable, click **Select**. The Select Environment Variables dialog opens.
5. Select the checkbox next to the desired variable, then click **OK**.
6. For each variable, choose whether to append it to the native environment or substitute it for the native environment.

22.4.4 Configuring Access Methods

Use the **Common** tab to specify whether this launch is local or shared, to add the launch to the Workbench toolbar favorites menus, and to indicate whether the program should be launched in the background or not.

1. By default this launch configuration is a local file available only to you. If you want to share it with others on your team, click **Shared**, then type or browse to the directory where you want to save the shared file.

2. If you want to be able to launch this program from the **Run** favorites menu (the drop-down menu on the Workbench toolbar), select **Run** in the **Display in favorites menu** box.

22.4.5 Running Your Native Application

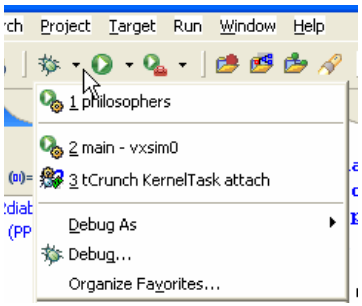
When you are finished configuring the launch configuration for your application, click **Apply** to save your settings but leave the dialog open, click **Close** to save your launch configuration for later use, or click **Run** to launch it now.

22.5 Relaunching Recently Run Programs

In a typical development scenario, you will run the same application many times in a single debugging session. After creating a launch configuration, you can click the **Run** or **Debug** icon or use a keyboard shortcut to run a process and attach the debugger in a few seconds.

To relaunch a recently run program:

- Press **CTRL+F11** to launch the last run-mode configuration you used, or **F11** to launch the last debug-mode configuration you used.
- Click the drop-down arrow next to the **Run** or **Debug** icon and select the configuration from the list. If you ran the configuration recently, it will appear on the menu. If you selected **Run** or **Debug** from the **Display in favorites menu** list (see *Configuring Access Methods*, p.288) it will always appear on the list, whether you have run it recently or not.



- To run a configuration not listed on the favorites menu, click **Run > Run** or **Run > Debug**, then choose the configuration from the **Configurations** list and click **Run** or **Debug**.

22.5.1 Increasing the Size of the Launch History List

Workbench stores a history of previously launched configurations. The default length of the launch history is 10, but you can increase the history length by selecting **Window > Preferences > Run/Debug > Launching** and increasing the number in the **Size of recently launched applications list** field.

22.6 Controlling Multiple Launches

You can create a Launch Control launch, consisting of a sequence of your launch configurations, each one of which is then considered a sub-launch. You can even add other Launch Control launches to a Launch Control configuration, the only restriction being that a Launch Control configuration cannot contain itself.

For detailed information on launch control settings, open the dialog and press the help key for your host.

Terminology

A *launch* is a specific instance of a *launch configuration*, and a launch configuration is a specific instance of a *launch type*. The launch is what occurs when you initiate a run or debug session.

A launch configuration is your definition of how the launch will occur, for example, what program will be run, what target it will run on, and what the arguments are.

A launch type defines the kind of launches that are supported by Workbench. There are several different kinds of launch types, for example, **Kernel Task** or **RTP on Target.** The launch type includes GUI elements that specify attributes specific to it.

You create a launch configuration based on a launch type, specifying the appropriate attribute values. You then initiate a launch based on a launch configuration. Launches also have a *mode*, the two standard modes being Run and Debug. A launch may be initiated by the **Run** or **Debug** buttons in Workbench (launches may be initiated other ways too).

Configuring a Launch Sequence

The following procedure assumes you have two or more launch configurations already defined.

1. Select **Run > Debug** and the Debug dialog opens.
2. Select **Launch Control** from the **Configurations** list on the left, and then click **New** at the bottom. A new launch control configuration with the default name **New Configuration** appears. Change the name as desired.
3. Select the **Launch Control** tab. Note that your current launch configurations are listed under **Available Configurations** on the left, and a space on the right is labeled **Configurations to Launch**.
4. Select each launch that you want to add to your new launch configuration and click **Add** to add it to the list of configurations to launch. When you have a list of configurations to launch, you can organize them in the order you want them to launch by selecting a configuration and clicking **Move Up** or **Move Down**. The sub-launch at the top of the list will come first and the one at the bottom last. You can remove any sub-launch from the Launch Control configuration by selecting it and clicking **Remove**.

You now have a Launch Control configuration that will launch a sequence of sub-launches in the order specified in the **Configurations to Launch** list. You can also specify commands to perform before launches, after launches, and in response to a launch failure or an application error report as discussed in the next section.

Each launch in a Launch Control will open a Console view for I/O and error messages as described in *22.7 Launches and the Console View*, p.298.

Pre-Launch, Post-Launch, and Error Condition Commands

To access the launch configuration commands, select a sub-launch in your **Configurations to Launch** list and click **Properties** (or double-click the sub-launch). A properties page containing command information is displayed. Here you can specify pre-launch, post-launch, and error condition commands,

which will inherit the environment variables shown below them unless you change them in the command. Your changes affect the launch you are working with only—other launches using the same configuration get the default values for the environment variables. Also, the set of environment variables differs for each launch configuration (see *Understanding the Command Environment*, p.296 for more on environment variables).

Preparing a Launch with a Pre-Launch Command

An example of the use of a pre-launch command is to prepare a target for use. For example, in a development environment you might have to reserve a target, and you would not want to attempt a launch without being sure you had a target to launch on. So a pre-launch command might be a script that reserves the board and then reboots it.

If the pre-launch command returns a non-zero return code then the launch is aborted and the error condition command is executed for each sub-launch previous to the failed sub-launch.

Using a Post-Launch Command

If your application requires additional set up after it has been launched, or if you would like to verify that it has launched correctly before proceeding to the next launch, use a post-launch command.

If the post-launch command returns a non-zero return code then the launch is aborted and the error condition command is executed for each sub-launch previous to the failed sub-launch as well as for the failed sub-launch.

Using the Error Condition Command

The error condition command of a launch is run when a launch fails, or a pre-launch or post-launch command returns a non-zero error code. This causes the error command of the current launch to run, and then each error command of any preceding launches to run. The error condition commands are executed in reverse order of the sequence in which the launches occurred. For example, if the fourth launch fails, the error condition command of the fourth launch is performed, then the error condition of the third launch, and so on. This is to deal with situations in which previous commands may have acquired locked resources--unlocking them in reverse order is important to prevent potential deadlock.



NOTE: To be precise, error commands are called in the reverse order that the pre-launch commands were called. An error command will never be called for a sub-launch that did not pass the pre-launch command step.

Inserting Commands using an Empty Sub-Launch

You can place a command into your Launch Control that is not associated with any particular sub-launch by adding an empty Launch Control to hold the command. Select Launch Control and click **New** and then specify a name for the dummy launch, for example, **Empty Launch**. Add the empty launch to the Launch Control and use the properties page to insert commands into the launch which aren't associated with any particular sub-launch.

Running All Pre-Launch Commands First

If you want to run each of the pre-launch commands for each launch first, check **Run Pre-Launch command for all launches first** on the main launch control page. The pre-launch commands will be executed in order, and only after they are all successfully completed will the first launch take place, followed by the second launch and so on. This provides for situations in which you do not want to continue with a complete launch Control sequence if any of the sub-launches cannot take place because, for example, a target is not available.

Launch Controls as Sub-Launches

You can use an existing Launch Control as a sub-launch, but do not attempt to create recursive launch controls in this way, as they will not run.

If the parent Launch Control's pre-initialize check box (**Run Pre-Launch command for all launches first**) is selected and the pre-initialize check box is set for the child Launch Control, the child will pre-initialize all of its sub-launches before operation continues on to the next sub-launch of the parent Launch Control. Otherwise, the child Launch Control will have its sub-launches initialize at the time that it is launched.

Understanding the Command Environment

The environment variables are collected from multiple locations and then provided on the Properties page as a convenience. Typically you will only read variable values, but you may want to change them in your pre-launch command. Your changes affect the launch you are working with only—other launches using the same configuration get the default values for the environment variables.

Environment variables are gathered from four different sources. First, variables may be defined on the Launch Control's **Environment** tab. These variables are not displayed on a sub-launch's Properties page because the information is readily available on the **Environment** tab. The next source for environment variables is from the sub-launch's **Environment** tab (if it has one). The third source for the list of environment variables is defined by the sub-launch's configuration type

attributes. Each sub-launch configuration type defines its own set of attributes (further documentation on sub-launch attributes can be found in the Eclipse documentation for Launch Configuration). The final source of environment variables are defined by Launch Control and provide general support for the launch. The variables defined by Launch Control for each sub-launch are:

- **com_windriver_ide_launchcontrol_launch_mode**
- **com_windriver_ide_launchcontrol_env_file**
- **com_windriver_ide_launchcontrol_skip_next**

The environment variable **com_windriver_ide_launchcontrol_launch_mode** identifies the mode of a launch. The mode may be either **debug** or **run**, depending on how a launch is initiated (for example selecting the **Run > Debug** dialog to initiate a debug mode launch and **Run->Run** to initiate a run mode launch). Changing **com_windriver_ide_launchcontrol_launch_mode** has no effect—it is only provided for information about a current launch.

Since the command's environment terminates after the command completes any variables which need to be changed for a launch must be written to a file. The name of this file is provided in the environment variable **com_windriver_ide_launchcontrol_env_file**. The format of this file is a list of key value pairs on separate lines. Each key and value is separated by an = and the key identifies the variable name (this is a standard Java properties file). After a command is completed Launch Control will read this file and update any variables as specified in the file.

Launch control also defines the **com_windriver_ide_launchcontrol_skip_next** variable. Setting this variable to **true** in the Pre-Launch command causes the remainder of the sub-launch to be skipped. Setting this variable in post-launch or error commands has no effect.

An example of how this could be used is to check for the existence of a server application in a pre-launch command. If the application is already running then specifying **com_windriver_ide_launchcontrol_skip_next=true** in the **com_windriver_ide_launchcontrol_env_file** will cause the launch of the application to be skipped without invoking an error.



NOTE: Note that the Wind River environment variables for individual launches are subject to change and you should not count on them being maintained across releases. For details on variables beginning with the string **org_eclipse** refer to the documentation available at **http://help.eclipse.org**.

22.7 Launches and the Console View

Workbench supports the Eclipse **Console** view with Virtual IO (VIO) features that allow you to monitor the standard output and error output of your applications and to enter standard input. VIO connects the **Console** view to a particular context (process or task). You can also have multiple **Console** views and "pin" them to a particular context. Most **Console** view settings are available in the **Common** tab of your launch configuration, and you can specify **Console** view preferences in your Workbench preferences.

Note that **Console** view VIO is tied to the debugger and cannot always serve the same purposes as a terminal connection to the target. You cannot use it, for example, to monitor the boot loader or set boot parameters. The **Console** view is associated with a particular debugger context and is not a general purpose terminal connection.

Launches and the Console View

Each launch opens a Console view for I/O and error messages, provided the **Allocate Console** check box is selected in the Common tab of the launch (the default setting).



NOTE: This refers to the Common tab of each individual launch configuration, not the Common tab of the Launch Control configuration.

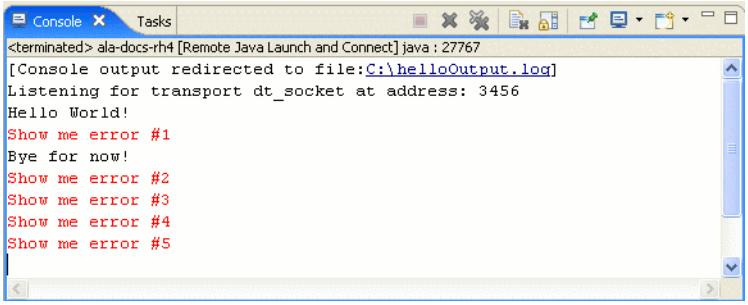
In the Common tab you can also specify a file where console output is appended or overwritten with each launch. The Console view itself offers several controls as described in the next section.

Note that you can also modify Console view settings such as buffer size and text colors by selecting your preferences at
Window > Preferences > Run/Debug > Console.

Console View Output

To open a **Console** view select **Window > Show View > Other > Basic > Console**. An example view is shown below.

Figure 22-1 **Example Console View**



The highlights of the view shown include the following:

- A title indicates which context (process or task) this view applies to.
- A comment indicates that in this case console file logging is occurring and identifies the log file location. Click on the filename to display it in the Editor.
- The standard output shown in the example is **Hello World!** and **Bye for now!** and is in black, the default color for standard output.
- The standard error outputs shown in the example are the **Show me error** messages which are in red, the default color for standard error output.

NOTE: The output appearing in the Console View can appear in a different order than the order the output was produced if both output and error output are present. The data from these two output types go through different channels and their transit times can be different.

Along with other standard functions, icons in the **Console** view toolbar allow you to pin the context to a Console view, select among different Console views, and create new Console views.

Select a specific process or task for a Console view by clicking the down arrow next to the **Display Selected Console** icon and making your selection. Click **Pin Console** to keep the Console view associated with that context. Select **Open Console > New Console View** to create additional Console views.

Refer to **http://help.eclipse.org** for further details on the Console view, or open the Console view and press the help key for your host.

22.8 Using Attach-to-Target Launches

Workbench automatically creates **Attach to Target** launch configurations when you attach to an individual process or kernel task from the Target Manager. They do not actually run an application, they just connect to your target and attach the debugger to the specified task or process that already exists. These configurations are visible only in Debug mode.

Once **Attach to Target** launches are created, you can:

- Review them and delete those that you no longer need.
- Change which target connection should be used to run the process.
- Rename your launch configurations, and if you think they are valuable, put them into your **Favorites** menu using the **Common** tab.
- Change the mapping between source paths compiled into your objects and source paths in your workspace by editing the Source Locator information in the **Sources** tab.
- Change the **Projects to Build** settings for the launch. This is particularly valuable for **Attach to Kernel** launches on the VxWorks simulator: you can disconnect your simulator, rebuild your kernel as part of the launch, and then let the launch automatically restart and reconnect the simulator. Automatically rebuilding shared libraries is another use of **Build before launching**.



NOTE: When you attach to a process or task with the same name using the same connection, Workbench automatically reuses all the settings from the previous launch.

However, Workbench creates a new launch (requiring you to reconfigure the settings) when it detects that the properties of the connection have changed: for example, if the connection was renamed, a different kernel image was used, or the target server arguments or other connection properties were changed.

One way to avoid accumulating many similar launches is to make your configuration changes in the launch itself, rather than right-clicking a process in the Target Manager and selecting **Attach**. That way Workbench will always have the correct settings for the process you want to run.

22.8.1 Attaching the Debugger to a Running Task or Process

To attach the debugger to a task or RTP that is already running, right-click it in the Target Manager and select:

- **Attach to Real-time Process** to attach to a Real-time Process on VxWorks.
- **Attach to Kernel Task** to attach to a kernel task on VxWorks.
- **Attach to Process** to attach to a process on Linux.

Whenever you manually attach an individual process or task, Workbench automatically switches to the Device Debug perspective (if it is not already open) and displays the task or process in the Debug view, the debugger attaches without stopping the program, and Workbench automatically creates a corresponding **Attach-to-Target** launch configuration with those properties. For more information about how to use **Attach-to-Target** configurations, see *22.8 Using Attach-to-Target Launches*, p.300.

Comparing Definitions: Running, Suspended, and Stopped Tasks

VxWorks and the Workbench Debug view both make a distinction between running, suspended, or stopped tasks, but their definitions are not identical.

VxWorks	Workbench Debug View	Definition
Running	Running	Task is active, and has focus.
Suspended	Running	Task is waiting while another task runs.
Stopped	Suspended	Task stopped at a breakpoint or other event, or was stopped by user.

22.8.2 Attaching the Debugger to the Kernel

The debugger functions differently depending on whether you attach to the kernel in task mode or system mode.

22.8.3 Attaching the Kernel in Task Mode

To attach to the kernel in Task Mode[6] (VxWorks), right-click the **Kernel Tasks** node in the Target Manager and select **Attach All Kernel Tasks**.

6. Task mode is also known as user mode.

The debugger will automatically track added and removed kernel tasks so that you can always debug the entire system. You can also stop (suspend) individual kernel tasks, unless they have the **VX_UNBREAKABLE** option set. When you stop a kernel task, the rest of the system will continue to run.

22.8.4 Attaching the Kernel in System Mode

To attach the kernel in System Mode (VxWorks and Linux dual-mode agent), right-click the CPU icon below the Connection icon and select **Attach-to-Kernel**.

This will create an **Attach-to-Target** launch configuration that automatically switches your target into System Mode before attaching the debugger. The Debugger will show a single node labelled **System Context** that represents the code that the CPU is currently executing. When you stop (suspend) the System Context, your entire System is stopped, including all the tasks, processes, and interrupt service routines. You can now also set breakpoints that will suspend the entire system when they are hit.

In addition to the single System Context node in the debugger, you can also attach to individual kernel tasks. This will create separate debug sessions. You can also set breakpoints that are specific to the task that is currently executing by selecting **restrict breakpoint scope to task** on the Scope tab of the breakpoint dialogs (for more information, open the line, expression, and hardware breakpoint dialogs and press the help key for your host).

Note that System Mode breakpoints (breakpoints that are planted while a System Mode attach is active) will only be active when your target is in System Mode. You can switch your target between System Mode and User Mode by choosing the gear-wheel icon in the Target Manager, or by ticking the **Debug Mode** menu items in the Debugger. For more information about Debug Mode functionality, see *24.5 Using Debug Modes*, p.320.

22.9 Suggested Workflow

Launch Configurations allow for a very tight Edit-Compile-Debug cycle when you need to repeatedly change your code, build and run it. You can use the **F11** (Debug Last Launched) key to build the projects you have specified, connect your target

(unless it is already connected), download, and run your most important program over and over again.

The only thing to keep in mind is that it may not be possible to rebuild your program or kernel while it is still being debugged (or its debug info is still loaded into the debugger). Workbench will warn you with a dialog and suggest proper actions in case a problem of such simultaneous usage is detected. Depending on the size of the modules you run and debug, it can be the case that the debug server cannot load all the symbolic information for your modules into memory. By default, the size limit is set to 60MB (this can be changed by selecting **Preferences > Target Manager > Debug Server Settings > Symbol File Handling Settings**.)

If a module is bigger than this limit, it will be locked against overwriting as long as the debugger has symbols loaded. This means that when you try to rebuild this module, you will see a dialog asking you to unload the module's symbol information from the debugger before you continue building. You can usually unload symbolic information without problems, provided that you do not have a debug session open in the affected module. If you have a module open, you should terminate your debug session before continuing the new build and launch process.

23

Managing Breakpoints

23.1 Introduction 305

23.2 Types of Breakpoints 305

23.3 Manipulating Breakpoints 309

23.1 Introduction

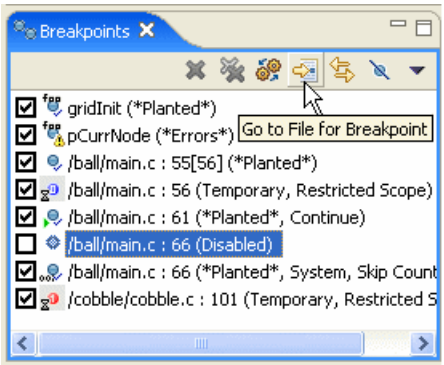
Breakpoints allow you to stop a running program at particular places in the code or when specific conditions exist. This chapter shows how you can use the Breakpoints view to keep track of all breakpoints, along with any conditions.

You can create breakpoints in different ways: by double-clicking or right-clicking in the Editor's left overview ruler (also known as the *gutter*), by opening the various breakpoint dialogs from the pull-down menu in the Breakpoints view itself, or by selecting one of the breakpoint options from the **Run** menu.

23.2 Types of Breakpoints

Figure 23-1 shows the Breakpoints view with various types of breakpoints set.

Figure 23-1 **Breakpoints View**



See the sections below for when and how to use each type of breakpoint. For a guide to the icons you will see in the Breakpoints view, open the view and press the help key for your host.

23.2.1 Line Breakpoints

Set a line breakpoint to stop your program at a particular line of source code.

Creating Line Breakpoints

To set a line breakpoint with an unrestricted scope (that will be hit by any process or task running on your target), double-click in the left gutter next to the line on which you want to set the breakpoint. A solid dot appears in the gutter, and the Breakpoints view displays the file and the line number of the breakpoint. You can also right-click in the gutter and select **Add Global Line Breakpoint**.

To set a line breakpoint that is restricted to just one task or process, right-click in the Editor gutter and select **Add Breakpoint for** *selected thread*. If the selected thread has a color in the Debug view, a dot with the same color will appear in the Editor gutter, with the number of the thread inscribed inside it.

Right-clicking in the Editor's gutter and selecting **Add Line Breakpoint**, or selecting **Add Line Breakpoint** from the Breakpoints view's pull-down menu will open the **Line Breakpoint dialog**, where you can create and adjust the properties of the breakpoint.

For more information about the settings in this dialog, open the dialog and press the help key for your host.

23.2.2 Expression Breakpoints

Set an expression breakpoint using any C expression that will evaluate to a memory address. This could be a function name, a function name plus a constant, a global variable, a line of assembly code, or just a memory address. Expression breakpoints appear in the Editor's gutter only when you are connected to a task.

Breakpoint conditions are evaluated after a breakpoint is triggered, in the context of the stopped task or process. Functions in the condition string are evaluated as addresses and are not executed. Other restrictions are similar to the C/C++ restrictions for calculating the address of a breakpoint using the Expression Breakpoint dialog.

Select **Add Expression Breakpoint** from the Breakpoints view's pull-down menu to open the **Expression Breakpoint dialog**, where you can create and adjust the properties for the breakpoint.

For more information about the settings in this dialog, open the dialog and press the help key for your host.

23.2.3 Hardware Breakpoints

Some processors provide specialized registers, called debug registers, which can be used to specify an area of memory to be monitored. For instance, IA-32 processors have four debug address registers, which can be used to set data breakpoints or control breakpoints.

Hardware breakpoints are particularly useful if you want to stop a process when a specific variable is written or read. For example, with hardware data breakpoints, a hardware trap is generated when a write or read occurs in a monitored area of memory. Hardware breakpoints are fast, but their availability is machine-dependent. On most CPUs that do support them, only four debug registers are provided, so only a maximum of four memory locations can be watched in this way.

There are two types of hardware breakpoints:

- A hardware *data breakpoint* occurs when a specific variable is read or written.

- A hardware *instruction breakpoint* or *code breakpoint* occurs when a specific instruction is read for execution.

Once a hardware breakpoint is trapped—either an instruction breakpoint or a data breakpoint—the debugger will behave in the same way as for a standard breakpoint and stop for user interaction.

Adding Hardware Instruction Breakpoints

There two ways to add a new hardware instruction breakpoint:

In the gutter (grey column) on the left of the source file, right-click and select **Add Hardware Code Breakpoint**. Or, double-click in the gutter to add a standard breakpoint and then, in the Breakpoints view, right-click the breakpoint you've just added and select **Properties**. In the last pane (**Hardware**) of the **Properties** dialog select **Enable Hardware Breakpoint**.

Adding Hardware Data Breakpoints

Set a hardware data breakpoint when:

- The debugger should break when an event (such as a read or write of a specific memory address) or a situation (such as data at one address matching data at another address) occurs.
- Threads are interfering with each other, or memory is being accessed improperly, or whenever the sequence or timing of runtime events is critical (hardware breakpoints are faster than software breakpoints).

Select **Add Data Breakpoint** from the Breakpoints view's pull-down menu to open the **Hardware Data Breakpoint dialog**, where you can create and adjust the properties for the breakpoint.

For more information about the settings in this dialog, open the dialog and press the help key for your host.

Converting Line or Expression Breakpoints Into Hardware Code Breakpoints

To cause the debugger to request that a line or expression breakpoint be a hardware code breakpoint, select the **Hardware** check box on the **Hardware** tab of the **Line Breakpoint** or **Expression Breakpoint** dialogs.

This request does not guarantee that the hardware code breakpoint will be planted; that depends on whether the target supports hardware breakpoints, and if so, whether or not the total number supported by the target have already been planted. If the target does not support hardware code breakpoints, an error message will appear when the debugger tries to plant the breakpoint.



NOTE: Workbench will set only the number of code breakpoints, with the specific capabilities, supported by your hardware.



NOTE: If you create a breakpoint on a line that does not have any corresponding code, the debugger will plant the breakpoint on the next line that does have code. The breakpoint will appear on the new line in the Editor gutter.

In the Breakpoints view, the original line number will appear, with the new line number in square brackets **[ ]** after it. See the third breakpoint in Figure 23-1.

Comparing Software and Hardware Breakpoints

Software breakpoints work by replacing the destination instruction with a software interrupt. Therefore it is impossible to debug code in ROM using software breakpoints.

Hardware breakpoints work by comparing the break condition against the execution stream. Therefore they work in RAM, ROM or flash.

Complex breakpoints involve conditions. An example might be, “Break if the program writes *value* to *variable* if and only if **function_name** was called first.”

23.3 Manipulating Breakpoints

Now that you have an understanding of the different types of breakpoints, this section will show you how to work with them.

23.3.1 Importing Breakpoints

To import breakpoint properties from a file:

1. Select **File > Import > Import Breakpoints**, then click **Next**. The Import Breakpoints dialog appears.
2. Select the breakpoint file you want to import, then click **Next**. The Select Breakpoints dialog appears.
3. Select one or more breakpoints to import, then click **Finish**. The breakpoint information will appear in the Breakpoints view, and the next time the context for that breakpoint is active in the Debug view, the breakpoint will be planted.

23.3.2 Exporting Breakpoints

To export breakpoint properties to a file:

1. Select **File > Export > Export Breakpoints**, then click **Next**. The Export Breakpoints dialog appears.
2. Select the breakpoint whose properties you want to export, and type in a file name for the exported file. Click **Finish**.

23.3.3 Refreshing Breakpoints

Right-clicking a breakpoint in the Breakpoints view and selecting **Refresh Breakpoint** causes the breakpoint to be removed and reinserted on the target. This is useful if something has changed on the target (for example, a new module was downloaded) and the breakpoint is not automatically updated.

To refresh all breakpoints in this way, select **Refresh All Breakpoints** from the Breakpoints view toolbar drop-down menu.

23.3.4 Disabling Breakpoints

To disable a breakpoint, clear its check box in the Breakpoints view. This retains all breakpoint properties, but ensures that it will not stop the running process. To re-enable the breakpoint, select the box again.

23.3.5 Removing Breakpoints

There are several ways to remove a breakpoint:

- right-click it in the Editor gutter and select **Remove Breakpoint**

- select it in the Breakpoints view and click the **Remove** icon
- right-click it in the Breakpoints view and select **Remove**

For more information about the Breakpoints view or any of the breakpoint dialogs, open the dialogs and press the help key for your host.

24 Debugging Projects

24.1 Introduction 313

24.2 Using the Debug View 314

24.3 Coloring Views 318

24.4 Stepping Through a Program 319

24.5 Using Debug Modes 320

24.6 Understanding Source Lookup Path Settings 326

24.7 Using the Disassembly View 326

24.8 Using the Kernel Objects View 328

24.9 Remote Kernel Metrics 331

24.10 Run/Debug Preferences 336

24.1 Introduction

Like other debuggers you may have used, the Wind River Workbench debugger allows you to download object modules, launch new processes, and take control of processes already running on the target.

Unlike other debuggers, it allows you to attach to multiple processes simultaneously, without affecting the state of the items you are attaching to or requiring you to disconnect from one process in order to attach to another.

This chapter shows you how to use the Debug, Disassembly, and Kernel Objects views to debug your programs. For a guide to the dialogs and icons you will see while using them, open the views and press the help key for your host.

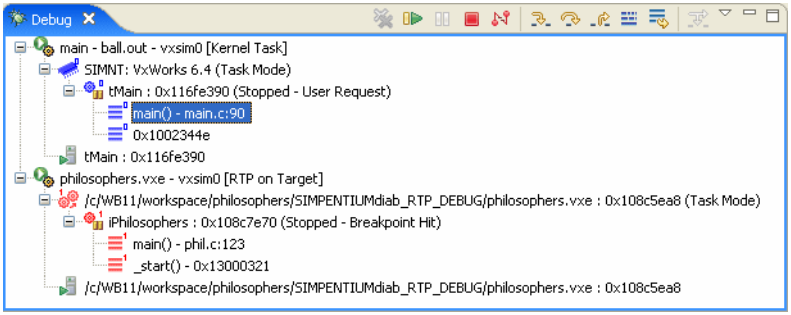
24.2 Using the Debug View

Use the Debug view to monitor, control, and manipulate the processes and tasks that you are actively debugging. Unlike the Target Manager, which shows all the processes that exist on the target, the Debug view shows only the ones that are currently under debugger control or were launched by Workbench.

To put a process or task under the control of the debugger and thus see it in the Debug view:

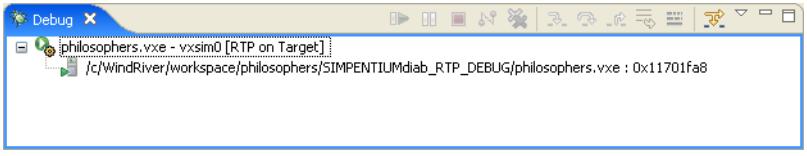
1. Connect to your target in the Target Manager view (see *Connecting to the Target*, p.259).
2. Launch one or more processes:
   - Using a launch configuration as described in *Relaunching Recently Run Programs*, p.292.
   - By attaching to an already running process, as described in *Attaching the Debugger to a Running Task or Process*, p.301
3. Once the debugger has attached to your process, it will appear in the Debug view as shown in Figure 24-1.

Figure 24-1 **Debug View**



Additionally, the Debug view shows processes that were launched on the target using Workbench, but which were not attached by the debugger. These launches have a special entry in the Debug view, as shown in Figure 24-2, and are only available to help you locate and terminate the process.

Figure 24-2 **Debug View Showing Process Not Under Debugger Control**



24.2.1 Understanding the Debug View Display

When using the Debug view, it is crucial that you understand what is represented by each level in the hierarchical tree of the process you are debugging. This is because the level of the current selection in the Debug view affects the activities that you can perform on it and controls the information displayed in other views.

Below are examples from the kernel task in Figure 24-1 for what might appear at each level of the tree, with a general description of each level.

main -ball.out - vxsim0 [Kernel task] = launch level
launch name [launch type]

> **SIMNT: vxWorks 6.*x* (Task Mode)** = debug target level
> *core name*:*OS name OS version (debug mode)*, can also be *process name*

tMain (Stopped - Breakpoint Hit) = thread level
thread name (state - reason for state change)

main( ) - main.c:59 = stack frame level
function(args) - file : line #, can also be *address*

In Workbench 2.6, stack arguments and argument values are not displayed in the Debug view by default. This default setting was implemented to improve debugging performance.

To activate stack-level arguments in the Debug view, select **Window > Preferences > Run/Debug > Performance**, then select the **Retrieve stack arguments for stack frames in Debug View** and **Retrieve stack argument values for stack frames in Debug View** checkboxes. Click **OK**.



NOTE: The stack arguments reflect the current value of the stack argument variables, not the initial value of the stack arguments immediately after entering the function call.

How the Selection in the Debug View Affects Activities

Choosing a specific level of your debug target controls what you can do with it.

Selected Level	Action Allowed
launch	Terminate or disconnect from all processes/cores for the launch debug target.
debug target	Terminate or disconnect from the debug target.
	Perform run control that applies to the whole process: suspend/resume all threads.
	Assign color to the debug target and all its threads/tasks.
thread	Terminate or disconnect; terminates individual tasks/threads, if supported by process/core.
	Run control for thread: resume/suspend/step.
	Assign color to thread.
stack frame	Select of the stack frame causes the editor to display instruction pointer and source for stack frame.
	Perform same run control as on the thread.

Assign color to thread.

Assign corresponding color for parent thread.

Monitoring Multiple Processes

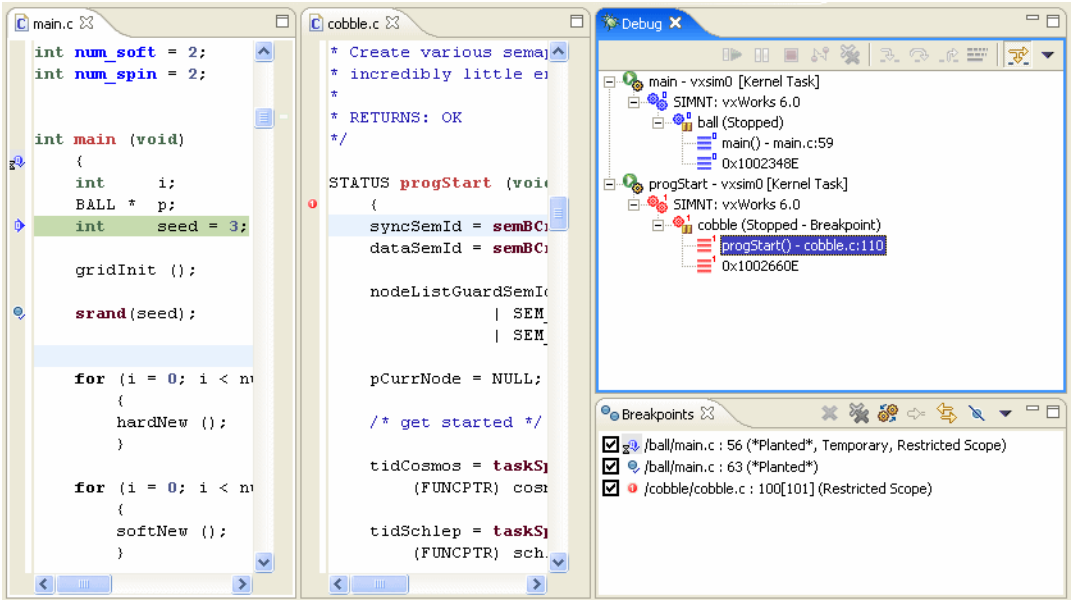
When you start processes under debugger control, or attach the debugger to running processes, they appear in the Debug view labeled with unique colors and numbers. Likewise, breakpoints that are restricted to a particular process display that process's color/number context in the Breakpoints and Editor views.

For example, in Figure 24-3:

- The first breakpoint in **main.c** (a blue circle containing a 0) is restricted to ball, the blue process numbered 0 in the Debug view.
- The second breakpoint (a solid blue-green circle) is unrestricted.
- The breakpoint in **cobble.c** (a red circle containing a 1) is restricted to cobble, the red process numbered 1 in the Debug view.

The color assigned to a process or thread can be changed by right-clicking the process or thread and selecting **Color >** *specific color*.

Figure 24-3 **Debug View with Breakpoint and Editor Views**



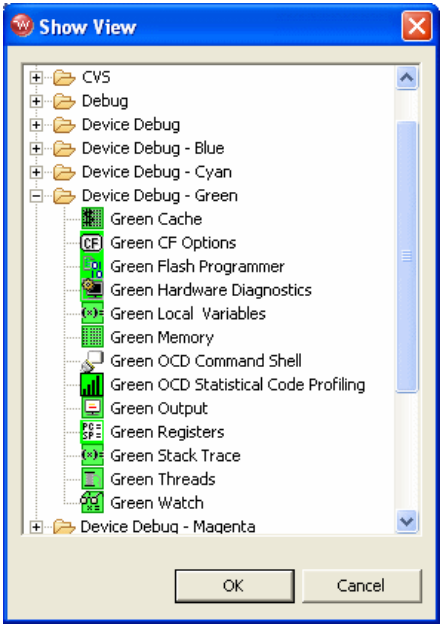
The Program Counter (the arrow in the left gutter in **main.c**) indicates the statement that will execute when the process resumes.

24.3 Coloring Views

The color context you assign to a process also carries through to other views whose scope is determined by the contents of the Debug view.

The data views that appear in the Device Debug perspective (such as the Threads or Stack Trace view) usually update to reflect whatever is currently selected in the Debug view. If you prefer, you can start colored views that are associated with a process of a particular color and update only when that process changes.

To open a view of a particular color, select
Window > Show View > Other > Device Debug - *color* **>** *view.*



For more information about data views in the Device Debug perspective, open the views and press the help key for your host.

24.4 Stepping Through a Program

Once a process has stopped under debugger control (most often, at a breakpoint), you can single-step through the code, jump over subroutine calls, or resume execution. What you can do depends on what you selected in the Debug view.

When the program is stopped, you can resume operation by clicking **Resume** on the toolbar of the Debug view. If there are no more remaining breakpoints, interrupts, or signals, the program will run to completion (unless you click **Suspend**).

To step through the code one line at a time, in the Debug view, click **Step Into**. If you have other data views open, such as the Registers or Local Variables views, they will update with current values as you step through the code.

The effect of **Step Into** is somewhat different if you click **Toggle Disassembly/Instruction Step Mode** in the Debug view, or when the current routine has no debugging information. When this mode is set, the step buttons cause instruction-level steps to be executed instead of source-level steps. Also, the Disassembly view will appear instead of the Editor view.

To single-step without going into other subroutines, click **Step Over** instead of **Step Into**.

While stepping through a program, you may conclude that the problem you are interested in lies in the current subroutine's caller, rather than at the stack level where your process is suspended. In this situation, if you click **Step Return** in **Debug**, execution continues until the current subroutine completes, then the debugger regains control in the calling statement.

These run control options, as well as others, are available from the **Run** menu as well as from the Debug view toolbar. For more information, open the Debug view and press the help key for your host.

24.5 Using Debug Modes

Depending on the type of connection you created between the debugger and the target, you may be able to operate the debugger in different modes. Different debug modes have different capabilities and limitations, which are mostly related to how the debugger interacts with the target and the processes that are being debugged. You can also create multiple debug connections to the same target, allowing you to debug in multiple modes simultaneously.

Target Connection Type	Supported Modes
WDB agent on VxWorks	**System Mode** ▪ Supports debugging the entire system using a single execution context. ▪ Supports limited debugging of individual kernel tasks. The debugger can retrieve stack traces for individual tasks, but if any of the tasks is resumed and suspended, even when stepping, the entire system is resumed and suspended.

Task Mode

- Supports debugging of kernel tasks. It allows suspending, resuming, and stepping kernel tasks individually, without affecting other kernel tasks.
- Supports debugging of RTPs.

kgdb on Linux

Kernel Mode

- Only supports debugging the kernel using a single execution context. When the system context is suspended, the kernel, kernel threads, and user processes are suspended also.

ptrace agent on Linux

User Mode

- Supports debugging user processes. Processes and threads within processes are suspended and resumed independently of each other.

Dual Mode on Linux

In dual mode, you must toggle between user and kernel mode depending on your debugging needs.

Kernel Mode (also called System Mode)

- Only supports debugging the kernel using a single execution context. When the system context is suspended, the kernel, kernel threads, and user processes are suspended also.

	User Mode ▪ Supports debugging user processes. Processes and threads within processes are suspended and resumed independently of each other.
OCD	**System Mode** ▪ Supports debugging the entire system using a single execution context.
OCD with OS Awareness for VxWorks	**System Mode** ▪ Supports debugging entire system using a single execution context, including retrieving the full stack trace when the system is suspended. ▪ Supports limited debugging of individual kernel tasks. The debugger can retrieve stack traces for individual tasks, but if any of the tasks is resumed and suspended, even when stepping, the entire system is resumed and suspended. ▪ Supports viewing of individual RTPs, but does not provide run control unless the target has been configured for one-to-one MMU virtual page mapping.
OCD with OS Awareness for Linux	**System Mode** ▪ Only supports debugging the kernel and kernel modules using a single execution context. ▪ Supports viewing of processes, but the debugger cannot be attached to them. ▪ Kernel objects are not available.

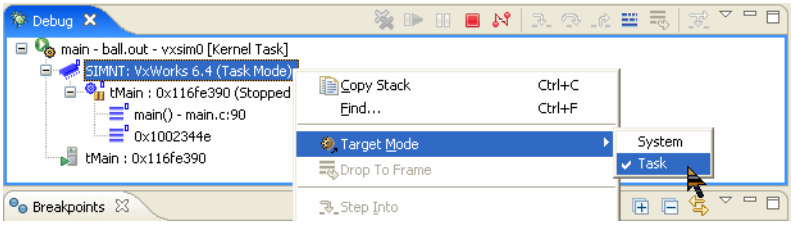
As a general rule, when you are debugging the target in user mode or task mode, the debugger interacts only with the process or processes being debugged. If you suspend this process, other processes keep running. This mode is less intrusive, as it allows you to control the selected process or thread while the rest of the system can continue to operate normally.

When you are debugging in system mode, the debugger interacts with the entire system at once, so if you suspend one task, all processes and kernel tasks running on the system are suspended as well. This gives you increased control and visibility into what is happening on the system, but it is also very disruptive.

For example, if the system maintains network connections with other systems, suspending it will cause the others to lose their network connections with the debugged system.

24.5.1 Setting and Recognizing the Debug Mode of a Connection

Right-clicking on a connection in the Target Manager or the Debug view and selecting **Target Mode** allows you to specify a debug mode for the connection. The currently active mode is indicated by a checkmark.

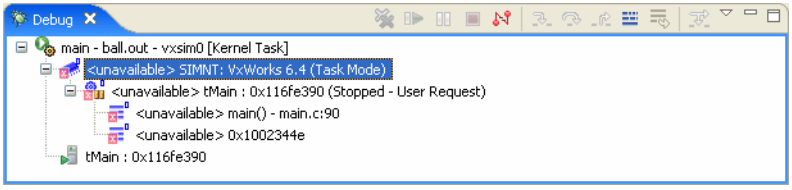


When you create a new debug connection through a launch, the connection debug mode (either system or task mode) is saved as a property of the launch. This mode is listed in parentheses at the end of the label of the target node in the Debug view.

Switching Debug Modes

For target connections that support switching between modes, if you switch the debug mode while a debug connection is active, this debug connection will become unavailable in the Debug view, as shown in Figure 24-4. When a debug connection is unavailable, no operations can be performed on it, except for disconnecting the debug connection.

Figure 24-4 **Debug View Showing Unavailable Connections**



In the Target Manager, if you switch the target to system mode, every node in the tree will have a system mode icon painted on top. If the system mode icon does not appear, then the node and processes are in task or user mode.

24.5.2 Debugging Multiple Target Connections

You can debug processes on the same target using multiple target connections simultaneously. An example of this setup is a Linux target that has a user mode **ptrace** agent installed for debugging processes, and an OCD connection for halting the system and debugging the kernel.

In this situation, if the system is halted using the OCD (system mode) target connection, the user mode **ptrace** agent will also be halted, and the user mode target connection will be lost. When the system is resumed, the user mode target connection will be re-established.

The Target Manager and the Debug view (if a debug session is active) both provide feedback in this scenario. The Target Manager hides all the process information that was visible for the target, and displays a label **back-end connection lost** next to the target node. The Debug view does not end the active debug session, but it shows it as being **unavailable**, in the same manner as if the debug mode was switched.

24.5.3 Disconnecting and Terminating Processes

Disconnecting from a process or core detaches the debugger, but leaves the process or core in its current state.

Terminating a process actually kills the process on the target.



NOTE: If the selected target supports terminating individual threads, you can select a thread and terminate only that thread.

24.5.4 Configuring Debug Settings for a Custom Editor

By default, the Workbench Editor opens when the debugger stops in a given file. To cause a different editor to open for particular file types, modify the mappings in **Window > Preferences > General > Editor > File Associations**.

Modifying these mappings takes care of editor selection and painting of the instruction pointer in the editor gutter. However, to associate other debugging actions with the new editor, you must modify the Eclipse extension point **org.eclipse.ui.editorActions**.

For example, the breakpoint double-click action associated with the Workbench Editor looks like this:

```
<extension point="org.eclipse.ui.editorActions">
   <editorContribution
       targetID="com.windriver.ide.editor.c"
       id="com.windriver.ide.debug.CSourceFileEditor.BreakpointRulerActions">
     <action
       label="Dummy.label"
       class="com.windriver.ide.debug.internal.ui.breakpoints.actions.ToggleB
reakpointRulerAction"
       actionID="RulerDoubleClick"
       id="com.windriver.ide.debug.ui.actions.toggleBreakpointRulerAction.c">
     </action>
</editorContribution>
```

Other features that are by default configured to work only with the Workbench Editor are **Run to line**, **Set PC to here**, and **Watch**. These features are configured through following extensions:

```
<viewerContribution
      targetID="#WREditorContext"
      id="com.windriver.ide.debug.ui.editprPopup.actions">
   <visibility>
      <and>
         <systemProperty
            name="com.windriver.ide.debug.ui.debuggerActive"
            value="true"/>
         <pluginState value="activated" id="com.windriver.ide.debug.ui"/>
      </and>
   </visibility>
   <action
        label="%WatchAction.label"
        icon="icons/actions/hover/watch_exp.gif"
        menubarPath="group.debug"
        helpContextId="com.windriver.ide.debug.ui.watchAction_context"
        class="com.windriver.ide.debug.internal.ui.actions.WatchAction"
        id="com.windriver.ide.debug.ui.editor.watchAction">
      <enablement>
        <systemProperty
           name="com.windriver.ide.debug.ui.debuggerActive"
           value="true">
        </systemProperty>
      </enablement>
   </action>
   <action
        label="%SetPcToHereAction.label"
        menubarPath="group.debug"
```

```
        helpContextId="com.windriver.ide.debug.ui.setPcToHereAction_context"
        class="com.windriver.ide.debug.internal.ui.actions.SetPcToHereAction"
        id="com.windriver.ide.debug.ui.editor.setPcToHereAction">
   </action>
   <action
        label="%RunToLineAction.label"
        icon="icons/actions/hover/run_to_line.gif"
        menubarPath="group.debug"
        helpContextId="com.windriver.ide.debug.ui.runToLineAction_context"
        definitionId="org.eclipse.debug.ui.commands.RunToLine"
        class="org.eclipse.debug.ui.actions.RunToLineActionDelegate"
        id="com.windriver.ide.debug.ui.editor.runToLineAction">
   </action>
</viewerContribution>
```

Please refer to Eclipse SDK documentation for more information on these extension points.

24.6 Understanding Source Lookup Path Settings

Source Lookup Path settings allow you to map source file paths that the debugger retrieves from an executable's symbol data (also known as the **debugger path**) to the correct location of the source files on the host file system and in your workspace.

The compiler generated these paths when the executable was built, but if you are debugging the executable on a different machine, then the paths to those files are no longer valid.

For information about how to set Source Lookup Path settings, open the source lookup dialog and press the help key for your host.

24.7 Using the Disassembly View

Use the Disassembly view:

- To examine a program when you do not have full source code for it (such as when your code calls external libraries).

- To examine a program that was compiled without debug information.
- When you suspect that your compiler is generating bad code (the view displays exactly what the compiler generated for each block of code).

24.7.1 Opening the Disassembly View

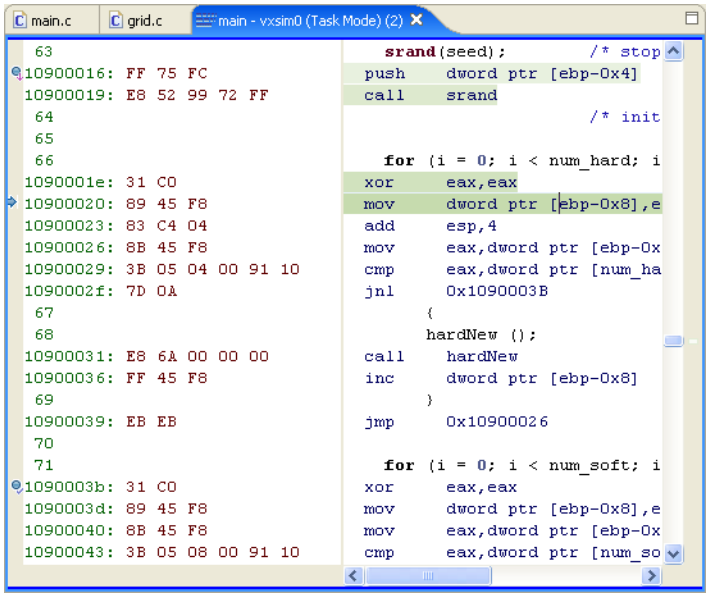
Unlike other Wind River Workbench views, you cannot access the Disassembly view from the **Window > Show View** menu—it appears automatically if the Debug view cannot display the appropriate source code file in the Editor (it appears as a tab in the Editor, labeled with the target connection being debugged).

You can open the Disassembly view manually by clicking the Debug view's **Toggle Assembly Stepping Mode** toolbar icon, and by right-clicking in the Stack Trace view, then selecting **Go To Code**.

24.7.2 Understanding the Disassembly View Display

The Disassembly view shows source code from your file (when available), interspersed with instructions generated by the compiler. As you step through your code, the Disassembly view keeps track of the last four instructions where the process was suspended. The current instruction is highlighted in the strongest color, with each previous step fading in color intensity.

Figure 24-5 **Disassembly View**



If the Disassembly view displays a color band at the top and bottom (here, the band is blue), then it is associated with the process with that color context in the Debug view; if no color band is displayed, then the view will update as you select different processes in the Debug view.

For more information, open the view and press the help key for your host.

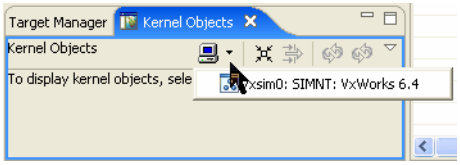
24.8 Using the Kernel Objects View

Use the Kernel Objects view to monitor data structures such as kernel tasks, RTPs, message queues, semaphores, and other resources.

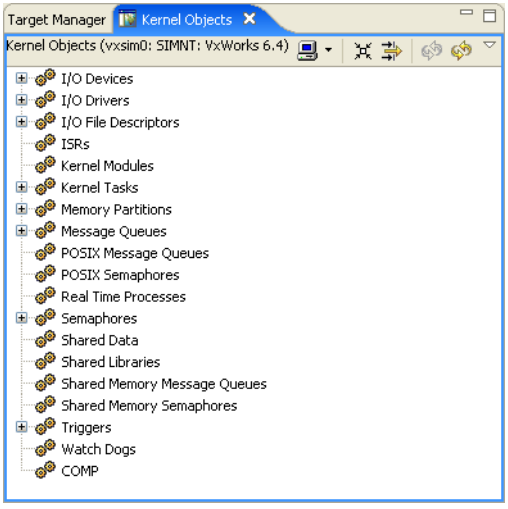
During multi-process debugging, you can use the Kernel Objects view to monitor a semaphore used to control a device that two processes are using. Or you can set an RTP that uses a system resource to watch that resource during **Step Over** system calls.

To open the Kernel Objects view:

1. Connect to your target in the Target Manager (see *19.4.2 Connecting to the Target*, p.259).
2. Click the **Kernel Objects** tab to bring it to the foreground, then click the pull-down arrow and select your target connection.



3. The Kernel Objects view appears.

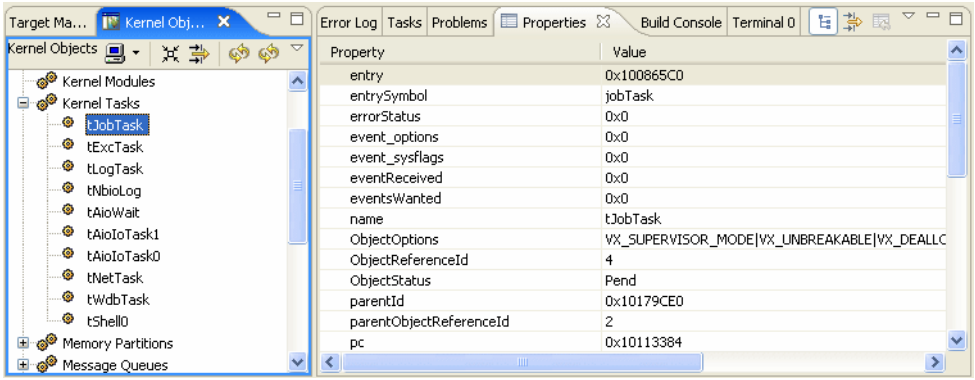


24.8.1 Understanding the Kernel Objects View Display

System resources are displayed in a hierarchical tree.

1. To see specific instances of each type of resource, or to display which tasks belong to which executable, click the plus sign to expand the tree.

2. To examine a resource in the Kernel Objects view, double-click it. Properties and their current values are displayed in the Properties view.



3. To copy a value to another view, right-click it and select **Copy**.

   To change a value, select it and type in a new value. If you have copied a value from another view, right-click the value in the Properties view and select **Paste**.

4. If you change the processes or tasks running on the target, select **Refresh Selected** or **Refresh All** to update the display in the Kernel Objects view.

5. If you want to remove some types of resources from the display, right-click them and select **Add to Filter**. Then when you toggle the **Filter** toolbar icon, these resources will appear or disappear so you can restrict the list to only the resources you want to monitor.



NOTE: To improve responsiveness, the Kernel Objects view updates the display or fetches information only when you specifically request it.

For a guide to the icons in the Kernel Objects view, open the view and press the help key for your host.

24.9 Remote Kernel Metrics

Remote kernel metrics (RKMs) are operating system signals (metrics) that are dynamically collected by the **RKM_monitor** target agent. The metrics can be displayed in real-time utilizing the full-color features of the StethoScope GUI included with Workbench.

The RKM monitor for VxWorks supports two types of connections: TCP/IP and WTX. Typically you would use the TCP/IP connection unless you are using the VxSim simulator, in which case you should use the WTX connection. Actual VxWorks target connections can use either connection but TCP/IP is faster and does not require the debug server to be running.

Building the RKM Monitor for VxSim

The following procedure uses VxSim and the WTX connection type to demonstrate RKM monitor usage under VxWorks.

1. Start Workbench.
2. Select **File > New > Example** from the main menu bar.
3. In the New Example dialog, select **Examples > VxWorks Downloadable Kernel Module Sample Project** and click **Next**.
4. Under **Available Examples**, you have two choices:
   - **The RKM (Remote Kernel Metrics) Monitor Program (TCP/IP)**
   - **The RKM (Remote Kernel Metrics) Monitor Program (WTX)**

   Select **The RKM (Remote Kernel Metrics) Monitor Program (WTX)** for this procedure because we will use VxSim as our target.

   Click **Finish**.

   A new **rkm_monitor_wtx (Wind River VxWorks 6.*x*)** project is created.
5. Select the project and set the active build spec to either **SIMNTdiab** or **SIMNTgnu** depending on your compiler and build the project.

Downloading the RKM Monitor and Libraries

The VxWorks RKM Monitor requires two StethoScope shared libraries in order to run. These libraries are **scopeutils.so** and **libscope*wtx.so**, located in the

installDir/**scopetools-stdone-5.5/target/arch/simntVx6.2gcc3.3.2/** directory, where the * wildcard is a version number, for example **710**.

Download both of these files before downloading the RKM monitor.

One way to download the libraries is as follows:

1. Start a VxSim connection.
2. Select **ScopeTools > StethoScop**e from the main menu.
3. Select **VxWorks** for the target type and click **OK**.
4. Set the options in the **StethoScope Setup Options** dialog box as follows:
   a. Select L**oad libraries only** under **Execution Mode**.
   b. If you wanted to start the StethoScope GUI, you would check **Start StethoScope GUI on the host** at the top of the dialog box. In this case we are only downloading the libraries, so leave it unchecked.
   c. Set **Scope index** to the default, **127**.
   d. Set **Connection type** to **WTX**, the connection type used for VxSim.

   Click on **OK** to automatically download the above libraries (if they are not already loaded). The StethoScope GUI will not be launched (unless you checked the **Start the StethoScope GUI on the host** box).

You may now download **RKM_monitor.** One way of doing that, which is also an alternate way of downloading the libraries, is discussed next.

Another way of downloading the libraries in Workbench is as follows:

1. Start a VxSim connection.
2. In the Target Manager View, select the target and right-click to bring up the context menu. Select **Download**.
3. Specify the full pathname of the **scopeutils.so** library and click **OK**.
4. Repeat the process for both the **libscope*wtx.so** library and **rlm_monitor_wtx.out**.

The libraries will appear in your Target Manager view if you expand **SIMNT**, and the monitor will appear if you expand **Kernel Tasks**. Once you have downloaded the libraries and the monitor, you can start the monitor as discussed next.

Starting the RKM Monitor

Starting the RKM Monitor Using Workbench Using Workbench

Once you have built the RKM monitor project as discussed in *Building the RKM Monitor for VxSim*, p.331, and have VxSim running (by creating a VxSim target connection), you can start the monitor as follows.

In the Project Navigator, expand **rkm_monitor_wtx.out**, right-click on the **rkm_monitor_wtx.out** executable, and select **Run Kernel Task** from the context menu. Select the VxSim connection in the **Connection to use** drop-down menu, and enter **RKM_monitor** for **Entry Point**. Click **Run**.

In the Target Manager, expand **Kernel Tasks** under **SIMNT**. You should see the **RKM_monitor** kernel task running.

Controlling the RKM Monitor Using VxSim

In the VxSim window, you can use the following commands to control the RKM monitor:

- **RKM_monitor**—Starts an RKM monitor with the default options.
- **RKM_list**—Lists all monitors which are currently running.
- **RKM_shutdown**—Stops all RKM monitors.
- **RKM_stop** *index*—Stops the RKM monitor using index number *index*, where *index* is a number from 0 to 127 (the default).
- **RKM_monitor** *options*—starts an RKM monitor with the specified options. Enter options within double-quotes.
- **RKM_monitor "-help"**—Displays the RKM monitor options.

For example, the following sequence of commands starts an RKM monitor with with a non-default port, shows that it is running, and then stops it.

```
-> RKM_list
value = 0 = 0x0
-> RKM_monitor "-index=125"
value = 0 = 0x0
-> RKM_list
Monitor[125] is running
value = 0 = 0x0
-> RKM_stop 125
value = 0 = 0x0
-> RKM_list
value = 0 = 0x0
->
```

Note that the RKM_monitor was started with a specific index number. In this way you can run multiple monitors on your target, which has several advantages. You might want to configure one monitor, for example, to collect a few signals for all processes at a low sampling frequency, and configure another to sample a complete set of metrics for a few processes at a high frequency.

By specifically selecting the signals you want to monitor, you can reduce the memory, CPU, and network resources required to monitor the large set of signals selected by default. In addition, the source for the RKM monitor is included so you can create versions that monitor specific signals that are not made available by the default configuration, or even monitor specific portions of your application.

For example, to start an RKM monitor with index 125 to monitor only the system metric tracking the number of tasks on the system, taking 10 samples every second:

```
->RKM_monitor "-index=125 -sysmetrics tasks -samples=10"
```

When you attach StethoScope to the RKM monitor invoked for specific metrics, you will be able to view only those metrics in StethoScope.

Starting the StethoScope GUI with a WTX Connection

You can start the StethoScope GUI using the Workbench main menu as described in *Using StethoScope to View Remote Kernel Metrics*, p.334, or you can create a desktop icon to start the StethoScope GUI and connect it to the target. Because the WTX version requires some VxWorks environment variables to be configured, it is easiest to create an icon which invokes the StethoScope program using the VxWorks Development shell. For example:

```
C:\PATH\wrenv.exe -p vxworks-6.3  scope.exe -index 127 -verbosity 0 -tgtsvr
vxsim0@HOST -wtxMode
```

Note that you have to enter the correct information for your path to **wrenv.exe** and also your host information.

Using StethoScope to View Remote Kernel Metrics

Your main navigation tool for StethoScope is the Signals Tree in the upper-left corner. As an example of how to use StethoScope with remote kernel metrics, expand **vxsim**, expand **vxKernel**, expand **System**, and then expand **Tasks**. Check the **Tasks** group to automatically check all of the metrics under **Tasks** (**tasks**, **ready**, **suspended**, **stopped**, **pending**, **delayed**).

Hover your mouse over the StethoScope toolbar to find the **Zoom to Fit** icon. Click on it to contain all the signals in the graph window. Note that each signal

has a unique color associated with it and this color is used for the lines in the graph. The MiniMonitor view lists the current values of monitored signals and the MiniDump view lists the value of each signal at each sampling.

By default, when any new processes are started they are added to the monitor. Note, however, that the buffer is reset when new signals are added, so you lose the history of what you had been monitoring. You can avoid this by, for example, monitoring only your own processes with appropriate command line options as described in *Controlling the RKM Monitor Using VxSim*, p.333.

For more information on the use of StethoScope, see your StethoScope documentation.

24.10 Run/Debug Preferences

For information about how to set debug and run control preferences, open the Debug view and press the help key for your host.

25 Troubleshooting

25.1 Introduction 337

25.2 Startup Problems 338

25.3 General Problems 341

25.4 Error Messages 343

25.5 Troubleshooting VxWorks Configuration Problems 355

25.6 Error Log View 358

25.7 Error Logs Generated by Workbench 358

25.8 Technical Support 365

25.1 Introduction

This chapter displays some of the errors or problems that may occur at different points in the development process, and what steps you can take to correct them. It also provides information about the log files that Workbench can collect, and how you can create a ZIP file of those logs to send to Wind River support.

25.2 Startup Problems

This section discusses some of the problems that might cause Workbench to have trouble starting.

Workspace Metadata is Corrupted

If Workbench crashes, some of your settings could get corrupted, preventing Workbench from restarting properly.

1. To test if your workspace is the source of the problem, start Workbench, specifying a different workspace name.

   On Windows

   Select **Start > Programs > Wind River > Wind River** Workbench **2.***x* **> Wind River** Workbench **2.***x*, then when Workbench asks you to choose a workspace, enter a new name (**workspace2** or whatever you prefer).

   Or, if the Workbench startup process does not get all the way to the Workspace Launcher dialog, or does not start at all, start it from a terminal window:

   ```
   > installDir\workbench-2.x\wrwb\2.x\x86-win32\bin\wrwb.exe -data newWorkspace
   ```

   On Linux or Solaris

   Start Workbench from a terminal window, specifying a new workspace name:

   ```
   > ./startWorkbench.sh -data newWorkspace
   ```

2. If Workbench starts successfully with a new workspace, exit Workbench, then delete the **.metadata** directory in your original Workbench installation (*installDir***/workspace/.metadata**).
3. Restart Workbench using your original workspace. The **.metadata** directory will be recreated and should work correctly.
4. Because the **.metadata** directory contains project information, that information will be lost when you delete the directory.

   To recreate your project settings, reimport your projects into Workbench (**File > Import > Existing Project into Workspace**). For more information about importing projects, open the Import File dialog and press the help key for your host.

.workbench-2.*x* Directory is Corrupted

1. To test if your *%USERPROFILE%***/.workbench-2.***x* directory is the source of the problem, rename it to a different name, then restart Workbench.



NOTE: Make sure you rename the *%USERPROFILE%***/.workbench-2.***x* directory (for example, on Windows XP it could be **C:\Documents and Settings***username***\.workbench-2.***x*).

Do not rename the *installDir***/workbench-2.***x* directory.

2. If Workbench starts successfully, exit Workbench, then delete the old version of your *%USERPROFILE%***/.workbench-2.***x* directory (the one you renamed).
3. Restart Workbench. The *%USERPROFILE%***/.workbench-2.***x* will be recreated and should work correctly.
4. Because the **.workbench-2.***x* directory contains Eclipse configuration information, any information about manually configured Eclipse extensions or plug-ins will be lost when you delete the directory.

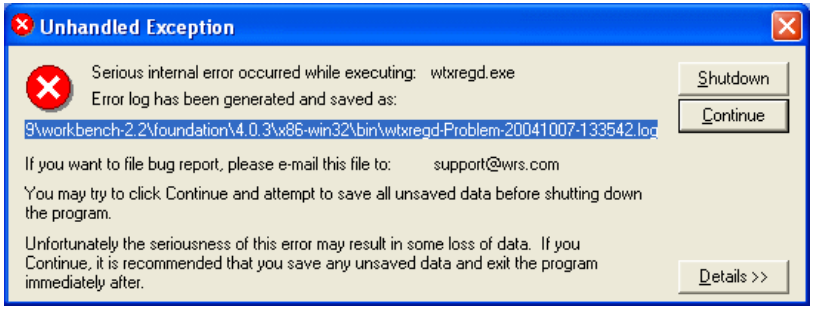
   To make them available again within Workbench, you must re-register them (**Help > Software Updates > Manage Configuration**). For more information about registering plug-ins, see *Adding Plug-in Functionality to Workbench*, p.372.

Registry Unreachable (Windows)

When Workbench starts and it does not detect a default Wind River registry, it launches one. After you quit Workbench, the registry is kept running since it is needed by all Wind River tools. You do not need to ever kill the registry.

If you do stop it, however, it stores its internal database in the file *installDir***/.wind/wtxregd.***hostname*.

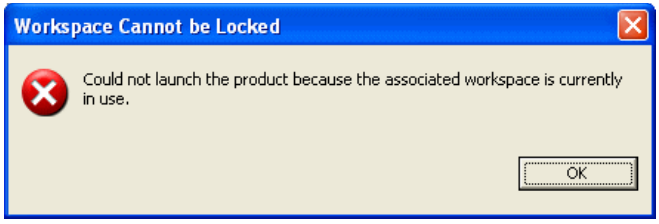
If this file later becomes unwritable, the registry cannot start, and Workbench will display an error.



This error may also occur if you install Workbench to a directory to which you do not have write access, such as installing Workbench as an administrator and then trying to run it as yourself.

Workspace Cannot be Locked (Linux and Solaris)

If you start Workbench and select a workspace, you may see a **Workspace Cannot be Locked** error.



There are three possible causes for this error:

1. Another user has opened the same workspace. A workspace can only be used by one user at a time.
2. You installed Workbench on a file system that does not support locking.

   Use the following command at a terminal prompt to start Workbench so that it creates your workspace on a file system which does allow locking, such as a directory on a local disk:

   `./startWorkbench.sh -configuration` *directory that allows locking*

   For example:

```
./startWorkbench.sh -configuration /usr/local/yourName
```

3. On some window managers (e.g. gnome) you can close the window without closing the program itself and deleting all running processes. This results in running processes maintaining a lock on special files in the workspace that mark a workspace as open.

   To solve the problem, kill all Workbench and Java processes that have open file handles in your workspace directory.

25.2.1 Pango Error on Linux

If the file **pango.modules** is not world readable for some reason, Workbench will not start and you may see an error in a terminal window similar to

```
** (<unknown>:21465): WARNING **: No builtin or dynamically loaded modules
were found. Pango will not work correctly. This probably means there was an
error in the creation of:
   '/etc/pango/pango.modules'
You may be able to recreate this file by running pango-querymodules.
```

Changing the file's permissions to **644** will cause Workbench to launch properly.

25.3 General Problems

This section describes problems that are not associated with any particular Workbench component.

25.3.1 Java Development Tools (JDT) Dependency

Some third party plug-ins are dependent on JDT. If a plug-in you are interested in requires JDT, you should download it from the official Eclipse Web site:

http://download.eclipse.org/eclipse/downloads/drops/R-3.0.1-200409161125/eclipse-JDT-3.0.1.zip

A list of official mirror sites is here:

http://www.eclipse.org/downloads

25.3.2 Help System Does Not Display on Solaris or Linux

Eclipse comes preconfigured to use Mozilla on Solaris and Linux, and it expects it to be in your path. If Mozilla is not installed, or is not in your path, you must set the correct path to the browser or Workbench will not display help or other documentation.

To manually set the browser path in Workbench:

1. Select **Window > Preferences > Help**.
2. Click **Custom Browser (user defined program)**, then in the **Custom Browser command field** type or browse to your browser launch program. Click **OK**.
   - On Solaris, a sample Netscape browser launch command is **"/usr/dt/bin/netscape" %1**, though you should enter the command line that is appropriate for your browser.
   - On Linux, sample Mozilla browser launch commands are **"/usr/bin/mozilla" %1** and **kfmclient openURL %1**, though you should enter the command line that is appropriate for your browser.

25.3.3 Help System Does Not Display on Windows

The help system can sometimes fail to display help or other documentation due to a problem in McAfee VirusScan 8.0.0i (and possibly other virus scanners as well).

For McAfee VirusScan 8.0.0i, the problem is known to be resolved with patch10 which can be obtained from Network Associates. As a workaround, the problem can be avoided by making sure that McAfee on-access-scan is turned **on** and allowed to scan the **TEMP** directory as well as ***.jar** files.

More details regarding this issue have been collected by Eclipse Bugzilla #87371 at **https://bugs.eclipse.org/bugs/show_bug.cgi?id=87371**.

25.3.4 Removing Unwanted Target Connections

If you have trouble deleting a target connection session for any reason, use **wtxtcl**.

1. Start **wtxtcl** from a terminal window.

```
% wtxtcl
```

2. List all entries in the registry.

```
wtxtcl> wtxInfo
```

3. Unregister the offending entry or entries (the full entry name must be used).

```
wtxtcl> wtxUnregister tgt_localhost@manebogad
```

25.4 Error Messages

Some errors display an error dialog directly on the screen, while others that occurred during background processing only display this icon in the lower right corner of Workbench window.



Hovering your mouse over the icon displays a pop-up with a synopsis of the error. Later, if you closed the error dialog but want to see the entire error message again, double-click the icon to display the error dialog or look in the *Eclipse Log*, p.359.

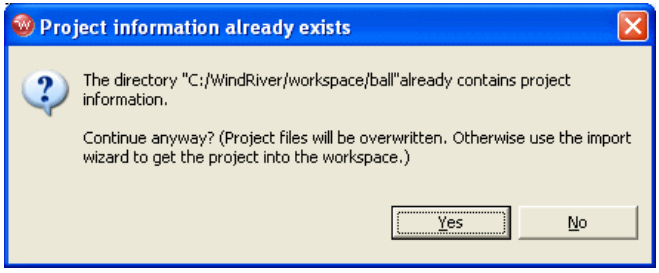
This section explains error messages that appear in each Workbench component.

25.4.1 Project System Errors

For general information about the Project System, see *4. Projects Overview*.

Project Already Exists

If you deleted a project from the Project Navigator but chose not to delete the project contents from your workspace, then you try to create a new project with the same name as the old project, you will see this error:



If you click **Yes**, your old project contents will be overwritten with the new project. If you want to recreate the old project in Wind River Workbench, click **No**, then right-click in the Project Navigator, select **Import**, then select **Existing Project into Workspace**.

Type the name of your old project or browse to the old project directory in your workspace, click **OK**, then click **Finish**. Your old project will appear in the Project Navigator.

Cannot Create Project Files in Read-only Location

When Workbench creates a project, it creates a **.wrproject** file and other metadata files it needs to track settings, preferences, and other project-specific information. So if your source files are in a read-only location, Workbench cannot create your project there.

To work around this problem, you must create a new project in your workspace, link in your source files using one of the following options:

Option 1—Use the Eclipse linked resource mechanism.

1. Create a project in your workspace (user-defined or flexible managed build[1]) by selecting **File > New >** *project type*.
2. Type in a name for your project, select **Create project in workspace**, then click **Next**.
3. Click **Next** to accept the default settings in the next dialogs, then click **Finish** to create your project.

1. Project types that offer a flexible managed build option are VxWorks Downloadable Kernel Module, Native Application, VxWorks Real-time Process, VxWorks Shared Library, and Standalone Application projects.

4. In the Project Navigator, right-click your new project and select **New > Folder**. The **Folder** dialog appears.
5. Type in a name for your folder, then click **Advanced** and select the **Link to folder in the file system** checkbox.
6. Type the path or click **Browse** and navigate to your source root directory, then click **OK** to create the new folder.
7. Click the plus next to the folder to open it, and you will see the source files from your read-only source directory. Eclipse calls items incorporated into projects in this way **linked resources**.



NOTE: This mechanism cannot be used for standard managed-build projects.

Option 2—Use Symbolic Links (Linux/Solaris only)

1. Create a new project in your workspace (any project type supports this).
2. In a command shell, create a symbolic link to the read-only directory in the project root directory.
3. In the Project Navigator, press **F5** to refresh the display. The directory and all sources in it appear.

25.4.2 Build System Errors

For general information about the Build System, see *16. Building Projects*.

Building Projects While Connected to a Target

If you try to build a project while you have a target connection active in the Target Manager, you may see an error. This happens when any of the files that need to be built contain symbol information, and therefore have been locked by the debugger.

You can continue your build by clicking **OK**, but be advised that you will need to disconnect your target and restart the build if you see an Build Console error message similar to **dld: Can't create file XXX: Permission denied**.

To avoid this problem, Workbench loads files up to a certain size completely into memory so no file lock is needed. To specify the largest symbol file that can be loaded into memory, select **Window > Preferences > Target Manager > Debug Server Settings > Symbol File Handling Settings** and specify a file size up to 60M.

Workflow for Cases Where You Need to Continually Rebuild Objects in Use by Your Target

1. Create a launch configuration for your debugging task. When you need to disconnect your target in order to free your images for the build process, the launch configuration allows you to automatically connect, build, download, and run your process with a single click.

   You can even specify that your project should be rebuilt before it is launched by selecting **Window > Preferences > Run/Debug > Launching**, and then selecting **Build (if necessary) before launching**. For more information about launch configurations, see *22. Launching Programs*.

- When you work with processes or RTPs, make sure that your process is terminated before you rebuild or relaunch. You can then safely ignore the warning (and check the **Do not show this dialog again** box).
- When you work with Downloadable Kernel Modules or user-built kernel images, just let the build proceed. If the **Link error** message appears, either disconnect your target or unload all modules, then rebuild or relaunch.

Workflow for Using On-Chip Debugging to Debug Standalone Modules Loaded on Your Target

1. Create a **Reset & Download**-type launch configuration for your application, and enable the **Build before launch** option (by selecting **Window > Preferences > Run/Debug > Launching > Build (if required) before launching**).
2. Run the launch configuration to debug your code. Make any changes to the source files and save them. Note that saving before unloading the symbols allows the debugger to track your breakpoints.
3. Before relaunching or rebuilding, unload the modules from the target by selecting them in the Target Manager and pressing the **Delete** key (you can multi-select if there are multiple modules).
4. Press the **Debug** button to relaunch your application. It will automatically rebuild, redownload, reset, and attach the debugger.

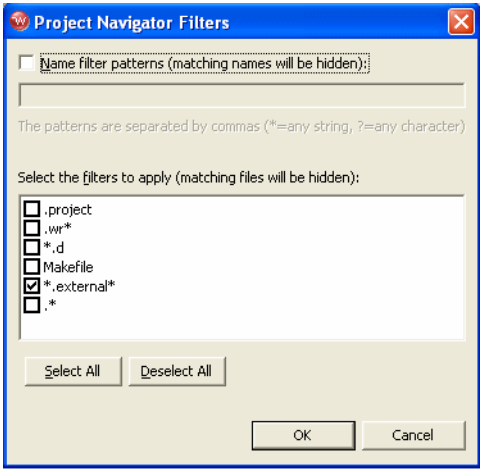
Problems Building Workbench 2.x Projects Imported Into Workbench 2.6

If you have trouble building projects that you imported from a previous version of Workbench, check if the **.wrproject** file contains an entry for **platform**. If not, the project is not compatible and has to be patched to work with the newest version of Workbench.

To patch the **.wrproject** file:

1. Open the file with the Workbench text editor by right-clicking the file in the Project Navigator, then selecting **Open With > Text Editor**.

   If the **.wrproject** file is not visible in the Project Navigator, click the downward arrow on the right side of the Project Navigator toolbar and select **Filters** to open the Project Navigator Filters dialog.



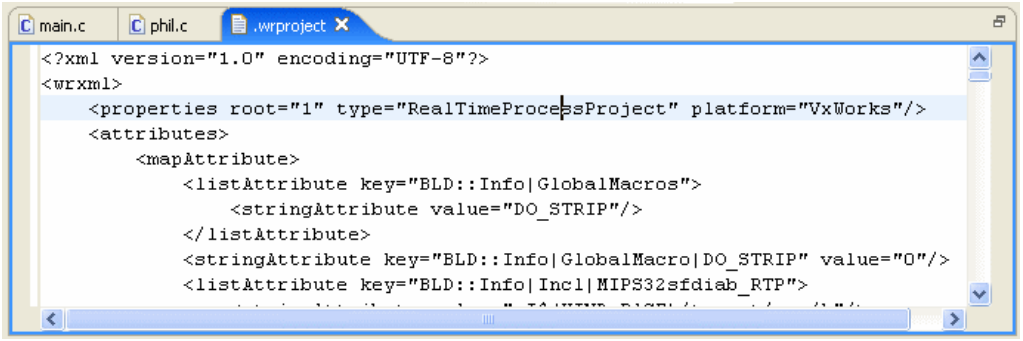
   Select the checkbox next to **.wr***, then click **OK**. The **.wrproject** file should now appear in the Project Navigator.

2. Locate the line at the beginning of the file similar to:

   ```
   <properties root="1" type="RealTimeProcessProject"/>
   ```

3. Add **platform="***projectplatform***"** to the end of the line, with *projectplatform* replaced by one of **VxWorks**, **Linux**, or **Standalone**, depending on the platform to which the project type belongs.

4. The result should appear similar to the following:



5. Save and close the **.wrproject** file. Your project should now build properly.

Build All Command Builds Projects Whose Resources have not Changed

Workbench may enter a state where selecting **Project > Build All** builds projects whose resources have not changed since the last build.

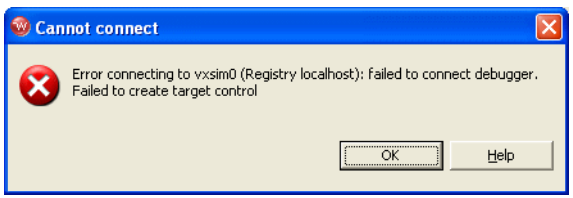
This happens only if Auto-Build (**Project > Build Automatically**) was previously enabled. If you switch this feature off, you must do a manual clean for all projects (**Project > Clean**) in order to re-enable building for previously built projects.

25.4.3 Target Manager Errors

For general information about the Target Manager, see *19. Connecting to Targets*.

Troubleshooting Connecting to a Target

If you see the following error:



Or if you have other trouble connecting to your target, try these steps:

1. Check that the target is switched on and the network connection is active. In a terminal window on the host, type:

   ping *n.n.n.n*

   where **n.n.n.n** is the IP address of your target.

2. Verify the target **Name/IP address** in the **Edit the Target Connection** dialog (right-click the target connection in the Target Manager then select **Properties**.)

3. Choose the actual target CPU type from the drop-down list if the **CPU type** in the **Edit the Target Connection** dialog is set to **default from target**.

4. Verify that a target server is running. If it is not:

   a. Open the Error Log view, then find and copy the message containing the command line used to launch the target server.

   b. Paste the target server command line into a terminal window, then hit **ENTER**.

   c. Check to see if the target server is now running. If not, check the Error Log view for any error messages.

5. Check if the **dfwserver** is running (on Linux and Solaris, use the **ps** command from a terminal window; on Windows, check the Windows Task Manager). If multiple **dfwservers** are running, kill them all, then try to reconnect.

6. When starting the VxWorks simulator on Solaris, the path environment variable must include **/usr/openwin/bin** so that it can find **xterm**. If **xterm** is not in the path, the simulator connection will fail.

7. Check that the WDB connection to the target is fully operational by right-clicking a target in the Target Manager and selecting **Target Tools > Run WTX Connection Test**. This tool will verify that the communication link is correct. If there are errors, you can use the WTX and WDB logs to better track down what is wrong with the target.

Exception on Attach Errors

If you try to run a task or launch an RTP and the Target Manager is unable to comply, it will display an **Exception on Attach** error containing useful information.

Build errors can lead to a problem launching your task or process; if one of the following suggestions does not solve the problem, try launching one of the pre-built example projects delivered with Workbench.

If the host shell was running when you tried to launch your task or process, try closing the host shell and launching again.

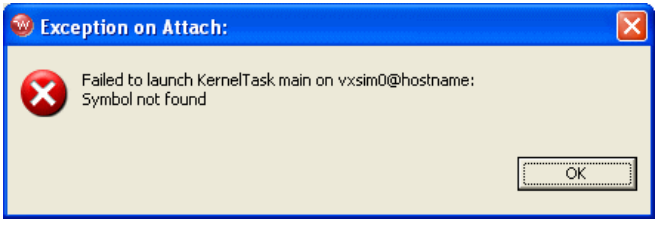
Error Launching a VxWorks Real-time Process on Linux

If you get an error when launching a VxWorks RTP from a Red Hat Workstation, update 3 host system, try these steps:

1. Delete **boothost:** from the beginning of the **Exec Path on Target** field of the **Run Real-time Process** dialog.
2. Add a new object path mapping to the target server connection properties that does not have **boothost:** in the host path.

Error When Running a Task Without Downloading First

You will see the following error if you try to run a kernel task without first downloading it to your target:



Processes can be run directly from the Project Navigator, but kernel tasks must be downloaded before running. Right-click the output file, select **Download**, fill in the Download dialog, then click **OK**.

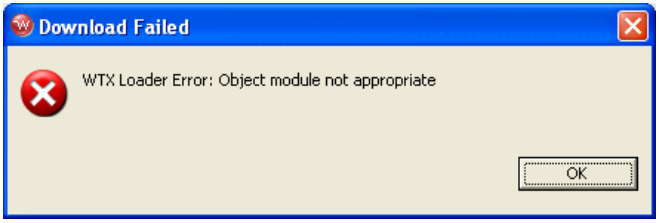
If you see this error and you did download the file, open a host shell for your connection, and try to run the task from the host shell. Type:

```
lkup entrypoint
```

to see if your entry point is there.

Downloading an Output File Built with the Wrong Build Spec

If you built a project with a build spec for one target, then try to download the output file to a different target (for example, you build the project for the simulator, but now you want to run it on a hardware target), you will see this error:

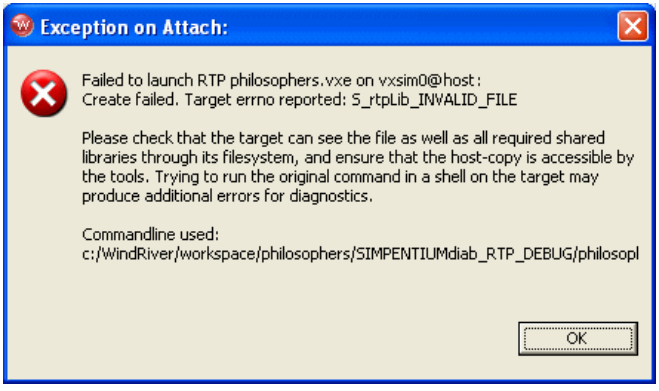


To select the correct build spec, right-click the output file in the Project Navigator, select **Set Active Build Spec**, select the appropriate build spec from the dialog, then rebuild your project.

Your project should now download properly.

Error if Exec Path on Target is Incorrect

If the **Exec Path on Target** field of the **Run Real-time Processes** dialog does not contain the correct target-side path to the executable file (if, for example, it contains the equivalent host-side path instead) you will see this error:



1. If the target-side path looks correct but you still get this error, check the following:

   a. Recheck the path you gave.
   Even if you used the **Browse** button to locate the file, it will be located in the host file system. The Object Path Mapping that is defined for your target connection will translate it to a path in the target file system, which is then visible in the Exec Path edit field. If your Object Path Mapping is wrong, the Exec Path will be wrong, so it is important to check.

Troubleshooting Running a Process

If you have trouble running your process from the **Run Process** or **Run Real-time Process** dialog, try these steps:

1. If the error **Cannot create context** appears, verify that the **Exec Path on Target** is a path that is actually visible on the target (and doesn't contain the equivalent host-side path instead).

   a. Right-click the process executable in the Project Navigator or right-click **Processes** or **Real-time Processes** in the Target Manager and select **Run Real-time Process.**

   b. Copy the exec path and paste it into the **Output View > Target Console Tab** (at the bottom of the view). Verify that the program runs directly on the target.

2. If the program runs but symbols are not found, manually load the symbols by right-clicking the process and selecting **Load Symbols**.

3. Check your **Object Path Mappings** to be sure that target paths are mapped to the correct host paths. See *20.2.3 Object Path Mappings Page*, p.270 for details on setting up your Object Path Mappings.

   a. Open a host shell and type:
   ```
   ls execpath
   ```
   If you have a target shell, type the same command.

   b. In the host shell, type:
   ```
   devs
   ```
   to see if the prefix of the Exec Path (for example, **host:**) is correct.

4. If the Exec Path is correct, try increasing the back-end timeout value of your target server connection (see *Advanced Target Server Options*, p.268 for details).

5. From a target shell or Linux console, try to launch the RTP or process.
6. Verify that the **vxWorks** node in the Target Manager view has a small **S** added to the icon, indicating that symbols have been loaded for the Kernel.
   a. If not, verify that the last line of your **Object Path Mappings** table displays a target path of **<any>** corresponding to a host path of **<leave path unchanged>**.

25.4.4 Launch Configuration Errors

If a launch configuration is not working properly, delete it by clicking **Delete** below the **Debug** dialog **Configurations** list.

If you cannot delete the launch configuration using the **Delete** button, navigate to *installDir*/**workspace/.metadata/.plugins/org.eclipse.debug.core/.launches** and delete the **.launch** file with the exact name of the problematic launch configuration



WARNING: Do *not* delete any of the **com.windriver.ide.*.launch** files.

Troubleshooting Launch Configurations

If you click the **Debug** icon (or click the **Debug** button from the **Launch Configuration** dialog) and get a “Cannot create context” error, check the **Exec Path** on the **Main** tab of the **Debug** dialog to be sure it is correct. Also check your **Object Path Mappings** (see *20.2.3 Object Path Mappings Page*, p.270 for information about Object Path Mappings).

If you still get the error, check to be sure that the process you are trying to run is a Real-time Process, and not a Downloadable Kernel Module or some other type of executable.

For general information about launch configurations, see *22. Launching Programs*.

25.4.5 Debugger Errors

Shared Library Problems

If you are having trouble working with shared libraries, try these steps:

1. If you are trying to run an executable and shared libraries located on your host machine's disk, make sure you can see the host machine's disk and the location of the shared libraries from the target.

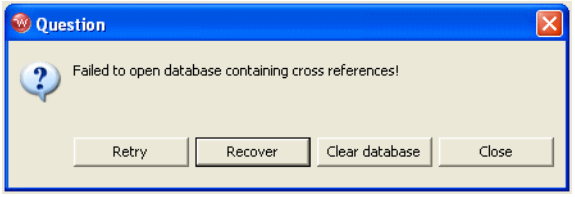
   Use a target shell, or the **@ls** command from a host shell, to check this.

2. Set **SHAREDLIB_VERSION** to **1** in order to generate the proper versioned shared object.

3. Make sure that a copy of **libc.so.1** is located in a place where the RTP has access to it. By default it should be located with the executable files, but you may locate it elsewhere as long as you use the compiler's **-rpath** option or the environment variable **LD_LIBRARY_PATH**.

25.4.6 Static Analysis Errors

If at any point Workbench is unable to open the cross reference database, you will see this error:



There are many reasons the cross reference database may be inaccessible, including:

- The database was not closed properly at the end of the last Workbench session running within the same workspace. This happens if the process running Workbench crashed or was killed.
- Various problems with the file system, including wrong permissions, a network drive that is unavailable, or a disk that is full.

You have several choices for how to respond to this error dialog:

- **Retry**—the same operation is performed again, possibly with the same failure again.
- **Recover**—the database is opened and a repair operation is attempted. This may take some time but you may recover your cross reference data.
- **Clear Database**—the database is deleted and a new one is created. All your cross reference data is lost and your workspace will be reparsed the next time you open the call tree.
- **Close**—the database is closed. No cross reference data is available, nor will it be generated. At the beginning of the next Workbench session, an attempt to open the database will be made again.

25.5 Troubleshooting VxWorks Configuration Problems

If you encountered problems booting or exercising VxWorks, there are many possible causes. This section discusses the most common sources of error. Please read *25.5.1 What to Check*, p.355 before contacting Wind River customer support. Often, you can locate the problem just by re-checking the installation steps, your hardware configuration, and so forth.

25.5.1 What to Check

Most often, a problem with running VxWorks can be traced to configuration errors in hardware or software. Consult the following checklist to locate a problem.

Hardware Configuration

- **If you are using an emulator**

  See the *Wind River ICE for Wind River Workbench Hardware Reference* or the *Wind River Probe for Wind River Workbench Hardware Reference* for information on troubleshooting those connections.

- **Limit the number of variables**

  Start with a minimal configuration of a single target.

- **Check that the RS-232 cables are correctly constructed**

  In most cases, the documentation accompanying your target system describes its cabling requirements. A common problem—make sure your serial cable is a NULL modem cable, if that is what your target requires.



NOTE: If you need to use a gender converter to connect your serial cable, it is most likely not the right kind of cable. NULL modem cables tend to have same gender connectors on each end, such as both female or both male. Straight through cables tend to have one male and one female connector. Changing the gender of a cable rarely has the desired results.

- **Check the boot ROM(s) for correct insertion**

  If the target seems completely dead when applying power (some have front panel LEDs) or shows some error condition (for example, red lights), the boot ROMs may be inserted incorrectly.

- **Press the RESET button if required**

  Some system controller boards do not reset completely on power-on; you must reset them manually. Consult the target documentation if necessary.

- **Make sure all boards are jumpered properly**

  Refer to the target information reference for your BSP and the target documentation to determine the correct dip switch and jumper settings for your target and Ethernet boards.

Booting Problems

- **Check the Ethernet transceiver site**

  For example, connect a known working system to the Ethernet cable and check whether the network functions.

- **Verify IP addresses**

  An IP address consists of a network number and a host number. There are several different classes of Internet addresses that assign different parts of the 32-bit Internet address to these two parts, but in all cases, the network number is given in the most significant bits and the host number is given in the least

significant bits. The simple configuration described in *3.4 Booting VxWorks*, p.50 assumes that the host and target are on the same network—they have the same network number. If the target Internet address is not on the same network as the host, the VxWorks boot program displays the following message:

```
Error loading file: errno = 0x33.
```

See the **errnoLib** reference entry for a discussion of VxWorks error status values.

- **Verify FTP server permissions**

  Check the FTP server configuration. See *Configuring FTP on Windows*, p.41 for more information on configuring the FTP server if you are using **WFTPD** (shipped by Wind River). Otherwise, consult your system documentation on the FTP Server shipped with it.

- **Helpful troubleshooting tools**

  When tracking down configuration problems, **ping**, **arp -a**, and **netstat -r** are useful tools. For more information, see *D. Glossary*.

Target Server Problems

- **Check back end serial port**

  If you use a WDB Serial connection to the target, make sure you have connected the serial cable to a port on the target system that matches your target-agent configuration. The agent uses serial channel 1 by default, which is different from the channel used by VxWorks as a default console (channel 0). Your target's ports may be numbered starting at one; in that situation, VxWorks channel one corresponds to the port labeled "serial 2."

- **Verify path to VxWorks image**

  The target server requires a host-resident image of the VxWorks run-time system. By default, it obtains a path for this image from the target agent (as recorded in the target boot parameters). In some cases (for example, if the target boots from a local device), this default is not useful.

  In that situation, create a new Target Server Connection definition in the Target Manager, and use the **-c** *filename* option in the **Advanced Target Server Options** field to specify the path to a host-resident copy of the VxWorks image.

Check the WFTPD Server Log

The WFTPD server log displays very helpful plain text messages. For information about how to enable logging FTP activities, see *Configuring FTP on Windows*, p.41.

25.6 Error Log View

Some errors direct you to the Error Log view, which displays internal errors thrown by the platform or your code. For more information about the Error Log, open the view and press the help key for your host.

25.7 Error Logs Generated by Workbench

Workbench has the ability to generate a variety of useful log files. Some Workbench logs are always enabled, some can be enabled using options within Workbench, and some must be enabled by adding options to the executable command when you start Workbench.

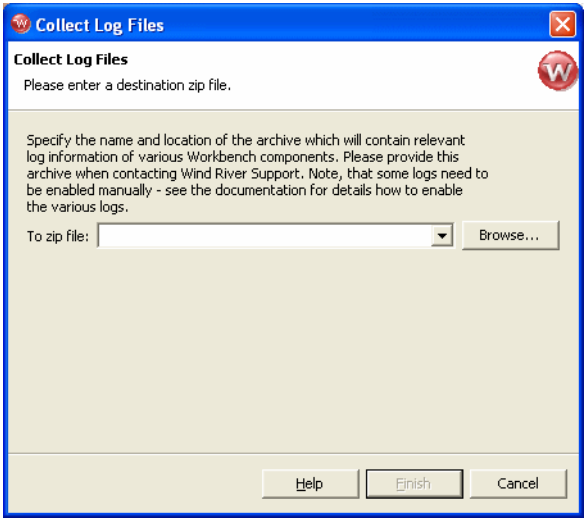
This section describes the logs, tells you how to enable them (if necessary), and how to collect them into a ZIP file you can send to Wind River support representatives.

25.7.1 Creating a ZIP file of Logs

Once all the logs you are interested in have been enabled, Workbench automatically collects the information as you work.

To create a ZIP file to send to a Wind River support representative:

1. Select **Help > Collect Log Files**. The dialog opens.



2. Type the full path and filename of the ZIP file you want to create (or browse to a location and enter a filename) then click **Finish**. The ZIP file is created in the specified location, and contains all information collected to that point.
3. To discontinue logging (for those logs that are not always enabled) uncheck the boxes on the Target Server Options tab, or restart Workbench without the additional options.

25.7.2 Eclipse Log

The information displayed in the Error Log view is a subset of this log's contents.

How to Enable Log

This log is always enabled.

What is Logged

- All uncaught exceptions thrown by Eclipse Java code.
- Most errors and warnings that display an error dialog in Workbench.
- Additional warnings and informational messages.

What it Can Help Troubleshoot

- Unexpected error popups.
- Bugs in Workbench Java code.
- Bugs involving inter-component communication.

25.7.3 DFW GDB/MI and Debug Tracing Logs

The DFW logs are a record of all communication and state changes between the debugger back end (the "debugger framework", or DFW) and other views within Workbench, including the Target Manager, debugger views, and OCD views.

How to Enable Log

These logs are always enabled.

To change the maximum debug server log file size, select **Window > Preferences > Target Manager > Debug Server Settings**. In the **Maximum Debug Server Log File Size** field, change the default size to the size you prefer (or to the size requested by a Wind River support representative).

Changing this field to **0** disables the collecting of **dfwserver.log** information.

What is Logged

Internal exceptions in the debugger back end, as well as all commands sent between Workbench and the debugger back end.

What it Can Help Troubleshoot

Debugger, Target Manager, and debugger back end-related bugs.

25.7.4 Debugger Views GDB/MI Log

This log shows the same information as reported in the *DFW GDB/MI and Debug Tracing Logs*, p.360.

How to Enable Log

You must enable this log before you start Workbench. Do this by adding these parameters to the Workbench executable command:

```
-vmargs -DDFE.Debug=true
```

What is Logged

Same as *DFW GDB/MI and Debug Tracing Logs*, p.360, except with Workbench time-stamps.

What it Can Help Troubleshoot

Debugger and Target Manager-related bugs.

25.7.5 Debugger Views Internal Errors Log

How to Enable Log

You must enable this log before you start Workbench. Do this by adding these parameters to the Workbench executable command:

```
-vmargs -DDFE.Debug=true
```

What is Logged

Exceptions caught by the Debugger views messaging framework.

What it Can Help Troubleshoot

Debugger views bugs.

25.7.6 Debugger Views Broadcast Message Debug Tracing Log

How to Enable Log

You must enable this log before you start Workbench. Do this by adding these parameters to the Workbench executable command:

```
-vmargs -DDFE.Debug=true
```

What is Logged

Debugger views internal broadcast messages.

What it Can Help Troubleshoot

Debugger views bugs.

25.7.7 Target Server Output Log

This log contains the messages printed by the target server while running. These messages typically indicate errors during various requests sent to it, such as load operations. Upon startup, if a fatal error occurs (such as a corefile checksum mismatch) then this error will be printed before the target server exits.

How to Enable Log

- Enable this log from the Target Manager by right-clicking the target connection, selecting **Properties**, selecting the **Target Server Options** tab, then clicking **Edit**.

  Select the **Logging** tab, then check the box next to **Enable output logging** and provide a filename and maximum file size for the log. Click **OK**.

- Enable this log from the command line using the **-l** *path/filename* and **-lm** *maximumFileSize* options to the target server executable. For more information about target server commands, see **Wind River Documentation > References > Host API and Command References > Wind River Host Tools API Reference > tgtsvr**.

What is Logged

- Fatal errors on startup, such as library mismatches and errors during exchange with the registry.
- Standard errors, such as load failure and RPC timeout.

What it Can Help Troubleshoot

- Debugger back end
- Target Server
- Target Agent

25.7.8 Target Server Back End Log

This log records all requests sent to the WDB agent.

How to Enable Log

- Enable this log from the Target Manager by right-clicking the target connection, selecting **Properties**, selecting the **Target Server Options** tab, then clicking **Edit**.

  Select the **Logging** tab, then check the box next to **Enable backend logging** and provide a filename and maximum file size for the log. Click **OK**.

- Enable this log from the command line using the **-Bd** *path/filename* and **-Bm** *maximumFileSize* options to the target server executable. For more information about target server commands, see **Wind River Documentation > References > Host API and Command References > Wind River Host Tools API Reference > tgtsvr**.

What is Logged

Each WDB request sent to the agent. For more information about WDB services, see **Wind River Documentation > References > Host API and Command References > Wind River WDB Protocol API Reference**.

What it Can Help Troubleshoot

- Debugger back end
- Target Server
- Target Agent

25.7.9 Target Server WTX Log

This log records all requests sent to the target server.

How to Enable Log

- Enable this log from the Target Manager by right-clicking the target connection, selecting **Properties**, selecting the **Target Server Options** tab, then clicking **Edit**.

  Select the **Logging** tab, then check the box next to **Enable WTX logging** and provide a filename and maximum file size for the log. Click **OK**.

- Enable this log from the command line using the **-Wd** *path/filename* and **-Wm** *maximumFileSize* options to the target server executable. For more information about target server commands, see **Wind**

River Documentation > References > Host API and Command References > Wind River Host Tools API Reference > tgtsvr.

What is Logged

Each WTX request sent to the target server. For more information about WTX services, see **Wind River Documentation > References > Host API and Command References > WTX C Library Reference > wtxMsg.**

What it Can Help Troubleshoot

- Debugger back end
- Target Server
- Target Agent

25.7.10 Target Manager Debug Tracing Log

This log prints useful information about creation and modification of Target Manager internal structures, as well as inconsistencies or warning conditions in the subsystems the Target Manager interoperates with.

How to Enable Log

You must enable this log before you start Workbench. Do this by adding these parameters to the Workbench executable command:

```
-debug -vmargs -Dcom.windriver.ide.target.DEBUG=1.
```

What is Logged

Target Manager internal debug errors.

What it Can Help Troubleshoot

Inconsistencies in the debugger back end.

25.7.11 Static Analysis Parser Logs

These logs contain information that can help a Wind River technical support representative to resolve problems with the source code parsers.

How to Enable Logs

Enable these logs by right-clicking a resource or resources in the Project Navigator or the Editor and selecting **Static Analysis > Generate Parser Logs**. From the list of logs that appears, select the logs that were requested by a technical support representative (the **jobs** log and the **performance** log are selected by default). Click **OK**.

What is Logged

- The **jobs** log describes what the parsers were instructed to do.
- The **performance** log lists the source files the parsers spent the most time with.
- The **PI4** interface and **symbols** logs contains the result of parsing.
- The **debug** log shows how the cpp-parser uses internal caching strategies.
- The **tokens** log lists every token processed by the cpp-parser.

What it Can Help Troubleshoot

Problems related to excessive parsing time. The performance log can help pinpoint if there are single files that require more parsing time, or if there are simply many resources to parse.

In a case where results of the analysis are different from what you expect but Wind River support cannot reproduce the problem, the PI4 and symbol logs are crucial in helping support representatives see the raw data the parsers are generating.

The debug and tokens logs are needed very seldom, but are useful to Wind River developers working on the parsers themselves.

25.8 Technical Support

If you have questions or problems with Workbench or with VxWorks after completing the above troubleshooting section, or if you think you have found an error in the software, please see the *Wind River Workbench Release Notes* for your platform for any additional information. Contact information for the Wind River Technical Support organization is also listed in the release notes. Your comments and suggestions are welcome.

PART VI

Using Workbench with Other Tools

26 Integrating Plug-ins 369

27 Using Workbench in an Eclipse Environment . 375

28 Using Workbench with Version Control 381

26 Integrating Plug-ins

26.1 Introduction 369

26.2 Finding New Plug-ins 370

26.3 Incorporating New Plug-ins into Workbench 370

26.4 Disabling Plug-in Functionality 373

26.5 Managing Multiple Plug-in Configurations 373

26.1 Introduction

Because Wind River Workbench is based on Eclipse, you can incorporate new modules into Workbench without having to recompile or reinstall it. These new modules are called *plug-ins*, and they can deliver new functionality and tools to your copy of Wind River Workbench.

Many developers enjoy creating new plug-ins and sharing their creations with other Eclipse users, so you will find many Web sites with interesting tools and programs available for you to download and incorporate into your Workbench installation.

Some plug-ins are dependent on Java Development Tools (JDT), which is automatically installed when you install Workbench (the install option is called **Wind River Java Development Tools**).

26.2 Finding New Plug-ins

In addition to the Eclipse Web site, **http://www.eclipse.org**, many other Web sites offer a wide variety of Eclipse plug-ins. Here are a few:

http://www.eclipse-plugins.info/eclipse/plugins.jsp

http://www.eclipseplugincentral.com/

http://eclipse-plugins.2y.net/eclipse/

http://www.sourceforge.net/

26.3 Incorporating New Plug-ins into Workbench

Many developers who download plug-ins prefer to create a new directory for each one, rather than unzipping the files directly into their Workbench installation directory. There are many advantages to this approach:

- The default Workbench installation does not change.
- You do not lose any of your plug-ins if you update or reinstall Workbench.
- Plug-ins do not overwrite each other's files.
- You know which files to replace when an update to the plug-in is available.

26.3.1 Creating a Plug-in Directory Structure

To make your plug-ins easier to manage, create a directory structure for them outside your Workbench installation directory.

1. Create a directory to hold your plug-ins. It can have any descriptive name you want, for example, **eclipseplugins**.
2. Inside this directory, create a directory for each plug-in you want to install. These directories can also have any descriptive name you want, for example, **clearcase**.



NOTE: Before continuing, download the plug-in's **.zip** or other archive file and look at its contents. Some plug-ins provide the **eclipse** directory structure and the **.eclipseextension** file for you, others do not.

- If the destination path for the files begins with **eclipse**, and you see an **.eclipseextension** file in the list, you may skip the rest of this section and extract the plug-in's files into the directory you created in step 2.
- If the destination path begins with **plugins** and **features**, then you must complete the rest of the steps in this section.

3. Inside each plug-in directory, create a directory named **eclipse**. This directory *must* be named eclipse, and a separate **eclipse** directory is required inside each plug-in directory.
4. Inside each eclipse directory, create an empty file named **.eclipseextension**. This file *must* be named **.eclipseextension** (with no **.txt** or any other file extension), and a separate **.eclipseextension** file is required inside each **eclipse** directory.
5. Extract your plug-in into the **eclipse** directory. Two directories, called **features** and **plugins**, appear in the directory alongside the **.eclipseextension** file.



NOTE: For any plug-in to work properly, its **features** and **plugins** directories as well as an empty file called **.eclipseextension** *must* be located inside a directory called **eclipse**.

26.3.2 Installing a ClearCase Plug-in

Once you have created a plug-in directory structure and have found a plug-in you want to use with Workbench, download and install it according to the instructions provided by the plug-in's developer (almost every plug-in comes with release notes containing installation instructions).

This section will show you how to download and install a plug-in on Windows.

Downloading the IBM Rational ClearCase Plug-in

Wind River recommends the IBM Rational ClearCase plug-in.

1. Follow steps 1 and 2 in *26.3.1 Creating a Plug-in Directory Structure*, p.370 (the IBM ClearCase plug-in creates the **eclipse** directory and the **.eclipseextension** file for you.)

   For the purposes of this example, name the top-level directory **eclipseplugins**, and name the plug-in directory **clearcaseIBM**.

2. Navigate to **http://www-128.ibm.com/developerworks/rational/library/1376.html** and click the **Plug-ins** link under **ClearCase**. The Rational ClearCase Plug-ins page opens.

3. Click the **Download** link to the right of the appropriate version of the package file. For this example, select **IBM Rational ClearCase SCM adapter for Eclipse 3.1: Windows** (this file works for Eclipse 3.2 as well).

4. Extract the **.zip** file to your **/eclipseplugins/clearcaseIBM** directory. The **eclipse** directory is created for you, and inside are two directories, called **features** and **plugins**, alongside the **.eclipseextension** file.

Adding Plug-in Functionality to Workbench

1. Before starting Workbench, make sure that **/usr/atria/bin** (where the ClearCase tools are installed) is in your path.

2. Start Workbench, then select **Help > Software Updates > Manage Configuration**. The **Product Configuration** dialog appears.

3. Select **Add an Extension Location** in the Wind River Workbench pane.

4. Navigate to your **eclipseplugins/***plug-in***/eclipse** directory. Click **OK**.

5. Workbench will ask if you want to restart. To properly incorporate ClearCase functionality, click **Yes**.

Incorporating the IBM Rational Plug-in

1. When Workbench restarts, activate the plug-in by selecting **Window > Customize Perspective**.

2. In the **Customize Perspective** dialog, switch to the **Commands** tab.

3. Select the **ClearCase** option in the **Available command groups** column, then click **OK**. A new **ClearCase** menu and icons appear on the main Workbench toolbar.

4. From the **ClearCase** menu, select **Connect to Rational ClearCase** to activate ClearCase functionality.

To configure the ClearCase plug-in, select **Window > Preferences > Team > ClearCase SCM Adapter**.

For more information about using the ClearCase plug-in, see **Help > Help Contents > Rational ClearCase SCM Adapter**.

For more information about ClearCase functionality, refer to your ClearCase product documentation.

26.4 Disabling Plug-in Functionality

You can disable plug-in functionality without uninstalling the downloaded files. This gives you the opportunity to re-enable them at a later time if you want.

1. To disable a plug-in, select **Help > Software Updates > Manage Configuration**. The **Product Configuration** dialog appears.
2. In the left column, open the folder of the plug-in you want to uninstall, select the plug-in itself, then click **Disable**.
3. Workbench will ask if you want to restart. To properly disable the plug-in's functionality, click **Yes**.

26.5 Managing Multiple Plug-in Configurations

If you have many plug-ins installed, you may find it useful to create different configurations that include or exclude specific plug-ins.

When you make a plug-in available to Workbench using the process shown in *Adding Plug-in Functionality to Workbench*, p.372, its extension location is stored in the Eclipse configuration area.

When starting Workbench, you can specify which configuration you want to start by using the **-configuration** *path* option, where *path* represents your Eclipse configuration directory.

On Windows:

From a shell, type:

```
% cd installdir\workbench-2.x\wrwb\platform\eclipse\x86-win32\bin
% .\wrwb.exe -configuration path
```

On Linux and Solaris:

Use the option as a parameter to the **startWorkbench.sh** script:

```
% ./startWorkbench.sh -configuration path &
```

For more information about using **-configuration** and other Eclipse startup parameters, see **Help > Help Contents > Wind River Partners Documentation > Eclipse Workbench User Guide > Tasks > Running Eclipse**.

27 Using Workbench in an Eclipse Environment

27.1 Introduction 375
27.2 Recommended Software Versions and Limitations 375
27.3 Setting Up Workbench 376
27.4 Using CDT and Workbench in an Eclipse Environment 377

27.1 Introduction

It is possible to install Workbench in a standard Eclipse environment, though some fixes and improvements that Wind River has made to Workbench will not be available.

27.2 Recommended Software Versions and Limitations

Java Runtime Version

Wind River tests, supports, and recommends using the JRE 1.5.0_08 for Workbench plug-ins.

Wind River adds a package to that JRE version, and not having that package will make the Terminal view inoperable.

Eclipse Version

Workbench 2.6 is based on Eclipse 3.2. Wind River patches Eclipse to fix some Eclipse debugger bugs. These fixes will be lost when using a standard Eclipse environment.

See the getting started for your platform for supported and recommended host requirements for Workbench 2.6.

Defaults and Branding

Eclipse uses different default preferences from those set by Workbench. The dialog described in *27.3 Setting Up Workbench*, p.376 allows you to select whether to use Workbench preferences or existing Eclipse preferences.

In a standard Eclipse environment, the Eclipse branding (splash screen, welcome screen, etc.) is used instead of the Wind River branding.

27.3 Setting Up Workbench

This setup requires a complete Eclipse and Workbench installation. Follow the respective installation instructions for each product.

1. From within Workbench, select **Help > Register into Eclipse**. The Register into Eclipse dialog appears.
2. In the **Directory** field, type in or **Browse** to your Eclipse 3.2 directory.
3. In the **Registration Options** section, select **Use Wind River default preferences**, or leave it unselected to maintain existing Eclipse preferences.

   If you decide to use Wind River default preferences, some changes you will notice are that autobuild is disabled, and the Workbench Application Development perspective and help home become the defaults.

4. If you decided to maintain existing Eclipse preferences you can still use the much faster Wind River (index based) search engine by leaving **Use Wind River search engine** selected. To use the Eclipse default search engine, unselect it.
5. If you want to track the installation process, leave **Log installation process** selected (click **Browse** to change the path where the file should be created). Uncheck it if you do not want Workbench to create a log file.
6. When you are done, click **Finish**. Workbench will be available the next time you launch Eclipse. No special steps are necessary to launch Eclipse.



NOTE: Any errors discovered during installation appear in the Error Log view.

27.4 Using CDT and Workbench in an Eclipse Environment

The following tips will help you understand how to use Eclipse C/C++ Development Tooling (CDT) and Workbench together in the same Eclipse environment.



NOTE: When starting Eclipse after registering Workbench, you will see three errors in the Error Log.

These errors are not a problem. They appear because Workbench ships some CDT plug-ins that are already on your system, and Eclipse is reporting that the new ones will not be installed over the existing ones.

27.4.1 Workflow in the Project Navigator

Some menus and actions are slightly different when using CDT and Workbench together.

Application Development Perspective (Workbench)

CDT projects appear in this perspective along with Workbench projects.

Building CDT Projects

The context menu of the Project Navigator contains entries for **Build Project** and **Rebuild Project**, but the **Rebuild Project** entry executes a normal build for CDT projects. The **Clean Project** entry is missing for CDT projects.

Running Native Applications

The **Run Native Application** menu is enabled for CDT projects. When executed, it creates a Workbench Native Application launch with correct parameters. Because Workbench Native Application launches do not support debugging, to debug your application you must create a CDT **Local C/C++ Application** launch from the **Run > Run As** menu.

Selecting Projects to Build

When selecting multiple projects (including Workbench and CDT projects) and executing any build action, the build action is only executed on Workbench projects.

Displaying File and Editor Associations

The Workbench Project Navigator displays icons for the default editor of a file, if file associations have been defined. If CDT is the default editor, the corresponding icons will also show up in the Application Development perspective.

C/C++ Perspective (CDT)

Static Analysis

Static analysis is available from the context menu of the Project Navigator.

Building Workbench Projects

CDT **Build Project** and **Clean Project** actions are enabled for Workbench projects, and they execute the appropriate build commands correctly.

Working with Workbench Binary Targets

There are no actions to directly run, debug or download a Workbench project's binary target in this perspective.

27.4.2 Workflow in the Build Console

Application Development Perspective (Workbench)

When adding a CDT project as a sub-project (project reference) to a Workbench project, the **Clear Build Console** flag is ignored when executing a build on this project.

C/C++ Perspective (CDT)

Executing a build on a Workbench project from this perspective correctly opens the Workbench Build Console.

General

When navigating to errors from the Workbench Build Console or the Problems view, the file containing the error opens in the assigned editor.

27.4.3 Workflow in the Editor

Opening Files in an Editor

The editor that should be used for files cannot be determined. It depends on the settings defined in the appropriate **plugin.xml** files, and on the order in which the Workbench and CDT plug-ins are loaded.

Only one default editor can be associated with each file type, and it is the same for both perspectives. Files can be opened with the **Open With** menu, allowing you to select the editor. When executed, that editor is associated with, and becomes the default for, this specific file.



NOTE: To assign a default editor for all files with a given signature, you must define a file association in the preferences by selecting **Window > Preferences**, then choosing **General > Editors > File Associations**.

For example, to add a default editor for all ***.c** files, click **Add** and enter ***.c**. The list of available editors appears. Select one, then click **Default**.

27.4.4 Workflow for Debugging

Workbench and CDT Perspectives

Regardless of any direct file association created using the **Open With** command, the default editor opens when debugging a file.

For example, associating ***.c** files with the default Workbench editor opens the Workbench editor in the CDT Debug and the Workbench Device Debug perspectives.

The reverse is also true: if you associate a file type with the CDT editor, it will open when those files are debugged even if you have made an association with a different editor using **Open With**.

28

Using Workbench with Version Control

28.1 Introduction 381

28.2 Using Workbench with ClearCase Views 381

28.1 Introduction

This chapter provides tips on using Workbench with version-controlled files, which Workbench project files you should add to version control when archiving your projects, and how to manage build output when your sources are version controlled.

28.2 Using Workbench with ClearCase Views

When using Workbench with ClearCase dynamic views, create your workspace on your local file system for best performance. For recommendations about setting up your workspaces and views, see **Help > Help Contents > Rational ClearCase SCM Adapter > Concepts > Managing workspaces**.

Wind River does not recommend that you place the Eclipse workspace directory in a view-private directory. If you create projects in the default location under the

workspace directory, ClearCase prompts you to add the project to source control. This process requires all parent directories to be under source control, including the workspace directory.

Instead, create workspace directories outside of a ClearCase view. If you want to create projects under source control, you should unselect the **Create project in workspace** checkbox in the project creation dialog and then navigate to a path in a VOB.

In addition, you should also redirect all build output files to the local file system by changing the **Redirection root directory** in the **Build Properties > Build Paths** tab of your product. All build output files such as object files and generated Makefiles will be redirected.

For more information about the redirecting build output and the redirection root directory, open the build properties dialog, press the help key for your host, and see the *Build Paths* section.

28.2.1 Adding Workbench Project Files to Version Control

To add Workbench project files to version control without putting your workspace into a ClearCase view, check-in the following automatically generated files along with your source files:

Project File	Description
.project	Eclipse platform project file containing general information about the project.
.wrproject	Workbench project file containing mostly general build properties.
.wrfolder	Workbench project file containing folder-level build properties (located in subfolders of your projects).
.wrmakefile	Workbench managed build makefile template used to generate Makefiles.
***.makefile**	Workbench managed build extension makefile fragments (e.g. for VxWorks Image projects or some Platform projects)
***.wpj**	VxWorks Image project file containing specific data not managed directly by Workbench but by the TCL engine.

For user-defined projects, all Makefile files need to be version controlled, too.

For VxWorks Image projects, it could occur that absolute paths are stored in the **.wpj** file, which breaks any team support. You should avoid manually adding source files to a VxWorks Image project that are referenced by absolute paths. The same is true for any build macro in any project type containing absolute paths—they should be substituted by environment variables (provided by **wrenv** for example) wherever possible.



NOTE: The **.metadata** directory should not be version controlled, as it contains mostly user- and workspace-specific information with absolute paths in it.

For more information about IBM Rational ClearCase, see **http://www-130.ibm.com/developerworks/rational/products/clearcase**.

Choosing Not to Add Build Output Files to ClearCase

After installing the ClearCase plugin, you may be prompted to add any build output files to ClearCase.

There are two ways to avoid this if you wish:

1. Using Workbench Preferences.
   a. Open the **Window > Preferences > Team > ClearCase SCM Adapter** preferences page.
   b. From the **When new resources are added** pull-down list, select **Do nothing**.
2. Using **Derived Resource** option.
   a. Configure your build so the build output goes into one (or a few) well-known directories such as **bin** or **output**.
   b. Check in the empty **bin** or **output** directories into ClearCase.
   c. In the Project Navigator, right-click the directory you checked in, select **Properties**, and on the **Info** page, select **Derived**.
   d. From now on, the Clearcase plug-in will not prompt you about **Derived** resources.



NOTE: If you use Workbench managed builds, they will automatically mark the build output directories as derived so ClearCase will not try to add the build output files to source control. If you use a different builder, you may have to configure it to mark resources as derived.

PART VII

Reference

A Command-line Updating of Workspaces 387

B Command-line Importing of Projects 391

C Configuring a Wind River Proxy Host 395

D Glossary 403

A

Command-line Updating of Workspaces

A.1 Overview 387

A.2 wrws_update Reference 388

A.1 Overview

The Workbench installation includes a **wrws_update** script that allows you to update workspaces from the command-line. This can be used, for example, to update workspaces in a nightly build script. The following section provides a reference page for the command.

A.2 wrws_update Reference

A script for updating an existing workspace is available in the Workbench installation and is named:

wrws_update.bat (Windows only)

wrws_update.sh (Windows, Linux, and Solaris)

This script launches a GUI-less Eclipse application that can be used to update makefiles, symbols (static analysis), and the search index.

Execution

Specify the location of the **wrws_update** script or add it to your path and execute it with optional parameters, for example:

```
$ wrws_update.sh -data workspace_dir
```

The workspace must be closed for the command to execute. If you do not specify any options to the command, all update operations are performed (**-all projects**, **-generate makefiles**, **--update symbols**, **-update index**).

Options

General Options

-h, --help
Print command help.

-q, **--quiet**
Do not produce standard output.

Eclipse Options

-data *workspace_dir*
The script uses the default workspace (if known), but it can also update other workspaces by specifying the **-data** *workspace_dir* option, just as Workbench does. (The script accepts the same command-line options as Workbench. For example, to increase virtual memory specify **-vmargs -Xmx***mem_size*.)

Global Options

-a, **--all-projects**

Update all projects, this option will force all closed projects to be opened. Opened projects will be closed after finishing the update.

-l, **--specify-list-of-projects** *argument*
Specify a list of projects to be updated. This option reduces the scope of the nightly update to the specified list of projects. Needed closed projects will be opened and unneeded opened ones closed. After finishing the update the previous state is restored. Separate the list with "," for example: **cobble,helloWorld**.

Build Options

-b, **--build-projects** *argument*
Launch build for projects. Several strings are valid as arguments, including: **build** (default), **clean**, and **rebuild**.

-e, **--enableTraceBuild**
Enable trace build output.

-f, **--debugMode** *argument*
Build using specific debug or non-debug mode where applicable. The *argument*, if specified, can be **0** or **1**, otherwise the current mode is used per project.

-u, **--buildArgs** *argument*
Specify a list of additional build options. Separate the list with "," for example: **-i,MY_VAR=***value*.

Nightly Update Options

-i, **--update-index**
Update search-database index.

-m, **--generate-makefiles**
Regenerate Makefiles where necessary.

-s, **--update-symbols** *argument*
Update symbol database (static analysis). To create the data from scratch, you can supply 'rebuild' as argument.

-t, **--create-team-symbols** *argument*
Export symbol databases for shared use in a team. The argument is a quoted comma-separated list of options. Valid options are **timestamp**, **readonly**, and

checksum. The default is **timestamp,readonly,checksum**. See the online documentation for details on these options.

-x, **--update-xref** *argument*
Update cross references (static analysis). To create the data from scratch, you can supply 'rebuild' as argument.

Output

Any errors that might occur during the updates are printed out to standard error output. Other information (for example, status, what has been done, and so on) are printed out to standard output.



NOTE: No configuration management-specific actions or commands are executed within this script and the launched application. Configuration management specific synchronizations or updates relevant to the workspace (for example, **cvs-update**, ClearCase view synchronization, and so on) have to be done before this script is started.

B

Command-line Importing of Projects

B.1 Overview 391

B.2 wrws_import Reference 392

B.1 Overview

The Workbench installation includes a **wrws_import** script that allows you to import existing projects into workspaces from the command-line. The following section provides a reference page for the command.

B.2 wrws_import Reference

A script for launching a GUI-less Eclipse application that can be used to import existing projects into the workspace is available in the Workbench installation and is named:

wrws_import.bat (Windows only)

wrws_import.sh (Windows, Linux, and Solaris)

Execution

Specify the location of the **wrws_import** script or add it to your path and execute it with optional parameters, for example:

```
$ wrws_import.sh -data workspace_dir
```

Options

General Options

-d, --debug *argument*
Provide more information. The argument, if given, specifies the level of verbosity. Default is **2**, the possible options are: [**2, 3, 4**].

-h, --help
Print command help.

-q, **--quiet**
Do not produce standard output.

Eclipse Options

-data *workspace_dir*
Specify the Eclipse workspace with this option.

Import Project Options

-f, **--files** *argument*

Specify a list of project files to be imported. Separate the items in the list with commas (**,**). For example: *dir1***/.project**,*dir2***/.project**.

-r, **--recurse-directory** *argument*
Specify a directory to recursively search for projects to be imported.



NOTE: This script will not stop or fail if some projects already exist in the Workspace, the way the **Import existing projects into workspace** wizard does. It will just print out the information and continue.

C

Configuring a Wind River Proxy Host

C.1 Overview 395

C.2 Configuring wrproxy 397

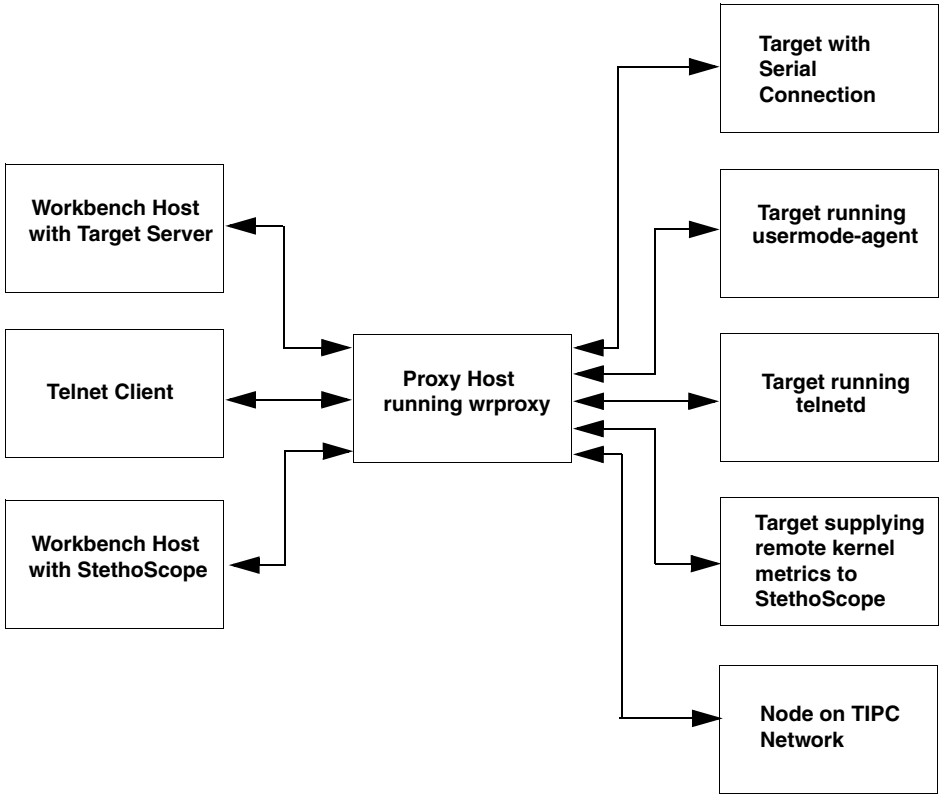
C.3 wrproxy Command Summary 399

C.1 Overview

The Wind River proxy allows you to access targets not directly accessible to your Workbench host. For example, you might run the proxy server on a firewall and use it to access multiple targets behind the firewall.

The proxy supports TCP, UDP, and TIPC (Linux only) connections with targets. Many different host tools and target agents can be connected. A simple illustration of this is shown in Figure C-1.

Figure C-1 **Wind River Proxy Example**



The proxy host itself can be one that runs any operating system supported for Workbench hosts or any host running Wind River Linux. You run the **wrproxy** command supplied with Workbench on the proxy host and configure it to route access from various tools to specific targets. The mapping is done by TCP/IP port number, so that access to a particular port on the proxy host is directed to a pre-defined target. You can start **wrproxy** and then manually configure it, or you can create a configuration script that **wrproxy** reads at startup.

C.2 Configuring wrproxy

The **wrproxy** command (or **wrproxy.exe** on Windows) is located in *installDir*/**workbench-***version*/**foundation/***version*/**x86-***version*/**bin/**. Copy it to the host that will serve as your proxy host. The following discussion assumes you have copied **wrproxy** to your proxy host and are configuring it from the proxy host.

Configuring wrproxy Manually

To configure **wrproxy** manually, start it with a TCP/IP port number that you will use as the proxy control port, for example:

```
$ ./wrproxy -p 1234 &
```

You can now configure **wrproxy** by connecting to it at the specified port.

Use the **create** command to configure **wrproxy** to map client (host tool) accesses on a proxy port to a particular target. The following example configures accesses to the proxy port 1235 to connect to the Telnet port of the host **my_target**:

```
$ telnet localhost 1234
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
create type=tcpsock;port=23;tgt=my_target;pport=1235
ok pport=1235
```

Refer to *create*, p.401 for details on **create** command arguments.

If you now connect to the proxy host at port 1235, you are connected to the Telnet port of **my_target**:

```
$ telnet localhost 1235
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

my_target login:
```

Creating a wrproxy Configuration Script

If you are typically using the same Wind River proxy configurations over time, it can be useful to use a startup script to configure it rather than doing it manually each time. You can cause **wrproxy** to read a startup script by invoking it as **wrproxy -s** *startupscript*. The script contains the commands that configure **wrproxy** as well as comments that begin with the **#** character. A simple startup script that

configures the same port setup performed manually in the previous example might look like this:

```
# This is an example of a wrproxy startup script

# Configure the proxy host port 1235 to connect to my_target Telnet

create type=tcpsock;port=23;tgt=my_target;pport=1235

# list the port configuration

list

# end of script
```

When you start **wrproxy** with this script, it gets configured as in the previous example and sends input and output to standard output:

```
$ ./wrproxy -s wrproxy_startup &
[2] 6660
Executing startup script...

create type=tcpsock;port=23;tgt=my_target;pport=1235
ok pport=1235
list
ok pport=1235;type=tcpsock;port=23;tgt=my_target
$
```

Since no control port was specified with the **-p** option at startup, the default port 17476 is used.



NOTE: There is no password management in **wrproxy**. If you want to be sure that no new connections (tunnels) are made remotely using the control port, use the **-nocontrol** option with the **-s** *startupscript* option which will disable the proxy control port.

The startup script accepts the **create**, **list**, and **delete** commands as described in *Configuration Commands*, p.399.

C.3 wrproxy Command Summary

The following section summarizes all of the Wind River proxy commands.

NOTE: For all commands, unknown parameters are ignored; they are not considered errors. In addition, the client should not make any assumption on the number of values returned by the command as this could be changed in the future. For example, the **create** command will always return the value for **pport** but additional information may be returned in a future version of the Wind River proxy.

Invocation Commands

The **wrproxy** command accepts the following startup options:

- **-p**[**ort**]—specify TCP control port. If not specified, the default of **0x4444** (17476) is used. This should be a unique number less than 65536 not used as a port by any other application, and it should be greater than 1024 which is the last of the reserved port numbers.
- -**V**—enable verbose mode.
- **-v**[**ersion**]—print **wrproxy** command version number.
- **-s** *startupscript*—specify a startup script that contains **wrproxy** configuration commands.
- **-h**[*elp*]—print **wrproxy** command help.
- **-nocontrol**—disable control port.

Configuration Commands

You can use the following commands interactively, and all except the **connect** command in a Wind River proxy startup script.

connect

Create a new Wind River proxy connection and automatically connect to it. Unlike the create command (see *create*, p.401) the connection is established immediately and all packets sent to the connection are immediately routed between the target and host.

Usage

connect type=*type***;mode=***mode***;proto=***proto* ; *connection_specific_parameters*

Where the arguments to the connect command are as follows:

type is:

- **udpsock**—UDP socket connection.
- **tcpsock**—TCP socket connection.
- **tipcsock**—TIPC socket connection (Linux only).

mode describes how the connection is handled between the proxy and the client (for example the Workbench host) and is:

- **raw**—raw mode (default).
- **packet**—packet size is sent first followed by packet content; the packet is handled only when fully received.

proto describes how the connection is handled between the proxy and the target and is:

- **raw**—proxy does not handle any protocol (default).
- **wdbserial**—(VxWorks targets only) proxy converts packet to wdbserial. When **proto** is **wdbserial**, some control characters are inserted by the proxy in the packet sent to the target so that the generated packet will be understood correctly by the target using a WDB serial backend. This is typically used to connect to a WDB agent running on a target through a serial line that is connected to the serial port of a port server (this serial line is then accessible by the proxy using a well-known TCP port of the port server).

Connection-specific Parameters

- **udpsock** and **tcpsock** connection:

  port=*port***;tgt=***tgtAddr*

  Where *port* is the TCP/UDP port number and *tgtAddr* is the target IP address.

- **tipcsock** connection (Linux only):

  tipcpt=*tipcPortType***;tipcpi=***tipcPortInstance***;tgt=***tgtAddr*

  Where *tipcPortType* is the TIPC port type, *tipcPortInstance* is the TIPC port instance and *tgtAddr* is the TIPC target address.

The response of the Wind River proxy to the **connect** command is a string as follows:

```
ok
```

or

```
error errorString
```

where *errorString* describes the cause of the error.

create

Create a new proxy port mapping to a target. The connection is not established immediately as with the connect command (see *connect*, p.399) but only when a client connects to the specified port number.

Usage

`create type=`*type***`;port=`***port***`;tgt=`***target***`;pport=`***pport***

where the arguments to the **create** command are as follows:

type=*type* is:

- **udpsock**—UDP socket connection.
- **tcpsock**—TCP socket connection. (Only **tcpsock** is allowed for a VxWorks proxy host.)
- **tipcsock**—TIPC socket connection.

port—this is the port to connect to on the target.



NOTE: If you do not assign a **port** number, the default value of **0x4444** is used.

tgt=*target*—is the host name or IP address of the target when **type** is **tcpsock** or **udpsock**, and **port** provides the UDP or TCP port number. When **type** is **tipcsock** this is the target TIPC address, and **tipcpi** provides the TIPC port instance and **tipcpt** provides the TIPC port type.

pport=*proxy_TCP_port_number*—specify the TCP port number that clients (host tools) should connect to for connection to *target_host*. This should be a unique number less than 65536 not used as a port by any other application, and it should be greater than 1024 which is the last of the reserved port numbers.



NOTE: If you do not specify a **pport** value, one will be assigned automatically and returned in the command output.

port=*target_TCP_port_number*—specify the TCP port to connect to on the target. This should be a unique number less than 65536 not used as a port by any other application, and it should be greater than 1024 which is the last of the reserved port numbers.

A simple example of using the **create** command to configure a Telnet server port connection is given in *C.2 Configuring wrproxy*, p.397.

delete

Delete the proxy configuration for a specific port.

Usage

delete pport=*port_number*

To delete the proxy configuration of a specific port, use the **`delete`** command with the port number, for example:

```
$ telnet localhost 1234
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
delete pport=1235
ok^]
telnet> q
Connection closed.
```

list

List your current configuration with the **`list`** command.

Usage

list

For example, to list your current configuration, connect to the proxy control port and enter the **list** command:

```
$ telnet localhost 1234
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
list
ok pport=1235;type=tcpsock;port=23;tgt=my_target
```

D Glossary

D.1 Introduction 403

D.2 Terms 404

D.1 Introduction

This glossary contains terms used in Wind River Workbench.

If the term you want is not listed here, you can search for it throughout all installed documentation.

1. At the top of the **Help > Help Contents** window, type your term into the **Search** field.
2. Click **Go**. Topics containing the term will appear in the Search Results list.
3. To open a topic in the list, click it.

   To switch from the Search Results list back to the help Table of Contents, click the **Show in Table of Contents** icon in the upper right corner of the help view.

D.1.1 Refining a Search

If the result set is very large, the information you are looking for might not appear in the top 10 or 15 results.

To refine a search to reduce the number of results:

1. Click the **Search Scope** link to open the search scope dialog.
2. Select **Search only the following topics** then click **New**.
3. In the working set content tree, select the topics to which you want to narrow the search, for instance **Wind River Documentation > References**.
4. Type a descriptive name in the **List name** field (such as **WR References**) then click **OK**.
5. Click **OK** to return to the help browser, where your new search scope appears next to the **Search scope** link.
6. Click **Go**. The results will be shown in the Search Results list.

For more information about online help, see **Help > Help Contents > Wind River Partner Documentation > Eclipse Workbench User Guide > Tasks > Using the help system**.

D.2 Terms

active view

The view that is currently selected, as shown by its highlighted title bar. Many menus change based on which is the active view, and the active view is the focus of keyboard and mouse input.

back end

Functionality configured into a target server on the host determines how it will communicate with a particular target agent on the target (for example, you use a **wdbrpc** back end for Ethernet connections, **wdbpipe** for VxWorks simulators, **wdbserial** for serial connections, and **wdbproxy** for UDP, TCP, and TIPC connections).

The target server *must* be configured with a back end that matches the target agent interface with which VxWorks has been configured and built.

board support package (BSP)

A *Board Support Package* (*BSP*) consists primarily of the hardware-specific VxWorks code for a particular target board. A BSP includes facilities for hardware initialization, interrupt handling and generation, hardware clock and timer management, mapping of local and bus memory space, and so on.

build spec

A particular set of build settings appropriate for a specific target board.

color context

The color assigned to a particular process in the Debug view; this color carries over to breakpoints in the Editor and to other views that derive their context from the Debug view.

cross-development

The process of writing code on one system, known as the host, that will run on another system, known as the target.

DKM

VxWorks Downloadable Kernel Module.

editor

An editor is a special type of view that is used to edit or browse a file or other resource. Each Workbench perspective displays an editor area even when no files are open.

Modifications made in an Editor follow an open-save-close life cycle model. Multiple instances of an editor type may exist within a Workbench window.

gutter

The left vertical border of the editor view where breakpoints and the program counter appear.

help key

The help key (or combination of keys) is determined by your host platform: press **F1** on Windows, **Ctrl-F1** on Linux, and **HELP** on Solaris.

kernel configuration editor

The editor that allows you to configure the kernel of a VxWorks Image project.

kernel module

A piece of code, such as a device driver, that can be loaded and unloaded without the need to rebuild and reboot the kernel.

launch configuration

A run-mode launch configuration is a set of instructions that causes Workbench to connect to your target and launch a process or application. A debug-mode launch configuration completes these actions and then attaches the debugger.

overview ruler

The right vertical border of the editor view where bookmarks and other indicators appear.

perspective

A perspective is a specific grouping of an editor and views that are useful when working on a particular task.

Default Workbench perspectives include the Application Development and Device Debug perspectives, but if you click **Window > Open Perspective > Other**, additional perspectives (such as those installed with ScopeTools) are available to you.

plug-in

An independent module, available from Wind River, the Eclipse Foundation, or from many Internet Web sites, that delivers new functionality to Workbench without the need to recompile or reinstall it.

program counter

The address of the current instruction when a process is suspended.

project

A collection of source code files, build settings, and binaries that are used to create a VxWorks system image, a kernel or RTP application, and so on.

Projects can be linked together in a hierarchical structure (displayed as a project/subproject tree in the Project Navigator) that reflects their inner

dependencies, and therefore the order in which they should be compiled and linked.

real-time process (RTP)

A VxWorks process that is specifically designed for real-time systems.

registry

The registry associates a target server's name with the network address needed to connect to that target server, thereby allowing you to select a target server by a convenient name.

system mode

When in system mode, the debugger is focused on kernel processes and threads. When a process is suspended, all processes stop. Compare with user mode.

target agent

The target agent runs on the target, and is the interface between VxWorks and all other Wind River Workbench tools running on the host or target.

target server

The target server runs on the host, and connects Wind River Workbench tools to the target agent. There is one server for each target; all host tools access the target through this server.

title bar

A view's title bar contains the view name, its icon, and the view toolbar. A highlighted title bar denotes the active view.

toolbar

A view's toolbar contains actions that apply only to that view (e.g. **Step Over** in the Debug view). The toolbar also contains a context menu that contains other actions for that view.

The main Workbench toolbar contains actions that apply to Workbench as a whole (e.g. **Search**) or that reflect the components that are installed (e.g. **Launch TraceScope**).

user mode

When in user mode, the debugger is focused on user applications and processes. When a process is suspended, other processes continue to run. Compare with system mode.

view

A view is a pane within the Workbench window that allows you to display, navigate, and manipulate the resources in your workspace. Only one view has focus (is active) at a time.

VIP

VxWorks Image Project.

window

The term window refers to the desktop development environment as a whole—the space Workbench takes up on your screen. A Workbench window can contain more than one perspective, but only one is displayed at a time.

working set

A working set is a group of resources you select because you want to view them or perform an operation on them as a group. For example, creating a working set allows you to speed up a search by restricting its scope. A working set can also help you focus by reducing the number of projects visible in the Project Navigator, the number of symbols displayed in the Outline view, and so on.

workspace

A workspace is the central location for all the resources you see in Workbench: your projects, folders, and files.

Workbench also uses the workspace to store settings that describe your working environment: which projects and views are currently open, how you have your views arranged within the perspective, whether you have breakpoints set, and so on.

The default location of the workspace is *installDir*/**workspace**, but it can be located anywhere. To keep your projects completely isolated from each other, you can create more than one workspace.

Index

A

adding
- application code to projects 162
- application initialization routines to VIPs 99
- applications to VIPs 99
- new files to projects 163
- subprojects 80

applications
- adding to VIPs 99
- initialization stubs 97
- projects, configuring 132

Attach to Target launches 300

B

back end, target server 266
ball sample program 15
basename mappings 273
board support package 100
- creating 100
- customizing manually 101
- migrating 100
- simulator 100
- Wind River Workbench 100

Bookmarks view 27
boot
- loader project 103
- build specs 107
- creating 104
- makefile 107
- overview 76
- project nodes 106
- target nodes 107

- mechanism, setting up 49
- programs
  - creating new 60
  - serial connection, configuring for 65
- ROMs
  - emulators, substituting ROM 50

booting
- command line parameters 59
- parameters
  - displaying current, at boot time 52
  - nonvolatile RAM, effect of 60
  - setting 52
  - VxWorks 58
- rebooting VxWorks 60
- TFTP, requiring 60
- troubleshooting 356
- VxWorks
  - commands 53

breakpoints
- conditional 307
- converting to hardware 308
- creating
  - expression 307
  - hardware 308

line 306
data 307
disabling 310
exporting 310
expression 307
hardware 307
importing 309
line 306
refreshing 310
removing 310
restricted 306
unrestricted 306
Breakpoints view 305
BSP
See board support package
build
applications for different boards 224
architecture-specific functions 228
complete product image 176
executables to dynamically link to shared libraries 229
failure due to locked files 345
flexible managed 205
adding build targets 208
build output 213
configuring 208
library for test and release 225
make rule in Project Navigator 232
managed
comparison of standard and flexible 207
flexible, using with linked resources 344
management 205
output
disabling prompt to add to ClearCase 383
folders 96
properties
accessing 214
dialog 214
global 215
project-specific 215
redirection root directory 118, 128, 138, 156
remote 238
connection 238
setting up environment 237
spec 216
creating 233
customizing 83
for new compilers, other tools 233
standard managed 205
configuring 208
support 205
disabled 206
target
excluding with regular expressions 211
troubleshooting
imported projects 346
user-defined 206

C

cables, connecting 46
ClearCase
disabling prompt to add build output files 383
installing plug-ins 371
using with Workbench 381
colored views 318
command line
import projects (wrws_import) 391
parameters 59
registry 257
update workspaces (wrws_update) 387
compiler
flags, add 222
new build spec 233
complex project structures 174
conditional breakpoints 307
configuring
application projects 132
file system project 191
flexible managed build 208
jumpers 46
kernel components 93
standard managed build 208
target
file system 110
hardware 46
VxWorks image project 99, 191
Console view 298

container
project
creating 177
per project type and external headers 178
subprojects 188
controlling multiple launches 293
core dump
connecting to a VxWorks 5.5.x 32
customize build specs, shared subprojects 83

D

data breakpoints 307
debug modes 320
debug server
loading symbols 260, 270
Debug view 314
debugger
debugging a VxWorks 5.5 application 29
disconnecting and terminating processes 324
single-stepping through code 319
deleting
breakpoints 310
flexible build targets 211
nodes
project 168
target 168
derived resource, not adding to ClearCase 383
development 181
disabled build support 206
Disassembly view 326
opening automatically 327
opening manually 327
Domain Name Service (DNS) 40
downloadable kernel module
application code 142
in Project Navigator 140
project
build specs 140
creating 134
files 141
nodes 140
target nodes 140
dual mode 39

E

Eclipse
basic concepts 7
log 359
using Workbench in 375
Editor 198
Kernel 94
program counter 318
environment commands (Launch Control) 296
environment variables
LD_LIBRARY_PATH 232
redirection root directory 118, 128, 138, 156
error condition command (Launch Control) 294
Error Log view 358
Exec Path on Target, troubleshooting
Linux 350
RTPs 351
execution environments, project-specific 83
exporting
breakpoints 310
object path mappings 271
expression
breakpoints 307
external headers 185

F

file
system
configuring the target 110
project files 112
project nodes 111
project, VxWorks 78
File Navigator view 198
File Transfer Protocol
See FTP server
files
manipulating 167
find and replace 201
flexible managed build
using with linked resources 344
folders, build output 96

FTP server
configuring 41
WFTPD 41

G

go into project 164

H

hardware breakpoints 307
headers, external 185
help system
accessing 12
display problems
Linux 342
Solaris 342
Windows 342

I

importing
breakpoints 309
build settings 163
object path mappings 271
resources 162
VxWorks image project 92
Include Browser view 196
initialization stubs, application 97

J

jumpers 46

K

kernel
configuration 95, 111
back ends 275
editor 94
image and symbols 267
shell 261
Kernel Configuration Editor 94
kernel metrics 331
Kernel Objects view 328
multi-process debugging 328
kernel signals 331

L

launch (terminology) 293
launch configurations
creating 284
native applications 290
Launch Control 293
launch sequence 294
LD_LIBRARY_PATH environment variable 232
library, shared
project structure 183
line breakpoints 306
linked resources, path variable 209
linking project nodes, moving and 167
linking to external sources
projects for read-only file locations 344
loading symbols to debug server 270
specifying an object file 260
location, resource 172
logical nodes 166
logs
creating a ZIP of 358
debugger back end
debug tracing 360
GDB/MI 360
debugger views
broadcast message debug tracing 361
GDB/MI 360
internal errors 361
Eclipse 359
static analysis parser 364
target manager debug tracing 364

target server
back end 362
output 362
WTX 363

M

make rule in Project Navigator 232
makefile
boot loader project 107
build properties 217
nodes
downloadable kernel modules 141
native application 159
RTP 121
shared libraries 130
VIP 97
managed build
flexible 205
configuring 208
using with linked resources 344
standard 205
configuring 208
memory
target server cache size 269
menu, Navigate 165
multiple
processes, monitoring 317
software systems 173
target operating systems or versions 216
multiple launch control 293
multi-process debugging
Kernel Objects view 328

N

native application
launching 290
project 151
application code 160
build specs 158
creating 152
files 159
nodes 158
target nodes 158
Navigate menu 165
navigation 164
New Connection wizard 258
nodes
moving and (un-)linking project 167
resources and logical 166

O

object path mappings
creating automatically 270
examples 271
exporting 271
for remote hosts 271
importing 271
why they are required 270
opening
build properties dialog 214
new window 165
project, in new window 164
operating systems, multiple 216
output folders, build 96

P

pango error 341
path variable 209
pathname prefix mappings 270
plug-ins
activating 373
adding an extension location 372
creating a directory structure 370
creating a Workbench plug-in for Eclipse 375
installing ClearCase 371
web sites 370
polled mode 39
post-launch command (Launch Control) 294
preconfigured project types, overview 74
pre-launch command (Launch Control) 294

processes
Attach to Target launches 300
disconnecting debugger 324
RTPs, running 289
profiles 90
VxWorks 5.5 compatible 91
VxWorks scalability levels 90
program counter 318
project
application code 74
boot loader 76, 103
creating 104
bsp, getting a functioning 100
build
properties, accessing 214
remote 238
system 82
closing 163, 164
configuring application 132
creating 162
for read-only sources 344
creating new 73
creating, boot loader 104
customizing VxWorks image 95
execution environment 83
files, version control of 382
go into 164
headers 178
importing Tornado 2 32
infrastructure design 176
linking application projects to VxWorks image 99
native application 151
nodes
manipulating 167
moving and (un-)linking 167
opening 163
preconfigured, overview 74
project structures 80
properties
creating project.properties file 84
limitations of project.properties files 85
using from the command line 85
using with a shell 85
wrenv syntax 84
real-time process 77
sample 75
scoping 164
shared library 78
sharing subprojects 83
structure
and build system 82
and host directory structure 81
structures, complex 174
troubleshooting imported 346
user-defined 206
VxWorks
file system 78
image 75
kernel configuration profiles 90
scalable 89
Project Navigator
boot loader projects 106
DOSFS file system projects 111
move, copy, delete 166
moving and (un-)linking project nodes 167
native application projects 158
real-time process projects 120
shared libraries 130
target nodes, manipulating 168
user-defined build-targets 232
VxWorks image projects 94
project.properties
creating 84
limitations 85
using from the command line 85
using with a shell 85
wrenv syntax 84

R

read-only sources
creating projects for 344
real-time process
and shared library 241
project 77
application code 122
build specs 120
creating 114

files 121
nodes 120
target nodes 120
See also RTPs
rebooting VxWorks 60
redirection root directory 118, 128, 138, 156
with ClearCase 382
registry 262
changing daemon default behavior 44
changing default 264
command line 257
data storage 263
error, unreachable 339
launching the default 262
remote, creating 263
shutting down 263
wtxregd 263
changing default options 44
regular expressions
to exclude contents of build target 211
remote
build 238
setting up environment 237
connection 238
rlogin 240
SSH 240
remote kernel metrics (RKM) 331
removing breakpoints 310
replacing text 201
resource locations 172
resources and logical nodes 166
RKM
See remote kernel metrics (RKM)
rlogin remote build connection 240
RTPs
and shared library 241
attaching to running 301
running 289

S

sample
ball program 15
projects 75
searching for text 201
serial lines
target server back end connection, as 64
testing 65
set, working 165
setting breakpoints
restricted 306
unrestricted 306
shared library 183
and real-time process 241
LD_LIBRARY_PATH environment variable 232
project
creating 124
nodes 130
project file 131
troubleshooting problems 354
simulator
adjusting priority of 278
establishing a new connection 277
VxWorks 100
software systems, multiple 173
source lookup path
adding sources 288
editing 326
source mode build 89
spec, build 216
SSH remote build connection 240
static analysis, description 195
StethoScope 334
structures, complex project 174
stubs, application initialization 97
sub-launch 293
subprojects 191
adding 80
container 188
symbol
file, specifying maximum size 345
table
configuring target server 267
Symbol Browser 197
symbols and kernel image 267
system mode 39
compared with task mode 320

T

target
agent
communication modes 38
introduction 37
board
configuring 46
establishing a connection 259
jumpers, setting 46
serial port 46
Terminal view 46
file system 110
name (tn) (boot parameter) 57
operating systems, multiple versions 216
server
back end settings 266
connecting
ethernet 61
serial 66
connections
establishing new 265
network 61
serial line 64
Tornado 29
core file 267
file system (TSFS) 268
making writable 269
introduction 37
kernel configuration back ends 275
memory cache size 269
symbol table 267
timeout options 269
troubleshooting 357
WDB Pipe back end 266
WDB Proxy back end 266
WDB Serial back end 266
Target Manager view 258
basename mappings 273
defining a VxWorks Simulator connection 277
New Connection wizard 258
object path mappings 270
examples 271
for remote hosts 271
pathname prefix mappings 270
shared connection configuration 274
tasks
attaching to running 301
state 301
team
sharing project.properties file 84
team, defining a path variable 209
Terminal view 46
text search 201
tgtsvr options 268
TIPC target server backend 266
Tornado
creating a target server connection 29
importing Tornado 2 projects 32
Workbench finding an installation of 28
troubleshooting
booting problems 356
building imported projects 346
creating a ZIP of log files 358
downloading 351
exception on attach 349
Exec Path on Target 351
hardware configuration 355
help system display problems
Linux 342
Solaris 342
Windows 342
Java Development Tools (JDT)
dependency 341
launch configurations 353
logs
debugger back end 360
debugger views
broadcast message debug tracing 361
GDB/MI 360
internal errors 361
Eclipse 359
Error Log 358
generated by Workbench 358
static analysis parser 364
target manager debug tracing 364
target server
back end 362
output 362
WTX 363

pango error 341
registry unreachable 339
removing unwanted target connections 342
running a process 352
shared library problems 354
startup errors 338
target connection 348
target server problems 357
VxWorks 355
workspace cannot be locked 340
TSFS
See target server, file system 268
tutorial
ball sample program 15
debugging a VxWorks 5.5 target 27
editing and debugging source files 22
Editor code assist 24
Type Hierarchy view 196

U

user mode 39
user-defined
build 206
project
creating 144
usrappinit.c 97
usrrtpappinit.c 98

V

version control, adding Workbench project files to 382
views
See Workbench views
VIO
See virtual I/O (VIO)
VIP
See VxWorks image project
virtual I/O (VIO) 298
VxWorks
boot loader project 76
booting 50
core dump, connecting to a VxWorks 5.5 32
file system project 78
creating 110
image
customizing 95
pre-built 52
image project 75
build specs 95
creating 88
files 97
in Project Navigator 94
kernel configuration profiles 90
linking application projects to 99
project nodes 95
source mode build 89
target nodes 95
rebooting 60
shared library project 78
simulator 100
defining a new connection 277
target server connection, 5.5.x 29
tutorial, VxWorks 5.5 27

W

WDB back end
Pipe 266
Proxy 266
Serial 266
WFTPD FTP server 41
Wind River
System Viewer
support libraries, excluding 89
writable target server file system 269
Workbench
Application Development perspective 15
bookmarks
creating 26
viewing 27
breakpoints
modifying 21
running to 20
setting 20

build errors 22
comparing files 23
connection definition, creating 17
creating a project 15
Editor
  bracket matching 25
  code completion 24
  Eclipse functionality 11
  parameter hints 25
finding a Tornado installation 28
help system
  accessing 12
  display problems
    Linux 342
    Solaris 342
    Windows 342
moving and sizing views 10
perspectives 8
project files, adding to version control 382
project source
  bookmarks
    creating 26
    viewing 27
  bracket matching 25
  breakpoints
    modifying 21
    running to 20
    setting 20
  code completion 24
  file history, viewing 23
  parameter hints 25
running ball sample program from build
    output 18
starting 14
target, connecting to
  connection definition 17
using in an Eclipse environment 375
using with ClearCase 381
views 10
  Breakpoints 305
  colored 318
  Debug 314
  Disassembly 326
  Editor 198
  Error Log 358
  File Navigator 198
  Include Browser 196
  Kernel Objects 328
  Symbol Browser 197
  Type Hierarchy 196
working sets 196
  using 165
workspace
  project location 72
  starting Workbench with a new 338
  switching to a different 173
  using one for multiple projects 174
wrenv
  syntax of project.properties file 84
wrws_import
  reference page 392
  script 391
wrws_update
  reference page 388
  script 387
wtxregd
  changing default options 44
  how to find API 257
  using a remote registry 263