WIND RIVER

Wind River® Workbench

USER'S GUIDE

3.1

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

Wind River Workbench User's Guide, 3.1

3 Dec 08
Part #: DOC-16292-ND-00

Contents

PART II: PROJECTS

PART I

Introduction

1 Overview

1.1 Introduction

Wind River Workbench 3.1 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. Wind River Workbench is available on Windows, Linux, and Solaris hosts.

This guide explains how to use the parts of Workbench that are not target OS specific.

1.1.1 Introducing Wind River Workbench

Workbench is an integrated development environment (IDE) for creating device software to run on embedded Wind River Linux or VxWorks systems. Workbench is optimized for both small programs and very large ones with thousands of files and millions of lines of code. It includes a full project facility, advanced

source-code analysis, simultaneous management of multiple targets, and a debugger with capabilities for managing multiple processes or threads on a single target or on multiple targets.

Workbench, VxWorks, and Wind River Linux have been developed in parallel to make it easy to develop Wind River targets.

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.

With Workbench, your application does not need to be fully linked. Partially completed modules can be downloaded for incremental testing. Modules do not need to be linked with the run-time system, 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 routines, spawn tasks, disassemble code in memory, set breakpoints, trace subroutine calls, and so on, all using the original symbol names.

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.

1.2 Starting Workbench

You can run Workbench on a Linux, Solaris, or Windows host.

1.2.1 Starting Workbench on a Linux or Solaris Host

You can find the shell command to start Workbench in your Workbench installation directory, represented in this document as *installDir*.

To start Workbench:

1. Make sure your path environment variable is set to include the path to your compiler. Typically, **which gcc** should yield **/usr/bin/gcc**.
2. From your Workbench *installDir*, issue the following command:

   ```
   $ ./startWorkbench.sh
   ```

   This is the basic startup command. You can supply arguments as described in *Running Eclipse* in *Eclipse Workbench User Guide:Tasks* in the online help. For example, these arguments give more heap space:

   ```
   $ ./startWorkbench.sh -vmargs -Xmx512m
   ```

   The resulting Welcome screen lets you click on options to see an overview, find tutorials, see what is new, and to run Workbench. To come back to this screen, select **Help > Welcome** from the Workbench window.
3. Click **Workbench**.

   Workbench displays the **Application Development** perspective. Workbench resumes the same perspective when you open it again, or you can select the default settings by choosing **Window > Reset Perspective**.

1.2.2 Starting Workbench on a Windows Host

To start Workbench on a Windows host:

1. From the **Start** menu, select **All Programs > Wind River > Workbench 3.*x* > Workbench 3.*x***.

1.2.3 Deciding which Workspace to Use

Workbench uses a *workspace* to hold different types of information, including:

- Information about a set of projects, including project names, lists of files for each project, and build specifications.

- Information about the current session, including the types and positions of your windows 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.

1.3 Wind River Documentation

A wide variety of documentation in many different formats is available to Workbench customers. See the *Getting Started Guide* for your VxWorks or Linux platform for a description of the full document set.

1.3.1 Introducing this Guide

This document is divided into the following parts:

Part I. Introduction (this chapter) provides an overview of Workbench documentation.

Part II. Projects provides detailed information on how to use projects in Workbench, including pre-defined projects, user-defined projects, and using the Project Explorer.

Part III. Development explains how to create projects and use the Project Explorer; how to navigate and edit source files; how to build projects; and how to follow common project building use cases.

Part IV. Target Management describes connecting to targets (with or without USB or TIPC), and how to create and manage your target connections.

Part V. Debugging provides an in-depth look at debugging operations, including launching programs, and managing breakpoints

Part VI. Using Workbench in a Larger Environment describes how to incorporate plug-ins (such as ClearCase) into Workbench, how to incorporate Workbench into an existing Eclipse environment, how to use Workbench with your version control system, and how to use Workbench with a team.

Part VII. Reference describes how to use the command line to update workspacesfor automated builds and how to import projects. It also provides a glossary.

Understanding the Typefaces in this Guide

In this document, italics mark terms being introduced, as well as placeholders for values you supply. Bold indicates literal values. For example, *installDir* refers to the location in which you have installed Workbench. (By default, this is **C:\WindRiver** on Windows hosts and **$HOME/WindRiver** on Linux and Solaris hosts.)

Bold also indicates menu choices, as in **File > New > Project**, meaning to select **File**, then **New**, then **Project**. Commands that you enter on a command line are shown in bold, and system output is shown in typewriter text, for example:

```
$ pwd
/home/mary/WindRiver
$
```

The screenshots in this document were taken on a host that may not be the same as the host you are using, therefore they they may differ slightly from what you see on your screen.

1.4 Accessing and Searching Workbench Context-Sensitive Help

For more information about Workbench functionality and user interface, you can access the context-sensitive help by pressing the help key for your host. On Windows press **F1**, and on Linux and Solaris press **CTRL+F1** 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**.

NOTE: The **Help** button on Solaris keyboards does not open Workbench help due to a problem in Solaris/GTK+. Instead, use **Ctrl+F1** to access help.

1.4.1 Searching for Information in the Documentation

Many Workbench terms are listed in *F. Glossary*.

If the term you want is not listed, there are two ways you can search for it throughout all installed documentation.

Help View

1. Press the help key for your host (see *1.4 Accessing and Searching Workbench Context-Sensitive Help*, p.7) to open the Help view within Workbench itself.
2. At the bottom of the Help view, click **Search**, then type the keyword or phrase you are looking for into the **Search expression** field. Click **Go**.
3. Links for relevant topics appear in the Help view. To open the document containing that topic, click the link.

To switch from the Search Results list back to the help Table of Contents, click the **All Topics** link at the bottom of the help view.

Help Browser

1. From the Workbench toolbar, select **Help > Help Contents** to open the help system in a standalone browser.
2. At the top of the browser, type your term into the **Search** field. Click **Go**.
3. Links for relevant topics appear in the **Search Results** list. To open the document containing that topic, click the link.

To switch from the Search Results list back to the help Table of Contents, click the **Contents** link at the bottom of the help browser.

1.4.2 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.

Restricting a Search to Local Help

To refine a local help search to reduce the number of results, follow these steps:

1. In the Help view, click **Default** next to the **Search scope** link to open the **Select Search Scope Sets** dialog.
2. Click **New** to open the **New Scope Set** dialog, type a name for your search scope, then click **OK**.
3. In the scope set list, select the scope set you want to define, then click **Edit**.
4. Select **Search only the following topics**, then select the local help sources to which you want to restrict the search, for instance **Wind River Documentation > References**. 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.

Restricting a Search to Another Information Source

By default, the Search Scope dialog opens to show options for searching local help. To select a different source of information for your search scope, follow these steps:

1. In the Help view, click **Default** next to the **Search scope** link to open the **Select Search Scope Sets** dialog.
2. Click **New** to open the **New Scope Set** dialog, type a name for your search scope, then click **OK**.
3. In the scope set list, select the scope set you just created, then click **Edit**.
4. From the **Search Scope** dialog, click **New**, then select **Info Center** or **Web Search**, then click **Finish**. Your new information source appears in the list on the left side of the dialog, and new options appear on the right.
5. Fill in the URL you want to connect to, adjust any other information as necessary, then click **OK**.
6. Click **OK** to return to the help browser, where your new search scope appears next to the **Search scope** link.
7. Click **Go**. The results will be shown in the Search Results list.

NOTE: If you have more than one Scope Set defined, your term or expression will be searched in all scopes unless you further restrict your search.

Click **Search Scope** to display all defined search scopes, uncheck the scopes you do *not* want to include, then click **Go** to rerun the search.

1.4.3 For More Information

For details about new Workbench features, including non-Eclipse enhancements to Workbench, see the release notes at **http://www.windriver.com/support/**).

For online documentation available from within Workbench, open **Help > Help Contents**.

For more information about Eclipse and the projects introduced here, see the Eclipse documentation available from the Workbench help system, as well as documentation at **http://www.eclipse.org/documentation/**.

PART II

Projects

2

Building and Debugging a Sample Project

2.1 Introduction

This chapter introduces you to Wind River Workbench so that you can become familiar with its perspectives, views, and development concepts, in a hands-on setting.

No special hardware or system setup is required. You can find the ball sample program in a Workbench installation directory.

In the course of working with this sample program, you will:

- Create a project
- Import source files
- Build the project
- Connect to a target
- Set breakpoints
- Step through code
- Set a watch on a variable

- Run code
- Edit source files
- Track build errors
- Debug a project
- Rebuild and rerun your code

2.2 Creating a Project and Running a Program

This chapter uses the **ball** sample program, written in C. The 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 view.

First, you create a new project in your workspace, then you import the ball source files into it.

NOTE: There are minor differences in setting up this project, depending on whether you are using VxWorks or Wind River Linux.

2.2.1 Resetting to the Default Perspective

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 have experimented with opening perspectives and moving views, switch back to the Application Development perspective by selecting **Window > Reset Perspective**.

2.2.2 Creating the ball Project

VxWorks and Wind River Linux take slightly different approaches to creating the sample project.

Creating the Project for the VxWorks Simulator

To create the project for the VxWorks simulator:

1. Select **File > New > VxWorks Downloadable Kernel Module Project**.
2. In the **Project Name** field, type **ball**.
3. Keep **Create project in workspace** selected.

   The default active build spec is for the VxWorks simulator, so you do not have to adjust any other settings.
4. Click **Finish**.

The ball project appears in the Project Explorer.

Creating the Project for a Linux Target

To create the project for Linux:

1. Select **File > New > Wind River Linux Application Project**.
2. In the **Project Name** field, type **ball**.
3. Keep **Create project in workspace** selected.
4. Click **Next**.
5. The supplied example uses a PowerPC target and associated cross-development tools. To specify the correct target and tools:
   a. Accept the build defaults: click **Next** and continue clicking **Next** until you come to the Build Specs window.
   b. In the **Active build spec** field, select **common_pc-glibc_std-i686-wrlinux_2_0**.
6. Click **Finish**.

The ball project appears in the Project Explorer.

2.2.3 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-3.***x*/**samples/ball** directory, then click **OK**.

NOTE: Be sure to use files from this directory and not those for the sample C++ program available from the **File > New > Example** menu.

5. Enable the **ball** checkbox, then click **Finish**.

If the contents of the ball project folder are not visible in the Project Explorer, click the plus sign or down arrow next to the folder name. Notice that project files are in black, build targets are green, and read-only files are gray.

2.2.4 Building the ball Project

To build the ball project:

1. Right-click the ball folder in the Project Explorer and select **Build Project**.
2. Click **Continue** if asked if you want Workbench to generate include paths.

Build output appears in the Build Console at the bottom of the screen, and the output file **ball.out** appears in the ball folder.

2.2.5 Connecting to the Target

In general, you must create a new target connection definition before you can connect to a new target. For more information on how to do that, see *10. Connecting to Targets*.

However, for this tutorial, you can use the predefined connection that appears in the Remote Systems view.

For VxWorks:

1. In the Remote Systems view, right-click **vxsim0 (Wind River VxWorks 6.x)**.

2. Choose **Connect 'vxsim0'**.

   A VxWorks simulator shell opens, and the connection details appear in the Remote Systems view.

NOTE: Minimize the shell if you wish, but do not close it.

For Linux:

1. In the Remote Systems view, right-click **LinuxHost**_*user* **(Host OS (Native Development))**.
2. Choose **Connect 'LinuxHost**_*user*'.

   The usermode-agent shell opens, and connection details appear in the Remote Systems view.

2.2.6 Running the ball Application in the Debugger

Follow these instructions to start the ball application in the debugger.

For Linux:

1. Right-click **ball.out** in the Project Explorer and select **Debug Process on Target**.
2. Click **Debug**.

For VxWorks:

1. Right-click **ball.out** in the Project Explorer and 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.
3. Click **Debug**.

Workbench automatically builds the **ball** project and switches to the Device Debug perspective, with **main.c** open in the Editor. The Debug view shows the program stopped at **main( )**. The Breakpoint view appears with no breakpoints listed. The Variables view shows the values of the variables that have been defined to this point.

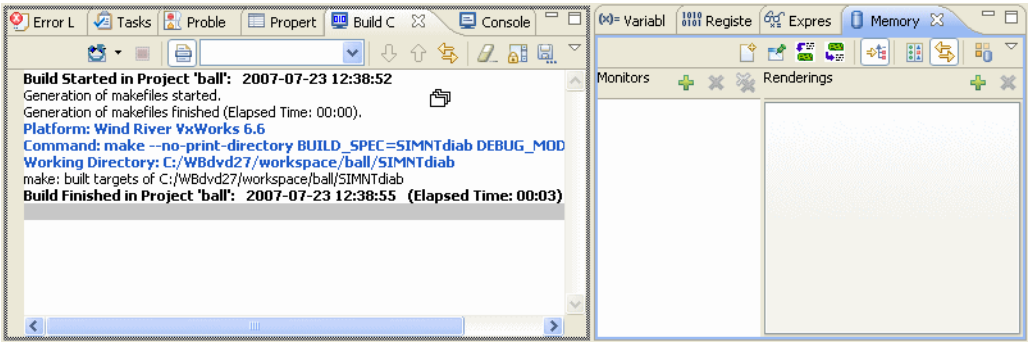
2.2.7 Setting Up the Device Debug Perspective

The action of the ball program is displayed by viewing the memory address of the **grid** global variable in the Memory view.

1. If it is not already open, select **Window > Show View > Other > Debug > Memory** to open the Memory view.

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

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. If it is not already visible, select **Window > Show View > Expressions**, then:

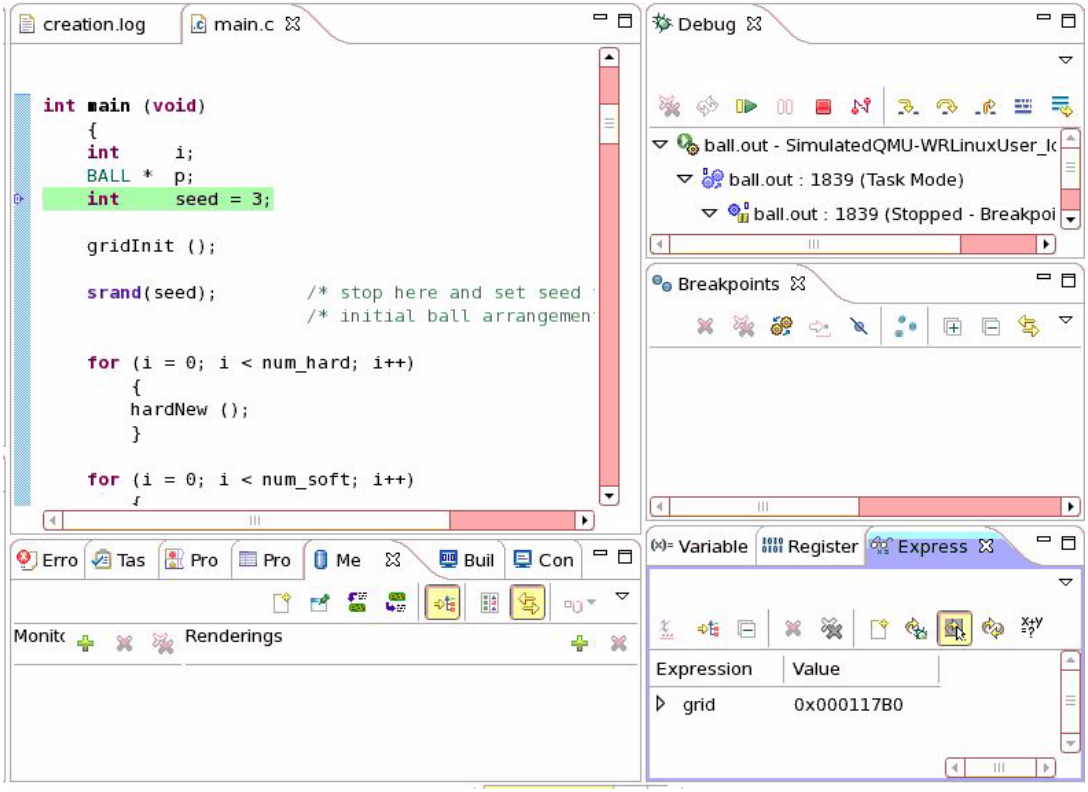
   a. Select the Expressions tab and expand it enough to see the Expression, Type, and Value columns.

   b. Click **Add new expression** in the Expression column.

   c. Type **grid** and press **ENTER**.

   The memory address of the **grid** global variable appears in the **value** column. It can vary from one session to another if, for example, you compile with the GNU compiler instead of the Wind River Compiler.

The Device Debug perspective now appears approximately as shown:



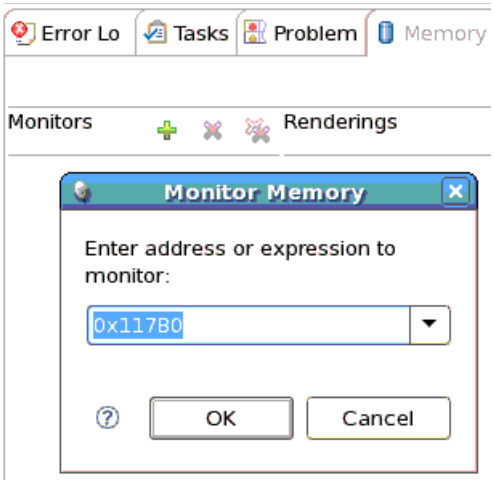
2.2.8 Stepping Through Code

This procedure takes you through the initialization of the Grid array in the ball sample program.

1. Press **F6** (or use the **Step Over** icon in the Debug view) twice to step from the entry point of **main( )** to the call to **srand( )**.

   Using **F6** twice causes Workbench to step over and complete the execution of **gridInit( )**. (All the run controls are available on the **Run** menu, and also as toolbar icons in the Debug view.)

2. In the Memory view, click the **Add Memory Monitor** icon in the Monitors toolbar, enter the value (address) of the **grid** array from *2.2.7 Setting Up the Device Debug Perspective*, p.18, and click **OK**.



3. Adjust the Memory view to show the box outlining the grid on which the balls will bounce:

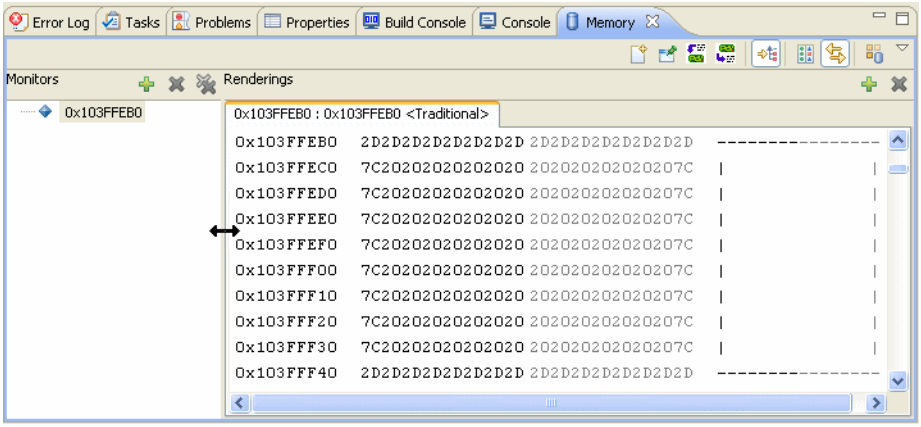
   a. Right-click in the **Renderings** column and select **Cell Size > 8 bytes**.

   b. Drag the right or left borders of the Memory view to make it wide enough, and drag the top and bottom borders to make it high enough for the box in the text area of the window to appear correctly.

   c. Adjust the relative sizes of the Monitors and Renderings panes within the Memory view before you can see the correct columns in the Renderings pane.

When set up for this tutorial, the view should look similar to Figure 2-1.

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



NOTE: Make sure the address you entered in the **Memory** window is correct for the **grid** global variable. If you see dots instead of the box, click the Debug view's toolbar **Step Over** once or twice. (Or press **F6.**)

The box may be empty now, but as the program runs, characters representing different types of balls (two zeros, two @ signs, and two asterisks) appear in the empty box, and collide with the walls and each other.

2.2.9 Setting and Running to a Breakpoint

The following procedure explains how to set a breakpoint and tell the application to run until the breakpoint is reached:

1. Scroll down in **main.c** past the three initialization **for** loops and set a breakpoint at the following **while** statement, by double-clicking in the gutter next to the **while** statement:

```
/*
 * To watch the balls move, display global "grid" in the
 * Memory window, and place a breakpoint with property
 * "Continue on Break" enabled at either the next statement
 * or at the "for" statement for a more detailed view.
 */

    while (! finished)
        {
        for (p = pMovableBalls; p != NULL ; p = p->pNextMovable)
            {
            p->pBallMove (p);
            }
        }

    return 0;
    }
```

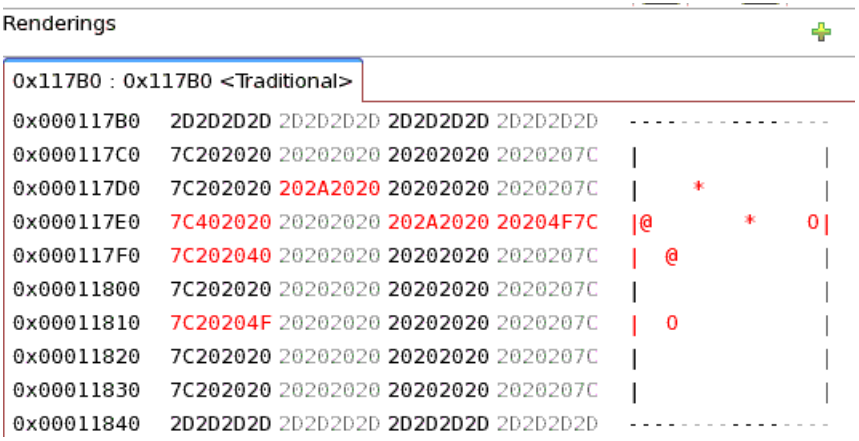
A blue dot appears in the gutter.

The Breakpoints view displays the module and line number of the breakpoint:

NOTE: If no comment appears above the **while** statement, click the **+** symbol in the gutter next to it to unfold the comment. You can click the **+** symbol again when you are done reading the comment to fold it away.

2. Press **F8** (or click the **Resume** icon in the Debug view's toolbar) to run to the breakpoint.

   Execution stops each time it hits the breakpoint. The Memory view changes each time you resume execution. The highlighting in the following figure indicates changes after the first refresh.



2.2.10 Modifying the Breakpoint to Execute Continuously

Next, change the behavior of the breakpoint so that the application does not stop executing at each break, but instead refreshes the display:

1. Right-click the breakpoint in the gutter, or in the Breakpoints view, and select **Breakpoint Properties**.

   The Line Breakpoint Properties dialog appears.

2. Click the **Continue on Break** check box.
3. Click **OK**.
4. Press **F8** or click the **Resume** button to watch the balls bounce in the Memory view.

To stop the program:

1. Open the **Breakpoint Properties** dialog again.
2. Clear **Continue on Break**, then click **OK**.

   The balls may bounce once more, but then stop.

2.3 Editing and Debugging Source Files

Workbench supports editing code, building your project, finding where the build fails, and tracking and fixing errors.

Before continuing, switch back to the Application Development perspective by clicking its icon in the upper-right corner of the Workbench window.

2.3.1 Using Bookmarks in Lines and 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. You can bookmark a file or a particular line of code within it. Open the Bookmarks view by selecting **Window > Show View > Bookmarks**.

The following procedures introduce an error in the source code and bookmark it. Later procedures explain how to recognize that the build does not complete because of the error; how to find the error; and how to rebuild the project.

Introducing an Error for this Tutorial

To create a failed build, introduce an error into the ball project code:

1. In the Project Explorer, 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.

Creating the Bookmark to the Error

To create the bookmark to the line containing the error:

1. Right-click the left gutter of the editor next to that line, then select **Add Bookmark**.

(The *gutter* is the shaded strip at the left edge of the Editor view. It is likely in blue, and it may be very thin.)

2. In the **Add Bookmark** dialog box, enter a bookmark name (or accept the default **gridInit( )**, and click **OK**.

   A small bookmark icon appears in the gutter.

3. To save the file with the error, click the **Save** button on the main toolbar.
4. Close all open files by clicking the **X** on the tab of their Editor view.

Locating and Viewing the Bookmark

To locate and view bookmarks:

1. To open the Bookmarks tab, select
   Window > Show View > Other > General > Bookmarks.

   The Bookmarks view shows all bookmarks in the project. Because there is only one bookmark in this project, only one bookmark appears in the list.

2. Double-click the entry. If you accepted the default name, it is **gridInit( )**.

   The **main.c** file opens in the editor with the bookmark line highlighted.

3. Use the **X** in the **main.c** tab to close the file without making any changes.

   (That is, leave the error so you can follow the tutorial steps below, which include correcting errors in builds).

2.3.2 Building a Project with Introduced Errors

In *Introducing an Error for this Tutorial*, p.24, you introduced an error in the ball program source code that you will fix in the following procedure, which demonstrates how to find and fix build errors using Workbench.

To fix build errors:

1. In the Project Explorer, right-click the ball folder and select **Build Project**.
2. Click **New Build Target** and click **Finish** in the resulting Build Target dialog (thereby accepting the default names).
3. Click **Continue** if a prompt appears concerning the include search path.

NOTE: If necessary, add **/usr/include** to the include search path.

Build output displays in the Console tab, and entries also appear in the Problems tab. Click these tabs to move back and forth between their contents (or rearrange your window to view them both simultaneously).

Because you created an error in the **main.c** file, errors are encountered during the build process. Notice that Workbench enters a task in the Problems view showing the type of error, the file the error is in, the location of the file, and the location of the error. It also shows warnings that occur in other locations in the code because of the error.

4. Double-click the error line in the Problems view.

   The editor focuses on (or just past) the erroneous line, with an identical white **X** in a red circle to mark the position of the error in the file.

5. Re-enter the semicolon (**;**) character you deleted in earlier steps.

   This was in **main.c**, after **gridInit( )**.

6. Right-click the bookmark icon in the gutter and select **Remove Bookmark**.
7. Save and close the file.

2.3.3 Rebuilding the Project Without Errors

To rebuild the ball project:

1. Right-click the ball folder at the top of the project tree.
2. Select **Rebuild Project**.

Now the project builds without errors.

2.3.4 Displaying a File's History

Workbench tracks all changes that are made to any files in the project.

At this point, several changes have been made to the **main.c** file.

To display the change history on the **main.c** file:

1. Right-click the file in the project tree and select **Compare With > Local History**.

The History tab opens. (It may be labeled just "H.") The upper area of the History view displays a list of the dates and times when the file was changed.

2. Double-click one of the list entries.

   A Comparison Viewer dialog opens, displaying the file on the left as it was at the date and time of the list entry that you double clicked. To the right of the file display is the file as it was in the immediately preceding version.

3. Move your cursor over the Compare Viewer's toolbar icons to read the functions provided, such as copying changes from the left to the right file, moving to the next difference or the next change, and so on.

4. Close the Compare Viewer.

   If you see a Save Resource error notice that **main.c** has been modified but is still open, click **Yes** to save the changes you made.

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 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.

The editor can emulate the **vi** or **emacs** editors. To set your preference, click the appropriate icon in the top toolbar. Refer to additional editor preferences in **Window > Preferences > General > Editor**, and to additional online information at **http://help.eclipse.org**.

2.4.1 Changing File Preferences

To open a source file and change its properties (for example, its font properties):

1. Expand the **ball** folder and double-click **main.c**.

   The **main.c** file appears in the Editor, using a fixed font and preference-based color syntax highlighting.

2. Select **Window > Preferences > General > Appearance > Colors and Fonts**.
3. Expand the **Basic** folder in the **Colors and Fonts** pane and select **Text Font**.
4. Click **Change** and select:
   - Family: Courier 10 Pitch (or some other font than the original)
   - Style: Regular
   - Size: 10
5. Click **OK**.
6. Move the **Preferences** dialog box out of the way to uncover the **main.c** editor window if necessary. Click **Apply** to see the changes without closing the dialog box.
7. Explore and experiment with other preferences. Click **Restore Defaults** if you prefer the default settings.
8. Click **OK**.

2.4.2 Navigating in the Source

You can find your way in source files using:

- The Outline view, which lists elements and symbols in the file in focus in the editor
- **Find** utilities to locate symbols, elements, and strings

Using the Outline View

In the Outline view, click any item in the list to focus the Editor on its declaration. Hover over the buttons in the toolbar to see how to filter the list of names in the view. You can sort them, hide fields and static members, and hide non-public members. (Note that the elements are sorted within types.)

The Outline view is limited to the names declared in the file that is open in the Editor. To find the declaration of any symbol appearing in the Editor, right-click the symbol in the Editor view, then click **Open Declaration**.

For example:

1. In the Outline view, click **main(void): int**.
2. In the Editor, click the call to **gridInit( )** to highlight that line.

3. Right-click **gridInit( )** and select **Declarations**. Specify the scope for your search: workspace, project, or working set.

   The Search tab displays the ball node, which you can open onto a list of the project files in which **gridInit( )** has been declared.

Finding Elements (Text Filtering)

To open a more advanced symbol or element search dialog box:

1. Select **Navigate > Open Element**.

   The **Open Element** dialog appears. The **Choose an element** field provides match-as-you-type filtering. For example, to find all names starting with **task**, start typing in the field. As you type the letter **t**, then **a**, and so on, the number of elements displayed narrows so you can select the one you want from the list.

2. Enter **grid*Ball**.

   As you enter a **Pattern** for the symbol, including wild cards, Workbench lists all matching elements. All those that match **grid*Ball** are displayed.

3. Highlight one and press **ENTER** to see its declaration.

The **Choose an element** field also supports wild-card matching

- A question mark (**?**) matches any single letter
- An asterisk (*****) matches any number of arbitrary letters
- A dollar sign (**$**) matches the end of a string

For example, to find all names that end with **0** (zero), use the filter ***0$**.

Finding Strings

To find and optionally replace any string in the active Editor view, use these functions:

- **Edit > Find/Replace** (or use **CTRL+F**)
- **CTRL+K** to **Find Next**
- **CTRL+SHIFT+K** to **Find Previous**

See the **Edit** menu for other choices.

The **Search** menu provides a **grep**-like search for strings in any file in the workspace or in any location.

2.4.3 Using Code Completion to Suggest Elements

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.

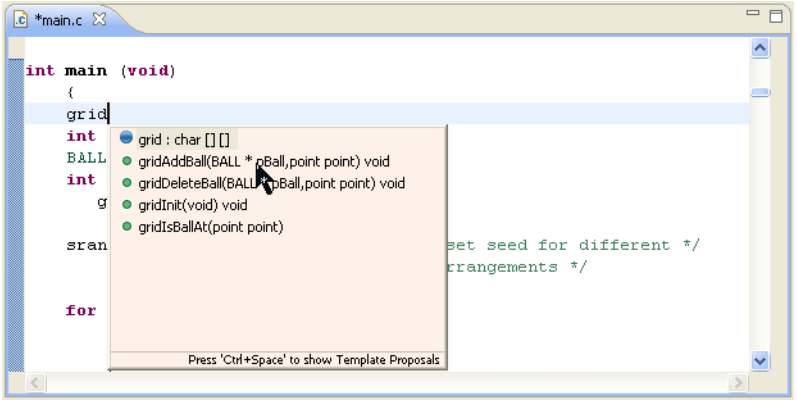
For example, in ball's **main.c**:

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: To change indentation, brace style, and other code formatting, select **Window > Preferences > General > Editors > Wind River Workbench Editor**.

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

   As you continue to type, your choices narrow.



3. Select **gridAddBall(BALL*, pBall,point point void)** from among the choices, then press **ENTER** to add the routine to the cursor location in the Editor.

2.4.4 Getting Parameter Hints for Routine Data Types

Parameter hints describe the data types that a routine 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 parameter hints.

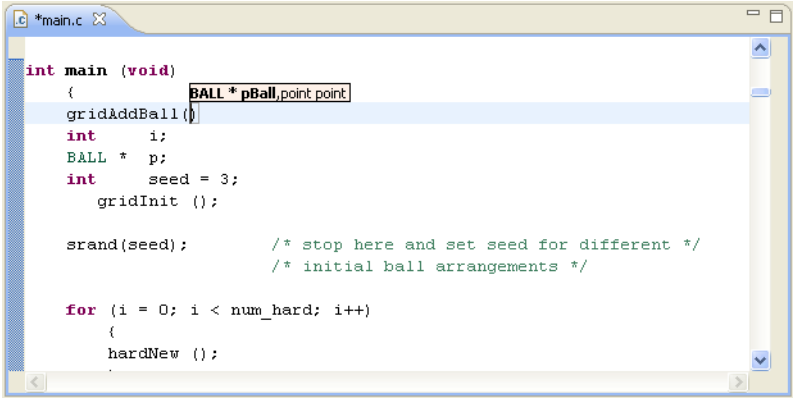
To see the data type descriptions for a routine's parameters:

1. Type a routine name followed by an open parenthesis, for example, **ballNew(** .

   When you type the open parenthesis, the editor supplies the closing one, and places your cursor between the two.

2. **Press CTRL+SHIFT+SPACE**.

The parameter hints appear, reminding you of the parameter types defined by the routine. You can copy and paste them if you wish.



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

2.4.5 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( )**.

2.4.6 Using Bracket Matching to Find Code Open and Close Sections

Bracket matching lets you find the end (or beginning) of sections of code such as functions, comments, and so on.

Bracket matching operates on the following characters: **( )** (parentheses); **[ ]** (brackets); **{ }** (braces); **" "** (quotes); **/* */** (comments); and **< >** *(C/C++ only)*.

To use bracket matching:

1. Position the cursor after one of the bracket matching characters, either the opening or closing character.

   A rectangle of very thin lines encloses the matching character. If your cursor is after an opening character, the rectangle appears after the corresponding closing one. If after a closing character, it appears after the corresponding opening one.

2. Press **CTRL+SHIFT+P**.

   The cursor moves to just after the matching character.

3. Press **CTRL+SHIFT+P** a second time.

   The cursor returns to the original position.

3 Projects Overview

3.1 Introduction

Workbench uses *projects* as logical containers and as building blocks that can be linked together to create software systems.

Using the Project Explorer, you can visually organize projects into structures that reflect their inner dependencies, and therefore the order in which they are to be compiled and linked.

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

3.2 Choosing Locations for Workspaces and Projects

When you start Workbench for the first time, it suggests that you create your workspace directory within the Workbench installation directory. However, this is not a requirement, or even necessarily desirable.

If you change the location of your workspace directory to outside of the Workbench installation tree, for example at the root of your existing source code tree, you can preserve the integrity of your projects after product upgrades or installation modifications. For multiple, unrelated source code trees, you can use multiple workspaces.

When you create a new project, Workbench creates a directory of the same name in the workspace. For Wind River Linux platform projects, Workbench creates an additional directory named with the project name and a **_prj** extension.

The default location for the project directory is in your workspace, but you can create it elsewhere as necessary. Below are 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 *6.3 Adding Resources and Files to Projects*, p.66).
- 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.

NOTE: If you created a workspace with a previous version of Workbench, the workspace structure must be updated before you can open it with the current version of Workbench.

A dialog appears informing you that this update may make it incompatible with previous versions; click **OK** to update and open the workspace, or **Cancel** to select a different workspace.

3.3 Creating New Projects

All application code is managed by projects of one type or another.

Although you can create projects anywhere, you would generally create them in your workspace directory (see *3.2 Choosing Locations for Workspaces and Projects*, p.34). If you follow this recommendation, there will generally be no need to navigate out of the workspace when you create projects. (If you do create projects outside the workspace, you must have write permission at the external location because Workbench project administration files are written there.)

You can use Workbench to create new projects, as well as projects from the installed examples, as follows:

- To create a new project, select **File > New > Wind River Workbench Project** to open the New Wind River Workbench Project wizard to help you create one of the pre-configured project types.

  You may see different project types depending on your installed software. (See *3.4 Project Types*, p.36 for an overview.) Platform developers have access to kernel source and kernel tools, whereas application developers do not.

- 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.

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 dialog, then press the help key for your host.

3.3.1 Modifying Project Settings

To modify the project creation wizard settings after the project exists:

1. In the Project Explorer, right-click the project to modify.
2. Select **Properties**.

   For more information about project properties, see *8.4 Accessing Build Properties*, p.104.

To modify project structural settings, such as the sub- and superproject context of the project you are creating:

1. Right-click a project folder.
2. Select **Project References > Add as Project Reference.**
3. Select another project to be the super-project.

3.4 Project Types

You can use Workbench to create, modify, execute, and debug projects of the following types:

- Linux-Specific Projects
- VxWorks-Specific Projects
- User-Defined Projects
- Native Application Projects

The following sections provide introductory descriptions of these project types. See *4. Creating Native Application Projects* and *5. Creating User-Defined Projects* for more information about native application and user-defined projects, and see the

appropriate version of *Wind River Workbench By Example* for details on the Linux- and VxWorks-specific projects.

3.4.1 Linux-Specific Projects

Linux projects are described fully in *Wind River Workbench By Example (Linux Version)*.

Wind River Workbench Projects

Wind River Workbench projects allow you to specify a target operating system, either the Workbench host (native development), or Wind River Linux Platforms 2.0. For either of these, you can create C or C++ applications, a static library, or a user-defined application. With the Wind River Linux Platforms 2.0, you can also create build types for platforms or kernel modules.

Wind River Linux Application Project

Wind River Linux application projects are developed on the host computer and deployed on the target. These projects dynamically generate build specs for Wind River Linux-supported board architectures, and kernel and file system combinations.

Wind River Linux User-Defined Projects

Wind River Linux user-defined projects are those that run on a Wind River Linux Platforms 2.0 target, with a user-defined build based on an existing makefile.

Wind River Linux Platforms Project

Wind River Linux Platforms projects are developed on the Linux host computer and deployed on pre-defined targets. Platform projects can include prebuilt or customized kernels, and pre-defined or customized file systems. Workbench provides special support for Wind River Linux Platforms projects including kernel and user space configuration tools, and multiple build targets.

Wind River Linux Kernel Module Projects

Wind River Linux kernel module projects allow you to specify superprojects as well as subprojects. You can specify a build (make) command, suffixes for source files, kernel root directory, target architecture, and cross-compile prefix.

Customer-Specific Linux Application Project

Customer-Specific Linux application projects are built and reside on the host computer, but run on the target kernel. For these projects, Workbench basically provides a generic managed build allowing you to derive your specific build specs from a set of predefined ones.

NOTE: To enable Customer-Specific Linux projects: select **Window > Preferences > Wind River > Capabilities**. Expand **Development**. Check **Customer-Specific Linux Development**. Click **Apply**.

Customer-Specific Linux Kernel Project

Customer-Specific Linux kernel projects let you configure (customize or scale) and build a Linux kernel to run on the target. These projects basically wrap the typical Linux kernel command line build. Because your subsequent Linux application projects will run on the Linux kernel on the target, this is often a necessary first project to build, unless you already have a target kernel.

3.4.2 VxWorks-Specific Projects

VxWorks projects are described fully in *Wind River Workbench By Example (VxWorks Version)*.

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.

VxWorks Source Build Project

Use a VxWorks Source Build project to rebuild the default VxWorks libraries to include support for products shipped with your Platform (for example, networking products or security products). This project is also the way to exclude unneeded components to make the VxWorks kernel image smaller. Once you have configured a VxWorks Source Build project to suit your needs, a VIP based on it will include those customizations.

VxWorks Boot Loader/BSP Project

Use a *VxWorks Boot Loader/BSP* project to create a VxWorks boot loader (also referred to as the VxWorks boot ROM) to boot load a target with the VxWorks kernel. You can also use this type of project to copy sources for an existing BSP into your project, then customize them without changing the VxWorks install tree.

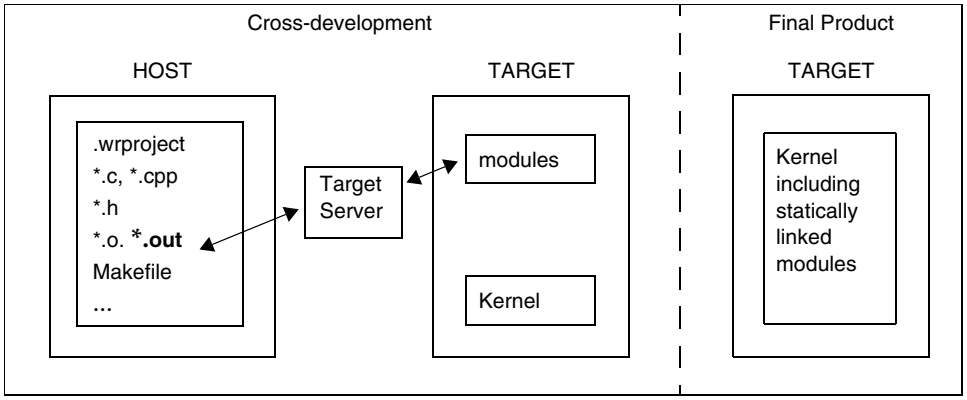
Boot loaders are used in a development environment to load a VxWorks image that is stored on a host system, where VxWorks can be quickly modified and rebuilt. Boot loaders are also used in production systems where both the boot loader and operating system image are stored on a disk.

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

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). Figure 3-1 illustrates a situation without a file system on the target.

Figure 3-1 **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.

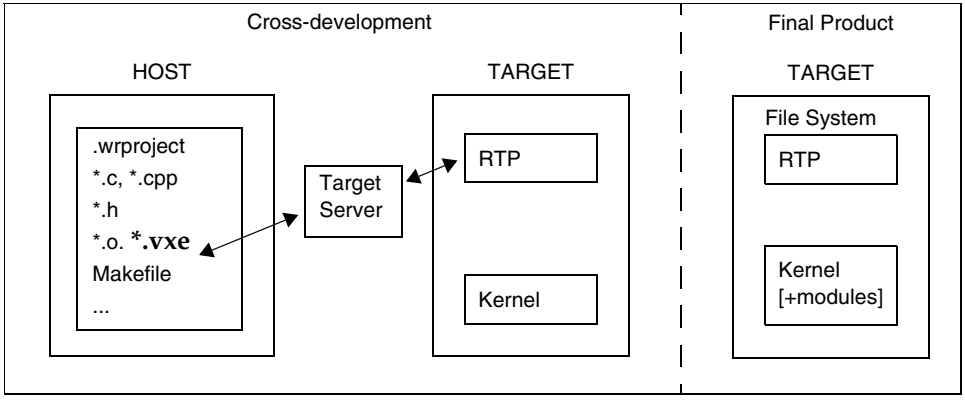
VxWorks Real-time Process Project

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.

Figure 3-2 shows how executables, when loaded into a Real-time Process, run as a separate entity from the kernel.

Figure 3-2 **Real-time Processes: Overview**



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 a VxWorks ROMFS File System project.

You can also use VxWorks Shared Library projects to create subprojects that are statically linked into other project types at build time.

VxWorks ROMFS File System Project

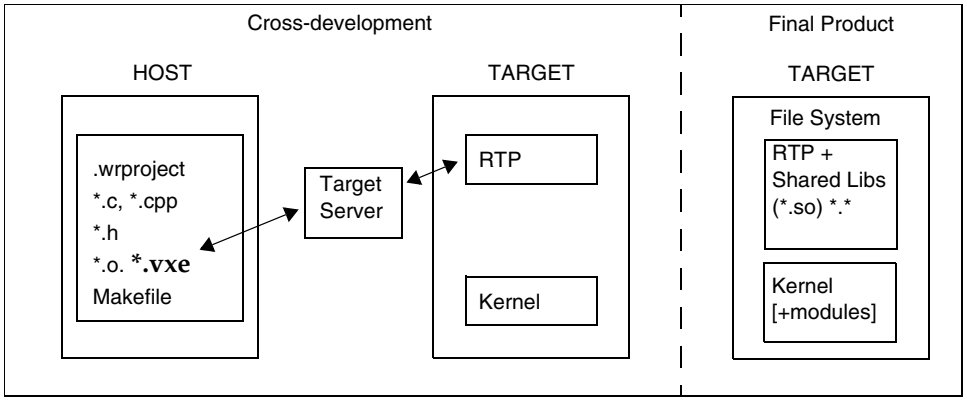
Use a *VxWorks ROMFS File System* project as a subproject of any other project type that requires target-side file system functionality.

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), 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: The name *ROMFS* has nothing to do with ROM media. It stands for *Read Only Memory File System*.

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



3.4.3 User-Defined Projects

User-Defined projects do not use Workbench build support or pre-configured features. They can be anything. It is up to you to organize and maintain the build system, the target file system contents, the makefile, and so forth.

The user interface does provide support for the following:

- Configuring the build command used to launch your build utility. This allows you to start builds from the Workbench GUI.
- Creating build targets in the Project Explorer that reflect rules in your makefiles. This allows you to select and build any of your make rules directly from the Project Explorer.
- Viewing build output in the Build Console.

You can specify the project tree structure and project references according to specific projects that you have already defined, which are then referenced as subprojects. The makefiles for user-defined projects can still use values from the build specs, to help set the correct cross-compile toolchain and architecture.

Refer to *5. Creating User-Defined Projects* for more information on working with this type of project.

3.4.4 Native Application Projects

Native Application projects are C/C++ applications developed for your host environment. Workbench provides build and source 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.

See *4. Creating Native Application Projects* for more about building native application projects.

3.5 Structuring Projects

All individual projects 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.

3.5.1 Adding Subprojects to a Project

Workbench provides the following ways to create a subproject/superproject structure:

- You can use the **Add as Project Reference** dialog. In the Project Explorer, right-click the project that you want to make into a subproject and choose **Project 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 Explorer, 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 subnodes of their parents (superprojects); see Figure 3-4 and Figure 3-5 for an example from VxWorks.

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.

3.5.2 Removing Subprojects

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

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

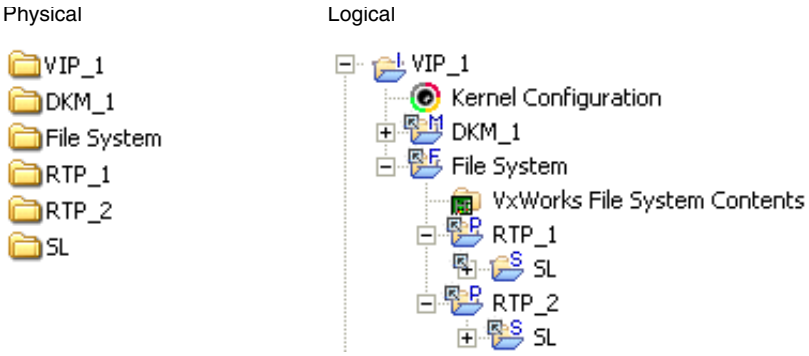
3.5.3 Project Structures and Host File System Directory Structure

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 3-4 illustrates an ideal host file system directory structure.

NOTE: The example is from VxWorks, but this section applies equally to Linux.

Figure 3-4 **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 Explorer, select individual projects, and using the instructions in *6.6.1 Adding Subprojects to a Project*, p.52, create the project structure that you need.

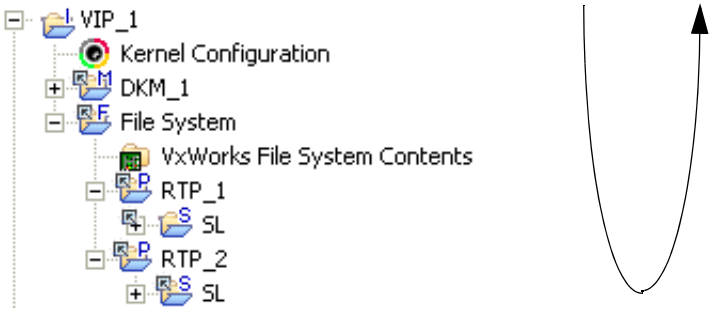
3.5.4 Project Structures and the Build System

As you can see in Figure 3-4, 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 3-5 illustrates the build order in this project structure.

Figure 3-5 **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 3-5 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**

3.5.5 Project Structures and Sharing Subprojects

Project structures can share subprojects. That is, a single physical project can be referenced by 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.

3.5.6 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 *9.5 Implementing Architecture-Specific Functions*, p.123 for an example.

3.6 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 Explorer, 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 Explorer, 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**).

   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,,3.0]
           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 Explorer (for example, it might say VxWorks 6.6).
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.

3.6.1 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.

3.6.2 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.

4

Creating Native Application Projects

4.1 Introduction

Use a *Native Application project* for C/C++ applications developed for your host environment.

Workbench provides build and source analysis support for native GNU 2.9*x*, GNU 3.*x*, and Microsoft development utilities (assembler, compiler, linker, and archiver) though you must acquire and install these utilities as they are not distributed with Workbench.

On Windows, Workbench supports the MinGW, Cygwin, and MS DevStudio compilers. Compilers for native development are distributed with Wind River Platforms, but not with Workbench, so you may have to acquire these tools yourself.

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. You

may use the GDB debugger built into Eclipse to debug native-mode, self-hosted applications such as Hello World.

4.2 Creating Native Application Projects

To create a Native Application project:

1. Choose **File > New > Native Application Project**.

   The **New Native Application Project** wizard appears.

2. Enter a **Project name**, and decide where to create your project in relation to the workspace.

   See also *3.2 Choosing Locations for Workspaces and Projects*, p.34 and *3.3 Creating New Projects*, p.35). When you are ready, click **Next**.

3. If you have created other projects, you are asked to define the project structure (the super- and subproject context) for the project you are creating.

   The text beside the **Link to superproject** check box refers to whatever project is currently highlighted in the Project Explorer (if you do not see this check box, no valid project is highlighted). If you select the check box, this will be the superproject of the project you are currently creating.

   The check boxes in the **Referenced subprojects** list represent the remaining projects in the workspace that can be validly referenced as subprojects by the project you are currently creating.

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

NOTE: All settings in the following wizard pages are build related. You can therefore verify/modify them after project creation in the Build Properties node of the project's Properties, see *8. Building Projects*.

4. A Native Application project is a predefined project type that uses Workbench **Build support**.

   The **Build command** specifies the make tool command line.

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

5. **Build Specs**: The complete list of available build specs appears here. By checkmarking individual specs, you enable them for the current project, which means that you will, in normal day to day work, only see relevant (enabled) specs in the user interface, rather than the whole list.

   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.

   The **Debug Mode** check box specifies whether or not the build output includes debug information.

   The build spec in the **Active build spec** field is used for builds, and is also propagated to the appropriate fields on the Build Tools, Build Macros, and Build Paths tabs.

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

6. **Build Target**: The **Build target name** is the same as the project name by default. You can change the name if necessary, but if you delete the contents of the field, no target will be created.

   Build tool — For a Native Application project you can select:

   - **C-Compiler, C++-Compiler, Assembler:** These tools produce a *BuildTargetName*.obj file on Windows or a *BuildTargetName*.o file on UNIX.
   - **Linker** (Linux) or **C-Linker** or **C++Linker** (VxWorks): This is the default selection. The linker produces a *BuildTargetName* (**.exe** for Windows native projects) executable file. This 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 (see step 3 above).

   - **Librarian**: This is the default selection if you specified that the project is to be linked into a project structure as a subproject. The Librarian produces a *TargetName***.a** (or **.lib** for Windows native projects) archive 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 (see step 3 above).

- 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.

7. For help configuring build macros and build paths, see *8.6 Configuring Build Macros*, p.107; and *8.7 Configuring Build Paths*, p.108.
8. When you are ready, you can review your settings using the **Back** button or click **Finish**.

   The Native Application project is created and appears in the Project Explorer, either at the root level, or linked into a project tree, depending on your selection in step 3 above.

4.3 Native Applications in the Project Explorer

After you create a Native Application project, several nodes appear in the Project Explorer. 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 *6. Working in the Project Explorer*.

4.3.1 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.

4.3.2 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.

4.3.3 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 is a brief summary of the project files displayed in the Project Explorer:

- .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.

4.4 Application Code for a Native Application Project

The Native Application project is created and appears in the Project Explorer, either at the root level, or linked into a project tree.

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 want to import these to the project. For more information, see *Importing Resources*, p.66, and *Adding New Files to Projects*, p.66.

5 Creating User-Defined Projects

5.1 Introduction

User-Defined Projects assume that you are responsible for setting up and maintaining your own build system, file system contents, makefile, 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 Explorer that reflect rules in your makefiles; this allows you to select and build any of your make rules directly from the Project Explorer.
- Build output is captured to the Build Console.

5.2 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**.

If you used the **-f** make option to specify a different filename in the New Project wizard's Build Command field, you could include a relative or absolute path to a subdirectory.

If the file does not exist, Workbench creates a skeleton makefile with a default **all** rule and a **clean**. This lets you use the **Build Project**, **Rebuild Project**, and **Clean Project** menu commands. It also prevents the generation of build errors. Since you are responsible for maintaining this makefile, you can write any other rules into it at any time.

If you base your user-defined project on an existing project, that project's makefile is copied to the new project's location, overwriting any makefile already there. (If necessary, you can change the name of the new project's makefile using the **-f** make option to avoid overwriting an existing makefile.)

5.3 Creating User-Defined Projects

Before creating the project, review the description of user-defined projects in *3.4.3 User-Defined Projects*, p.42.

1. Create a user-defined project by selecting **File > New > User-Defined Project**.

   The New User-Defined Project wizard appears.

2. Select a target operating system, then click **Next**.
3. Type a name for your project.
4. Decide where to create your project, whether in the current workspace, elsewhere in the file system, or within the workspace but with the project's sources at an external location. If you select the last option, click **Browse** and specify the directory where your sources are located.
5. When you are ready, click **Finish**.

   Your project appears in the Project Explorer.

5.4 Configuring User-Defined Projects

After you create your project, you can configure its build targets, build specs, and build macros.

1. Right-click your project in the Project Explorer and select **Properties**.
2. From the **Properties** dialog, click **Build Properties**.

NOTE: Build tools and build paths cannot be configured for user-defined projects.

In Linux, the makefiles for user-defined projects can use values from the build specs to help set the correct cross-compile toolchain and architecture from the target's sysroot.

For more details, see *8.3 Configuring User-Defined Builds*, p.104 and *8.4 Accessing Build Properties*, p.104, and press the help key for your host.

5.4.1 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 and current objects should be passed to the next level.
4. Specify whether received build targets should be passed to the next level.
5. 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.
6. You may continue configuring your project by selecting another build tab, or if you are finished, click **OK** to close the Build Properties.

5.4.2 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 Explorer and select them from the context menu.

   The **Build Folder** and **Clean Folder** options are available when you select a folder in the Project Explorer.

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 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 Explorer. 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. Continue configuring your project or 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.

This makes custom build targets different from the default ones created when you select **File > New > Build Target**, or when you name a build target during project creation.

The default build target is a dedicated make rule at the level at which the build target is defined (whether that is the project, folder, or subfolder level). A custom build target can be used on multiple levels, either as a command or a make rule.

5.4.3 Configuring Other Build Options

See the following sections in *8. Building Projects* for information about configuring other build options for your user-defined projects:

- *8.5 Working with Build Specs*, p.106
- *8.6 Configuring Build Macros*, p.107
- *8.7 Configuring Build Paths*, p.108

5.5 Debugging Source

When debugging source files in a user-defined project, you must add the project to the source lookup path to ensure that the debugger can resolve breakpoints and find files.

To add source lookup settings for a running process:

1. Right-click a launch configuration, a target, or a thread in the Debug view.
2. Select **Edit Source Lookup**.

   The **Edit Source Lookup Path** dialog appears.
3. Click **Add**.

   The **Add Source** dialog appears.
4. Select **Project** and click **OK**.
5. Select your project from the selection dialog, then click **OK** again.
6. Click **Up** or **Down** to adjust the order of entries in the list.

   The source lookup containers are searched in the order in which they appear in the **Source Lookup Path** dialog.

7. Check the **Search for duplicate source files on the path** to force the debugger to search for and display all files that match the given debugger path, rather than stopping as soon as it finds one.

For more information about source lookup paths, open the **Edit Source Lookup Path** dialog and press the help key for your host.

PART III

Development

6

Working in the Project Explorer

6.1 Introduction

The Project Explorer is your main graphical interface for working with projects. You use it to create, open, close, modify, and build projects. You can 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 Explorer to open a drop-down menu of these options.

6.2 Creating Projects

Creating projects is discussed in general under *3. Projects Overview*. Specific descriptions for creating individual project types are provided in the other chapters in *Part II. Projects* and in the platform-specific *Wind River Workbench By Example* guides.

6.3 Adding Resources and Files 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, or other resources such as CVS projects, executables, and other Workbench projects, you will want to import these to the project.

Importing Resources

You can import various types of existing resources into your projects by choosing **File > Import**. Importing resources copies them into your project.

For details about the entries in the Import dialog, open it and press the help key for your host.

NOTE: If Workbench encounters a problem while importing resources (for example, the project already contains a file with the same name), it returns an error. 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**.

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 a **File name**. If a file with the same name already exists, the New File wizard warns you so you can choose a different name.

For a description of the **Advanced** button, and what it reveals, press the help key for your host, then select New File wizard. Pay particular attention to the **Linked resources** link under **Related concepts**.

6.4 Opening and Closing Projects

You can open or close a project by selecting it in the tree and choosing **Project > Open Project** (if it is currently closed), or **Project > Close Project** (if it is currently open).

You can also use the corresponding commands on the Project Explorer's right-click context menu.

6.4.1 Closing a Project

When you close 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.

6.5 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 select the root project you are leaving, and right-click **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 *6.4.1 Closing a Project*, p.67).

- **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 the project and select **Open in New Window**). This opens the project in a new window, 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**

  You can use working sets to 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 Explorer content (and other query requests) using the pull-down at the top-right of the Project Explorer.

  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 type of working set you want to create. For example, select **Resource**, click **Next**, then select 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, it is inserted into the Project Explorer's drop-down menu, so that you can directly access it from there. The currently selected working set is marked with a dot.

- **Use the Navigate Menu**

  For day-to-day work, there is generally no need to see the contents of your software systems as presented in the Project Explorer.

  Using the **Navigate > Open Resource** (to navigate to files) and **Navigate > Open Element** (to jump straight to a symbol definition) may often prove to be the most convenient and efficient way to navigate within, or among, systems.

6.6 Moving, Copying, and Deleting Resources and Nodes

The resources you see in the Project Explorer are normally displayed in their logical, as opposed to physical, configuration (see *3.4 Project Types*, p.36). Depending on the type of resource or purely logical element you are manipulating, different things will happen. The following section briefly summarizes what is meant by resource types and logical nodes.

6.6.1 Understanding Resources and Logical Nodes

Resources are items such as *files*, *folders*, and *projects* 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. If the project is closed, you cannot see or manipulate its contents. A discussion of closed projects is provided under *6.4.1 Closing a Project*, p.67.

Logical nodes is a collective term for nodes in the Project Explorer 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.

  For example, 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.

6.6.2 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.

6.6.3 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 want to make a project into a subproject of one or more other projects, right-click the first project node, select **Project References > Add as Project Reference**, select the project to be the superproject, then click **OK**. A reference is inserted from the subproject to the superproject. 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 (by right-clicking the project, then selecting **Rename**), it will also be renamed in all other contexts.

Moving Project Nodes

To move a project, right-click the project folder node and select **Move**. Clear the **Use default location** check box; you will be asked to enter (or 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 Explorer. You can verify the new location in the **Project Properties**.

Changing Project Node References

If you right-click a project and choose **Add as Reference**, a dialog opens to allow you to place the currently selected project reference into other projects as a subproject.

If you right-click a subproject and choose **Remove Reference**, Workbench removes the currently selected project from its structural (logical) context as a subproject and moves it to the root level as a standalone project in the Project Explorer.

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 all instances of the subproject, then right-click it and press **Delete**.
- Get a *flat* view of your workspace. To do this, open the drop-down list at the top-right of the Project Explorer's toolbar, then choose **Project Presentation > Flat**. This hides the logical project organization and provides a flat view with a single instance of the subproject. Then you can select it and press **Delete**.

When you delete a project you are asked whether or not you want to **Delete project contents on disk**. 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.

However, if you do decide to delete the project contents, and you had created your project in an external location (for example, in the same directory as your source files) this may delete your original sources or the contents of a project in a different workspace, rather than the project in your current workspace. So before deleting your projects, be very sure you know where they are located and what they are linked to.

6.6.4 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.

Editing Build Targets

To edit the contents of a build target, right-click the build target and select **Edit Content**. For more information about adding and editing the contents of build targets, see *Adding Build Targets to Managed Builds*, p.99.

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 lets you see at a glance whether the target has been built or not, even if you have uncluttered your view in the Project Explorer by hiding build resources and/or collapsing the actual target node.

To hide build targets, click the drop-down menu at the top-right of the Project Explorer, then choose **Customize View**. From the Available Customizations dialog, select **Wind River build targets** (or anything else you want to hide) and click **OK**. If you have collapsed the node, the **+** sign will indicate that the build-target exists.

6.7 Parsing Binary Images

Both the Wind River Compiler and the GNU compiler (**gcc**) offer parsing tools to display information from binary image files, such as executables or object files. These tools provide detailed information about binary image files to help you find problems in section allocations or memory layout.

In previous releases of Workbench, these tools were available only on the command line, as the **ddump** command (for the Wind River Compiler) and the **objdump** command (for **gcc**.)

Now you can use the binary image parsing tools in the Workbench GUI, without going to the command line. To see the parser output for any binary image file, follow these steps:

1. In the Project Explorer, double-click the binary file, located under the **Binaries** node or within the build spec trees.
2. Workbench parses the file with the appropriate tool and displays the output in the Workbench Editor.

Files that can be parsed include executables, kernel modules, real-time processes (RTPs), and object files.

Configuring the Binary Parser Globally

You can configure what results the parsing tools should return by selecting **Window > Preferences > Wind River > Binary Parser**.

Wind River Compiler Defaults

By default, the Wind River Compiler **ddump** command uses the following arguments:

- **-f** (display file header)
- **-h** (display all section headers)
- **-N** (display symbol table information)

To change these defaults, open **Window > Preferences > Wind River > Binary Parser** and edit the **Wind River Compiler ddump** field. For information on **ddump** arguments, see the *Wind River Compiler User's Guide: DDUMP File Dumper*.

GNU Compiler Defaults

By default, the **gcc objdump** command uses the following arguments:

- **-C** (demangle low-level symbol names into user-level names)
- **-x** (display all available header information, including the symbol table and relocation entries)
- **-S** (display source code intermixed with disassembly)

To change these defaults, open **Window > Preferences > Wind River > Binary Parser** and edit the **GNU objdump** field. For information on **objdump** arguments, see the **objdump** man page.

Configuring the Binary Parser by Project

To configure the binary parser on the project level, right-click your project in the Project Explorer and select **Properties**, then select **Binary Parser**.

The **Enable binary parser** checkbox is selected by default. To turn the binary parser off, clear this checkbox. This is a team-shared setting, since it modifies the **.cproject** file. If you want to version control that file, it must be checked out as part of the operation.

To change the default arguments on the project level, select the **Enable project specific settings** check box. With this check box selected, the **Command Arguments** fields become active. If you select this check box, Workbench will take

its command arguments for this project from this dialog, and not from the Workbench **Preferences** dialog.

7 Using Advanced Navigation and Editing

7.1 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 **Open Element** dialog (available from the **Navigate** menu). Also, see *Symbol Browsing*, p.79.
- On the other hand, if you prefer navigating within and between files, you can use the File Navigator. See *Using the File Navigator*, p.80.

For more information about these views, open them then press the help key for your host

Workbench navigation depends on *source analysis*, which is the parsing and analysis of symbol information in source code. This information is used to provide code editing assistance features such as multi-language syntax highlighting, code completion, parameter hints, and definition/declaration navigation for files within your projects.

Apart from what source analysis provides that you can see directly in the Editor, it also supplies the data for code comprehension and navigation features such as include browsing , and call trees, as well as resolving includes to provide the compiler with include search paths.

NOTE: Syntax highlighting is provided for file system files that you open in the Editor, but no other source analysis features are available for files that are outside your projects.

7.2 Using Advanced Context Navigation

Various filters are available on each tool's local toolbar. Hover the mouse over each button to see its tool tip describing what it does. At the top-right, a drop-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 drop-down menu.

Generally, you want to navigate to symbols, or analyze symbol-related information, from an Editor context. The entry points are as follows:

- 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 (that is, those files outside your projects).
F4	Open the type hierarchy of the current selection (see *Using the Type Hierarchy View*, **p.81**).
CTRL+ALT+H	Open the call hierarchy (tree) of the current selection (from the Call Hierarchy view, press the help key for your host).

CTRL+ALT+I	Open the include browser to view the includes of the current selection (see *Using the Include Browser*, **p.81**).
CTRL+I	Indent the selected lines according to the code style profile selected in **Window > Preferences > C/C++ > Code Style**.
CTRL+O	Opens a quick outline dialog, similar to the Outline view (described in *Using the File Navigator*, **p.80**), but specific to the current selection rather than the file as a whole.

- keyboard shortcuts that open dialogs from which you can access symbols in any of your projects:

SHIFT+F3	Display the **Open Element** dialog.
SHIFT+F4	Display the **Open Type Hierarchy** dialog.
ALT+SHIFT+H	Display the **Open Call Hierarchy** dialog.
CTRL+SHIFT+R	Display the **Open Resource** dialog.

These options are also available from the **Navigate** toolbar menu.

Symbol Browsing

Workbench now uses the Eclipse CDT Indexer and Editor for source analysis and symbol browsing.

To open an editor for a symbol (also known as an element), select **Navigate > Open Element** or use the C/C++ Search tool by selecting **Search > C/C++**.

Very large element loads can cause delays of up to several minutes while Workbench loads the elements. Loading smaller batches of elements can decrease this delay.

To make symbol loading easier on startup:

1. Select **Window > Preferences > C/C++ > Indexer**.
2. Adjust the settings in the **Cache Limits** dialog, so that symbols are indexed in smaller batches.

Text Filtering

The **Choose an element** field at the top of the Open Element dialog provides match-as-you-type filtering. For example, to find all names starting with **task**, start typing in the field. As you type the letter **t**, then **a**, and so on, the number of elements displayed narrows so you can select the element you want from the list.

The field also supports wild card matching. Type a question mark (**?**) to match any single letter; type an asterisk (*****) to match a string of arbitrary letters; type a dollar sign (**$**) to match the end of a string.

For example, to find all names that end with **0** (zero), use the filter ***0$**.

Using the File Navigator

If you have not yet opened the File Navigator:

1. Choose **Window > Show View > Other**.
2. 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 Workbench window, along with the Project Explorer and the Debug Symbol Browser.

The File Navigator presents a flat list of all the files in the open projects in your workspace. You can constrain the list by using working sets, which you configure and select using the File Navigator's local pull-down menu.

To use the File Navigator:

- Notice that its left column shows the file name, and is active.
- Double-click a file name to open the file in the Editor.
- Right-click a file to build the project (which compiles it, 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 **Choose an element** field in the Open Element dialog (see *Finding Elements (Text Filtering)*, p.29).

Using the Type Hierarchy View

Use the Type Hierarchy view to see hierarchical **typedef** and type-member information.

To open the Type Hierarchy view, choose one of the following:

- Right-click a symbol in the Editor, Outline, or Symbol Browser view and select **Type Hierarchy view**.
- Click the toolbar button on the main toolbar.
- Select **Navigate > Open Type Hierarchy**.

For more information, see the *Wind River Workbench User Interface Reference: Type Hierarchy View*.

Using the Include Browser

By default, the Include Browser appears as a tab at the bottom-right.

To open it, choose one of the following:

- Right-click a symbol in the Editor, Outline, or Symbol Browser view and select **Open Include Browser**.
- Right-click a file in the File Navigator or Project Explorer and select **Include Browser**.
- Select **Navigate > Open Include Browser**.

Use the Include Browser as follows:

- To see which file includes, or is included by, the file you are examining.
- Use the buttons on the browser's local toolbar to toggle between showing include and included-by relationships.
- Double-click an included file to open it in the Editor at the include statement.

7.3 Using the Editor's Advanced Features

The Editor is your primary view for editing and debugging source code. The Editor is language-aware and can parse different types of files such as C/C++, Ada, and Assembler files. (Workbench no longer includes a specific Ada editor, so Ada syntax highlighting is not available.)

For help using standard editing features, see *2.3 Editing and Debugging Source Files*, p.24, and *2.4 Using the Editor's Code Development Features*, p.27.

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).

For additional editor preferences, see **Window > Preferences > General > Editors** and **Window > Preferences > C/C++ > Editor**. Also, additional online information is available at **http://help.eclipse.org**.

Inserting Text Using Code Templates

The Editor uses templates, provided by Eclipse, 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. Type the first letter of the pattern, then press **CTRL+SPACE**. The templates that begin with that letter appear; click a template to view its contents, or double-click it to insert it into your file.

To view the preferences associated with code templates, select **Window > Preferences**, then **C/C++ > Editor > Templates**. For additional information, press the help key for your host.

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 Explorer, 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 *15.3.5 Configuring Debug Settings for a Custom Editor*, p.199.

7.3.1 Building Projects from the Editor

You can build a project from within the Editor.

When you are finished editing a file, press **CTRL+SHIFT+A** to build the project to which the file belongs.

If the **Link with Editor** option is

- enabled (by clicking the icon on the Project Explorer toolbar), the corresponding project of the file being shown in the Editor will be built (as it is selected there).
- not enabled, the current selection in the Project Explorer is used to build the corresponding project there. If there is no selection in the Project Explorer, and an Editor has the focus, the corresponding project of the file being shown in the Editor will be built again.

NOTE: To check if something gets built, bring the Build Console to the front. If nothing is built, be sure that you have selected the project in the Project Explorer (this should be selected if you opened the file from there).

7.4 Searching for and Replacing Elements

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. You can filter matches 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.

When the search tool finds a match to your query, the Search view displays the matching lines, and the matches and line numbers are highlighted in a different

background color. If there is more than one match in a file, the number of matches in the file is displayed.

For instructions on how to change the colors used to mark matches and line numbers, open the Search view and press the help key for your host.

7.4.1 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**.
- Or one of the scoping options from the global **Search** menu.

For more information, open the search dialog and press the help key for your host.

7.5 Expanding and Exploring Macro References

You can see the full expansion of a macro reference in your code by hovering over the macro with the cursor. This displays the macro code in a popup window.

To step through the individual steps of a macro expansion, you can invoke the Macro Expansion Explorer by pressing **F2** from the popup window, or by selecting one or more macro references, then right-clicking them and selecting **Explore Macro Expansion** (or pressing **CTRL + =**).

The Macro Expansion Explorer consists of three panes.

- The top pane shows the definition of the currently expanded macro.
- The bottom-left pane shows the original source code before the expansion.
- The bottom-right pane shows the source code after the full expansion.

The macro expansion steps can be navigated back and forth using **Alt+Left Arrow** and **Alt+Right Arrow**. Pressing **F3** (Open Declaration) closes the Macro Expansion

Explorer and opens the macro definition of the currently selected macro in the Editor.

7.6 Configuring the Indexer for Source Analysis

Editing, navigating, and code completion rely on parsing and analyzing project source code. This source analysis (formerly known as static analysis) is done by the Eclipse C/C++ Indexer.

The resulting index is the basis for any source navigation and editing capabilities, such as navigation between declaration and definition of methods, showing included files in the Include Browser, showing the call tree of functions and methods, showing the type hierarchy of classes, code completion in the editor, hover information for function and method occurrences (a tool tip showing the signature and comments of the related declaration or definition), and so on.

Workbench uses the Eclipse Fast C/C++ Indexer by default. You can, however, switch off source analysis for any project by selecting the **No Indexer** option on the indexer preferences page. For more information about the Indexer, see the *C/C++ Development User Guide* under **Help > Help Contents**.

NOTE: If you encounter problems with source navigation tools for certain files, and the information in this section does not solve them, see also *A.4 Fixing Indexer Issues*, p.249.

7.6.1 Setting Indexer Preferences

You can set basic indexer preferences such as whether to index a project, or to index all files in a project, on either a global (workspace) or per-project basis.

To set global indexer preferences, open the preferences dialog (by selecting **Window > Preferences > C/C++ > Indexer**).

To enable label decorations so you can see which files have been indexed, select **Window > Preferences > General > Appearance > Label Decorations**, then select **C/C++ Indexed Files** and click **OK**.

For information about the preferences dialog that appears, press the help key for your host.

Setting Global (Workspace) Preferences

To set global (workspace) indexer preferences:

1. Select **Window > Preferences > C/C++ > Indexer** to open the preferences dialog.
2. Select the files you want Workbench to parse.
   - Select **Index all files** if you want to parse all source and header files.

     By default, the only files indexed are those added to a managed project's build target, or, for user-defined projects, those included by enabling build-output analysis.
   - Select **No Indexer** in the drop down, if you want no files to be parsed.
3. Set **Indexing strategy** to **after every file change**, if you do not want the index upated automatically.
4. Accept the **Cache limits** as specified: share of heap memory, absolute file size, and header file cache size.
5. Click **Apply**.

Setting Project-Specific Properties

To set project-specific indexer preferences (properties):

1. Right-click a project in the Project Explorer, then select **Properties > C/C++ General > General > Indexer**.
2. Select **Enable project specific settings**.
3. Select the files you want Workbench to parse.
   - Select **Index all files** if you want to parse all source and header files.

     By default, the only files indexed are those added to a managed project's build target, or, for user-defined projects, those included by enabling build-output analysis.
   - Select **No Indexer** in the drop down, if you want no files to be parsed.
4. Click **Apply**.

7.6.2 Editing Build Properties

Indexing works differently depending on the type of project:

- Managed projects allow full configuration of build settings using the Workbench build properties page. Include paths and symbols (preprocessor macros) defined in the build properties are directly passed to the indexer. The build scope (the files to be built) defines which files the indexer processes by default. Other files are not parsed.
- User-defined projects do not allow full configuration of build settings using the Workbench GUI. Manual editing of makefiles may be required in order to get a fully functional build. The indexer can, however, derive include paths, symbols, and build scope through build output analysis.
- VxWorks Image Projects allow configuration of build settings using the build properties page, but they are treated as user defined projects with respect to indexer setup. This is because the resulting set of include paths and symbols is more accurate when build output analysis is enabled.

For all these projects, you can define additional include paths and symbols in the Paths and Symbols property page.

Setting Build Properties for Managed Projects

Project creation wizards create default build targets, unless you clear the **Build Target name** field on the Build Target wizard page.

A build target defines the build scope that is used by the indexer to process source code. Files outside this build target are not parsed per default.

If there are no build targets, all files are parsed, otherwise you would end up with an empty index, and no source analysis or navigation.

A build target must contain at least one source file. If you delete all files from a build target, and there are no non-empty build targets in the project, then nothing is parsed, and the source navigation tools such as Type Hierarchy and Call Tree will not work correctly.

To create a new build target, select **New > Build Target** from the main menu, or right-click the **Build Targets** node under the project in the Project Explorer and select **New Build Target**. Then follow the instructions in the wizard.

You can use the build properties to control the whole build process. For that purpose, select the project in the Project Explorer view and navigate to the **Build**

Properties page. The available tabs are Build Support, Build Targets, Build Specs, and Build Macros. You can use these as follows:

- Create and/or enable proper build specs on the Build Support and Specs tab.
- Edit build tool settings on the Build Tools tab, if necessary.
- Edit build macros on the Build Macros tab. In many cases, the **DEFINES** build macro is already predefined. Use this to define preprocessor macros (symbols), if available. All symbols (including the **-D** prefix) defined by build macros and passed to build tools are also passed to the indexer, and thus are used for parsing source code of the given resource. For instance, you may set **DEFINES** to **-DMACRO1 -DMACRO2** and pass **DEFINES** to the C or C++ compiler on the Build Tools tab, in the Command or Tool Flags section.

  In addition to the symbols defined on the Build Macros tab, the symbols defined implicitly by the selected build tools are also passed to the indexer.
- Edit build paths on the Build Paths tab. Use **Generate** to have include paths determined for unresolved include directives. Build paths of the default build spec are passed to the indexer, as well as to the build tools.

  In addition to the include paths defined on the Build Paths tab page, the include paths defined implicitly by the selected build tools are also passed to the indexer.

NOTE: The indexer uses only the symbols and include paths defined for the default build spec to parse source code for a given project.

To generate include paths and resolve include directives for managed projects:

1. Open the Resolve Include Directives dialog by choosing one of these paths:
   - Right-click the project in the Project Explorer, select **Build Options > Generate Include Search Paths**.
   - Or, open the project properties, select **Build Properties**, and on the Build Paths tab click **Generate**.
2. If necessary, specify include directives to ignore.
3. Click **Next**.
4. On the Resolve Include Directives dialog, if there are unresolved directives, you might click, for instance, **Resolve All**. The resulting include paths will be stored in the build properties of the selected project.

   You may also choose **Resolve**, **Add**, **Substitute**, and so on.

5. Click **Next**.

The results are stored in the build properties of the project.

Setting Build Properties for User-Defined Projects

For user defined projects, you must edit makefiles.

You may need to edit the build command and default build rules in the build properties. For that purpose, select the project in the Project Explorer view and navigate to the **Build Properties** page.

Then:

- Set a proper build command in the **Build command** field on the Build Support tab, if necessary.
- Set a **Build Project rule** and **Build folder rule** on the Build Targets tab, if you use any other than the **all** rule in your makefile.

Build output is analyzed to determine symbols and include paths that the indexer should use to parse the files of the project. Only files parsed during a previous build run are parsed.

NOTE: If no files have ever been built for a project, all files are parsed, guaranteeing that you end up with an index, source analysis, and navigation capabilities.

7.6.3 Setting Up Paths and Symbols

Usually you do not need to set up the indexer manually by defining additional symbols and include paths. The indexer derives the necessary settings automatically from the build properties of managed projects, or from the build output of user-defined projects.

In addition, for user-defined projects, include paths are automatically calculated after creating a project, in order to have a valid setup even before the project is built. Files and folders are automatically considered only if they have been added or linked to the project at project creation time.

At any time later, you can let Workbench detect include paths for a project, using the Include Search Path (ISP) wizard. In order to do so, follow the directions in the following subsections for both managed and user-defined projects.

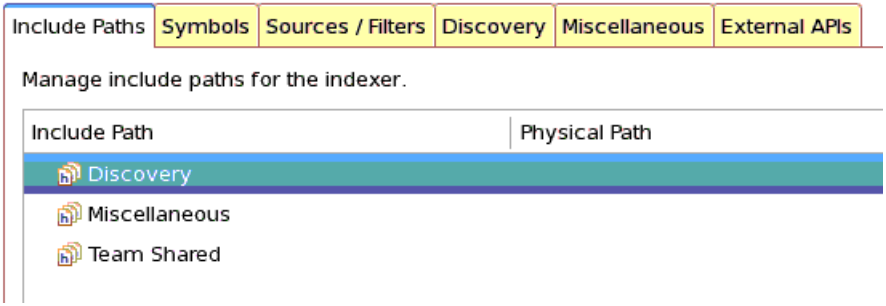
NOTE: The ISP wizard available under **Build options > Generate Include Search Paths** can be used only for managed projects. It stores its results in the build properties of the corresponding project.

However, the ISP wizard available from the Include Paths tab is available for both managed and user-defined projects, and it stores its results in the team-shared include paths of the corresponding project.

To access the ISP wizard:

1. Right-click the project.
2. Choose **Properties > C/C++ General > Paths and Symbols**.
3. Notice the tabs in the resulting Paths and Symbols page.

The following subsections describe how to use these tabs, as shown on the following figure.



Managing Include Paths for the Indexer (Include Paths Tab)

The Include Paths tab shows include paths passed to the indexer, and allows you to edit team-shared include paths. Categories displayed on this tab include:

- Discovery—Shows build-output analysis for user-defined projects. To enable this analysis, select the Discovery tab (see *Setting Up a Build-Driven Index (Discovery Tab)*, p.93).
- Build—Shows include paths for managed builds, along with the corresponding physical path in the filesystem. To configure these paths, select **Build Properties**, then select the **Build Paths** tab (for information about the settings on the Build Paths tab, open it and press the help key for your host).

- Miscellaneous—Shows preprocessor options for the indexer. To configure these options, as well as set the compiler name or location, select the Miscellaneous tab (see *Specifying User-Private Paths and Symbols (Miscellaneous Tab)*, p.93).
- Team Shared — A set of include paths shared automatically in a team.

You can select an include path on the Include Paths page, click **Copy**, and paste the resulting path into your makefile, if necessary.

Using Team Shared Paths

Choose the Team Shared set of include paths, and then select one of the actions in the right column: **Add Folder**, **Add File**, **Generate**, **Substitute**, or **Remove**. For more information about these actions, open the Include Paths tab and press the help key for your host.

For example:

1. Select an item or items in the **Team Shared** section, then click **Generate**.
2. On the first wizard page, select options for analyzing the include statements of your sources, then click **Next**.
3. On the Resolve Include directives page, if there are unresolved directives, click **Resolve** or **Resolve All**. To examine them before deciding whether to resolve them or not, click Show in Editor.

   The resulting include search paths will appear at the bottom of the page, and are stored in the build properties of the selected project.
4. To update any of the include search paths or resolved directives, select one and click **Add**, **Substitute**, or **Remove**.
5. When everything is configured properly, click **Finish**.

Any include paths resulting from Steps 3 and 5 will be stored as team-shared settings in the **.cproject** file under the project folder. You can now check this in with your project in your version control system.

NOTE: You must *not* modify the **.cproject** file manually because it contains essential metadata for the project.

To share an index, see *7.6.5 Sharing Source Analysis Data with a Team*, p.96.

Configuring Indexing of Symbols (Symbols Tab)

In the Symbols tab of the Paths and Symbols dialog, you can choose how to manage symbols that are in a Build, Miscellaneous, or Team Shared section.

For Team Shared symbols, you can use the following buttons:

- **Add**, **Remove** — to add or remove team-shared macro definitions.
- **Edit** — to modify such symbols or to navigate to the properties page where they can be defined.
- **Share** — to copy include paths of other sources into the team-shared section.

Values need not be quoted or escaped. They are passed to the indexer as they are. For example, if you click **Add**, then set **MACRO5** as name and **'c'** as value, this corresponds to **#define MACRO5 'c'**.

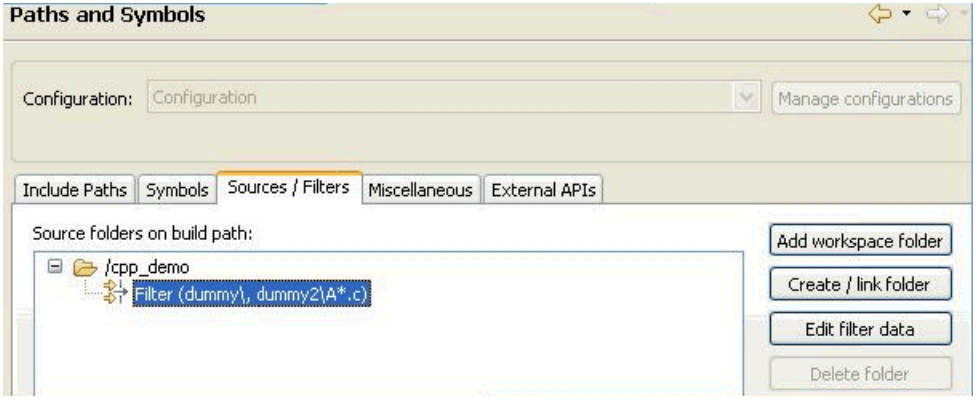
Configuring Sources and Exclusion Filters (Sources / Filters Tab)

In the Sources/Filters tab of the Paths and Symbols dialog, you can specify exclusion filters for the project.

Excluded files and folders are not parsed, even if the **Index all files** option is set for the project (see *7.6.1 Setting Indexer Preferences*, p.85).

The root nodes are source folders. Per default, there is only one source folder—the project root folder. You should never replace this by another source folder. Additional source folders, however, are allowed.

Figure 7-1 **Sample Exclusion Filter**



In order to specify an exclusion filter, select a source folder, and click **Edit filter data**. You can then add files and folders to be excluded. Wildcards are supported.

In the example shown in Figure 7-1, the filter entry **dummy/** causes the **dummy** sub-folder under the cpp_demo project to be excluded. The **dummy2/A*.c** part excludes all files of the **dummy2** sub-folder starting with **A** and having a **.c** extension. If the filter were ****/dummy3**, it would exclude all **dummy3** sub-folders of the cpp_demo project or of any sub-folder of **cpp_demo**.

If you need to include a sub-folder **dummy4** of an excluded folder **dummy3**, then simply add **dummy4** as source folder by using the **Add workspace folder** button. **dummy4** will then be treated as exception to the **dummy3** exclusion filter.

Setting Up a Build-Driven Index (Discovery Tab)

The Discovery tab of the Paths and Symbols dialog provides options for controlling the build-output analysis for user-defined projects. The tab is only visible for this kind of project.

Use the Discovery tab to enable build-output analysis where you want to set up the project according to your build settings. You cannot modify the results. Click **Clear** to reset the discovered include paths, symbols, and build scope, if necessary.

When the **Enable analysis of build output from console and build-output file** option is enabled, include paths, symbols, and the build-scope (the set of built files) are gathered during a build. The results are used to set up the indexer correctly. Although user-defined projects already come with a set of include paths found during project creation (by analyzing source code), the results delivered by build-output analysis are much more accurate, since the results reflect the actual build settings. The impact on the build-performance is minimal, so you should leave this option enabled. Disable this option only if the results are not sufficient.

The **Discovery statistics** section reports the number of discovered include paths, symbols and source files, and helps you to assess the correctness of the results. In addition, include paths and symbols are presented on the Include Paths and Symbols tabs.

Use the **Analyze build-output file** section to perform build-output analysis on a build -output file. If a build-output file already exists for your project, then this is the fastest way to set up the indexer correctly, without the need to build the whole project.

Specifying User-Private Paths and Symbols (Miscellaneous Tab)

The Miscellaneous tab of the Paths and Symbols dialog specifies user-private include paths and symbols.

Preprocessing Options

The Preprocessing options edit field takes most common preprocessor options, such as **-I**, **-include**, **-D**, and so on. Include paths (**-I** *includePath*) and include files (**-include** *headerFile*) can contain placeholders, such as **$(WIND_BASE)**. Use **-include** if you want the indexer to treat every source file in the project as if it were including the given header file. This does not have any affect on any build settings.

Symbol options (**-D***name* or **-D***name*=*value*) do not need any quoting for integer values. For instance, **-DMACRO1** and **-DMACRO1=1** correspond to **#define MACRO1** and **#define MACRO1 1**. One level of quotes is removed according to compiler command line options.

In order to specify character values for symbols, use single quotes surrounded by double quotes (for instance, **-DMACRO2= "'c'"**). For string values use double quotes surrounded by single quotes (for instance, **-D'MACRO3="Text"'** or **-DMACRO3='"Text")'**.

The compiler inspection field is disabled for managed projects. For user-defined projects, it is filled automatically with the compiler name or path used for building the project (build-output analysis must be enabled). The compiler is used to detect built-in include paths and symbols. If the field is empty, no compiler-internal include paths and symbols are passed to the indexer. Change the compiler name only if necessary.

Build Spec Used by the Indexer

You can specify which build spec should be used by default when parsing a project, either the active build spec, the default build spec (both as specified in the project's build properties), or a fixed build spec that you select from the drop-down list.

Changing this setting only has an effect on the files that are parsed afterwards. In order to update the index for the entire project, right-click the project, then select **Index > Rebuild**.

NOTE: Settings on the Miscellaneous tab are lost when the project is removed from the workspace.

Specifying External APIs (External APIs Tab)

An *external API* is an index holding a pre-parsed platform API for a project type. Most managed project types come with an external API package.

To enable an external API, select the **Enable project-type specific external API** check box.

The external APIs are used for code completion and parameter hinting (hover info) in the source editor. In addition, they can be used to find out where specific platform API functions are declared. For example, when you use **Open Element** or **C/C++ Search** to search for a function named **printf**, you get the locations of the function declarations with this name, even if the project itself does not contain the corresponding header files. This allows you to find out which header files to include to build your sources successfully.

Turning off the external APIs for a project is useful, if, for example, you are working on the source files that are actually the basis for the external APIs. This avoids having two different indexes for the source files in the project (one for the sources in the project, and one for the corresponding external API that might reference outdated versions of the same source files).

7.6.4 Updating a Project's Index

Changing include paths or symbols using the **Build Properties** page or the **Paths and Symbols** page has an immediate effect only on parsing modified and newly added files. For performance reasons, the index is not rebuilt automatically after such changes. You have three different options to update the index manually in order to get more accurate source navigation:

- **Rebuild** — Right-click the project in the Project Explorer and select **Index > Rebuild** in order to clear the index and re-parse all files in the current build scope (for build-driven setup) or all files in the project (no build target, or the **Index All Files** option is enabled for a project, or build-output analysis is disabled or did not return any results).

  The current include paths, symbols, and build scope will be used to re-parse all files, thus source navigation gives the most accurate results after performing a full rebuild.

- **Update with Modified Files** — Right-click the project in the Project Explorer and select **Index > Update with Modified Files** in order to parse modified and newly added files.

- **Freshen All Files** — Right-click the project in the Project Explorer and select **Index > Freshen All Files** in order to re-parse previously parsed files with the current include path and symbol settings.

For all options, the current include path and symbol settings defined for the project are used when parsing files.

7.6.5 Sharing Source Analysis Data with a Team

You can create a project index to be shared with project team members. Such a team-shared index holds a snapshot of source-analysis data retrieved by parsing the project.

When a project is imported, this significantly reduces the initial parsing time for a project, because only modified files are parsed (as compared with parsing all files when the index was created).

After the project is imported, only the pre-filled project index is used — the team-shared index has no further relevance.

In order to create and share an index:

1. Right-click a project in the Project Explorer, then select **Export**.
2. In the Export dialog, select **C/C++ > Team Shared Index**.
3. Click **Next**.
4. Mark other open projects if you want to export their indexes.
5. Select an export destination relative to the project root folder.
6. Use **Insert variable** if you need to store the index outside of the project, in order to avoid absolute paths. Make sure that the export destination is available by all team-members (access to an NFS or Windows/Samba share).
7. Check the created **.settings** project subdirectory, checking all files in this folder into your version control system to share it with other team members.

The team-shared index will be loaded automatically whenever the project is opened. No user interaction is required.

The file **.settings/org.eclipse.cdt.core.prefs** under your project folder holds the location of the team-shared index (slot **indexer/indexImportLocation**). Do not edit this entry, unless the location of the index has moved outside the project, and you do not wish to export the index again.

8 Building Projects

8.1 Introduction

The process of building in Workbench starts during project creation, when you decide what type of project you want.

Along with creating project-specific files, Workbench creates makefiles, assigns default build settings, and offers several levels of build support: Managed, User-defined, or Disabled. The type of build support available depends on what type of project you chose.

Available build support is as follows:

Managed Build

Workbench controls all phases of the build. Managed build support is available for all project types except User-Defined projects.

NOTE: *Managed build* refers to the type of build that was called *flexible managed build* in previous releases of Workbench. What used to be called *standard build* or *standard managed build* has been deprecated, but you can still create one by choosing **Window > Preferences > Wind River > Build** and checking the box for **Enable deprecated standard managed build**, if your build structure is similar to the file system structure.

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 Explorer that reflect rules in your makefiles, so you can select and build any of your make rules directly from the Project Explorer.
- 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: Once a project is created, you can decrease build support (by disabling the build support for a managed build project), but you cannot increase it (by turning on managed build support for a project that did not have it originally).

If you later want Workbench to manage your build, you must 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.

You can set the build properties to be used for new projects by navigating to **Window > Preferences > Wind River > Build > Build Properties** and selecting different project types from the drop-down menu.

You can set build properties for a specific existing project by selecting **Project > Properties > Build Properties**.

8.2 Configuring Managed Builds

Once your project is created, a managed build project appears in the Project Explorer. The project contains the usual project-specific files, but you must create a build target manually.

Adding Build Targets to Managed Builds

Once your project is created, you will see a **Build Targets** node inside it.

To add a build target to your project:

1. Right-click the **Build Targets** node and select **New Build Target**.

   The **New Build Target** dialog appears. By default the **Build target name** and **Binary output name** are the same as the project name, but if you are going to create multiple build targets, enter more descriptive names.

NOTE: 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.

2. 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 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.

   i. To add contents to your virtual folder, right-click it in the Project Explorer 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 Explorer 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 Explorer 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, right-click it in the Project Explorer, select **Rename** (or 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 content 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 Explorer 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 Explorer 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**.

Leveling Attributes

The leveling chain for 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 Managed Builds*, p.99. The information that can be leveled allows you to build files on a per build-spec basis.

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 image superprojects; it is not possible to pass managed build targets to other managed build superprojects.

To reference other managed build targets, add them to the contents of a build target as described in *Adding Build Targets to Managed Builds*, p.99.

Understanding Managed Build Output

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.

This 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, and deposits them in the appropriate debug-mode directory.

In general, the build output is structured as follows:

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 within 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
```

8.3 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 *5. Creating User-Defined Projects*.

To create the build target, right-click your project in the Project Explorer and select **Build Project** or press **CTRL+SHIFT+A**.

To update the build settings, right-click your project in the Project Explorer 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.

8.4 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.
- 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.

8.4.1 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.

8.4.2 Project-specific Build Properties

To access build properties from the Project Explorer, 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 only to the existing project that is selected in the Project Explorer.

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.

8.4.3 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. This file should therefore be version controlled along with the project. Note that you can revert to the inherited settings by clicking the eraser icon next to a field.

8.4.4 Multiple Target Operating Systems and Versions

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.

For existing projects, the target operating system and version are displayed by default in the Project Explorer next to the project name. You can toggle the display of this information by selecting **Window > Preferences**, then **General > Appearance > Label Decorations**, then selecting or clearing the **Project Target Operating Systems** check box.

If you choose not to display this information in the Project Explorer, you can verify the target operating system of the project in the project's Properties dialog (right-click the project, then select **Properties**, then **Project Info**).

8.5 Working with 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 *9.7 Defining Build Specs for New Compilers and Other Tools*, p.129).

It is important to remember that the build spec used when you build must match the target board; that is, it must match the kernel or platform project with which the application project is associated.

8.5.1 Defining and Importing Build Specs

Use the Build Specs 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.

8.5.2 Regenerating Build Spec Cache Information (VxWorks)

Pre-generated build specs are shipped as part of the distribution, rather than being generated after the product is installed.

If you change any makefile fragments in a VxWorks installation, you need to regenerate the cache information before the new settings take effect for newly-created workspaces.

To regenerate the cache, use the following commands:

```
% cd installDir/vxworks-6.x/setup
% vx_postinstall.bat (on Windows) or vx_postinstall.sh (on Linux or Solaris)
```

To import the changed settings to existing workspaces so you can use them in your current projects, select **Window > Preferences > Wind River > Build > Build Properties**, then select a project type from the drop-down list, then click **Restore Defaults**.

8.6 Configuring Build Macros

Use the Build Macros tab of the Build Properties dialog 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**.

8.7 Configuring Build Paths

Use the Build Paths tab of the Build Properties dialog 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**.

8.8 Makefiles

The build system uses the build property settings to generate a self-contained makefile named **Makefile**. For 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** (as described in *8.7 Configuring Build Paths*, p.108), 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 the file **Makefile** 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 *9.6 Creating User-Defined Build Targets in the Project Explorer*, p.126.

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 *9.8 Developing on Remote Hosts*, p.131.

8.8.1 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 the parser-generator **yacc** (Yet Another Compiler Compiler) which is not contained in the Workbench 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 Explorer, 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: Execute this build step prior to the project build, or the files generated by **yacc** will not be used by the managed build. (The managed build generates the corresponding makefile before the build starts and before all files that are part of the project at the time are taken into account.)

6. When the build is finished, right-click the **yacc_example** folder and select **Build Project** or press **CTRL+SHIFT+A**.

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**.

9 Building: Use Cases

9.1 Introduction

This chapter suggests some of the ways you can complete various build-specific tasks in Wind River Workbench.

9.2 Adding Compiler Flags

This section describes how to add and edit the compiler flags on specific projects.

You can add compiler flags:

- By using the GUI to help you put a specific, known compiler flag in the right place as shown in *Adding a Compiler Flag by Hand*, p.116.
- By using the GUI to help you specify the correct flag for what you want to do, as shown in *Adding a Compiler Flag with GUI Assistance*, p.117.

NOTE: For Wind River Linux projects, global changes are made in board templates so that the settings can be used by both the command line build system and Workbench, as described in *Wind River Linux Platforms User's Guide*.

Adding a Compiler Flag by Hand

If, for example, you are familiar with the GNU compiler command line and you just want to know *where* to enter a known flag such as **-w**:

1. In the Project Explorer, right-click an application project and select **Properties**.
2. In the Properties dialog box, select the **Build Properties** node.
3. In the **Build Properties** node, select the **Build Tools** tab.
4. In the **Build Tools** tab:
   a. Set the **Build tool** to **C-compiler**
   b. Set the **Active build spec** to, for example, **PENTIUM-gnu-native** (for Linux), or **SIMPENTIUMgnu_RTP** (for VxWorks).
   c. 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%** placeholder 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**.**

Adding a Compiler Flag with GUI Assistance

If you are not familiar with the specific command line tool options that you want to use, the GUI may be able to help. For example:

1. In the Project Explorer, right-click an application project, and select **Properties**.
2. Click **Build Properties** and select the **Build Tools** tab.
3. In the **Build Tools** tab:
   a. Set the **Build tool** to **C-compiler**
   b. Set the **Active build spec** to, for example, **PENTIUM-gnu-native** (for Linux), or **SIMPENTIUMgnu_RTP** (for VxWorks).
   c. To view the GNU compiler options, click the **Tool Flags** button.
   d. In the GNU Compiler Options dialog box, click your way down the navigation tree on the left, and take a look at the corresponding options on the right.
   e. When you get to the **Compilation > Diagnostics** node, select **Suppress all compiler warnings**.

      Notice that **-w** now appears in the list of command line options at the right of the dialog box.
   f. Click **OK**.
4. Back in the **Build Tools** tab of the Build Properties dialog box, you will see that the **-w** option you selected now appears in the field next to the **Tool Flags** button.

   The contents of the **Tool Flags** field is expanded to the **%ToolFlags%** placeholder you see in the **Command** field above it.

9.3 Building Applications for Different Target Architectures

The target nodes under projects in the Project Explorer display the name of the currently active build spec. You may want to switch build specs to build projects for different architectures.

If, for example, you want to build an application for testing on a simulator on the localhost, 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 select **Build Options > Set Active Build Spec**.
2. In the dialog box 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 box, you will notice that the label of the target node has changed. If you selected **Debug mode** in the dialog box, debug is added after the the build spec name.
3. Build the project for the new architecture.

NOTE: To select the Active Build Spec directly from the Project Explorer, click the checkmark  icon in the Project Explorer toolbar.

9.4 Creating Library Build Targets for Multiple Applications

This example demonstrates a single project with which you can build five applications using two libraries. Since it is a Native Application project, it requires a native compiler and debugger, and as described in *4. Creating Native Application Projects*, you must acquire and install these tools yourself, as they are not shipped with Workbench.

In *9.4.1 Creating the ComplexSystem Example Project*, p.119, you will create the example project that is shipped with Workbench.

In *9.4.2 Creating the ComplexSystem Project Manually*, p.119 are step-by-step instructions for creating this same structure manually. You will create your project at the root location of the sources, and your library code will be located parallel to the application code.

9.4.1 Creating the ComplexSystem Example Project

To create the ComplexSystem example project, follow these steps:

1. Select **File > New > Example** to open the New Example wizard.
2. Select **Native Sample Project**, then click **Next**.
3. Select **The Complex Project Demonstration Program**, then click **Finish**. The **ComplexSystem** project appears in the Project Explorer.
4. To see the output of the applications (the application and library source files used to build it), double-click the project's **Binaries** node, then right-click an application executable and select **Run as > Local C/C++ Application**.
5. Select your native debugger, then click **OK**.
6. Workbench builds the project, with build output appearing in the Build Console. The output from the application appears in the Console view (if the Console is not visible, select **Window > Show View > Console**).

9.4.2 Creating the ComplexSystem Project Manually

You will find the source files you need to create this project in *installDir***workbench-3.1\samples\ComplexSystem**.

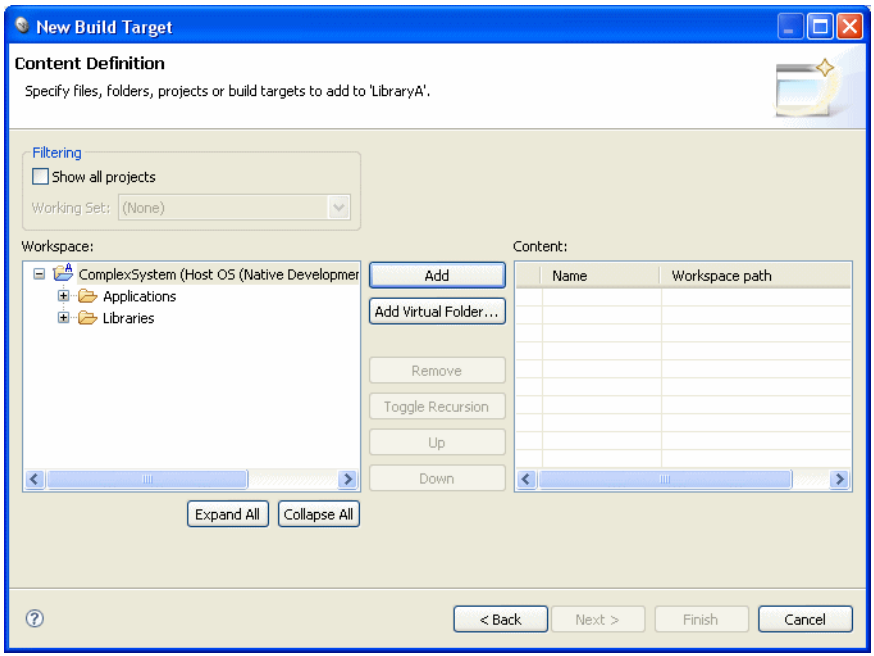
Step 1: Create a native application project to hold your files.

The native application project type (rather than the library project type) is appropriate here because you need to use both the **C++-Linker** and the **Librarian** build tools.

1. Select **File > New > Native Application Project**.
2. Name the project **ComplexSystem**.
3. Select **Create project at external location**, then click **Browse** and navigate to the directory shown above containing the example source files.
4. Click **Next** until you reach the **Build Target** dialog.
5. Since you will be creating multiple build targets later, remove the suggested build target name from the wizard (leave it blank).
6. Click **Finish**. The project appears in the Project Explorer, and the Workbench project files appear in the directory with the source files.

Step 2: Create the build target for the first library.

1. Right-click the green **Build Targets** node under the ComplexSystem project and choose **New Build Target**.
2. Name the new build target **LibraryA** (in the **Build target name** field).
3. Set the **Build tool** to **Librarian**.
4. Clear the **Use default content** check box.
5. Click **Next**. The Content Definition dialog opens.



6. In the Workspace pane, select **Libraries > LibraryA**, then click **Add**.
7. Click **Finish**.

The LibraryA build target appears under the Build Targets node, and references the LibraryA source folder.

Step 3: Create the build target for the second library.

1. Repeat the instructions in Step 2, this time naming the new build target **LibraryB** and selecting **Libraries > LibraryB** from the Content Definition dialog.
2. Click Finish.

The LibraryB build target appears under the Build Targets node, and references the LibraryB source folder.

Step 4: Create the build target for the first application.

1. Right-click the **Build Targets** node and choose **New Build Target**.
2. Name the new build target **Application1**.
3. Set the **Build tool** to **C++-Linker**.
4. Clear the **Use default content** check box.
5. Click **Next**.
6. In the Workspace pane of the Content Definition dialog, select **Applications > Application1**, then click **Add**.
7. Application1 uses the library build target LibraryA, so select **Build Targets > LibraryA**, then click **Add**.

NOTE: Be sure to select the LibraryA build target, not the LibraryA source folder.

8. Click **Finish**.

The Application1 build target appears under the Build Targets node, and references both the Application1 source folder and the build target LibraryA.

Step 5: Create the build target for the second application.

1. Repeat the instructions in Step 4, this time naming the new build target **Application2** and selecting **Applications > Application2** from the Content Definition dialog.
2. Application2 uses the library build target LibraryB, so select **Build Targets > LibraryB**, then click **Add**.
3. Click **Finish**.

The Application2 build target references both the Application2 source folder and the build target LibraryB.

Step 6: Create the build target for the third application.

1. Repeat the instructions in Step 4, this time naming the new build target **Application3** and selecting **Applications > Application3+4 > ComplexSystemApplication3.c** from the Content Definition dialog.
2. Application3 uses the library build target LibraryA, so select **Build Targets > LibraryA**, then click **Add**.
3. Click **Finish**.

The Application3 references both the Application3 source file and the build target LibraryA.

Step 7: Create the build target for the fourth application.

1. Repeat the instructions in Step 4, this time naming the new build target **Application4** and selecting **Applications > Application3+4 > ComplexSystemApplication4.c** from the Content Definition dialog.
2. Application4 uses the library build target LibraryB, so select **Build Targets > LibraryB**, then click **Add**.
3. Click **Finish**.

The Application4 build target references both the Application4 source file and the build target LibraryB.

Step 8: Create the build target for the fifth application.

1. Repeat the instructions in Step 4, this time naming the new build target **Application5** and selecting **Applications > Application5** from the Content Definition dialog.
2. Application5 uses both library build targets, so expand the **Build Targets** node, select **LibraryA**, press **Shift**, select **LibraryB**, then click **Add**.
3. Click **Finish**.

The Application5 build target references the Application5 source folder and both library build targets.

Step 9: Build the project.

1. Right-click the ComplexSystem project folder, then select **Build Project** (or click **Build Project** on the Project Explorer toolbar).
2. A dialog appears asking if you want to generate include search paths for your project. You must do that or the build will fail, so click **Generate Includes**.

3. On the first screen of the Generate Include Search Paths dialog, leave all options as-is, then click **Next**.
4. On the second screen, there are unresolved include directives. Click **Resolve All** to resolve them automatically.

   Workbench resolves the include directives, which now appear in the lower pane. Click **Next**.
5. Decide whether to overwrite or merge the new search paths with existing include search paths. For more information about this choice, press the help key for your host.
6. Click **Finish**. The build will proceed.

Each library is built only once, and is linked to all applications as needed.

9.5 Implementing Architecture-Specific Functions

You can enable or disable build specs at the project, folder, build-target, and file levels. This allows architecture-specific implementation of functions within same project.

Consider a simplified managed build project with two subfolders, **arch1** and **arch2**, that each use code-specific differences for different target architectures. You can set up a project to build a software target that requires the implementation of a function specific to different target boards, where you only need to change the active build spec at the project level.

In this example, suppose that the function **int arch_specific (void)** is declared in **arch.h**, and that:

- **arch1.c** implements **int arch_specific (void)** for the **PENTIUMdiab_RTP** VxWorks build spec, or the **common_pc** Linux build spec.

  This is the only build spec enabled for the **arch1** folder.
- **arch2.c** implements **int arch_specific (void)** for the **PPC32diab_RTP** VxWorks build spec, or the **arm_versatile_926ejs** Linux build spec.

  This is the only build spec enabled for the **arch2** folder.

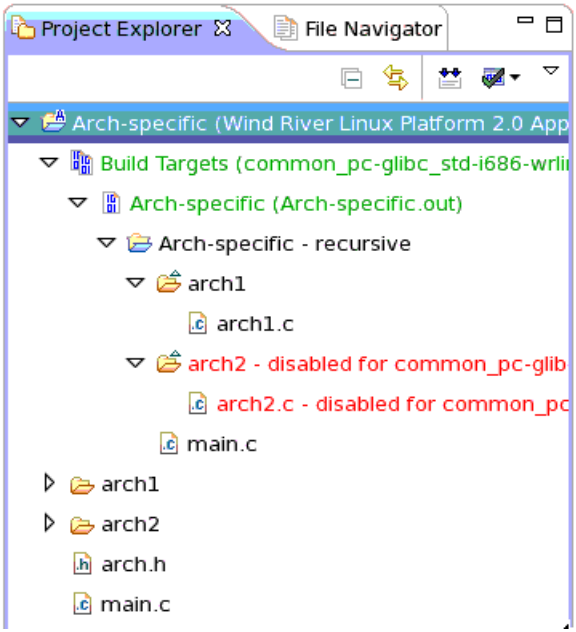
The inner build spec relationships are outlined in Table 9-1.

Table 9-1 **Project Content and Build Spec Configuration**

Directories/Folders	Files	Enabled Build Specs
/project	**main.c, arch.h**	Linux: **common_pc** and **arm_versatile_926ejs** VxWorks: **PENTIUMdiab_RTP** and **PPC32diab_RTP**
/project/arch1	**arch1.c**	Linux: **common_pc** only VxWorks: **PENTIUMdiab_RTP** only
/project/arch2	**arch2.c**	Linux: **arm_versatile_926ejs** only VxWorks: **PPC32diab_RTP** only

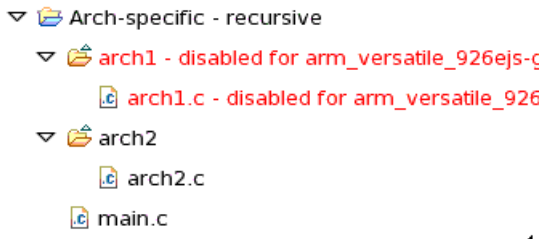
To build the project for the **common_pc** architecture (for example):

1. Highlight the project folder.
2. Click **Set Active Build Spec** on the Project Explorer toolbar.
3. Specify **common_pc** as the active build spec for the project.



The subfolder **arch1** will be built, and its objects will be linked into the project build target. The **arch2** subfolder will not be built, and its objects will not be linked into the project build target because the **comon_pc** build spec is not enabled for **arch2**.

Similarly, if you set **arm_versatile_926ejs** as the active build spec for the project, the **arch2** subfolder will be built, but the **arch1** subfolder will not:



9.6 Creating User-Defined Build Targets in the Project Explorer

In the Project Explorer, you can create custom build targets that reflect rules in makefiles. This is especially useful if you have user-defined projects, which are projects for which Workbench does not manage the build.

You might find this feature useful in other projects as well.

9.6.1 Custom Build Targets in User-Defined Projects

Assume you have two rules in a makefile: **clean** and **all**.

To define a custom build target for either one or both of these rules:

1. Right-click a project or folder and select **New > Build Target**.
2. In the dialog box that appears, enter the rule(s) for which you want to create a target. If you want to execute multiple rules, separate each one with a space.

   In our example, enter **clean all** to cause both these rules to be executed when you build your new user-defined target. (Any rules you specify must exist in your makefile.)
3. Set the **Build tool** to **User-defined**.
4. 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**.

9.6.2 Custom Build Targets in Workbench-Managed Projects

The make rules you want to execute must appear in the project's **.wrmakefile** before you can use them with a build target.

1. Right-click a project or folder and select **New > Build Target**.
2. In the dialog box that appears, enter the rule name(s) you created in **.wrmakefile**. If you want to execute multiple rules, separate each one with a space.
3. Set the **Build tool** to **User-defined**.
4. 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**.

9.6.3 Custom Build Targets in Wind River Linux Platform Projects

For Wind River Linux Platform projects, you edit **Makefile.wr**, not **.wrmakefile**.

The contents of **Makefile.wr** (build properties and build targets) should be edited only through the project's **Properties > Build Properties** dialogs.

NOTE: This example shows how to add a build target to invoke the **uClibc** configuration for PCD-based projects. This technique applies to bringing any other Wind River Linux or custom command line feature to Workbench Platform Projects.

First, write the rules you want add to **Makefile.wr**.

1. Double-click **Makefile.wr** and add the following at the bottom of the file in the Editor view:

   ```
   uclibc-config :
        xterm - e make -C $(DIST_DIR) uclibc.menuconfig
   ```

   An **xterm** is used because this build rule, **menuconfig**, requires a shell that can support **xterm** commands, which is beyond the capabilities of the **Build Log** view within Workbench.

NOTE: At the top of **Makefile.wr** are the definitions **DIST_DIR** and **TARGET_DIR**. These provide the redirection to the Wind River Linux Platform project's content directory.

2. Right-click a project or folder and select **New > Build Target**.
3. In the dialog box 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. For this example, add the build target **uclibc-config**.
4. Set the **Build tool** to **User-defined**.
5. 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**.

9.6.4 User Build Arguments

In its default state, the User Build Arguments drop-down list appears as an empty field at the top of the Build Console.

You can use it to enter and apply one or more arguments (such as a rule or rules, or macro re-definitions) that change the execution of any existing make rule, or override any macro, or affect anything else that is understood by make. This happens at every build, regardless of what is being built.

When you enter something and launch a build, Workbench adds your new entry to the growing drop-down list so it is available for reuse with later builds.

To enter new arguments:

1. Type one or more arguments into the text field, whether it is blank or already has contents. Use spaces to separate multiple arguments.
2. In the Project Explorer, right-click the project you want to build, then select **Build Project**.[1]

   The build causes the text field's new arguments to be stored in the User Build Arguments list. They are appended to (and thus override) the existing makefile entries. This occurs on the fly at every build.

To use arguments already in the drop-down list:

1. Select the appropriate argument or arguments in the list.
2. Click the **Run Last Build again** icon from the Build Console toolbar, or click the down arrow to its right and select the project to build.

   The current setting of the User Build Arguments field applies to the build, that is, the **Run Last Build again** action does not remember the setting that applied when you initially ran it.

The user build arguments functionality does not provide any value, macro, or shell substitution. For these, set up an intermediate makefile (e.g., **Makefile.wr** for platform projects, or perhaps **Makefile.user**).

1. Other ways to launch a build are clicking the **Build all selected projects** toolbar icon, or pressing **CTRL+SHIFT+A**.

9.7 Defining Build Specs 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 the pre-configured build spec of an existing tool chain and architecture, and modify the copy.

Step 1: Copy an Existing Build Spec.

1. Open any application project's build properties, as described under *8.4 Accessing Build Properties*, p.104.

   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 Support and Specs** tab and look at the existing specs.

   The pre-configured build spec names follow a naming convention of *ArchitectureToolChain_ProjectType*, for example, **PENTIUMgnu_RTP**, meaning this spec is configured for a Pentium target board, using GNU tools to create a Real-time Process (RTP).

3. 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.

4. Click **OK**, then enter a name for the copy you are creating in the next dialog and click **OK** again.

5. 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 is normally correct, which is one advantage of copying an existing spec, rather than creating a new one.

Step 2: Configure the Build Tool.

The build system uses generic build tools, for example, **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, make a copy and modify them accordingly.

   For example, to add a compiler for a new language, **foo_language**, you would click **Copy**, give the new compiler a name (for example, **Foo-Compiler**) then click **OK**. Then configure the suffixes and other settings as required.

2. In the **Build spec specific settings**, 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 command in this field. However, for flexibility Workbench provides some predefined macros (referenced using the syntax **%***MacroName***%**), and you can define new macros on the **Macros** tab (referenced using the syntax **$(***MacroName***)**). You can also specify **Tool Flags**, **Debug mode**, and **Non Debug mode** flags.

     For a list of predefined macros as well as other details about these settings, 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 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 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.

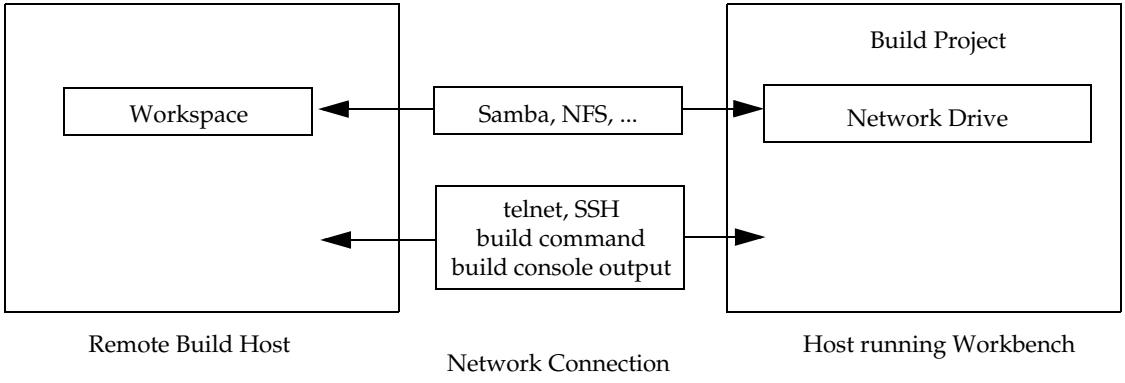
After you have configured the build spec for the first tool in the chain, for example, the compiler, go back to the **Build Tools** tab (see Step 2, above) to configure any additional tools, such as the **Linker** or **Librarian**.

NOTE: The current setting of the User Build Arguments field applies to all builds, including those from the **Run Last Build again** button.

9.8 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.

Figure 9-1 **Remote Build Feature**



In the case of a managed build, Workbench generates the makefiles on the local machine that is running Workbench. You map a path from the workspace root location to where generated makefiles will be correctly placed for builds executed on a remote machine.

When launching the build, Workbench establishes a network connection (**telnet** or **SSH**) to the build host. The actual build command is executed there by using an intermediate script so you can set up the environment for the build process.

9.8.1 General Requirements

- The Workbench host tools and toolchain must be installed on the remote machine.
- The workspace root directory must be accessible from both machines.
- Only Workbench projects under the workspace root can be remotely built. Linked resources are not supported for files outside the workspace.
- A **telnet** or **SSH** remote connection to the build machine must be possible.

9.8.2 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.

9.8.3 Setting Up a Connection to a Remote Environment

Before you can connect to a remote system, you must create the remote connection definition. If necessary, you may also need to edit the remote command script to include the appropriate environment variables or other commands.

Creating a Remote Connection Definition

1. Right-click your project in the Project Explorer, then select **Build Options > Remote Connection > Configure**. The Remote Connections dialog opens.
2. Click **Add**, then double-click **General**, then select the type of remote system you want to connect to. Click **Next**.
3. Most connection types require that you define the following connection information, then click **Next**.

   Host name
   The host name of the remote system (can also be an IP address).

   Connection name
   An arbitrary name for this connection, if you want a name other than the host name to appear in the connections list.

   Description
   Additional descriptive information about this connection.

   If you like, select **Verify host name** to have Workbench check the host name before connecting.
4. On the next screen of most connection types, you can define how to work with files, for example using **FTP** or **SSH**. To update property values, click the value and enter a new one, or select a new value from the drop-down list. Click **Next** or **Finish** (depending on your connection type).
5. On the next few screens of some connection types, you can define how to work with processes, shells, and terminals. To update property values, click the value and enter a new one.
6. When you have entered all required information for your chosen connection type, click **Finish**.

NOTE: You can also add connection definitions in the Remote Systems view by clicking **Define a connection to remote system** from the toolbar. Double-click **General**, then follow the rest of the instructions in this section.

Adding a Remote Workspace Location to a Connection Definition

1. Once the connection to the remote system has been defined, right-click your project then select **Build Options > Remote Connection > Configure**.
2. Select the connection in the top pane of the Remote Connections dialog, verify the Host name and User name, then click **Edit** and enter the absolute path to the remote workspace directory (environment variables are not supported). Click **OK**.
3. To save your settings and close the dialog, click **Close**.
   To save your settings and connect to the remote system, click **Connect**.

Remote settings are stored, and are specific to this workspace. They are not accessible from any other workspace.

Removing a Remote Connection Definition

To remove a connection definition, right-click it in the Remote Systems view and select **Delete**.

Editing the Remote Command Script

The **Edit remote command script** option (**Project > Build Options > Remote Connection**) opens **remote_cmd.sh** in the Editor. This makes it easy to include additional commands and environment variables to set up your environment on the remote machine prior to a build or run.

The following example version has been edited (as indicated by bold font) to set the display and update the PATH variable.

```
#!/bin/sh

WORKSPACE_ROOT="%WorkspaceRoot%"
export WORKSPACE_ROOT

PATH=/MyTools/gmake_special/bin:$PATH
```

```
export PATH

# translate CR to NL for better build output handling
stty ocrnl 1>/dev/null 2>&1


cd $WORKSPACE_ROOT

cd "$1"
shift 1

exec "$@"
```

You can add any commands you need, but all commands must be in **sh** shell style.

9.8.4 Building Projects Remotely

Once you have defined a connection to a remote system and configured any necessary environment variables, you can connect to that system to do a remote build.

1. Connect to a remote system using its default settings by right-clicking a project and selecting **Build Options > Remote Connection**, then selecting one of the available connections.

   To select a different remote workspace from the one listed in the connection definition, click **Configure**, select the remote system, then click **Edit** and enter a different **Remote workspace location** on the remote system.

2. Once your connection is selected, click **Connect** to connect to the remote system.
3. The build is executed on the remote host, with the build output listed in the standard Workbench build console.
4. To return to local development, select **Local Host** from the list of connections, then click **Connect**.

9.8.5 Running Native Applications Remotely

Running native applications remotely is quite similar to running applications locally: a C/C++ Remote Application launch configuration must be created that defines the executable to be run.

1. Right-click your project in the Project Explorer, then select **Run As > Run Configurations**.

2. Double-click **C/C++ Remote Application** to create a new launch configuration.
3. From the **Connection** drop-down list on the Main tab, select the remote connection you want to use.
4. To enter a new remote workspace location for this connection, click **Properties**. Leave **Skip download to target path by default** selected (because the file is already on the remote machine), then click **OK**.
5. If you selected a project, it appears in the Project and Name fields.

   If you did not select a project, click **Browse** and select one from the dialog.
6. To find the application you want to run on the remote machine, click **Search Project** to see the executables in your project, or click **Browse** if the executable is located elsewhere.
7. In the **Remote Absolute File Path for C/C++ Application** field, type in the full path of the executable on the remote host, or click **Browse** and navigate to the executable.
8. Leave **Skip download to target path** selected, then click **Apply** to save these settings and leave the dialog open, or click **Run** to launch the application on the remote host.

The application runs on the remote host, and the output of the application appears in the Console view.

For more information about creating launch configurations, see *14. Launching Programs*.

9.8.6 Example Using Samba on Remote Linux Host

This section presents a procedure in which you configure Samba on a remote Red Hat Linux host. You then use Workbench on a Windows host to create a project, build it, run the application, and debug it. This example assumes the remote Linux host supports **ssh**.

Configure the Remote Linux Host

The following steps assume an RHEL 4 Linux host. If you are using a different version of Linux you will have to translate the specific commands below for the Samba tools available on your host. The **smb** service should be running on your

Linux host (see **System Settings > Server Settings > Services** or **/usr/bin/system-config-services**).

1. Create a workspace directory for a Linux user for which you have log on privileges, or create a new one such as **wbuser**. This is the workspace that you want to export. For example, in your home directory, you could create a subdirectory called **remote_workspace**.

NOTE: This workspace cannot be used remotely by Windows and simultaneously by the local Linux Workbench installation. So either you must not be running Workbench on the Linux host or it must be using a different workspace than the one you specify here.

2. As the root user, start the Samba configuration tool:

```
# /usr/bin/system-config-samba
```

3. Select **Preferences > Samba Users** and click **Add User**. Add the Linux user that has the workspace to be exported.
4. Enter a password for the user. This is the Samba password and can be different than your Windows or Linux password. Click **OK**.
5. Select **Preferences > Server Settings.**
   a. In the **Basic** tab enter a workgroup that is accessible from Windows, for example **workgroup**.
   b. In the **Security** tab, you should be able to leave the settings as **User** authentication mode, **Yes** for password encryption, and **No guest account**.
   c. Click **OK**.
6. Click **Add** to add a Samba share. This is the workspace that you want to export.
   - For **Directory**, enter the full path to the workspace to export, for example **/home/wbuser/remote_workspace**.
   - For **Basic Permissions** check **Read/Write**.
7. Click the **Access** tab and allow access to everyone, or specifically select the user(s) you want to have access. If you want to add specific access for a user that does not appear in this tab, you must add them with **Preferences > Samba Users** as done previously.
8. Click **OK**.
9. Start **usermode-agent**, for example:

```
$ installDir/linux-2.x/usermode-agent/bin/ia/i386/usermode-agent &
```

Configure the Windows Host

The following steps assume a Windows XP host. If you are using a different version of Windows there may be some difference in the commands required.

1. Right-click **My Computer** (or select **Tools** in Windows Explorer) and select **Map Network Drive**.
2. Choose a network drive, for example **W:**, and enter your hostname or IP address and the share (workspace), for example:

   ```
   \\remotebox\remote_workspace
   ```

3. Click **Finish**.

NOTE: If you limited access to specific users, you will be prompted to log in with the user name and Samba password.

Configure Workbench

1. Start Workbench on your Windows host and use the exported workspace. If Workbench prompts you for a workspace on startup, enter the drive you mapped to the workspace, for example, **W:**. Alternatively, once Workbench has started, specify the drive from within Workbench by selecting **File > Switch Workspace**.
2. Select **Project > Remote Connection** and click **Add**.
   a. Enter a name you would like to give the remote connection.
   b. Choose the connection type. This example uses **SSH**.
   c. Enter the host name or IP address of the remote Linux host.
   d. Enter the user name of the login account on the remote Linux host.
   e. Enter the full path of the workspace on the remote Linux host, for example, **/home/wbuser/remote_workspace**.
   f. You can ignore the setting for the X server because this example displays output in the Console view. For details on using an X server with a remote host configuration, see *Using an X Server with a Remote Host*, p.140.
   g. Click **Connect**, and click **Yes** when prompted to save your settings.

3. Enter your remote Linux host password when prompted.

NOTE: You are now connecting to the remote Linux host with **ssh** (or **telnet**) so this must be your Linux password which may be different than your Samba login.

Create an Example Project

1. Right-click in the Project Explorer and select **New > Example > Native Sample Project**. Click **Next**.
2. Select the **Hello World Demonstration Program** and click **Finish**.

   This creates the project. You can now look in the **remote_workspace** directory on your remote Linux host and see that a **hello_world** subdirectory is in it. You can also see the same contents in the Windows drive you have mapped to the share.
3. Click the **Set Active Build Spec and Debug Mode** icon in the Project Explorer and set the active build spec to Linux GNU.
4. Right-click the **hello_world** project and select **Build Project**.The build is performed on the remote Linux host.

Run the Application on the Remote Host

1. Right-click the executable or the **DEBUG** directory and select **Run Native Application**. click the **Remote Settings** tab. In this example we are *not* assuming that the Windows host is running an X server, so delete the string **xterm -e %Application%** in the section under **Remote Program** so that it is empty.
2. If you do not have a Console view in your current Workbench window, select **Window > Show View > Console**. You may want to move it to a place where it is always visible if it is not already.
3. Click **Run**. Build ouput appears in the Build Console view and after a few moments the program output **hello world** appears in the Console view. This output has come from the execution of the program on the remote Linux host.

Debug the Application on the Remote Host

1. Click the **Create a New Target Connection** icon in the Remote Systems view. If you already have a target connection that you want to use, be sure the object path mappings and source path lookup are set correctly as described in this procedure.
2. Select the user mode connection type and click **Next**.
3. Ignore the usermode agent options screen and click **Next**.
4. Enter the remote Linux host name or its IP address and click **Check** to verify it. Click **Next**.
5. Specify your object file path mappings by associating the path to the workspace on the remote Linux host with the drive share under Windows. For example:

   Target path: **/home/wbuser/remote_workspace**

   Host path: **W:**
6. Click **Next** until you can click **Finish** to complete the connection.
7. Select **Run > Debug** and select your new connection. Click **Debug**.

NOTE: If you get a **Source not found** error in the Editor space, click **Edit Source Lookup**, click **Add**, click **Workspace**, and click **OK**. Then expand **Workspace**, select your project folder, and click **OK**. (You can also add this source lookup path to an existing connection by selecting **Run > Debug** and using the **Source** tab.)

You can now proceed to debug the program as usual. Note that program output apears in the Console view.

Using an X Server with a Remote Host

If you have an X server available on your Windows host, such as Exceed or Cygwin X, you can display the remote Linux output in an xterm on the Windows host if you are configured as follows:

1. In **Project > Remote Connection**, the **Display (X server)** field is by default set to *IP_address*:**0.0**. This should be acceptable unless you are running multiple X servers on your Windows host.
2. The Windows X server must allow clients from your remote host to display locally. For example, if you are using Cygwin X, enter **xhost +** in an **xterm**

window to allow access to all X clients, or **xhost +***IP_address*, using the IP address of your remote Linux host for restricted access.

3. If you are using the **hello_world** sample program, the output will probably display too fast to be seen because the xterm closes immediately after displaying the message.
   a. Open the project in Workbench and double-click **helloworld.c** to open it in the Editor.
   b. Add the following **include** statement:
      ```
      #include <unistd.h>
      ```
   c. And add the following line after the **printf** statement:
      ```
      sleep(10);
      ```
   d. Save **helloworld.c**.
4. Right-click the executable or the **DEBUG** directory and select **Run Native Application**, and click the **Remote Settings** tab.

   The **Remote Program** field should be set to **xterm -e %Application%** which causes the ouput to appear in an **xterm** on the Windows host.
5. Click **Run**.

In a moment, an **xterm** window should appear with the message **Hello World**, and close after ten seconds.

NOTE: If you get an error such as **connection to** *IP_address***:0.0 refused by server** it means your X server is not allowing the remote Linux host to connect.

Note that debug output will still appear in the Console view.

Building Locally on the Share

You can use the same share to build locally. For example, assuming you are configured with the same setup as the previous example, do the following:

1. Select **Project > Remote Connection**, select **Local Host** and click **Connect**.
2. Click the **Set Active Build Spec and Debug Mode** icon in the Project Explorer and set the active build spec to Windows GNU.
3. Right-click the **hello_world** project and select **Build Project**.The build is performed on the local Windows host.

Alternatively, you can configure additional connections to remote hosts and toggle between them by selecting them in **Project > Remote Connection.**

PART IV

Target Management

10 Connecting to Targets

10.1 Introduction

You manage the interaction between host tools and the target system using a Workbench target connection. A target connection must be defined and established before the host tools can communicate with the target.

You do all host-side configuration work and connection-related activity in the Remote Systems view. Connections are registered and made accessible to users by the Wind River Registry as described in *10.5 The Registry*, p.148. Connection data may be maintained in a central location and shared between hosts with the use of remote registries as described in*10.5.2 Remote Registries*, p.149.

10.2 The Remote Systems View

A connection to a target (such as a remote system, a target server, or a simulator) must be defined and established before tools can communicate with a target system. You do this in the Remote Systems view, which appears by default at the bottom-left corner of the Application Development and Device Debug perspectives. If the view is not visible, choose **Window > Show View > Remote Systems**, or, if not listed there, choose **Window > Show View > Other**.

The most import tasks done in the Remote Systems 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 *14. Launching Programs*).

10.3 Defining a New Connection

To open the New Connection wizard, use the **Define a connection** toolbar icon, or right-click anywhere in the Remote Systems view and select **New > Connection**.

The first thing the New Connection wizard asks you to do is to select the type of connection you want to define. Once you select a connection type, Workbench presents different options and you enter different values for the remaining screens of the wizard.

NOTE: The connection types that are available to you depend on what Wind River products you have installed; you may not see all of these.

General

In this folder are remote systems connections such as FTP and SSH. For information about these connection types, see the *RSE User Guide* (press the help key for your host, then click **All Topics** at the bottom of the view).

Wind River Linux

In this folder are Wind River Linux-related connection types such as core dump, host development, user mode, kernel mode, and QEMU. For more information about these connection types, see the Linux version of *Wind River Workbench By Example*.

VxWorks 6.x

In this folder are VxWorks-related connection types such as core dump, VxWorks simulator, and target server. For more information about these connection types, see the VxWorks version of *Wind River Workbench By Example*.

NOTE: For more information about the target server itself, press the help key for your host, click **Search** at the bottom of the view, and type **tgtsvr** into the **Search expression** field.

On Chip Debugging

In this folder are connections to the ICE, ICE2, ISS, and Probe. For more information about these connection types, see .

10.3.1 Modifying Connection Properties

Properties you set during the creation of a new connection using the New Connection wizard can be modified by right-clicking the connection in the Remote Systems view and then selecting **Properties** from the context menu.

If you change properties while a target server or other connection is active, you may have to disconnect and reconnect in order for changes to take effect.

10.4 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. To do this, you first need to connect to the target.

Because **Immediately connect to target if possible** is selected by default on the final screen of the New Connection wizard, Workbench automatically attempts to connect to the target. If you do not want Workbench to establish the connection at that time, unselect this option.

To manually connect to and disconnect from targets, select a connection node in the Remote Systems view and then use the appropriate toolbar button, or right-click and select **Connect** or **Disconnect**.

10.5 The Registry

The Wind River Registry is a database of target servers, boards, ports, and other items used by Workbench to communicate with targets.

Before any target connections have been defined, the default registry—which runs on the local host—appears as a single node in the Remote Systems view. Additional registries can be established on remote hosts.[2]

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 mean that the target is actually connected.

- The registry keeps track of the currently running target servers and manages access to them.
- The registry allows Workbench 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 continues running in case it is needed by other tools.

2. For Linux, the default registry is a target-server connection for Linux user mode.

For more information about the registry, press the help key for your host, click **Search** at the bottom of the view, and type **wtxregd** or **wtxreg** into the **Search expression** field.

10.5.1 Launching the Registry

If it is not already running, you can launch the default registry by opening the Target menu or right-clicking in the Remote Systems view and selecting **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.

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

- On Windows, the registry icon is displayed in the Windows system tray.
- On Linux and UNIX, execute **wtxregd status** and look for **wtxregd.ex**.

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.

10.5.2 Remote Registries

If you have multiple target boards being used by different team members, it makes sense to maintain connection data in a central place that is accessible to everybody. This is called a remote registry, and it saves everyone from having to remember communications parameters such as IP addresses and other settings for every board that they might need to use.

To create a new remote registry on a networked host, follow these steps:

1. Workbench must be installed and the registry must 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.
2. To connect to the remote registry from another host, right-click the **Local > Wind River Registries** entry in the Remote Systems view, then select **New > Remote Registry** from the context menu.

3. In the dialog that appears, enter either the host name or the IP address of the remote host. Click **OK**.

   Workbench immediately attempts to connect to the remote registry. A valid connection will display the registry in the Remote Systems view, and any active connections will be shown. Connect to the target just as you would to a target in your local registry.

   If the host is invalid, or if no registry is identified on the remote host, this information is displayed in the Remote Systems view.

10.5.3 Shutting Down the Registry

Because other tools use the registry, it is not automatically shut down when you quit Workbench. However, there are times when you should manually shut down the registry, such as when switching between Tornado and Workbench registries (you cannot run both at the same time), and when updating or uninstalling Workbench (or other products that use 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 **wrenv.sh -p workbench-3.***x* **wtxregd stop** in a terminal window, 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.

10.5.4 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 *10.5.2 Remote Registries*, p.149).

- 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**.

To verify that the registry host ws changed correctly, type **Wind::registry** without parameters to see the name of the currect default registry host.

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-3.***x*/**foundation/4.***x*/**resource/wtxregd/wtxregd.conf**).

For available options and other information about the operation of the registry, type *installDir*/**workbench-3.***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**.

Example Usage

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

On Windows:

```
wtxregd -d C:\temp\registry-db
```

or

```
wtxregd -d %HOME%\registry-db
```

On UNIX:

```
wtxregd.ex -d ${HOME}/registry-db
```

or configure the directory in **wtxregd.conf** to use:

```
-d ${HOME}/registry-db
```

then start **wtxregd** using:

```
wtxregd start
```

PART V

Debugging

13

Working with Breakpoints

13.1 Introduction

Breakpoints allow you to stop a running program at particular places in the code or when specific conditions exist.

This chapter shows how to use the Breakpoints view to keep track of all breakpoints, along with their conditions (if any). It also discusses adding dynamic printf event points to your code.

You can create breakpoints by any of the following means:

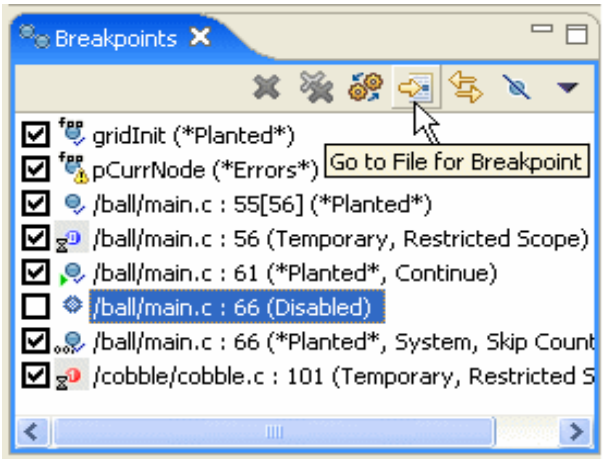
- double-clicking or right-clicking in the Editor's left vertical ruler (also known as the *gutter*)
- opening the various breakpoint dialog boxes from the pull-down menu in the Breakpoints view itself

- selecting one of the breakpoint options from the **Run > Breakpoints** menu

13.2 Types of Breakpoints

Figure 13-1 shows the Breakpoints view with various types of breakpoints set.

Figure 13-1 **Breakpoints View**



See the sections below for when and how to use each type of breakpoint. For a guide to the icons in the Breakpoints view, open the view and press the help key for your host.

13.2.1 Line Breakpoints

Set a line breakpoint to stop your program at a particular line of source code.

To set a line breakpoint with an unrestricted scope (so it will be hit by any process or task running on your target):

1. 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.

2. You can also right-click in the gutter and select **Breakpoints > Add Breakpoint (Scope=Unrestricted).**

To set a line breakpoint that is restricted to just one task or process:

1. Right-click in the Editor gutter.
2. Select **Breakpoints > Add Breakpoint (Scope="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.

To adjust the properties of the breakpoint as you create it:

1. Right-click in the Editor's gutter.
2. Select **Breakpoints > Add Breakpoint**, or select **Add Line Breakpoint** from the Breakpoints view's pull-down menu.

   The Line Breakpoint dialog opens.

13.2.2 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.

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 box.

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.

NOTE: Expression breakpoints appear in the Editor's gutter only when you are connected to a task.

13.2.3 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 support them, only four debug registers are provided, so 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 a data breakpoint or an instruction breakpoint—the debugger will behave in the same way as for a standard breakpoint and stop for user interaction.

WARNING: Do not use hardware breakpoints with the WDB-based tools and an On-Chip Debugging (OCD) tool at the same time. Simultaneous use may lead to unpredictable debugging behavior, as both facilites access hardware breakpoint registers.

Adding Hardware Instruction Breakpoints

You can add a new hardware instruction breakpoint by right-clicking in the gutter on the left of the source file, and selecting **Breakpoints > Add Breakpoint (Hardware)**.

Or, you can follow the following steps:

1. Double-click in the gutter to add a standard breakpoint.
2. In the Breakpoints view, right-click the breakpoint you just added.
3. Select **Properties**.

4. In the last tab (**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).

To add a hardware data breakpoint:

1. Go to the Breakpoints view.
2. Click the down arrow in the top right of this view.
3. Select **Add Data Breakpoint** to display the hardware data breakpoint dialog box.

   You are presented with four tabs.
4. Use the **General** tab to enter the variable you want to monitor in the **Address Expression** box.
5. Use **Status** and **Scope** tabs the same way for hardware code breakpoints.
6. Use the **Hardware** tab to specify what you want to monitor. Select the check box before an option, then enter a value or choose from the drop-down list.

   For example, select **Access Size**, then choose **Byte**, **Half-Word**, or **Word** from the drop-down list. Do the same for any other values you want to monitor for this breakpoint.

Converting Breakpoints to Hardware Breakpoints

To cause the debugger to request that a line or expression breakpoint be a hardware code breakpoint, select **Enable Hardware Breakpoint** 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 is already 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 13-1.

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."

13.2.4 Dynamic printf Event Points

Workbench allows you to dynamically insert instrumentation points (called *dynamic printf event points*), based on breakpoints, to display variables when reaching certain pieces of code. Currently the only supported variable types are **int** and **char***.

The goal of a dynamic **printf** event point is to remove the need to add a **printf( )** statement in an application, and instead use a dynamic **printf** that can be inserted at run time and does not require recompilation.

Using this dynamic **printf**, you can print the value of a specific variable (including the value of the field of a complex C structure) at a specific line of code. When no debugging information is available (for example, if your module has not been built

with debug information support), the dynamic **printf** event points can only display global variables or the content of specified areas of memory.

Whenever a dynamic **printf** event point is hit, a routine is called to read the requested information; this information is stored and then provided to another task that handles the display of that information.

Since a dynamic **printf** event point does not require stopping the current task, you can insert a dynamic **printf** event point on an unbreakable task. However, the dynamic **printf** event points are not raised if the routine is called from an interrupt handler.

You can implement dynamic printf event points using both software and hardware breakpoints. Enable, disable, edit, or remove event points the same way as with regular breakpoints.

Working With Dynamic printf Event Points

To add a dynamic **printf** event point, right-click in the editor or in the gutter and select **Add Dynamic printf** from the context menu.

This brings up the dynamic **printf** dialog. Use this dialog to specify the **printf** arguments and other attributes. Click **OK** to create the event point. The event point appears in the gutter, like any other breakpoint.

If you place the cursor on a variable of type **int** or **char*** when creating a dynamic **printf** event point, Workbench automatically fills in the **printf** arguments with default arguments based on the name and type of the variable. For example, if you select the integer variable counter and then create a **dprintf** event point, the default arguments are

```
"counter=%d\n", counter
```

If you select a variable that is part of a field reference expression, Workbench takes the complete expression as the **printf** argument. For example, if you select **data**, Workbench uses **node->data** as the **printf** argument.

If the variable you select is a pointer type, Workbench uses any suitable surrounding pointer dereference or array subscript expression as the **printf** argument.

If you do not select a variable, Workbench looks at comments in or before the line in focus for a dynamic **printf** argument hint. This is a string of the format

```
DP "format", arg1, arg2, …
```

or

```
printf("format", arg1, arg2, …)
```

If Workbench finds such a string, it uses that string to fill in the default arguments for the dynamic **printf** event point. Dynamic **printf** event points use a syntax similar to the standard **printf( )** command. You must provide a format string that controls the output and formatting of the arguments, and a list of arguments that represent the variables or values to be printed.

Dynamic **printf** events are also reported in Wind River System Viewer.

13.3 Managing Breakpoints

You can take the following actions to manage breakpoints:

- import and export
- refresh
- disable
- remove

13.3.1 Importing Breakpoints

To import breakpoint properties from a file:

1. Select **File > Import > General > Breakpoints**, then click **Next**.

   The Import Breakpoints dialog box appears.

2. Select the breakpoint file you want to import, then click **Next**.

   The Select Breakpoints dialog box 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.

13.3.2 Exporting Breakpoints

To export breakpoint properties to a file:

1. Select **File > Export > General > Breakpoints**, then click **Next**.

   The Export Breakpoints dialog box appears.
2. Select the breakpoint whose properties you want to export.
3. Type in a file name for the exported file.
4. Click **Finish**.

13.3.3 Refreshing Breakpoints

If something has changed on the target (for example, a new module was downloaded) and a breakpoint is not automatically updated, you can remove and reinsert it:

1. Right-click a breakpoint in the Breakpoints view.
2. Select **Refresh Breakpoint**.

NOTE: The **Refresh Breakpoint** selection only appears if the breakpoint is planted.

To refresh all breakpoints in this way, select **Refresh All Breakpoints** from the Breakpoints view drop-down menu.

13.3.4 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.

13.3.5 Removing Breakpoints

To remove a breakpoint

1. Select it in the Breakpoints view.
2. Click **Remove** on the toolbar.

13.4 Knowing Which Debugger Gets the Breakpoints

If you are using the GDB debugger for native-mode (self-hosted) applications, note that some breakpoint actions always create breakpoints exclusively for the Wind River debugger.

This applies to all actions in the Breakpoints and Tracepoints submenus. If a debugger has at least one debug session active, the active debugger is chosen. If there are mixed debug sessions, the breakpoints are seen by the Wind River debugger. In particular, the Wind River debugger is assumed or chosen in the following situations:

- If no project is associated with the open file, because CDT does not support project-less debugging.
- If there is no active debugger, and the target seems to be a device. Otherwise if the project is a CDT project, a CDT breakpoint will be created.
- If a Wind River perspective is open (such as Application Development, Device Debug, On Chip Debug, or Hardware Debug), otherwise the CDT debugger is chosen.

For more information about debugging self-hosted applications, see *15.4 Debugging Self-Hosted Applications*, p.200.

13.5 Limitations on Breakpoints During SMP Task Debugging

In general, task mode debugging on symmetric multiprocessing (SMP) systems is very much like task mode debugging on uniprocessor (UP) systems.

However, there are limitations on when and where you can place breakpoints when working on SMP systems.

Breakpoints cannot be placed on these routines

During breakpoint exception handling, a number of kernel APIs are called before all breakpoints are removed from the target memory, so you cannot put breakpoints on these routines.

```
taskCpuLock( )/taskDbgUnlock( )
intCpuLock( )/intCpuUnlock( )
```

```
usrBreakpointSet( )
vxTas( )
```

Breakpoint exception while holding an ISR-callable spinlock

Workbench ignores this type of breakpoint and resumes the execution of the context (in other words it steps over this type of breakpoint) since an ISR attempting to take the same spinlock will spin forever.

Breakpoint exception while holding a task-callable spinlock

Workbench ignores this type of breakpoint and resumes the execution of the context (in other word it steps over this type of breakpoint). The task that holds the spinlock can be stopped while running on **CPU0** and the scheduler can decide to resume it on **CPU1**. This type of scenario (taking and releasing a spinlock on different CPUs) is a kernel fatal error and must be prevented.

14 Launching Programs

14.1 Introduction

A launch configuration is similar to a macro, because it allows you to group together the actions required to build your program, connect to your target, start your process, and if you wish, attach the debugger. Your configurations are stored persistently, so they can be rerun by clicking a single button or can be shared with your team.

This chapter explains how to 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 detailed descriptions of the tabs in this dialog, as well as a guide to the icons you will see, open the launch configuration dialog, click in the tab you want information about, then press the help key for your host.

14.2 Defining Terminology

Because these terms sound so similar, it may be useful to define them:

- A *launch* is what occurs when you initiate a run or debug session. A launch is a specific instance of a launch configuration.
- A *launch configuration* is your definition of how the launch will occur. It specifies what program will be run, what target it will run on, and what the arguments are. A launch configuration is a specific instance of a launch type.
- A *launch type* defines the kind of launches that are supported by Workbench. There are several different kinds of launch types, for example, Java Application, Process on Target, and Launch Control.

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.

- *Run-mode* connects to your target, then launches a process.
- *Debug-mode* is like run-mode, but it automatically attaches the Wind River debugger after completing all other actions.

A launch may be initiated by the **Run** or **Debug** buttons in Workbench (launches may be initiated other ways too). Note that some launch types may only be available in one mode.

14.3 Creating a Launch Configuration

Many of the launch types require you to enter similar information, so the following sections will help you create a launch configuration for them. For additional details, open the launch configuration dialog, select a tab, and press the help key for your host.

To create a launch configuration from the launch configuration dialog:

1. Select **Run > Run Configurations** or **Run > Debug Configurations**. The **Create, manage, and run configurations** dialog box appears.
2. Select the type of launch type you need, then click **New** (or double-click the launch type).
3. Tabs appear and display the appropriate fields and options you can use for your launch configuration; for details see *14.3.1 Customizing a Launch Configuration*, p.170.
4. Enter information as necessary, then click **Run** or **Debug**.

To create a launch configuration from a build target:

1. Right-click a build target in the Project Explorer.
2. Select the appropriate **Run** or **Debug** command from the context menu.
3. If a similar launch configuration exists, a selection dialog appears. Choose one of the following options, then click **OK**:
   - Use the selected launch configuration.
   - Edit the configuration, for example, to change the target.
   - Copy the configuration and then edit it (thereby preserving the original).
   - Create a new launch configuration.
4. If you decide to edit, copy and edit, or create a new launch configuration, enter information as appropriate for your program, then click **Run** or **Debug**.

In addition to the launch configurations you create manually, a launch configuration is automatically created whenever you run a process or task from Workbench. For details see *14.5.1 Editing an Automatically Created Launch Configuration*, p.176.

14.3.1 Customizing a Launch Configuration

Once you click **New** to create a new launch configuration, or select an option from the launch selection dialog, several tabs appear. These tabs may differ from one launch configuration to another, but the following sections describe the ones that appear most frequently.

If you selected a build target in the Project Explorer, the name of the build target appears at the top of the dialog box in the form *name - connection_name*. If you did not select a build target, or want to modify the name that appears, type a descriptive name for your new launch configuration.

Specifying Connection, Output File, and Breapoint Information with the Launch Context Tab

The **Launch Context** tab displays information about the output file that you want to run during the launch, and the connection that should be used.

Configuring the Connection to Use

The default that Workbench will use is the target that is currently connected. If no connections are active, the default is the target that is selected in the Remote Systems view.

1. Keep the default launch context settings, or if you have more than one connection defined in the Remote Systems view, you may select a different one.
2. To configure the connection, including updating target server options and object path mappings, select it and click **Properties**. If necessary, click **Create a new connection**.

Specifying the Output File or Process to Run

The **Entry Point** field is always available, but the **Exec Path on Target**, **Arguments**, and other fields in this section are only active when you are connected to a target. To retrieve the connection-specific properties from the target, and adjust them if necessary, click **Connect**.

For details about these fields, press the help key for your host.

Specifying Where Your Program should Break

With **Break on Entry** selected and a routine entered in the box, the process will break on the entry to the routine for debugging operations. If you want the

program to run to the first breakpoint you set, rather than breaking immediately after startup, clear this check box.

For kernel task launches, if you want Workbench to automatically attach spawned Kernel Tasks, select that option.

Specifying a Build Target to Download

For kernel task launches only, if you want Workbench to download a particular build target each time the launch is used, specify it on the Downloads tab. If you selected a build target in the Project Explorer before opening the launch configuration dialog, the file appears in the Downloads list automatically.

You can also create launches for kernel tasks that are already downloaded, or are resident in flash memory or are part of the kernel image, but 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.

NOTE: If the build target you specify requires that a module be downloaded, but that module is already in use when you initiate the launch, a dialog appears warning that all running tasks from the module will terminate unexpectedly.

Specifying the Projects to Build

The **Projects to Build** tab displays the projects that Workbench will build before launching the process in this configuration.

If you selected a build target in the Project Explorer, its project appears in the Projects to Build list automatically. You can specify that other projects should also be built (if necessary) before launching the current project with the **Projects to Build** tab. This only applies if you have selected **Build (if required) before launching** in the **Window > Preferences > Run/Debug > Launching** dialog box.

1. To add to the list, click **Add Project**, select one or more projects from the dialog box, then click **OK**.
2. To rearrange the build order, select a project then click **Up**, **Down**, or **Remove**. Then click **Apply** and **Close**. When this configuration is launched, those projects will be built first (if required), in the specified order, prior to launch of this project.

Note that the Projects to Build list takes project-subproject relationships from the Project Explorer into account. Thus, when **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 is therefore disabled.

Identifying Source File Locations Using the Source Tab

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. But if the build-target was compiled on a different host, you need to configure the Source Lookup Path. See *15.5 Changing Source Lookup Path Settings*, p.204 for more information about the source locator.

The **Source** tab displays the search order for source files during debugging. The search order is determined by a location's position in the list.

1. On the **Source** tab, click **Add** to configure the source lookup path.
2. Select the type of source to add to the lookup path; for a description of each type, open the source lookup dialog and press the help key for your host.
3. Once you add a new source to the lookup path, you can adjust its position in the search order by clicking **Up** or **Down** to change its position in the list.
4. Select **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.

Configuring Access Methods Using the Common Tab

The **Common** tab allows you to specify whether this launch configuration is **Local** or **Shared** (local is the default), whether you want to access it from the Workbench toolbar buttons, and if the program should be launched in the background.

1. To share this launch configuration with others on your team, click **Shared**, then type in or browse to the directory where the shared configuration should be saved.
2. If you want to be able to access this launch configuration from the **Run** or **Debug** favorites menus (the drop-down lists next to the bug button on the

Workbench toolbar), select **Run** or **Debug** in the **Display in favorites menu** box.

3. If you want the process to launch in the background, ensure that box is checked.
4. Click **Apply** to save your settings but leave the dialog box open, click **Close** to save your launch configuration for later use, or click **Run** or **Debug** to launch it now.

14.4 Using Launch Configurations to 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** toolbar icon to relaunch the most recently executed process (and attach the debugger as appropriate).

Other ways to launch a recently run process include the following:

- 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 pull-down arrow next to the **Run** or **Debug** icon on the Workbench toolbar, then select the process 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 while creating the launch configuration (see *Configuring Access Methods Using the Common Tab*, p.172), it will always appear on the list, whether you have run it recently or not.
- To run a configuration not listed on the menu, select **Run > Run Configurations** or **Run > Debug Configurations**, choose the configuration you want from the list, and click **Run** or **Debug**.

Increasing the Launch History

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.

Launch Configuration Preferences

There are two different places to set launch configuration preferences.

- To apply working sets or specify various filters for your launch configurations, see **Window > Preferences > Run/Debug > Launching > Launch Configurations**.
- To configure the criteria Workbench uses to determine whether an existing launch configuration matches a program-connection combination you are trying to launch (and therefore whether that launch configuration should be reused or a new one should be created), as well as how launches should be grouped and displayed, see **Window > Preferences > Wind River > Target Management > Launch Configurations**.

Troubleshooting Launch Configurations

If you press the **Run** or **Debug** button (or click the **Run** or **Debug** button from the Launch Configuration dialog box) and get a "Cannot create context" error, check the **Exec Path** on the **Main** tab of the **Debug** dialog box (see *Specifying Connection, Output File, and Breapoint Information with the Launch Context Tab*, p.170) to be sure it is correct. Also check your Object Path Mappings.

14.5 Launching Programs Manually

Once a launch configuration has been established, you can run programs using the **Run** or **Target** selections on the menu bar, or using the **Run** and **Debug** buttons on the toolbar. Workbench will automatically try to connect to the target, if a connection is not already running.

Workbench checks the executable file running as a process on the target against the counterpart present on the host and accessible by the host's debugger server. These executables may be the same, if both the target and the host have access to the same file system. But they can differ, too. Workbench pops up a warning if it

finds differences in their section sizes (such as in the **.text**, **.bss** and **.data** sections). This might warn you, for instance, that you have rebuilt the host executable, but you have forgotten to update the executable running on the target.

NOTE: The target executable must have the same sections, but it need not have any debug information, if, for example, the target has limited resources.

To launch a process from the Remote Systems view, follow these steps:

1. Right-click **Processes**, then select **Run | Debug Process**. The launch configuration selection dialog box appears.
2. Choose whether to run an existing launch configuration, or if you will need to edit or create a new one. Click **OK**.
3. Select **Exec Path on Target** then type the path and filename (as seen by the target) into the field.

   Alternately, select **Exec Path on Host** and click **Browse Files** or **Select a Build Target**. Once you select a file, Workbench automatically translates the host path to the appropriate target path.
4. To immediately put the program under debugger control at launch, select **Break at** and enter the entry point of your program; to let it run, clear the **Break at** check box. Click **Run** or **Debug**.

   Workbench runs the process on the target; the executable and its host location, along with the individual tasks, appear below **Processes** in the Remote Systems view. If a red **S** appears, then symbol information has been loaded into the debugger.

   If you selected **Break at** and provided an entry point, four other things happen as well:

   - Workbench automatically switches to the Device Debug perspective (if it is not already open).
   - The process is displayed in the Debug view.
   - A temporary breakpoint is planted and appears in the Breakpoints view.
   - The program executes up to the entry point and breaks.

Whenever you manually run a process, a corresponding Attach to Target launch configuration is automatically created.

14.5.1 Editing an Automatically Created Launch Configuration

You do not create Attach to Target launch configurations manually. Instead, these configurations are created automatically when you attach the debugger to a process or kernel task in the Remote Systems view.

Attach to Target launch configurations are special in some ways:

- They do not actually run a program, but just connect a target and attach the debugger to some context that must already exist.
- They are visible only in Debug mode.

Once an Attach to Target launch is created, you can review and edit it in the Launch Configurations dialog box as described in *14.3.1 Customizing a Launch Configuration*, p.170. You can also rename them if necessary, and add them to your Favorites menu using the Common tab.

NOTE: One property of Attach to Target launch configurations that differs from other launch types is that the information on the Launch Context tab is for review only, and cannot be changed. This is because it reflects information about an actual running process.

14.6 Attaching the Debugger to a Running Process

You can attach to a running process to debug it as follows:

1. In the Remote Systems view, expand **Processes** for the target connection and locate the process you want to attach to.
2. Right-click the process and select **Attach to Process**.

If you compiled the process with debug symbols, the symbols should load to allow source-level debugging.

Right-clicking a process executable from the Project Explorer and selecting **Run Process on Target** or **Debug Process on Target** opens the appropriate launch configuration dialog box. See *14.3 Creating a Launch Configuration*, p.169 for more information about working with these dialog boxes.

14.7 Controlling Multiple Launches

You can create a Batch Launch launch configuration, consisting of a sequence of your other launch configurations, each one of which is then considered a sub-launch. You can even add other Batch Launch launches to a Batch Launch configuration, the only restriction being that a Batch Launch configuration cannot contain itself.

For detailed information on launch control settings, open the launch configuration dialog, click in the tab you want information about, then press the help key for your host.

Configuring a Launch Sequence

The following procedure assumes you have two or more launch configurations already defined (see *14.3 Creating a Launch Configuration*, p.169).

1. Select **Run > Debug Configurations** to open the **Create, manage, and run configurations** dialog.
2. Select **Launch Control** from the list on the left, and then click **New**. 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 *14.8 Launches and the Console View*, p.181.

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 only the launch you are working with—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.179 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, puts **usermode-agent** in the root file system, reboots the board, and starts **usermode-agent**.

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 are not 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 **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 only the launch you are working with—other launches using the same configuration get the default values for the environment variables.

Environment variables are gathered from four different sources:

- From 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.
- From the sub-launch's Environment tab (if it has one).
- From the sub-launch's configuration type attributes. Each sub-launch configuration type defines its own set of attributes: see the Eclipse documentation for Launch Configuration for details on sub-launch attributes.
- From the Launch Control launch configuration. The variables are defined by Launch Control for each sub-launch, and provide general support for the launch:
  - **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 **Run > Debug Configurations** to initiate a debug mode launch and **Run- > Run Configurations** to initiate a run mode launch). Changing **com_windriver_ide_launchcontrol_launch_mode** has no effect—it only provides information about a current launch.

Since the command's environment terminates after the command completes, any variables that 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 equals sign (=) 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**.

14.8 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.

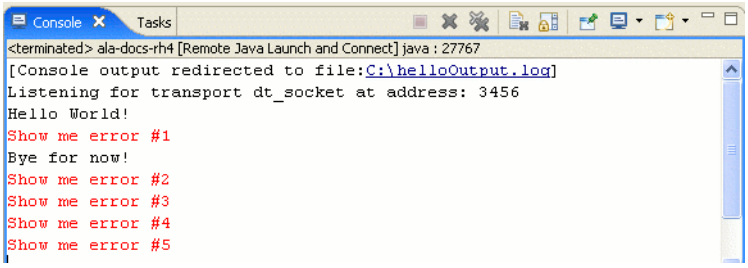
In the Common tab you can also specify a file where console output is appended or overwritten with each launch. The Console view itself offers several controls as described in the next section.

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 > General > Console**. An example view is shown below.

Figure 14-1 **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 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 press the help key for your host.

14.9 Attaching to the Kernel

You can attach to the kernel in KGDB debugging mode.

14.10 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 watch is that you cannot rebuild your program or kernel while it is still being debugged (or its debug info is still loaded into the debugger). 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 **Window > Preferences > Wind River > 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 box 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.

15 Debugging Projects

15.1 Introduction

The Workbench debugger lets you download object modules, launch new processes, and take control of processes running on the target. Unlike other debuggers, you can attach to multiple processes simultaneously, without affecting them or being required to disconnect from one process in order to attach to another.

This chapter introduces the Debug and Disassembly views, and shows you how to use them to debug your programs.

NOTE: You must use the **-g** compiler option to use many debugger features. The compiler settings used by the Workbench project facility's managed builds include debugging symbols.

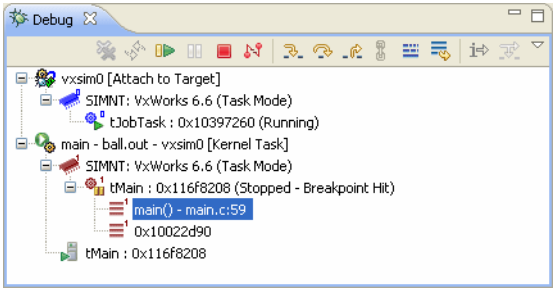
15.2 Using the Debug View

Use the Debug view to monitor and manipulate the processes running on your target. Unlike the Remote Systems view, 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 be able to see it in the Debug view, follow these steps:

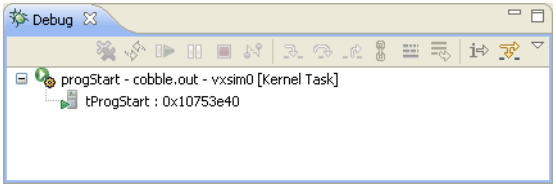
1. Connect to your target in the Remote Systems view (see *10.4 Establishing a Connection*, p.147).
2. Launch one or more processes:
   - Using a launch configuration as described in *14.3 Creating a Launch Configuration*, p.169.
   - Manually, as described in *14.5 Launching Programs Manually*, p.174.
   - By attaching to an already running process, as described in *14.6 Attaching the Debugger to a Running Process*, p.176.
3. Once the debugger has attached to your process, it will appear in the Debug view.

Figure 15-1 **Debug View**



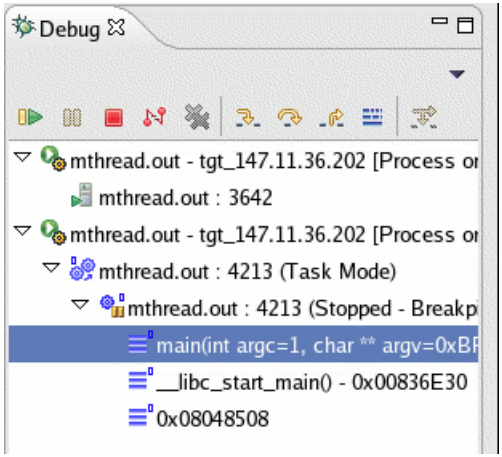
Additionally, the Debug view shows processes that were launched on the target using Workbench, but which were not attached by the debugger. These launches have a special entry in the Debug view, as shown in Figure 15-2, and are only available to help you locate and terminate the process.

Figure 15-2 **Debug View Showing Process Not Under Debugger Control**



The Debug view displays processes differently depending on whether the debugger is attached or not. Figure 15-3 shows two instances of the **mthread** process: the first instance does not have the debugger attached, and the second does.

Figure 15-3 **Debug View with Unattached and Attached Processes**



15.2.1 Understanding the Debug View Display

The Debug view displays a hierarchical tree for each process being debugged.

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.

The following examples show what might appear at each level of the tree, with a general description of each level.

For the ball sample in the Linux environment, the tree looks like this:

ball (2) [Process on Target] = launch level
launch name [launch type]

> **16,ball.out (MPC8260: Linux 2.6)** = debug target level
> *process name (core name:OS name OS version)*
>
> > **16,ball.out (Stopped - Breakpoint)** = thread level
> > *thread name (state - reason for state change)*
> >
> > > **main( ) - main.c:59** = stack frame level
> > > *function(args) - file : line* #, can also be *address*

For the ball sample in the VxWorks environment, the tree looks like this:

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*

Stack arguments and argument values are not displayed in the Debug view by default, to improve debugging performance.

To view stack-level arguments in the Debug view, select **Window > Preferences > Wind River > Run/Debug**, then select the **Retrieve stack arguments for stack frames in Debug View** and **Retrieve stack argument values for stack frames in Debug View** check boxes. Click **OK**.

NOTE: The stack arguments reflect the current values of the stack argument variables, not their initial values 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 15-4 (from a VxWorks view):

- 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.

For example, in Figure 15-5 (from a Linux view), three processes are shown in the Debug view:

- The **ball** process, in pink in the Debug view, has been launched in debug mode and the program counter is shown, in pink, in the **main( )** routine.
- The **forkexec** process is shown in blue. It has stopped at a breakpoint set at the **fork** system call. The breakpoint is shown as a solid circle and the program pointer is shown in blue with the number 0 in it. Note that the number 0 is also shown with the parent process in the Debug view.
- The third process, the forked child process, is shown in red 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 15-4 **Debug View with Breakpoint and Editor Views (VxWorks)**

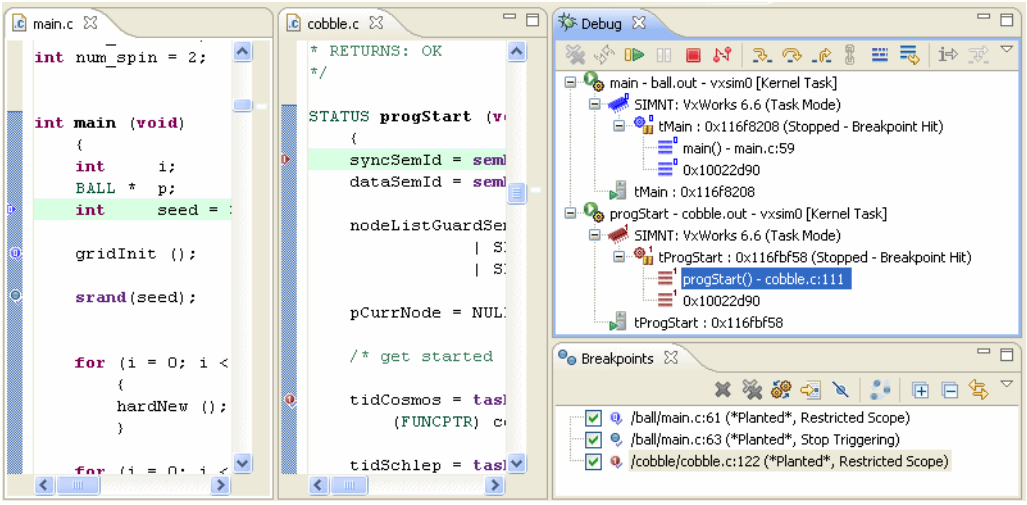
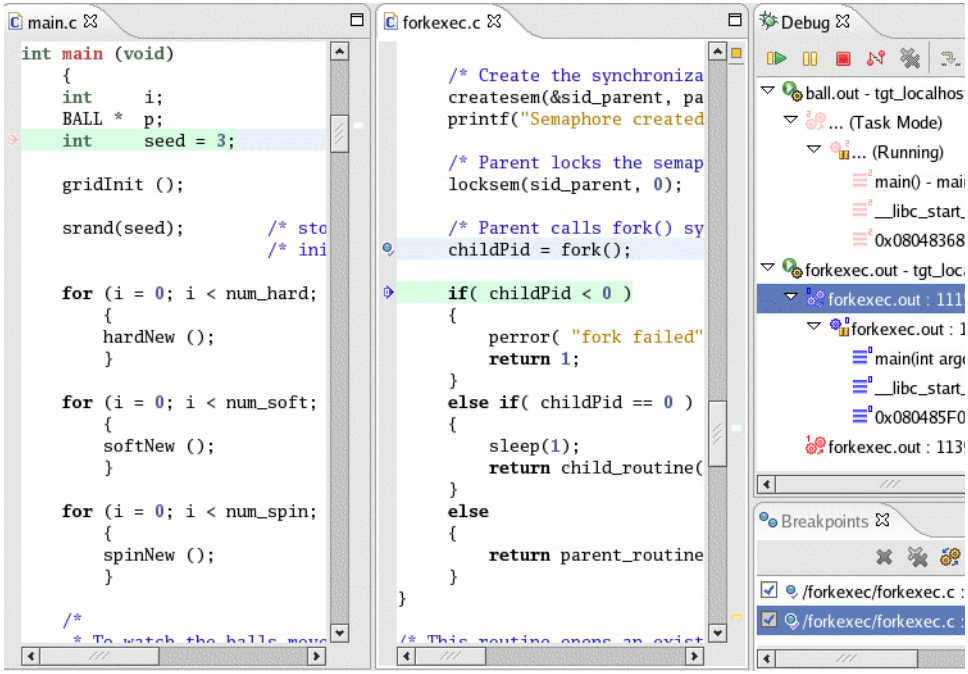


Figure 15-5 **Debug View with Editor and Breakpoint View (Linux)**



The context pointer (the arrow in the left gutter in **main.c**) indicates the statement that will execute when the process resumes.

For more information about how to set up Debug view settings, open the Debug view and press the help key for your host.

15.2.2 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 the **Suspend** button).

To step through the code one line at a time, click the Debug view's **Step Into** button. If you have other data views open, such as the Registers or 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** on the Debug view toolbar, 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 be shown instead of the Editor.

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. Click the Debug view's **Step Return** button so that 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.

15.3 Using Debug Modes

Depending on the type of connection the debugger has to the target, the debugger may be capable of operating 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.
usermode agent on Linux	**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.

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.
usermode agent on Linux	**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.

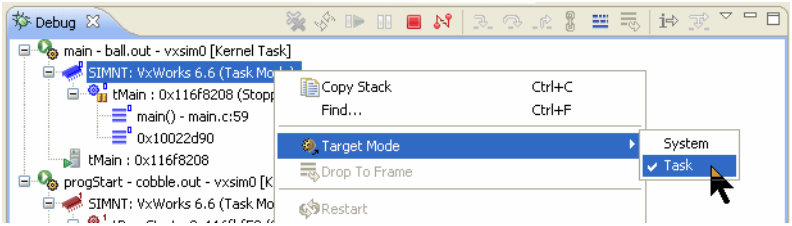
As a general rule, when the target is being debugged in user mode or task mode, the debugger interacts only with the process or processes being debugged. If this process is suspended, other processes keep running. This mode is less intrusive, as it allows the user to control the selected process or thread while the rest of the system can continue to operate normally.

When you are debugging in system or kernel mode, the debugger interacts with the entire system at once, so if one task is suspended, all processes and kernel tasks running on the system are suspended as well. This allows for increased control of (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.

15.3.1 Setting and Recognizing the Debug Mode of a Connection

Right-clicking a connection in the Remote Systems or the Debug views and selecting **Target Mode** allows you to specify a debug mode for the connection. The currently active mode is indicated by a checkmark.

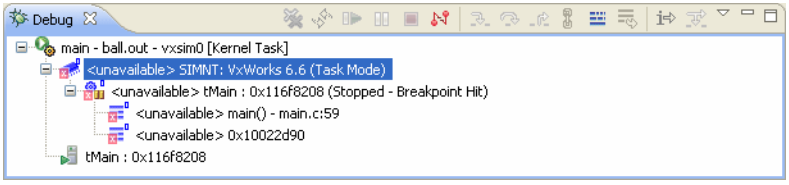


When you create a new debug connection through a launch, the connection debug mode (system or task) is saved as a property of the launch. This mode is listed in parentheses at the end of the label of the target node in the Debug view.

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 15-6. When a debug connection is unavailable, no operations can be performed on it, except for disconnecting the debug connection.

Figure 15-6 **Debug View Showing Unavailable Connections**



In the Remote Systems view, 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.

15.3.2 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 Remote Systems and the Debug views (if a debug session is active) both provide feedback in this scenario. The Remote Systems view 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.

15.3.3 Suppressing Target Exception Dialogs

When you are running a task on a target and an exception occurs, Workbench is preconfigured to display a dialog prompting you to attach to the task in order to debug it.

If you are writing an application for which a particular exception is acceptable (for example, divide by 0) or you want the target OS to stop the task on an exception but you do not necessarily want to debug it, then you can suppress the display of target exceptions in the target connection properties dialog.

1. Right-click your target connection in the Remote Systems view, then select **Properties**.
2. The properties dialog displays a set of tabs. Using the small arrows on the right of the dialog, scroll to the right until you see the **Debug Options** tab.
3. Unselect **Show dialog on target exceptions**, then click **OK**.

NOTE: You can also unselect this option when creating a new target connection. The Debug Options screen is almost the last screen of the New Connection wizard.

15.3.4 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.

15.3.5 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 > Editors > 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..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>
```

Refer to Eclipse SDK documentation for more about these extension points.

15.4 Debugging Self-Hosted Applications

Self-hosted applications are those that are built for, and run on, your local machine. This is also known as native development, to contrast it with the cross development you do on your local machine when you build applications that will run on a different processor, a remote target, or a simulator.

For information about how breakpoints are set when using more than one debugger, see *13.4 Knowing Which Debugger Gets the Breakpoints*, p.164.

15.4.1 Debugging with GDB

On Linux, you may use the GDB debugger built into Eclipse to debug native-mode, self-hosted applications, such as the **Hello World** example included with Workbench. On Windows, you must acquire and install GDB yourself.

To debug **Hello World** using the GDB debugger:

1. Create a project using the example sources by selecting **File > New > Example > Native Sample Project > The Hello World Demonstration Program**, then clicking **Finish**.
2. Build the **hello_world_Native** project by right-clicking it and selecting **Build Project**, or by clicking **Build all selected projects** on the Project Explorer toolbar.
3. Debug the project by right-clicking the project folder again, then selecting **Debug As > Local C/C++ Application**.
4. Choose **gdb Debugger** (or the appropriate version of **gdb** for your system) from the Launch Debug Configuration Selection dialog, then click **OK**.

   The process executes until it comes to the **main( )** routine in the program, and because **Local C/C++ Application** is a CDT launch type, the Debug (rather than the Device Debug) perspective opens.

   The Debug perspective shows the Debug view in the upper left, the editor displays the source file, and other views show typical debugging operations.
5. In the Debug view, note:
   - The process ID (PID) in the entry **hello_world_Nat**: *PID* **(Task Mode)**
   - The entry **hello_world_Nat**: *PID* **(Stopped - Step End)**
   - The breakpoint itself appears as **main( )** *full path*/**helloworld.c:8**, meaning that execution has stopped at Line 8 in **helloworld.c**.
6. In the Debug view, hover the mouse cursor over the buttons in the view's toolbar to see the debug functions that you can perform: Restart, Resume, Terminate, Step Into, Step Over, and Instruction Stepping Mode.
7. Click the **Step Over** button (or press **F6**). The program counter advances one line in the editor, and **Hello World** displays in the Console view (you may have to click the Console tab in the lower tabbed notebook to bring it to the foreground).
8. Step through the program, or press **F8** or click **Resume** to complete it.

When the program has completed, the Debug view displays its exit status: **<terminated>hello_world_Native [C/C++ Local Application]**.

9. To remove old information from the Debug view, click **Remove All Terminated Launches** on the Debug view's toolbar.

15.4.2 Debugging with the Wind River Debugger (Linux Hosts Only)

To run and debug **hello_world** on a Linux host:

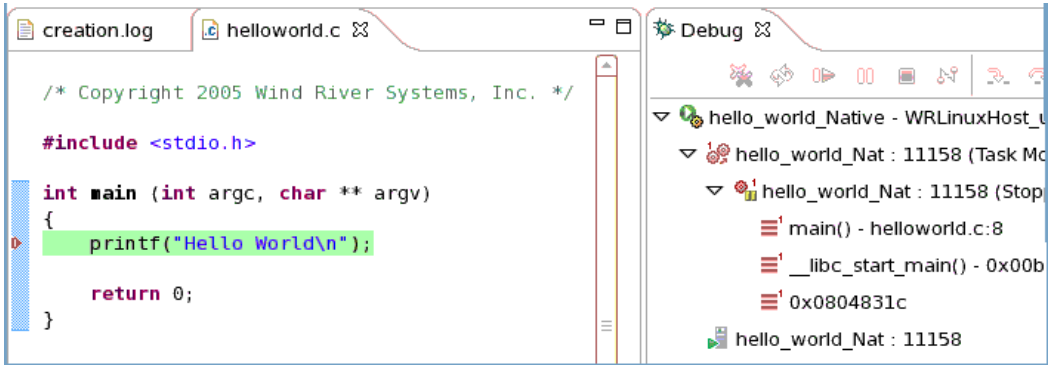
1. Create a connection to the target:
   a. In the Remote Systems view (on the lower left), select **WRLinuxHost**_*username*.
   b. Click the green connection button in the Remote Systems toolbar to open a default output window.

      The **usermode-agent** window opens.
2. In the Remote Systems view, right-click **WRLinuxHost**_*username* again and select **Debug > Debug Process on Target**.
3. In the Main tab, find the text box labeled **Exec Path on Target**. If there is no exec path, browse to the workspace location of the **hello_world** executable, select **hello_world**, and click **OK**.

NOTE: The location is *workspace*/**hello_world_Native/Linux-gnu-native-3.x/hello_world_Native /Debug/hello_world_Native**, or a similar path depending on your configuration.

4. Click **Debug** in the lower right corner of the Debug dialog.

   The process executes until it comes to the **main( )** routine in the program. Workbench's Device Debug perspective shows the Debug view in the upper right. The editor displays the source file, and other views show typical debugging operations.



5. In the Debug view, note:
   - The process ID (PID) in the entry **hello_world_Nat**: *PID* **(Task Mode)**
   - The entry **hello_world_Nat**: *PID* **(Stopped - Breakpoint Hit)**
   - The breakpoint itself appears as **main( ) - helloworld.c:8**, meaning that execution has stopped at Line 8 in **helloworld.c**.
6. To display all the processes running on the target and the state of the **hello_world_Nat** process:
   a. Expand **Processes** in the Remote Systems view.
   b. Scroll down if necessary until you find the process ID of **hello_world_Nat**.
   c. Expand it.

   The process is shown as **[Stopped]** because it has hit a breakpoint.
7. Move to the Debug view and hover the mouse cursor over the buttons in the view's toolbar to see the debug functions that you can perform: Resume, Terminate, Disconnect, Step (Into, Over, and Return), Toggle Assembly Stepping Mode, and Drop to Frame.

   To see function keys assigned to common activities, open the **Run** menu on the Workbench toolbar.
8. Click the **Step Over** button (or press **F6**). The program counter advances one line in the editor, and **Hello World** displays in the Console view (you may have to click the Console tab in the lower tabbed notebook to bring it to the foreground).

9. Continue stepping through the program, or press **F8** or click the **Resume** button to complete it.

   When the program has completed, the Debug view displays its exit status: **<terminated, exit value: 0>hello_world_Nat:** *PID*.

10. To remove old information from the Debug view, click **Remove All Terminated Launches** on the Debug view's toolbar.

NOTE: You may also debug self-hosted applications from another instance of Workbench running on another host, in addition to using the local Workbench. For example, this lets you use a 64-bit host that you share with other 32-bit hosts, so you can develop 64/32 bit applications when 64-bit hosts are rare.

To debug self-hosted applications from another host, start the usermode agent on the "self-hosted" host as if it were a target, then connect to it from another host.

15.5 Changing Source Lookup Path Settings

Source Lookup maps debugger source file paths to the actual file locations in the workspace and the host file system. The paths for source files are read by the debugger from the symbol data (also known as the debugger path) to the correct location of the executable being debugged.

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.

NOTE: Source Lookup Path settings are different from Object Path mappings.

Object Path mappings describe the relationship between the location of executables and symbol files on the target file system and their location on the host file system.

Source Lookup Path settings are the mapping of source file paths retrieved from the executable's symbol data to the correct location of the source files on the host file system, so the debugger can find the files in their new locations.

15.5.1 Selecting Source Lookup Containers

The Source Lookup Path consists of a list of Source Lookup Containers, each of which points to a location in the workspace or the file system where the source files can be found. When you search for a source file, each container in the list is searched consecutively until the source file is found.

The following source containers are supported:

- **Debugger Path** uses the original source path compiled into the program to find sources in the project system. If the file is not found in the project system, the debugger searches the local file system and opens the file from there.
- **Disassembly** opens in the Disassembly view files matching the given debugger path.
- **Filesystem Directory** recursively searches the given filesystem directory; see the *File System Files in Workbench* section of *Wind River Workbench User's Guide: Debugging Projects*.
- **Filesystem Directory Substitution** substitutes a file system folder for part of the debugger path (the original location of the sources when the program was compiled).
- **Project*** searches a project in your workspace.
- **Workspace*** searches all projects in your workspace.
- **Workspace Folder*** recursively searches the given folder in your workspace; all sub-folders are searched as well.
- **Workspace Folder Substitution** substitutes a workspace folder for part of the debugger path, and checks if the file with given path exists in the workspace.

Adding Source Lookup Settings

To add source lookup settings for a running process:

1. Right-click a launch configuration, a target, or a thread in the Debug view, then select **Edit Source Lookup**. The **Edit Source Lookup Path** dialog appears.
2. Click **Add**. The **Add Source** dialog appears.
3. Select the type of source to add to the source lookup path, complete any dialogs that required additional selections, then click **OK**.

NOTE: Selecting the **Disassembly**, **File System Directory Substitution**, and **Workspace Folder Substitution** options opens an additional dialog that lists matching compilation units.

By default, this list does not populate automatically because Workbench can take several minutes to update it when working with very large files.

- If your code base contains very large files, click **Update** when you want to populate this list manually.
- If this is not a problem for your sources, select **Update list automatically**.

4. The source lookup containers are searched in the order in which they appear in this dialog, so click **Up** or **Down** to adjust the order of entries in the list.
5. Check the **Search for duplicate source files on the path** to force the debugger to search for and display all files that match the given debugger path, rather than stopping as soon as it finds one.

More about File System Directory Substitution

As an example of how this substitution works, to create a substitution rule that takes the debugger path of

```
/home/john/project
```

and maps it to a host path of

```
c:\unix_directories\john\project
```

you would enter each path in the dialog, then click **OK**.

Thereafter, when the debugger looks for

```
/home/john/project/include/config/my_board.h
```

it will substitute

```
c:\unix_directories\john\project\include\config\my_board.h
```

15.5.2 Reverse Source Lookup

Just as the source containers are used to map the debugger path to the host location, they are also used in reverse, to map the host location to a debugger path in order to plant breakpoints.

For **Workspace Folder Substitution** and **Filesystem Folder Substitution**, if the host location matches the target folder, the folder/directory part of the host path

is removed, and the debugger path is substituted. For **Debugger Path**, the path is used as is. For all other containers, they are checked against the host location, and if they match, just the filename part of the host location is sent to the debugger.

NOTE: Not all locations contain enough information for the debugger to calculate a full debugger path, such as those noted with an asterisk (*) in *Selecting Source Lookup Containers*, p.205 (**project**, **workspace**, and **workspace folder**).

This lack of information can possibly lead to errors in those operations that require correct input paths, like planting breakpoints, so these source containers should be used with caution. Instead, use **Workspace Folder Substitution** and **Filesystem Directory Substitution** containers whenever possible.

Searching for Duplicate Source Files

Only the file name part of the debugger path is sent to the debugger when planting breakpoints. This means that if there are multiple files with the same name in different projects or folders, the breakpoint could be planted incorrectly.

To avoid this, select the **Search for duplicate source files on the path** checkbox. This forces the debugger to search for all files that match the given debugger path, and if duplicates are found, a dialog appears asking you to choose the correct file.

Browsing to Source

The **Browse to Source** wizard allows you to select the correct version of a file, and then it creates the appropriate source container for you. This is useful if the debugger opened the wrong version of a file and you want to create a source lookup rule so that mistake will not happen again.

The wizard opens automatically when you select **Browse to Source** from the **Missing Source Editor**, but you can open it manually by right-clicking on a stack frame in the Debug view and selecting **Browse to Source**.

Source Location

Debugger Path

The path to the file as seen by the debugger.

Host Path Type

Specify whether the source file is located in the **Workspace** or the **Filesystem**.

Host Path

- **Workspace**

  If you selected **Workspace**, the **Host path** field displays the project folder, and the field at the bottom of the wizard is called **Folder selection**.

  Select the correct version of the file, then click **Next**.

- **Filesystem**

  If you selected **Filesystem**, the **Host path** field is empty, and the field at the bottom of the wizard is called **Directory contents**. Type the path to the correct directory on the host, or click **Browse** and navigate to it.

  Select the file, then click **Next**.

Source Container to Create

Source container choices

The list of possible source containers that will locate the file you selected. There are advantages and disadvantages to each type of container (as discussed in the Note in *Reverse Source Lookup*, p.206).

Select the type of source container you would like to create, then click **Next**.

Review New Source Container List

All Containers

The list of all source containers, with asterisks next to those that will find the selected source file.

Select the source file to use for this file, then click **Finish**.

NOTE: You can review this list, and if necessary select a different source container, by right-clicking a stack frame in the Debug view and selecting **Properties > Source Lookup**.

Editing Source Lookup Settings

To edit the source lookup settings for a running process:

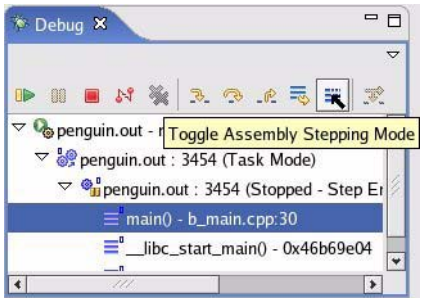
1. Right-click a launch configuration, a target, or a thread in the Debug view, then select **Edit Source Lookup**. The **Edit Source Lookup Path** dialog appears.
2. Select the source lookup container whose settings you want to modify, then click **Edit**. The appropriate source container dialog appears.
3. Adjust the settings as necessary, then click **OK**.
4. If necessary, click **Up** or **Down** to adjust the order of entries in the list.
5. Check the **Search for duplicate source files on the path** to force the debugger to search for and display all files that match the given debugger path, rather than stopping as soon as it finds one.

Workbench stores source lookup settings with other launch configuration data, so you can also access these settings through the **Source** tab of the launch (for more information about launch configurations, see *14. Launching Programs* or open the launch configuration dialog and press the help key for your host.

15.6 Stepping Through Assembly Code

To view assembly code interleaved with the corresponding source code:

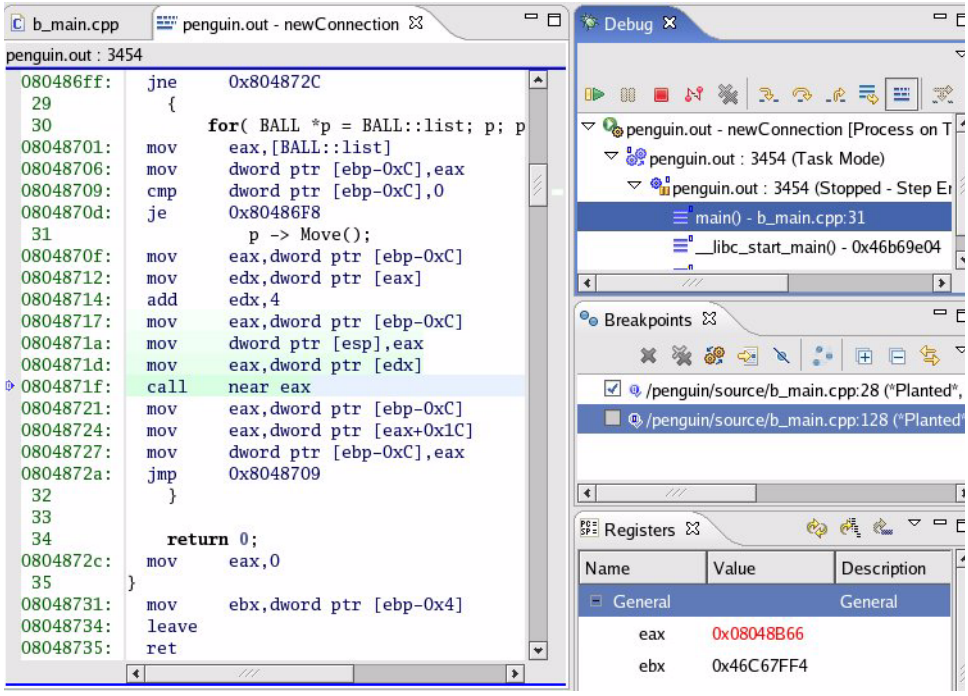
1. Click the **Toggle Assembly Stepping Mode** button as shown in the following example, which shows penguin in the Debug view:



2. Step or proceed to a breakpoint such as shown in Figure 15-7 as a mix of assembly and C++ source code.

   In this case, the code has been single-stepped to the **call** instruction. Note that the previous steps are shown in lessening degrees of shading—this helps you see where you have been when you step into and out of routines.

Figure 15-7 **Single Stepping Though Assembly Code**

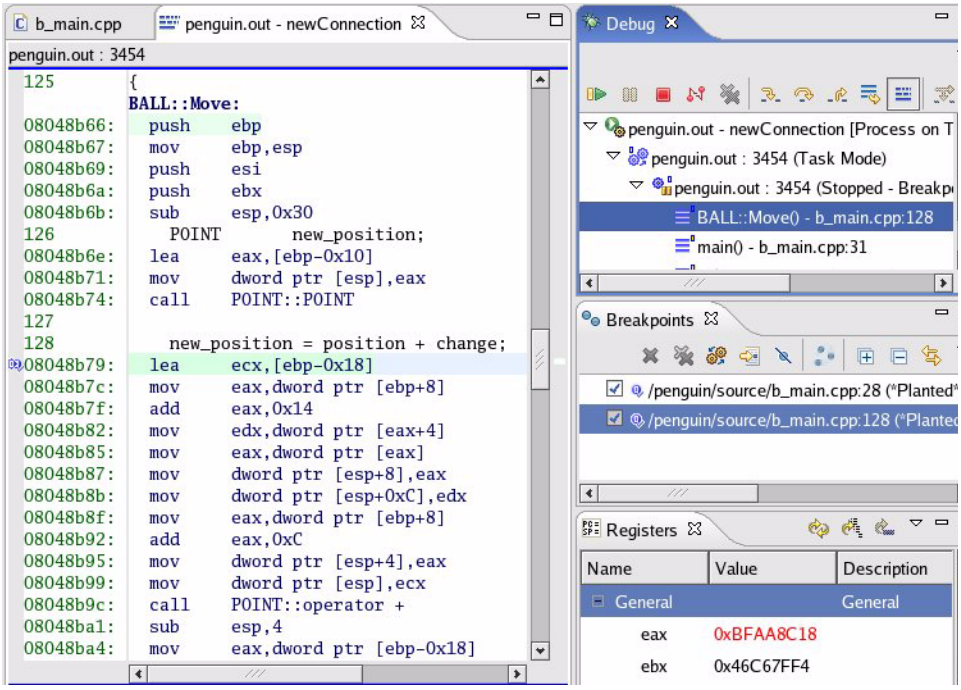


NOTE: The assembly code shown will differ depending on your target.

With the assembly language code visible, you can step in and out of the inherited parent class methods for C++ classes as your C++ code is executing.

For example, if you step into the call for the **p** object's **Move** method, you step into the **Move** method code for the parent class **BALL**, as shown in Figure 15-8.

Figure 15-8 **Example of C++ Assembly Code**



Also you can see, at line 126, the call to the parent class constructor **POINT::POINT**, which creates a new **POINT** object for the local variable **new_position**. Additionally, you can see the call (for line 128) to the parent class operator method **POINT::operator +**.

NOTE: This is all hidden when you debug just at the source code level.

15.7 Using the Disassembly View

Use the Disassembly view for the following purposes:

- 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).

15.7.1 Opening the Disassembly View

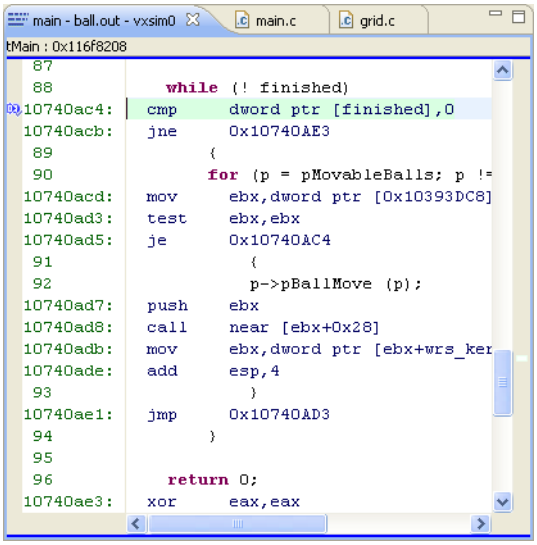
The Disassembly view appears automatically if the Debug view cannot display the appropriate source code file in the Editor (it appears as a tab in the Editor, labeled with the target connection being debugged).

You can open the Disassembly view manually by clicking the Debug view's **Toggle Disassembly/Instruction Step Mode** toolbar button, or by selecting **Window > Show View > Other > Debug > Disassembly**.

15.7.2 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 15-9 **Disassembly View**



If the Disassembly view displays a color band at the top and bottom (here, the band is blue), then it is pinned to 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.

15.8 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.

PART VI

Using Workbench in a Larger Environment

16 Integrating Plug-ins

16.1 Introduction

Because Wind River Workbench is based on Eclipse, you can incorporate other Eclipse-based modules into Workbench without having to recompile or reinstall it. These 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.

For plug-ins dependent on Java Development Tools (JDT) , see *16.6 Installing JDT for Third-Party Plug-ins and Debugging*, p.222 for help downloading the JDT.

16.2 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://www.sourceforge.net/**

16.3 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.

16.3.1 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**.

16.3.2 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. To install it:

1. Follow steps 1 and 2 in *16.3.1 Creating a Plug-in Directory Structure*, p.218.

   For the purposes of this example, name the top-level directory **eclipseplugins**, and name the plug-in directory **clearcaseIBM**.

2. Navigate to **http://www.ibm.com/developerworks/rational/downloads/** and select the **Plug-ins** tab, then select **ClearCase plugins**. The IBM Rational ClearCase plug-ins page opens.

3. Click the **Get the downloads** link, then click the **HTTP** link to the right of the appropriate version of the package file.

   - Adapter V 7.0.0.x for Eclipse 3.3: Linux/Solaris
   - Adapter V 7.0.0.x for Eclipse 3.3: Windows

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 the ClearCase tools directory is in your path:

   - **/usr/atria/bin** for Linux or Solaris
   - **C:\atria\ClearCase\bin** for Windows (this may vary)

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**.

Activating the IBM Rational Plug-in

When Workbench restarts, activate the plug-in:

1. Select **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.

After you have activated the plug-in:

- Select **Window > Preferences > Team > ClearCase SCM Adapter** to configure it.
- Press the help key for your host, select **All Topics**, then see the **Rational ClearCase SCM Adapter** documentation for more information about using it.

For more information about ClearCase functionality, refer to your ClearCase product documentation.

16.4 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, right-click the plug-in you want to disable, then select **Disable**.
3. Workbench will ask if you want to restart. To properly disable the plug-in's functionality, click **Yes**.

16.5 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 *16.4 Disabling Plug-in Functionality*, p.221, 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-3.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, press the help key for your host, select **All Topics**, then **Wind River Partners Documentation > Eclipse Workbench User Guide > Tasks > Running Eclipse**.

16.6 Installing JDT for Third-Party Plug-ins and Debugging

Previous versions of Workbench included a modified version of the Java Development Tools (JDT), but this has been removed for better integration with Eclipse and certain third party packages.

However, some third party plug-ins have a dependency on JDT.

To use the Workbench Update Manager to install the JDT:

1. Open the Update Manager by selecting **Help > Software Updates > Find and Install**.

2. Select **Search for new features to install**, then click **Next**.
3. Select **The Eclipse Project Updates**, check **Automatically select mirrors**, and click **Finish**.

   The Search Results dialog opens.
4. Check **Show the latest version of a feature only**.
5. Expand **The Eclipse Project Updates**, then **Eclipse SDK 3.3.1**.
6. Check **Eclipse Java Development Tools 3.3.1.r331** and click **Next**.
7. Accept the JDT license agreement, then click **Next**.
8. Click **Finish**.

   The Feature Verification dialog eventually opens.
9. Click **Install All**.
10. Click **Yes** to restart Workbench.

You can also use the Update Manager to search for new features and other updates.

17

Using Workbench in an Eclipse Environment

17.1 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.

17.2 Recommended Software Versions and Limitations

Java Runtime Version

Wind River tests, supports, and recommends using the JRE 1.5.0_11 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 3.1 is based on Eclipse 3.4. Wind River patches Eclipse to fix some Eclipse debugger bugs. These fixes will be lost when using a standard Eclipse environment.

For supported and recommended host requirements for Workbench 3.1, see the Getting Started for your VxWorks or Linux Platform product.

Defaults and Branding

Eclipse uses different default preferences from those set by Workbench. The dialog described in *17.3 Setting Up Workbench*, p.226 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.

17.3 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 > Install into Eclipse**. The Install into Eclipse dialog appears.

2. In the **Directory** field, type in or **Browse** to your Eclipse 3.4 directory.
3. In the **Installation 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 log 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.

17.4 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 installing 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.

17.4.1 Workflow in the Project Explorer

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 Explorer 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 Explorer 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)

Source Analysis

Source analysis (rebuild, update, and freshen) is available from the **Index** entry on the context menu of the Project Explorer.

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.

17.4.2 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.

17.4.3 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**.

17.4.4 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**.

18 Using Workbench with Version Control

18.1 Introduction

This chapter provides tips on using Workbench with version-controlled files, which Workbench project description files you should add to version control when archiving your projects, and how to manage build output when your sources are version controlled.

18.2 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** check box 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.

18.2.1 Specifying the Path to an External JVM

To launch Workbench from within a ClearCase view, you must specify a path to a JVM outside of ClearCase using the **-vm** *javapath* option.

On Windows

If you launch Workbench from the Start menu, select **All Programs > Wind River > Workbench 3.x**, then right-click **Workbench 3.x** and select **Properties**. Add **-vm** *javapath* to the end of the path shown in the **Target** field.

If you created a desktop icon for Workbench during installation, follow the same process. Right-click the icon and select **Properties**, then add **-vm** *javapath* to the **Target** field.

On Linux and Solaris

Append **-vm** *javapath* to the Workbench launch command:

```
% startWorkbench.sh -vm /usr/bin/java
```

18.2.2 Adding Workbench Project Files to Version Control

To add Workbench project description files to version control without putting your workspace into a ClearCase view, check-in the following automatically generated files along with your source files. They should be checked out and made writeable during a build:

Project File	Description
.cproject	CDT project file containing CDT-specific information about the project.
.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 (for example, for VxWorks Image projects, some Platform projects, or BSP validation test suite support).
Makefile	Should be version controlled *only* for user-defined projects.
***.wpj**	VxWorks Image project file containing specific data not managed directly by Workbench but by the TCL engine.
usrAppInit.c	A stub where you can add DKM application initialization routines.
usrRtpAppInit.c	A stub where you can add RTP application initialization routines.

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. **Files That Should Not Be Version Controlled**

Some files are regenerated each time you build your project, so they should not be version controlled:

Project File	Description
linkSyms.c	A configuration file 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.
Makefile	For managed build projects, this file is regenerated each time the project is built, so do not add custom code to this file (add it to **.wrmakefile** instead).
Makefile.mk	This file is called from the project's **Makefile**. It includes a list of components and build parameters, and connects the Workbench project to the VxWorks build system.
prjComps.h	A configuration file that contains the preprocessor definitions (macros) used to include VxWorks components.
prjConfig.c	A configuration file that contains initialization code for components included in the current configuration of VxWorks.
prjParams.h	A configuration file that contains component parameters.
default*	Directories that contain build output files.
.metadata	Directory that contains mostly user- and workspace-specific information with absolute paths in it.

18.2.3 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 Explorer, 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.

18.3 Using Workbench with CVS

Unlike ClearCase, adding projects to version control using CVS does not impact performance, so you can create your project in your workspace if you wish.

The *Eclipse Workbench User Guide*, available from the Workbench help system, provides information about using Workbench with CVS. You can access this information in two ways from the Workbench Help view.

From the Help Table of Contents

This method is useful if you want to read an entire section of the document.

1. Open the Help view by pressing the help key for your host.
2. At the bottom of the view, click **All Topics**, then navigate to **Wind River Partners Documentation > Eclipse Workbench User Guide**.
3. Sections related to using CVS include:

   Getting Started > Team CVS tutorial

   Concepts > Team programming with CVS

   Tasks > Working in the team environment with CVS

Reference > Team support with CVS

Using the Help View's Search Feature

This method is useful if you want to find a specific piece of information.

1. Open the Help view by pressing the help key for your host.
2. At the bottom of the view, click **Search**, then type in **CVS**.
3. Several links, along with a small amount of text from each section, appears for you to choose from.

19

Using Workbench in a Team Environment

19.1 Introduction

If you are working as part of a development team, this chapter will explain some of the things you and your colleagues can do to ensure that your environments and source files stay in sync.

19.2 Sharing Workbench Settings With Your Team

You can import, export, and share several types of Workbench settings.

19.2.1 Sharing Workbench Preferences

Each time you or your team members customize the preference settings in **Window > Preferences**, you drift farther from a synchronized team environment. Some settings, such as the color used to display text in a particular editor, may not impact the team's effectiveness very much.

However, if you change a setting like **Insert spaces for tabs**, it could cause the code you write to differ significantly from the team's default style.

Workbench makes it easy to export your preferences, which can then be imported by your team.

1. After updating and saving your preferences, select **File > Export**.
2. From the Export dialog, select **General > Preferences**, then click **Next**.
3. Decide whether you want to export all your preference settings, or choose specific preferences to export.
4. Type the path or click **Browse** and navigate to the location where the preferences file should be stored. If this is a new file, type in a file name (the file extension of **.epf** is already filled in) and click **Save**, or if you want to overwrite an existing preferences file, select it and click **Save**.

19.3 Multiple Users and Installations of Workbench

Different configurations are possible for the number of Workbench users and the number of Workbench installations on a single host. The following considerations apply.

NOTE: Each user should work in a user-specific workspace. Sharing of workspaces is not possible. To specify a workspace, either use the **-data** startup option, select a workspace in the **Workspace selection** dialog during startup, or select **File > Switch Workspace** in Workbench.

Single User with Single Installation

This configuration presents no special installation or use considerations, assuming the users on each host perform their own installation and have standard access permissions to the installation location. No special startup arguments are necessary and the default workspace may be used.

Multiple Users with Multiple Installations

The same conditions apply here as they do for a single user with a single installation, with the exception that a single administrative user often performs the different Workbench installations. In this case it is important that the proper permissions be granted to each user for access to his or her particular installation.

Multiple Users with a Single Installation

Multiple users can share a single Workbench installation as long as the access rights allow them to read all files of the installation (same group as the user who installed Workbench).

For performance reasons, it is desirable to have the workspace on a local file system. Some Eclipse-specific data is stored in the user's home directory by default. If this is not desired because of slow network access, use the **-configuration** startup option to redirect this data. See *Eclipse Workbench User Guide: Running Eclipse* in the Workbench online help for more information on startup options.

Single User with Multiple Installations

If a single user is installing Workbench more than once, it is important that the configuration area not become corrupted. Different versions of Workbench will not conflict, but for multiple installations of the same version for the same user, different configuration areas should be specified at startup with the **-configuration** option.

Eclipse Team Features

Workbench supports all the team features of the standard Eclipse installation as documented in the online help supplied with Workbench.

PART VII

Reference

A Troubleshooting

A.1 Introduction

This appendix 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.

If you are experiencing a problem with Workbench that is not covered in this appendix, please see the Wind River Workbench Release Notes for your platform.

A.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 > Workbench 3.1 > Workbench 3.1**, 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 box, or does not start at all, start it from a terminal window, specifying a new workspace name:

   ```
   $ installDir\workbench-3.1\wrwb\platform\eclipse\wrwb-x86-win32.exe -data
   newWorkspace
   ```

   On Linux or Solaris

   Start Workbench from a terminal window, specifying a new workspace name:

   ```
   $ ./startWorkbench.sh -data newWorkspace
   ```

   NOTE: For details on Workbench startup options, press the help key for your host, select **All Topics**, then see **Wind River Partners Documentation > Eclipse Workbench User Guide > Tasks > Running Eclipse**.

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-3.1 Directory is Corrupted

1. To test if your *homeDir*/**.workbench-3.1** directory is the source of the problem, rename it to a different name, then restart Workbench.

NOTE: Make sure you rename the *homeDir*/**.workbench-3.***x* directory (for example, on Windows XP it could be **C:\Documents and Settings***username***.workbench-3.1**).

Do not rename the *installDir*/**workbench-3.1** directory.

2. If Workbench starts successfully, exit Workbench, then delete the old version of your *homeDir*/**.workbench-3.1** directory (the one you renamed).
3. Restart Workbench. The *homeDir*/**.workbench-3.1** will be recreated and should work correctly.
4. Because the **.workbench-3.1** directory contains Eclipse configuration information, any information about manually configured Eclipse extensions or plug-ins will be lost when you delete the directory.

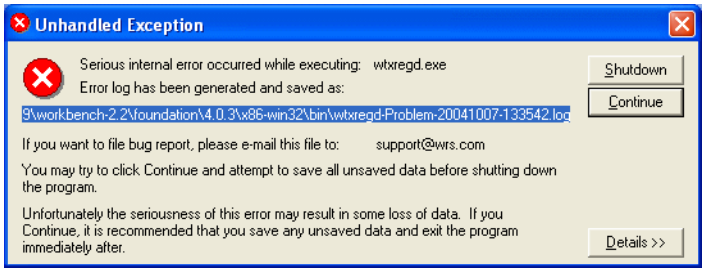
   To make them available again within Workbench, re-register them (**Help > Software Updates > Manage Configuration**).

Registry Unreachable (Windows)

If Workbench starts and it does not detect a running 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 ever need to kill the registry.

If you do stop it, however, it stores its internal database in the file *installDir*/**.wind/wtxregd.***hostname*.

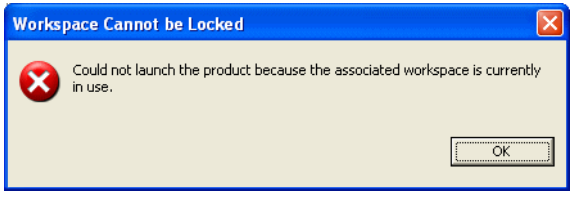
If this file later becomes unwritable, the registry cannot start, and Workbench will display an error.



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 **-configuration** option to start Workbench at a terminal prompt 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
   ```

NOTE: For details on Workbench startup options, press the help key for your host, select **All Topics**, then see **Wind River Partners Documentation > Eclipse Workbench User Guide > Tasks > Running Eclipse**.

3. On some window managers (e.g. gnome) you can close the Workbench 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.

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.

A.3 General Problems

This section describes problems that are not associated with any particular Workbench component.

A.3.1 Help System Does Not Display on Solaris or Linux

Workbench 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 install it and 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. Select **Use external browser**.
3. Click the **Web Browser** link to configure your browser launch program.
4. Select **Use external Web browser**, then select one of the listed browsers or click **New** to add one.
5. In the **Add External Web Browser** dialog, type in the browser's name, then type in the path or click **Browse** and navigate to your browser's executable.
6. In the Parameters field, type in any parameters required by the browser's launch command.
   - 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 browser launch commands are **"/usr/bin/firefox" %1**, **"kfmclient openURL %1"**, and **"/opt/mozilla/mozilla %1"**. Enter the command line as appropriate for your browser.
7. Click OK twice to return to Workbench.
8. To access the context-sensitive help for a particular view or dialog, click the view or open the dialog, then press **Ctrl+F1**.

NOTE: The **Help** button on Solaris keyboards does not open Workbench help due to a problem in Solaris/GTK+. Instead, use **Ctrl+F1** to access help.

A.3.2 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**.

A.3.3 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
   ```

A.3.4 Resetting Workbench to its Default Settings

If Workbench crashes, some of your settings could get corrupted, preventing Workbench from restarting properly. To reset all your settings to their defaults, delete your *homeDir*/**.workbench-3.1** directory which will be recreated when Workbench restarts.

⚠ **CAUTION:** Remove the directory **.workbench-3.1** (begins with a period) in your home directory, not the directory **workbench-3.1** in the Workbench installation directory.

A.4 Fixing Indexer Issues

If you encounter problems when using source navigation tools with certain files (such as missing or incorrect information in the Include Browser, Type Hierarchy, Call Tree, or Outline views) these files may not have been parsed, or were parsed with wrong include paths or symbols. This can also cause problems when using Workbench code-completion features.

This section describes various solutions to indexer and source analysis issues.

A.4.1 Indexing Problems with Managed Projects

If the files are not part of the current build-target, or all build targets in the project are empty:

1. Add the files to the build-target. This changes the build, so you need to update the index afterwards.
2. Activate the **Index all files** option of the indexer:
   a. Open the project's Properties dialog by right-clicking the project, then selecting **Properties**.
   b. Select **C/C++ General > Indexer**.
   c. Select **Enable project specific settings**, then select **Index all files** to cause all files of the project to be parsed.

A.4.2 Indexing Problems with User-defined Projects

The following problems might show up specifically in user-defined projects.

Source Files Not Yet Built

If the source files have not yet been built:

1. Build all source files of your project and then update the index.
2. Disable build-output analysis:
   a. Open the project's Properties dialog by right-clicking the project, then selecting **Properties**.
   b. Select **C/C++ General > Paths and Symbols**, then click the **Discovery** tab to bring it to the foreground.
   c. Clear the **Enable analysis of build output** check box, then click **Clear**. This disables build-driven indexer setup and causes all files of the project to be parsed, except standalone header files. You need to update the index afterwards.
3. Activate the **Index all files** option of the indexer:
   a. Open the project's Properties dialog by right-clicking the project, then selecting **Properties**.

b. Select **C/C++ General > Indexer**.

c. Select **Enable project specific settings**, then select **Index all files** to cause all files of the project to be parsed.

Unsuccessful Build Output Analysis

If build-output analysis has not been successful (for example, because of unsupported make rules):

1. Make sure that the **--no-print-directory** make option is not set.
2. Disable build-output analysis:
   a. Open the project's Properties dialog by right-clicking the project, then selecting **Properties**.
   b. Select **C/C++ General > Paths and Symbols**, then click the **Discovery** tab to bring it to the foreground.
   c. Clear the **Enable analysis of build output** check box, then click **Clear**. This disables build-driven indexer setup and causes all files of the project to be parsed, except standalone header files. You need to update the index afterwards.
3. Activate the **Index all files** option of the indexer:
   a. Open the project's Properties dialog by right-clicking the project, then selecting **Properties**.
   b. Select **C/C++ General > Indexer**.
   c. Select **Enable project specific settings**, then select **Index all files** to cause all files of the project to be parsed. Note that this increases the memory consumption of Workbench.

A.4.3 Other Indexing Problems

The problems in this section may appear regardless of whether the project is managed or user-defined.

If the files have been excluded on the **Sources / Filters** tab on the **C/C++ General > Paths and Symbols** project property page:

1. Open the project's Properties dialog by right-clicking the project, then selecting **Properties**.

2. Select **C/C++ General > Paths and Symbols**, then click the **Sources / Filters** tab to bring it to the foreground.
3. Expand any source folder and check the filters below it.
4. Click **Edit filter** data to change exclusion filters.

Note that filtered paths are not parsed, and that you need to upate the index after making these changes.

Outdated Index

If the index is not up to date:

1. Right-click the project in the Project Explorer and select **Index > Update with Modified Files** in order to parse modified and newly added files.
2. Right-click the project again and select **Index > Rebuild** in order to clear the index and reparse all files in the current build scope (for build-driven setup) or all files in the project (if there is no build target, or the **Index all files** option is enabled for a project, or build output analysis is disabled or did not return any results).

NOTE: Changing include paths or symbols, for example by using the **Build Properties** page or the **Paths and Symbols** property page, only has an immediate affect on parsing modified and newly added files. Rebuild the index manually if you need accurate source navigation.

Incorrect Include Paths and Symbols

If the include paths and symbols are not set correctly:

- For managed projects, build properties are passed to the indexer. If the project successfully builds, the include paths and symbols are most likely correct. Let Workbench detect include paths to resolve include directives , or specify additional paths manually in the build properties.
- For user-defined projects, the indexer is set up automatically with include paths found through inspection of source-code right after project creation. To get better results, you can perform a full build with enabled build-output analysis (right-click the project in the Project Explorer, then select **Properties,** then **C/C++ General > Paths and Symbols**, and switch to the **Discovery** tab).

Trouble Parsing Source Code

If the indexer has trouble parsing the source-code:

1. Open **Window > Preferences** and go to section **C/C++ > Language Mappings**.
2. Change the language mappings for **C Header File** and **C Source File** to Wind River Compiler C++ Dialect.

If the problem still appears, send a report to Wind River, including log files: select **Help > Collect Log Files**. If possible, include a small project that shows the problem.

A.5 Optimizing Workbench Performance

Many Workbench projects are very large, and can present a challenge to Workbench performance.

This section explains how to optimize Workbench for large projects.

Module Optimization Levels and Jumping Program Counters

Module Optimization Levels and Jumping Program Counters

Normally, the kernel is built with various optimization flags set. If you step through kernel code, you will see the program counter marker jump around as you step through the optimized and re-ordered instructions. This is normal, and proves that optimization was applied.

This also occurs for kernel modules if you compile without overriding the optimization level, becaus,e by default, all kernel modules built against a kernel will inherit its optimization. You can be overcome this by changing a build command field before you build, so that the build does not try to optimize code.

Right-click the Kernel Module Project (Linux) or the DKM (VxWorks) , and select **Properties > Build Properties**

In Linux, change the Build command to:

```
make COPTIMIZE="-O0"
```

NOTE: The end of the command is a dash, a capital letter **O**, and a zero.

In VxWorks, change the Build command to:

```
make COPTIMIZE="-O0"
```

When you next build the module, it will have minimum optimization.

Module Optimization Levels and Skipped Breakpoints

Sometimes you may find behavior like a step-next that does not break on the next step. In this example, the compiler may have created two in-line copies when optimizing the code, where the breakpoint was set in the first copy, but the code may branch into the second copy. This is a limitation of some ELF formats, and a resulting optimization and context awareness issue for the debugger.

You can try switching to mixed assembly and source stepping, where you step the actual assembly instructions, but the best choice is to remove the optimization for the module and retest.

Manual Refresh of the Build Linked Directory in Workbench

To help the workflow, Workbench has disabled the automatic project refresh after builds so that the user can immediately choose the next task. As a result, you need to manually refresh the **build** directory whenever you wish to see any new or revised file in it (this refreshes any stale file handles in the Project Explorer).

If you need to browse the **build** directory, use Source Analysis (in Linux, you may also use the Quilt patch mechanism), right-click the project and select:

- **Add link to build folder** (Linux)
- TBD (VxWorks)

Disabling Workspace Refresh on Startup

You may choose to disable the automatic workspace refresh when starting Workbench, particularly when you have one or more platform or DKM projects.

To do this, select **Window > Preferences > General > Startup and Shutdown**, and de-select **Refresh workspace on startup**.

When you need to browse the build tree, you can always manually refresh that directory by right-clicking on the project (or specifically the **build** entry) and selecting **Refresh**.

Workbench Freezes: No Response to Mouse Clicks

Sometimes Workbench will not immediately refresh its screen or respond to mouse or keyboard actions. This is often due to a garbage collection timeout, or running low on real and virtual memory.

To track this behavior, enable Workbench heap status monitor, by selecting **Windows > Preferences > General > Show Heap Status**. The heap monitor will appear on the lower right-hand corner.

A.6 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.267.

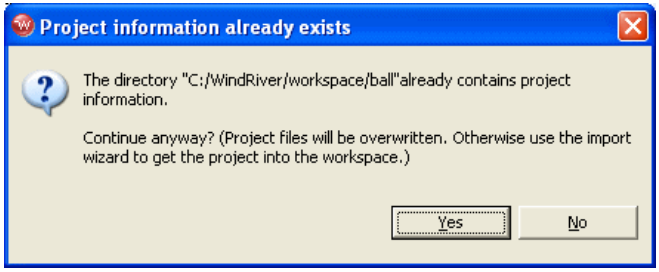
This section explains error messages that appear in each Workbench component.

A.6.1 Project System Errors

For general information about the Project System, see *3. Projects Overview*.

Project Already Exists

If you deleted a project from the Project Explorer 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 the following 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 Workbench, click **No**, then right-click in the Project Explorer, 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 Explorer.

Cannot Create Project Description 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, then create a folder that links to the location of your source files using one of the following options:

Create Project Files in the Workspace Rather than in Source Location

1. Create a project in your workspace by selecting **File > New >** *project type*.
2. Type in a name for your project, select **Create project in workspace with content at external location**.

3. Type the path or click **Browse** and navigate to your source root directory, then click **OK**.
4. Click **Next** and adjust any settings as necessary on the next few screens.
5. Click **Finish** to create your project.
6. Click the plus next to the folder to open it, and you will see the source files from your read-only source directory.

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 Explorer, press **F5** to refresh the display. The directory and all sources in it appear.

NOTE: This mechanism can be used only for user-defined projects.

A.6.2 Build System Errors

For general information about the Build System, see *8. Building Projects*.

A.6.3 Building Projects While Connected to a Target

If you try to build a project while you have a target connection active in the Remote Systems view, 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 > Wind River > 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

The best workflow for cases where you continually need to rebuild objects that are in use by your target is as follows:

- 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 specify that your project should be rebuilt before it is launched by selecting **Window > Preferences > Run/Debug > Launching**, and then selecting **Build (if required) before launching**. For more information about launch configurations, see *14. 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 and Download**-type launch configuration for your application, and enable the **Build before launch** option (by selecting **Window > Preferences > Run/Debug > Launching**, then selecting **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 Remote Systems view and pressing the **Delete** key (you can multi-select if there are multiple modules).
4. Press the **Debug** button on the Workbench toolbar (it looks like a green bug) to relaunch your application. It will automatically rebuild, redownload, reset, and attach the debugger.

Problems Building Workbench 2.x Projects Imported Into Workbench 3.1

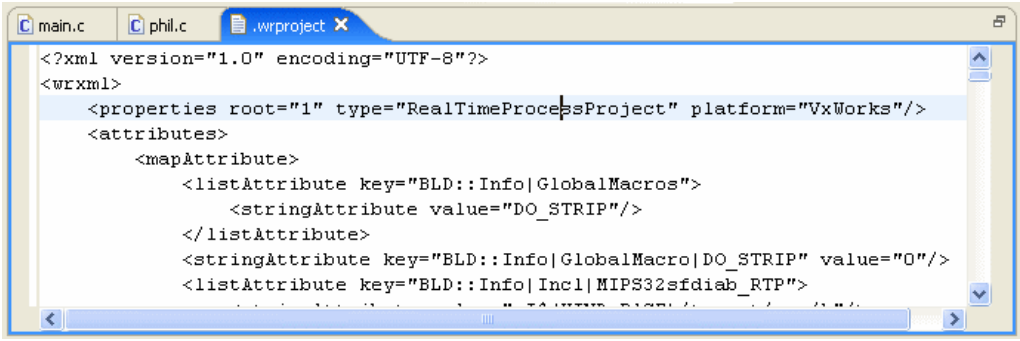
If you have trouble building projects that you imported from a previous version of Workbench, check if the **.wrproject** file contains an entry for **platform**. If not, the project is not compatible and has to be patched to work with the newest version of Workbench.

To patch the **.wrproject** file:

1. Open the file with the Workbench text editor by right-clicking the file in the Project Explorer, then selecting **Open With > Text Editor**.
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 also 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.

A.6.4 Remote Systems View Errors

For information about the Remote Systems view, see *10. Connecting to Targets*.

Troubleshooting Connecting to a Target

If you see an error similar to "Error connecting to *target*: failed to connect debugger. Failed to create target control" 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 Remote Systems view 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 press **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 Remote Systems view 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 track down what is wrong with the target.

RPC Timeout Errors

If you get an **RPC timeout** error when connecting to a target, it means that either the agent or the target server is not responsive. There can be many reasons for this:

- The agent died, or is busy.
- The target crashed.
- The target server is doing something heavy (like a load operation).
- The host's CPU is loaded with another process.

To work around this problem, try changing the default timeouts:

1. Change the **WTX client timeout** by selecting **Window > Preferences**, then selecting **Wind River > Target Management** and adding a few seconds to the fields under **Communication timeouts**.
2. Change the **Backend request timeout** by right-clicking your target connection in the Remote Systems view, then selecting **Properties**.
   a. On the **Target Server Options** tab, click **Edit** beside the **Options** field.
   b. On the Common tab, add a few seconds to the **Backend request timeout**.

If adjusting the timeouts does not help, you can use the WTX and WDB log files, or the target server output, to track down the problem. For more information about collecting log files, see *A.8 Error Logs Generated by Workbench*, p.266.

Exception on Attach Errors

If you try to run a task or launch an RTP and the Remote Systems view 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 an error similar to "Failed to launch Kernel Task *name* on *connection@host*: Symbol not found" if you try to run a kernel task without first downloading it to your target.

Processes can be run directly from the Project Explorer, 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 an error similar to "WTX Loader Error: Object module not appropriate".

To select the correct build spec, right-click the output file in the Project Explorer, 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 an error similar to "Failed to launch RTP *name* on *connection@host*. Create failed. Target name reported: S_rtpLib_INVALID_FILE".

Unlike with kernel modules, RTP executable files are accessed and loaded from the target, not from the host running Workbench, so if the target-side path looks correct but you still get this error, 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; correctly specifying the path may involve including the proper device name in front of the path. For example:

```
$ host:d:/my.vxe
```

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 does not contain the equivalent host-side path instead).
   a. Right-click the process executable in the Project Explorer or right-click **Processes** or **Real-time Processes** in the Remote Systems view and select **Run Real-time Process.**
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.
   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.
5. From a target shell or Linux console, try to launch the RTP or process.
6. Verify that the **vxWorks** or kernel node in the Remote Systems view has a small **S** added to the icon, indicating that symbols have been loaded for the kernel.

   If not, verify that the last line of the **Object Path Mappings** table displays a target path of **<any>** corresponding to a host path of **<leave path unchanged>**.

A.6.5 Launch Configuration Errors

If a launch configuration is not working properly, delete it by selecting it in the Launch Configurations dialog, then clicking **Delete** (the red **X**) above the configurations pane.

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.

NOTE: 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**.

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 *14. Launching Programs*.

A.6.6 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**.

A.6.7 Source Analysis Errors

If at any point Workbench is unable to open the cross reference database, you will see an error similar to “Failed to open database containing cross references”.

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** – Perform the same operation again, possibly with the same failure.
- **Recover** – Open the database and attempt a repair operation. This may take some time but you may recover your cross reference data.
- **Clear Database** – Delete the database and create a new one. All cross reference data is lost and your workspace will be reparsed the next time you open the call hierarchy.

- **Close** – Close the database. 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.

A.7 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.

A.8 Error Logs Generated by Workbench

Workbench has the ability to generate a variety of useful log files. Some 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.

NOTE: To discontinue logging for those logs that are not always enabled, clear the boxes on the Target Server Options tab, or restart Workbench without the additional options.

A.8.1 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 box 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), select any projects whose build log and config.log you want to include in the ZIP, then click **Finish**.
3. The ZIP file is created in the specified location, and contains all information collected to that point.
4. To discontinue logging (for those logs that are not always enabled), restart Workbench without the additional options, or follow these steps:
   a. Right-click the connection in the Remote Systems view, then select **Properties**.
   b. On the Target Server Options tab, click **Edit** in the **Advanced Target Server Options** section.
   c. On the Logging tab, uncheck the boxes of the logs you want to disable.

A.8.2 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 alerts
- bugs in Workbench Java code
- bugs involving inter-component intercomponent communication

Supported?

Yes.

A.8.3 DFW GDB/MI Log

The DFW GDB/MI log is 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 Remote Systems, debugger, and OCD views.

How to Enable Log

This log is always enabled.

What is Logged

All commands sent between Workbench and the debugger back end.

What it Can Help Troubleshoot

Debugger and Remote Systems view-related bugs.

Supported?

No. You may send this log to Wind River support, but no instructions are provided for how to interpret the information contained in it.

A.8.4 DFW Debug Tracing Log

How to Enable Log

This log is always enabled.

To change the maximum debug server log file size, select **Window > Preferences > Target Management > 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, Remote Systems, and debugger back end-related bugs.

Supported?

No. You may send this log to Wind River support, but no instructions are provided for how to interpret the information contained in it.

A.8.5 Debugger Views GDB/MI 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

Same as *DFW GDB/MI Log*, p.268, except with Workbench time-stamps.

What it Can Help Troubleshoot

Debugger and Remote Systems view-related bugs.

Supported?

No. You may send this log to Wind River support, but no instructions are provided for how to interpret the information contained in it.

A.8.6 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.

Supported?

No. You may send this log to Wind River support, but no instructions are provided for how to interpret the information contained in it.

A.8.7 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.

Supported?

No. You may send this log to Wind River support, but no instructions are provided for how to interpret the information contained in it.

A.8.8 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 Remote Systems view 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, press the help key for your host, click **Search**, and type **tgtsvr** into the **Search expression** field.

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

Supported?

No. You may send this log to Wind River support, but no instructions are provided for how to interpret the information contained in it.

A.8.9 Target Server Back End Log

This log records all requests sent to the WDB agent.

How to Enable Log

- Enable this log from the Remote Systems view 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, press the help key for your host, click **Search**, and type **tgtsvr** into the **Search expression** field.

What is Logged

Each WDB request sent to the agent. For more information about WDB services, press the help key for your host, click **Wind River Documentation > References > Host Tools > Wind River WDB Protocol API Reference**.

What it Can Help Troubleshoot

- debugger back end
- target Server
- target agent

Supported?

No. You may send this log to Wind River support, but no instructions are provided for how to interpret the information contained in it.

A.8.10 Target Server WTX Log

This log records all requests sent to the target server.

How to Enable Log

- Enable this log from the Remote Systems view 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, press the help key for your host, click **Search**, and type **tgtsvr** into the **Search expression** field.

What is Logged

Each WTX request sent to the target server. For more information about WTX services, see **Wind River Documentation > References > HostTools > WTX C Library Reference > wtxMsg.**

What it Can Help Troubleshoot

- debugger back end

- target server
- target agent

Supported?

No. You may send this log to Wind River support, but no instructions are provided for how to interpret the information contained in it.

A.8.11 Remote Systems Debug Tracing Log

This log prints useful information about creation and modification of Remote Systems view internal structures, as well as inconsistencies or warning conditions in the subsystems the Remote Systems view 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

Remote Systems view internal debug errors.

What it Can Help Troubleshoot

Inconsistencies in the debugger back end.

A.9 Technical Support

If you have questions or problems with Workbench or with your target operating system 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 Customer Support organization is also listed in the release notes. Your comments and suggestions are welcome.

B

Command-line Updating of Workspaces

The Workbench installation includes a **wrws_update** script that allows you to update workspaces from the command-line, for example to update workspaces with a nightly build script. The following section provides a reference page for the command.

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 (source 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
```

NOTE: The workspace must be closed for the command to execute. This includes closing all instances of the Workbench GUI that are accessing this workspace.

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 items in the list with commas (**,**). For example:

```
$ cobble,helloWorld
```

If the build target of a managed build project depends on files, folders, or sub-targets from other projects in the workspace, they must be included in the list of projects.

For example, a project named **ManagedBuildProj** references build targets from a subproject, **DependProj1**, and source files from another project, **DependProj2** (managed build only). Therefore, they must be included in the list of projects, as shown here on Windows:

```
$ wrws_update -data C:\build -l DependProj1,DependProj2,ManagedBuildProj -m
```

Build Options

-b, **--build-projects** *argument*
Launch build for projects. Several strings are valid as arguments, including: **build** (default), **clean**, and **rebuild**. All open projects in the workspace are built in the correct build order. It is not required to specify a list of projects using the **-l** option.

-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 items in the list with commas (**,**). For example:

```
$ -i,MY_VAR=value
```

Launch Options

-r, **--run-config** *argument*
Run the launch configuration named *argument* previously created using the Workbench GUI. Execution will always be in run mode. If the launch configuration name has spaces within the name, you must enclose the name in double quotes and escape the enclosing quotes, for example:

```
$ -r \"launch config name with spaces\"
```

Nightly Update Options

-i, **--update-index**
Update search-database index.

-m, **--generate-makefiles**
Regenerate makefiles where necessary.

-s, **--update-symbols** *argument*
Update symbol database (source 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 (source analysis). To create the data from scratch, 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) must be done before this script starts.

C

Command-line Importing of Projects

C.1 Introduction

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.

C.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. If this option is not given, the default workspace is used.

Import Project Options

-f, **--files** *argument*
Specify a list of project files to be imported. Separate the items in the list with commas (**,**). All files must be specified using an absolute path. For example:

```
$ dir1/.project, dir2/.project
```

-r, **--recurse-directory** *argument*
Specify a directory to recursively search for projects to be imported. All files must be specified using an absolute path.

-v, **--define-variables** *argument*
Specify a list of Eclipse path variables to be defined. Separate the entries in the list with commas (**,**). For example:

```
$ pathvar1=path1,pathvar2=path2
```

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.

D

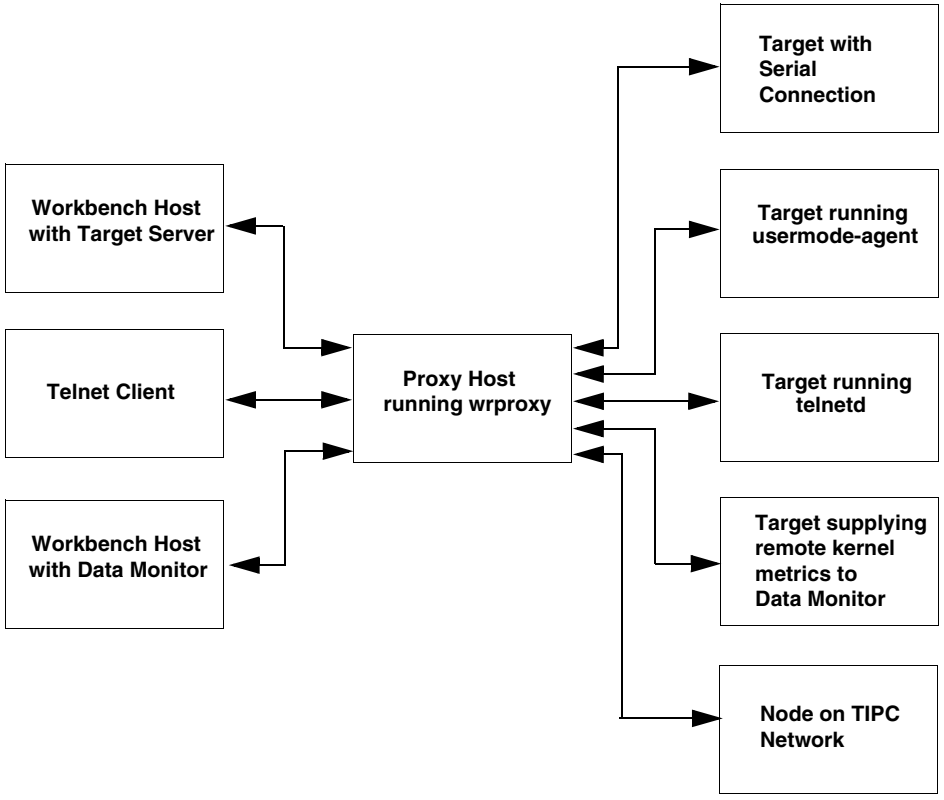
Configuring a Wind River Proxy Host

D.1 Introduction

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 D-1.

Figure D-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.

D.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) access on a proxy port to a particular target. The following example configures access 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.289 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 each time, it can be useful to use a startup script to configure it rather than doing it manually. 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.287.

D.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 always returns the value for **pport** but more information could 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 except for the **connect** command you can use all of them 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.289), 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*

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.287) 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 is:

- **udpsock** – UDP socket connection
- **tcpsock** – TCP socket connection (only **tcpsock** is allowed for a VxWorks proxy host.)
- **tipcsock** – TIPC socket connection

port 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 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 specifies 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 *D.2 Configuring wrproxy*, p.285.

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
```

E

Configuring Firewalls for Host-Target Interaction

E.1 Introduction

Workbench has the ability to interact with a target over numerous ports. These ports could be physical ports (serial, parallel, communication) or TCP/IP ports (TCP and UDP). Data passage using TCP/IP can be denied by firewalls that can "close" ports. This chapter lists ports on the host machine that are required to be open by processes/daemons responsible for information exchange between a connected target and the host.

E.2 System Limitations

Ports 1 through 1024 are reserved by the host OS for various protocols. Unless otherwise specified, none of the Workbench components should be set to use port numbers below 1024.

Ports below 1023 that will have to be opened, unless port mapped to a different number, include the following:

TCP Port Number	Description
21	The default port for **FTP**. If using an **FTP** server that forces passive mode, higher numbered ports may have to be opened.
22	The default port for **SSH**. Both **SSH1** and **SSH2** connections use this port.
23	The default port for **Telnet**.
513	The default port for **rlogin**. Note that on UNIX systems, local port range 512-1023 is restricted to root for security reasons.
514	The default port for **rsh**.

E.3 Wind River Components

WDB agent

Using the **wdbrpc** backend, **wdbrpc** acts as a server for requests sent by the target server.

Default

UDP port 0x4321 (17185) on the target. Dynamic (between 400 and 1024 UDP) on host.

Change by invoking **tgtsvr** with the **-p** option:

```
tgtsvr [-p.ort portNumber]
```

System Viewer

Socket Over TCP/IP

The Event Receive utility allows a remote host to listen for an incoming connection from a target that has an event buffer to upload.

Default

TCP 6164 on the host. Each new log will use the last used port incremented by 1.

Change by selecting **Analyze >> Event Receive** and modifying the value for **Port Number**.

File Over netDrv

The **netDrv** driver typically uses the **ftp** or **rsh** protocols to transfer data between the target and host.

Default

TCP 21 or TCP 514 on the host.

wtxregd

Daemon registry service that maintains a database of target servers, boards, and so on.

Default

TCP 2340 on the host.

KGDB

Agent-proxy

Bridges host and target if subnets are different.

Default

TCP 3333 on bridge. UDP 6443 on target.

Change by executing **agent-proxy** and passing new values.

```
agent-proxy 3333 [target-ip] udp:6443.
```

The KGDB connection can also be accomplished using Telnet, in which case TCP 23 will be used.

QEMU Deployment

KGDB over Ethernet

Default

UDP 6443 on host.

Change by editing the target connection's Connection Properties by right-clicking the connection in the Remote Systems view and selecting **Properties**.

Default QEMU configuration

Target QEMU Port	Port Number
TARGET_QEMU_PROXY_PORT	4442
TARGET_QEMU_PROXY_LISTEN_PORT	4446
TARGET_QEMU_DEBUG_PORT	1234
TARGET_QEMU_AGENT_RPORT	UDP 4444 mapped to 17185
TARGET_QEMU_KGDB_RPORT	UDP 4445 mapped to 6443
TARGET_QEMU_TELNET_RPORT	TCP 4441mapped to 23
TARGET_QEMU_SSH_RPORT	TCP 4440 mapped to 22
TARGET_QEMU_MEMSCOPE_RPORT	5698
TARGET_QEMU_PROFILESCOPE_RPORT	5678

Change by executing **make config-target** inside the build directory.

If there were multiple targets, by default, they would use the above port numbers incremented by 1.

This is changed by using the **TOPTS** flag with the **-i** option.

```
Make start-target TOPTS='-i #'
```

Wind River Proxy Host

wrproxy.

Default

TCP 17476.

Change by executing **wrproxy** with the **-p** option.

```
wrproxy -p 17444
```

Scopetools

Default

Using **tgtsvr**

Dynamic on host (UDP 400 through 1024).

UDP 17185 on target.

Using **TCP**

Dynamic on host (TCP 1024 through 5000).

Dynamic on target (TCP 49152 through 49407).

F Glossary

This glossary contains terms used in Wind River Workbench.

F.1 Searching for Terms in Online Documentation

If the term you want is not listed in this glossary, you can search for it throughout all online documentation.

1. If it is not already open, open the Help view by pressing the help key for your host.
2. At the bottom of the view, select **Search**.
3. Type the term you are looking for into the **Search expression** field.
4. Click **Go**. Links to topics containing the term will appear, along with a short example of the surrounding text.
5. To open the document containing a topic, click the topic in the list.

- To switch from the document you were reading back to the search results, click **Back**.
- To see where this topic's document appears in the help Table of Contents, click the **Show in All Topics** icon in the upper right corner of the help view.

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, you can create a Search Scope Set that will search only Wind River documentation:

1. Click **Search** at the bottom of the Help view.
2. Click the **Default** link next to **Search Scope**.
3. In the Select Search Scope Sets dialog box, click **New**.
4. Type in a name for your new Scope Set, for example **WR Documentation**. Click **OK**.
5. Select your new Search Scope Set, then click **Edit**.
6. In the Search Scope dialog box, click **Search only the following topics**.
7. From the Working set content pane, select the document sets to which you want to narrow the search, for instance **Wind River Documentation > References**.
8. Click **OK** twice to return to the help browser, where your new search scope appears next to the **Search scope** link.
9. Click **Go**. The results will appear in the Search list.

For more information about online help, click **All Topics**, then navigate to **Wind River Partner Documentation > Eclipse Workbench User Guide > Tasks > Accessing help**.

F.2 Glossary of 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, in VxWorks, 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 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

The type of project built: *managed build* by default (formerly *flexible build*); and also a deprecated but still available *standard managed build* (sometimes known as a *standard build*). There are also *user-defined* and *disabled builds*.

build spec

A particular set of build settings appropriate for a specific target board.

This provides functionality that is configured into a target server to allow it to communicate with various target agents, based on the mode of communication that you establish between the host and the target (network, serial, and so on).

CDT (C/C++ Development Tooling)

The Eclipse C/C++ IDE.

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.

debuggable objects

Debuggable objects are the executable application files, kernels, kernel modules, and libraries that can be accessed by both the host and target. These objects are ideally compiled without optimization, but with the appropriate debug flags (for example with **-g**, or **-g-dwarf-2**). They can not be stripped of symbols.

disabled build

Project builds for which Workbench provides no build support at all. Useful for projects or folders that contain, for example, only header or documentation files that do not need to be built.

Editor

The Editor is a visual component within Wind River Workbench. It is typically used to edit or browse a file or other resource. Each Workbench perspective displays the Editor area even when no files are open.

Modifications made in the Editor follow an open-save-close life cycle model. Multiple instances of an Editor type may exist within a Workbench window.

element

An element is an entity that holds source analysis information of any kind, standing for a declaration or occurrence of a constant, preprocessor option, variable, function, method, type, or namespace in a parsed source code file.

gutter

The left vertical border of the editor view where breakpoints and the program counter appear.

flexible managed build

Deprecated name for what is currently called *managed build*.

help key

The help key (or combination of keys) is determined by your host platform: press **F1** on Windows, or **Ctrl+F1** on Linux and Solaris.

Press the help key in Workbench to get context-sensitive help.

host shell

A Wind River command shell that provides a command-line environment for GDB and KGDB debugging. The host shell also provides Tcl scripting support.

hunk

In Linux, a hunk is a contiguous group of source lines generated when the **diff** program is applied to compare files. The **patch** program and the Quilt patch program based upon it use **diff** to create patches, which are then internally represented as one or more hunks to apply to a file to patch it.

JDT

Java Development Toolkit provided by the Eclipse organization (**http://www.eclipse.org**).

kernel mode

For Linux 2.6 and higher kernels, two connection modes are supported: kernel and user mode connections. Kernel mode connections allow **kgdb** debugging of the kernel in a manner analogous to debugging applications in user mode.

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 instructs 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.

managed build

A build for which Workbench controls all phases, available for all project types except user-defined projects.

native mode development environment

Linux: A development environment requiring a usermode agent program to be running on the target, in a Linux operating system. In this environment, the debugger and application are compiled with the same toolchain, thus no emulator is required when running in self-hosted mode.

A native mode development environment can only be used for application development.

object path mappings

The object path mappings specify where the debuggable objects are to be found for both the debugger running on the host and the target. In Workbench, this is set within the Remote Systems view's **Target Connection Properties**.

overview ruler

The vertical borders on each side of the Editor view. Breakpoints, bookmarks, and other indicators appear in the overview ruler.

perspective

A perspective is a group of views and Editors. One or more perspectives can exist in a single Workbench window, each perspective containing one or more views and Editors. Within a window, each perspective may have a different set of views but all perspectives share the same set of Editors.

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 Data Analysis Tools) 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 used to create a downloadable application or bootable 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 Explorer) that reflects their inner dependencies, and therefore the order in which they should be compiled and linked.

project description files

Automatically-generated files that contain information about a project, such as project properties, build information, makefile fragments, and other metadata.

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.

self-hosted development environment

The opposite of cross-development. The debugger and the application are running on the same machine.

source lookup path

The source lookup path specifies the location that the Workbench debugger uses to identify and open each source file as it is being debugged. This is set in the Debug view in Workbench.

standard build; standard managed build

Synonymous, deprecated project build types suitable for projects with build structures similar to the file system structure.

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 or Wind River Linux 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 (for example, **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 such as Search that apply to Workbench as a whole or that reflect the components that are installed.

TIPC

Transparent inter-process communication protocol typically used by nodes within a cluster. Wind River provides a proxy and usermode agent program that allow Workbench to access targets within the TIPC cluster.

user-defined build

Project builds for which you set up and maintain your own build system and Makefiles, and for which Workbench provides minimal support beyond a GUI

launch, make rules expressed in the Project Explorer, and build output to the Build Console.

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. For Linux 2.6 and higher kernels, user mode is a separate connection type. Compare to kernel mode.

view

A view is a visual component within Workbench, typically used to navigate a hierarchy of information (such as the resources in your workspace). Only one view has focus (is active) at a time.

VIP

VxWorks Image Project.

window

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

Resources you select to view or operate on as a group. For example, you can create a working set 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 Explorer, the number of symbols displayed in the Outline view, and so on.

workspace

The central location for all the resources you see in Workbench: your projects, folders, and files. Workbench 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**. To keep your projects completely isolated from each other, you can create more than one workspace.

To share the build objects of your projects with a target, the workspace (directory) may be in a file system that is exported to the target, or you may redirect build objects from your workspace to a location exported to the target.

NOTE: This use of the term **workspace** is entirely different from the flash workspace, which is a small area of RAM needed to run the flash algorithm; that sense of the term is restricted to flash programming.

Index

C

D

E

F

H

I

J

L

M

N

O

P

R

S

T

U

V

W