WIND™

WIND RIVER®
LINUX

7 COMMAND-LINE TUTORIALS

7.0

Copyright Notice

Corporate Headquarters

Wind River
500 Wind River Way
Alameda, CA 94501-1153
U.S.A.
Toll free (U.S.A.): 800-545-WIND
Telephone: 510-748-4100
Facsimile: 510-749-2010

For additional contact information, see the Wind River Web site:

www.windriver.com

For information on how to contact Customer Support, see:

www.windriver.com/support

22 May 2015

Contents

1

Developing Platform Projects

Creating and Configuring a Platform Project

You create and configure a platform project with a BSP, kernel, and file system to match your target platform system requirements.

Step 1 Create a project directory.

You must create this outside of the installed development environment. For example, you could have a subdirectory of your home directory named **Builds/**, under which you would create your specific platform project directories.

In this example, we will create and navigate to a directory named after the target hardware board support package (BSP) and file system we are creating the platform project for.

```
$ mkdir -p ~/Builds/qemux86-64_small
$ cd ~/Builds/qemux86-64_small
```

Step 2 Define the shared state cache (*sscache*) environment variable.

The shared state cache speeds up successive builds of a project so that only elements that have changed are rebuilt. Run the following commands from the project directory to define the **sscache** variable:

```
$ mkdir -p ~/Builds/sscache
$ export SSTATE_DIR=~/Builds/sscache
```

NOTE: The shared state cache can also be specified in the configure script as follows:

--with-sstate-dir=~/Builds/sscache.

For additional information on using **configure** to create platform projects, see: *Wind River Linux User's Guide*.

NOTE: The content of the environment variable will override any previously configured path.

Step 3 Optionally define the *ccache* environment variable.

While **ccache** is optional and not required to configure and build platform project images, it helps speed up the build process for repeated build and/or delete cycles in the project directory. To use **ccache**:

- To limit potential negative effects on the cache speed, the **ccache** and the platform project directories should reside on the same physical volume and file system. Do not to install the **ccache** directory on a NFS-mounted system, since this scenario is not fully tested.
- Your development workstation must have the **ccache** command installed. For example, on a Debian-based host, you would use the command **sudo apt-get install ccache** to install ccache.
- The ccache directory can be shared across multiple platform projects.

Run the following commands from inside the project directory to define the *ccache* variable:

```
$ mkdir -p ~/Builds/ccache
$ export CCACHE_DIR=~/Builds/ccache
```

You can also use additional configuration options on the command line instead to define the **ccache** directory as follows:

- **--enable-ccache=yes** — use this options to enable the **ccache**.
- **--with-ccache-dir=***cache-dir-name* — use this option to associate a previously-defined **ccache** directory for the platform project, where *cache-dir-name* is the previously created **ccache** directory. The default is **host-cross/var/cache** inside the directory of your platform project, which the build process creates automatically.
- **--with-ccache-size=***5G* — use this optional setting to limit the size of the **ccache**. The default is 1G.

Step 4 Configure the project.

NOTE: This tutorial uses the configure script, however, you can also use the Platform Project Configure tool. For more information, see: *Wind River Linux User's Guide*.

To configure a project using the **configure** script, specify the board, kernel, and file system. For example, enter the following command to specify the qemux86-64 (common PC, 64-bit) board with a standard kernel and small file system:

```
$ installDir/wrlinux-7/wrlinux/configure \
--enable-board=qemux86-64 \
--enable-kernel=standard \
--enable-rootfs=glibc_small \
--enable-parallel-pkgbuilds=4 \
--enable-jobs=4
```

In addition to the board and file system configuration options, this command also includes two options to help speed up the build process:

- **--enable-parallel-pkgbuilds=4** — Sets the number of packages that can be built in parallel to speed up the build process. As a rule of thumb, this number should equal the number of CPUs available in your workstation. Using this option above sets up Wind River Linux to build up to four packages in parallel for your project.
- **--enable-jobs=4** — Sets the number of jobs (threads) that can be used to build a single package. Using the configure option above sets up Wind River Linux to use up to four execution threads when building an individual package.

NOTE: In addition to support for the options above, BitBake automatically sets parameters based on the number of cores on the machine. For more information, see: *Wind River Linux User's Guide*.

You may need to adjust these two option settings to achieve optimal performance for your specific configuration, particularly if you are doing multiple simultaneous builds. You can validate the configuration for your purposes by examining the resulting **local.conf** file.

The build system starts to configure your project. This may take a few minutes.

Postrequisites

For future projects, you may want to experiment with new settings for these options to establish the best performance for your development host.

For more information on the use of the **configure** command and the **Platform Project Configure** tool, see: *Wind River Linux User's Guide*.

Once your project configuration completes, the next step is to build the project. See *Building a Platform Project Image* on page 7.

Building a Platform Project Image

Once you configure a platform project, you must it build before further development.

To build your platform project using the command-line:

Step 1 Run the **make** command to create the file system.

Once the project has been configured, enter the following command in the project directory:

```
$ make
```

NOTE: The **make** command must be run from the project directory, and not from inside the Wind River Linux installation directory.

This will take from minutes to hours, depending on your development resources and project configuration.

When the build finishes, you will have a kernel and file system ready for deployment to a target or an emulator as described in *Deploying a Platform Project Image* on page 11.

When the build is complete, the platform project is ready to be deployed to the target. The kernel is in (or linked from) your project directory, for example:

***projectDir*/export/qemux86-64-bzImage-WR7.0.0.0_standard** .

The target file system is in **export/dist/**. The contents of the directory may include the following:

README* bin/ boot/ dev/ etc/ /home lib/ lib64/ media/ mnt/ proc/ sbin/ sys/ tmp/ usr/ var/

The file system is also available as a compressed archive file for ease of transporting it to the target, for example, ***projectDir*/export/qemux86-64-glibc-small-standard-dist.tar.bz2**

Step 2 Familiarize yourself with the output of the build process.

Once the build process completes there are a few directories and files to look at in your project directory:

For a summary of notable directories, see: *Wind River Linux User's Guide*.

Configuring a New Project to Add Application Packages

You can add application packages using the **configure** command.

In the following example, you are going to configure a project to add the **gdb** (GNU Debugger) to it. When you add a package in this manner, it is automatically included in the file system of the platform project, and built (or rebuilt) each time you run the **make** command to recreate the file system.

NOTE: The functionality (the target-resident debugger) added in this example is currently only supported on targets with the x86 architecture (32- and 64-bit), but the workflow for adding a non-default layer and templates is the same with all architectures and BSPs.

Step 1 Configure the project.

To configure a glibc_small platform project that includes the **gdb** package, you use the **--with-package=gdb** configure option. For example:

```
$ installDir/wrlinuxDir/wrlinux/configure \
    --enable-board=qemux86-64 \
    --enable-kernel=standard \
    --enable-rootfs=glibc_small \
    --enable-parallel-pkgbuilds=4 \
    --enable-jobs=4 \
    --with-package=gdb
```

In addition to the standard configuration arguments of a board, kernel, and file system, this configuration adds the **gdb** package.

When you configure a project with a specific package, that package is added to the platform project, and available for use once the project is built and deployed to a target.

This same approach works for templates and layers as well, with the **--with-template=** and **--with-layer=** configure options. For additional information on configuring projects with templates and layers, see: *Wind River Linux User's Guide*.

Step 2 Build the project by entering the make command:

```
$ make
```

This will take from minutes to hours, depending on your development resources. When it is finished, you will have a kernel and file system that includes **gdb**.

Step 3 Verify that the package was added successfully. See *Verifying that the Project Includes the New Application Package* on page 10.

Adding New Application Packages to an Existing Project

You can add application packages to an existing platform project.

In the following example, you will add **gdb** to an existing, previously configured and built, platform project. See:

- *Creating and Configuring a Platform Project* on page 5
- *Building a Platform Project Image* on page 7

This example assumes that you do not already have **gdb** included with your platform project.

Step 1 Build the **gdb** package.

From the project directory, enter the following command to build the **gdb** package:

```
$ make gdb
```

Building the package takes a couple of minutes, during which you will see the progress on your terminal.

Step 2 Add the **gdb** package to the platform project build.

a) Add the **gdb** package.

From the project directory, enter the following command to add the **gdb** package:

```
$ make gdb.addpkg
```

The system will return the following output:

```
make: Entering directory `/Builds/qemux86-64_small/build'
==Checking ../layers/local/recipes-img/images/wrlinux-image-glibc-small.bbappend==
==Checking for valid package for gdb==
...
=== ADDED gdb to ../layers/local/recipes-img/images/wrlinux-image-glibc-
small.bbappend ===
```

Package changes like this are added to the following file, where *file-system* represents the name of the root file system used to configure the platform project:

***projectDir*/layers/local/recipes-img/images/wrlinux-image-*file-system*.bbappend**

b) Verify that **gdb** was added successfully.

```
$ cat layers/local/recipes-img/images/wrlinux-image-glibc-small.bbappend
```

The system returns the following output, after the #### **END Auto Generated by configure** #### line:

```
#### END Auto Generated by configure ####
IMAGE_INSTALL += "gdb"
```

This indicates that the package will be included in the build.

Step 3 Rebuild the root file system.

Enter the **make** command in the project directory:

```
$ make
```

Rebuilding the file system should take just a couple of minutes this time, because only the newly added elements need to be built.

Step 4 Verify that the package was added successfully.

See *Verifying that the Project Includes the New Application Package* on page 10.

Verifying that the Project Includes the New Application Package

After you add an application package to your project, you can verify that it is working properly.

In this example, you will verify that a package was added successfully to the platform project.

This procedure assumes you have added the **gdb** package from *Configuring a New Project to Add Application Packages* on page 8, or *Adding New Application Packages to an Existing Project* on page 9

Step 1 Verify that the **gdb** is available now by looking at the generated root file system. From the project directory, enter:

```
$ ls export/dist/usr/bin/gdb
```

The system returns the following output to confirm that the **gdb** executable exists:

```
export/dist/usr/bin/gdb
```

Step 2 Verify that the **gdb** command is available on the running target.

a) Deploy the platform project on a target. See *Deploying a Platform Project Image* on page 11.

b) Run the **gdb** command on the target:

```
gdb
```

The system returns the following and display the (gdb) prompt, confirming that **gdb** is working:

```
GNU gdb (Wind River Linux Sourcery CodeBench 4.6-60) 7.2.50.20100908-cvs
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-wrs-linux-gnu".
For bug reporting instructions, please see:
<support@windriver.com>.
(gdb)
```

Deploying a Platform Project Image

Once you have configured and built a platform project, you can deploy it to a simulated target platform to test it.

The example in this section uses the qemux86-64 platform project created in *Building a Platform Project Image* on page 7.

See *Creating and Configuring a Platform Project* on page 5 for project creation details.

To deploy a platform project to QEMU using the command-line:

Step 1 In your project directory where you ran the configure and make commands, enter one of the following commands to boot your kernel with QEMU:

Options	Description
Run the platform project in the current terminal window:	`$ make start-target` This option displays serial output from the target only, and is for command-line specific actions that do not require VGA graphics support.
Run the platform project in a separate terminal window:	`$ make start-target TOPTS=-gc &` This option displays the QEMU graphics console, which emulates a VGA display. The kernel boots in the emulation, and then mounts the file system using NFS. The file system is located in the platform project directory as discussed in *Building a Platform Project Image* on page 7.

Step 2 Log in.

Login as user **root**, password **root**.

Step 3 Explore the system.

With the platform project running on the target, you can use Linux commands to explore the system.

Step 4 End the session using one of the following options.

Options	Description
From the QEMU terminal, enter:	`# poweroff` This allows the target to shutdown cleanly, and also clears the QEMU process.
For a platform project running in separate window, you can simply enter the following in the terminal window from which launched the target:	`# make stop-target` This is the equivalent of removing power from a running system.

With either choice, the QEMU session will halt and return control of the command prompt to the original terminal window.

Postrequisites

Once your project is created and launches successfully on a target, you may want to customize it by adding applications or packages to the new file system.

2

Developing Application Projects

About the Local Layer and Adding Applications

When you configure a platform project image, Wind River Linux automatically creates a local layer that can be used as the starting point to develop your own applications.

The local layer resides in the ***projectDir*/layers/local** directory. For additional information on the content and structure of the layer, see the section *About the layers/local Directory* in the *Wind River Linux User's Guide*. When you add a Wind River Linux-supplied sample application to your platform project, it is automatically added to the ***projectDir*/layers/local** directory.

Layer configuration is managed using the ***projectDir*/layers/local/conf/layer.conf** file. There is generally no need to modify this file, because the default configuration includes everything necessary to process local application recipes.

Every time you build a platform project, any application that you previously built and added to your platform project (see *Creating and Deploying an Application* on page 14) with a recipe file located in the local layer will also be built and included with the platform project image. For additional information the contents of the **layer.conf** file and creating recipes for your own application projects, see the section *About .conf Files and Platform Projects* in the *Wind River Linux User's Guide*.

Creating and Deploying an Application

You can build and deploy an application from the command line.

A sample Hello World application is provided with Wind River Linux when you build a platform project of any kind. In this section, you will add it to the platform project created and built in the following sections:

- *Creating and Configuring a Platform Project* on page 5
- *Building a Platform Project Image* on page 7

Because the ***projectDir*/layers/local** layer is already integrated into the build process, all you have to do is to activate the **hello** package as you would do with any other package provided by default with the distribution.

The difference here is that now you know where the source code is and that you have direct control over it through your own local configuration.

NOTE: To add an application to a platform project that is not part of an existing package, or does not have a recipe file associated with it, see the section *About Application Development*in the *Wind River Linux User's Guide*. For information on using the SDK, see *Exporting the SDK* in the *Wind River Linux User's Guide*.

Step 1 Build the **hello** package.

From the project directory, enter the following command to build the **hello** package:

```
$ make hello
```

Step 2 Add the hello package to the platform project build.

a) Enter the following command from the project directory to add the **hello** package:

```
$ make hello.addpkg
```

The system will return the following output:

```
make: Entering directory `/Builds/qemux86-64_small/build'
==Checking ../layers/local/recipes-img/images/wrlinux-image-glibc-small.bbappend==
==Checking for valid package for hello==
...
=== ADDED hello to ../layers/local/recipes-img/images/wrlinux-image-glibc-
small.bbappend ===
```

Package changes like this are added to the recipe file at the following location, where *file-system* represents the name of the root file system used to configure the platform project:

***projectDir*/layers/local/recipes-img/images/wrlinux-image-*file-system*.bb**

b) Verify that **hello** was added successfully:

Run the following command:

```
$ cat layers/local/recipes-img/images/wrlinux-image-glibc-small.bbappend
```

The system returns the following output, after the line that declares `####` **END Auto Generated by configure** `####`:

```
#### END Auto Generated by configure ####
IMAGE_INSTALL += "hello"
```

This indicates that the package will be included in the build.

Step 3 Rebuild the root file system.

Enter the following command in the project directory:

```
$ make
```

Rebuilding the file system should take just a couple of minutes this time, because only the newly elements added need to be built.

Step 4 Verify that the package was added successfully by looking at the platform project's root file system:

From the project directory, enter:

```
$ ls export/dist/usr/bin/hello
```

The system returns the following output to confirm the hello executable exists:

```
export/dist/usr/bin/hello
```

NOTE: The source file of the application is automatically added to the ***projectDir***/**layers/local/recipes-sample/hello** directory. For all sample applications, Wind River Linux automatically creates a BitBake recipe file (**.bb**) in the folder for the application.

Step 5 Verify that the application runs on the target.

a) Deploy the platform project on a target.

b) Run the **hello** command on the target:

```
# hello
```

The system returns the following to confirm that the application was added successfully:

```
Hello World
```

3

Debugging an Executable

Debugging an Executable

Typically, you debug an application once you have created or added it to a platform project image.

You can debug target applications from the host or on the target itself. This section shows how to use **gdb** on the target and assumes you have configured and built a platform project.

Debugging requires an application that has been built with debugging symbols.

Use the following procedure to add an application and connect **gdb** to it:

Step 1 Add an application to the build that includes debugging symbols.

a) Add the **hello-dbg** package.

Enter the following in the platform project directory:

```
$ make hello-dbg.addpkg
```

The system returns the following output.

```
make: Entering directory `/Builds/qemux86-64_small/build'
==Checking ../layers/local/recipes-img/images/wrlinux-image-glibc-small.bbappend==
==Checking for valid package for hello-dbg==
...
=== ADDED hello-dbg to ../layers/local/recipes-img/images/wrlinux-image-glibc-
small.bbappend ===
```

Package changes like this are added to the ***projectDir*/layers/local/recipes-img/images/wrlinux-image-*file-system*.bbappend** file, where *file-system* represents the name of the root file system used to configure the platform project.

a) Verify that hello was added successfully:

Run the following command:

```
$ cat layers/local/recipes-img/images/wrlinux-image-glibc-small.bbappend
```

The system returns the following, after the line that reads

END Auto Generated by configure ####:

```
#### END Auto Generated by configure ####
IMAGE_INSTALL += "hello-dbg"
```

This indicates that the package will be included in the build.

Step 2 Rebuild the root file system.

Enter the following command in the project directory:

```
make
```

Rebuilding the file system should take just a couple of minutes this time, because only the new elements added need to be built.

Step 3 Verify that the package was added successfully.

Examine the platform project's root file system of the platform project. From the project directory, enter:

```
$ ls export/dist/usr/bin/hello
```

The system returns the following output to confirm that the **hello** executable exists:

```
export/dist/usr/bin/hello
```

Step 4 Verify that the application runs on the target.

a) Deploy the platform project on a target. See *Deploying a Platform Project Image* on page 11.

b) Run the hello command on the target by typing:

```
# hello
```

The system returns the following to confirm that the application was added successfully:

```
Hello World
```

Step 5 Use **gdb** on the target.

Connect **gdb** to the application. For example, using the hello application from *Creating and Deploying an Application* on page 14, you would enter:

```
# gdb /usr/bin/hello
```

The system returns the following output, and display the (**gdb**) prompt:

```
GNU gdb (Wind River Linux Sourcery CodeBench 4.6a-107) 7.4.50.20120716-cvs
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-wrs-linux-gnu".
For bug reporting instructions, please see:
```

```
<support@windriver.com>.
(gdb)
```

Step 6 Enter **quit** at the prompt to end the debugging session.

Postrequisites

Refer to standard GNU documentation and search on the web for how to use **gdb**.

For more general guidance and examples of debugging applications on the target from the host using command-line tools, see: *Wind River Linux User's Guide*.