WIND™

WIND RIVER® LINUX

MIGRATION GUIDE

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

Introduction and Overview

About Migrating Wind River Linux Projects

Learn about your migration options and limitations, and where to find related information for your project requirements.

Migration Scope: What You Can Migrate from Wind River Linux 4.x

This guide is designed to be a supplement to the *Wind River Linux User's Guide* and other documentation, to address specific procedures and issues related to product migration. For information on using the product in general, please refer to the relevant documentation as described in *Wind River Linux Documentation* on page 39. This includes information such as host system requirements, product structure, the development environment, and procedures and reference material for using Wind River Linux.

Wind River Linux documentation is available in the product installation, through Workbench, and also through Wind River Online Support.

Specifically, this guide refers to migrating Wind River Linux platform and application projects. This includes:

- Layers and packages from Wind River Linux 4.3 platform projects using the migration tools. See *About Migrating Platform Projects and Elements* on page 25.

  The import tools are designed for user space applications and do not attempt to import kernel patches or BSPs that may be present in a layer. In this case, the migration scripts will provide warnings when it encounters layer content that it does not support, and will ignore that content and not try to import it to your Wind River Linux 7.x project.

- Application projects (from source), and SRPM packages from Wind River Linux 4.x platform projects using the Package Importer tool. See *Migrating Source Applications* on page 36.

  Note that migrating compiled applications or packages from Wind River Linux 4.x projects is not supported.
- Factors for migrating BSPs and kernels from Wind River Linux 4.x. See *About Kernel and BSP Changes* on page 23.
- Manually updating layers from Wind River Linux 4.x, for projects with complex layer requirements. See *About Build System Changes* on page 17.

  Note that the Migration tool supports the migration of basic platform project layers from Wind River Linux 4.x layers, but due to changes in the layer structure, it may be necessary to migrate complex layer content manually. The *Migration Limitations - What is Not Supported* section below provides a list of layer content that requires manual conversion.

Migrating from Wind River Linux 4.x poses some challenges in that the build system for Wind River Linux 7.x is completely different. For an overview of the changes, see *Product Structure for Migration Purposes* on page 7. For a description of the build system changes, see *BitBake and LDAT Comparison* on page 18.

.

Migration Limitations: What is Not Supported

A full platform project migration is not supported. You must configure and build your base platform project using a Wind River Linux 7.x-supported BSP, and migrate the parts, such as layers, applications, and packages, specific to your Wind River Linux 4.x project. See *About Migrating Platform Projects and Elements* on page 25 for additional information.

With regards to a platform project's layer content, the following layer-specific features are not yet implemented:

- pkglist.remove, pkglist.only files
- toolslist.add, toolslist.remove, toolslist.only files
- config.sh
- BusyBox or Linux template directories
- Linux templates
- Profile, rootfs or board templates
- Packages that include Makefile fragments from other layers
- Converting LDAT variables in the **fs-final*.sh** script
- Automatic `LIC_FILES_CHKSUM` generation

If your platform project layers include support for any of the unsupported features in this list, you will need to import the content manually to your Wind River Linux 7.x platform project.

Product Structure for Migration Purposes

Understanding the product structure changes between Wind River Linux 4.x and 7.x will help prepare you to better understand migration requirements.

Product Structure Changes Overview

With the introduction of Wind River Linux 7.x and the Yocto Project BitBake build system, the former Wind River Linux LDAT build system used by Wind River Linux 4.x significantly changes the structure of a configured and built platform project. These changes include:

Layer and template structure and content

This structure is very different in that the Yocto Project build system relies on the concept of recipes and configuration files to define platform project configuration information. This change results in different layer and template directories, as well as how configuration information is specified for the build system. For a comparison, see *BitBake and LDAT Comparison* on page 18.

BSP structure and content

Like layers and templates, the new build system uses recipes and configuration files to define BSP configuration information. This allows developers to append existing Wind River Linux 7.x BSPs with project-specific changes, simplifying the process of modifying a BSP to meet a given need. To learn how to modify an existing BSP, see *About Kernel and BSP Changes*.

Replacing sysroots with a software development kit (SDK) for Linux and Windows development hosts

The inclusion of a new SDK greatly simplifies application and package development, as once a platform project is built, the SDK is provided as an installable shell script. This script automatically installs the SDK, and sysroot information, and sources the development environment. As a result, the build paths and other elements of the SDK differ from their Wind River Linux 4.x counterparts. For additional information, see *About Application Development Changes* on page 35.

For information on the Wind River Linux 7.x development and build environments, see the *Wind River Linux User's Guide: About the Development Environment Directory Structure* and *Wind River Linux User's Guide: About the Build Environment Directory Structure*.

Migration Strategy Reference

Use this reference to determine the best migration strategy for your application or package.

Current Format	Migration Tool(s)	Migration Strategy	Document Reference(s)
LDAT classic tar file layer	**importlayer.py**	Reorganize the directory layout of the package, and begin the translation of the Makefile into a BitBake recipe with the relevant tool. Test the generated layer in a platform project to debug the recipe created by the tool.	To migrate a layer, see *Using the Migration Tools to Migrate Layers* on page 33
LDAT spec-driven SRPM package	Package Importer tool **rpmparse.py**	Remove the embedded tar file from the SRPM, and then import it into a test project using the Package Importer tool. Use **rpmparse.py** to generate the BitBake recipe from the embedded spec file. Manually combine the recipe files from both tools and test.	To import a LDAT SRPM package, see the *Wind River Linux User's Guide: Importing a SRPM Package from the Web*. For instructions on updating the package's BitBake recipe from information in its ***.spec** file, see *Updating a BitBake Recipe from a SRPM *.spec File* on page 29. For information on using **rpmparse.py**, see *Migration Script Reference* on page 14.
Applications built with the sysroot and copied into the file system at the end of the platform build	SDK **fs_final.sh**	Build the SDK (**export_sdk**) from the Wind River Linux 7.x platform project and use the SDK's sysroot to compile your code. Place your existing **fs_final.sh** script file, which copies your content into the file system, into its new	For instructions on building and exporting the SDK, see the *Wind River Linux User's Guide: Exporting the SDK*. For instructions on using the **fs_final.sh** script with Wind River Linux 7.x, see the *Wind River Linux User's Guide: Adding an Application to*

Current Format	Migration Tool(s)	Migration Strategy	Document Reference(s)
		***projectDir*/layers/local** location and test.	*a Root File System with fs_final.sh scripts*.
Workbench Wind River Linux 4.x application project	Workbench	Import the new sysroot into Workbench if necessary. Copy the source into a new empty Wind River Linux 7.x application project and debug. Optionally package the new projects by importing them into the platform project with the Package Importer tool (**import_package**)	Refer to the *Wind River Workbench by Example Guide (Linux Version)* for information on importing sysroots and creating application projects. To import a source application as a package, see the *Wind River Linux User's Guide: Importing a Source Package from the Web* for guidelines.
Workbench Wind River Linux 4.x kernel module project	Workbench	Copy the source into a new empty Wind River Linux 7.x kernel module project that is a subproject of a Wind River Linux 7.x platform project and debug.	Refer to the *Wind River Workbench by Example Guide (Linux Version)* for additional information.
Publicly available tar file package	Package Importer tool	Use an existing package in Yocto, or Open Embedded. If the same package is not available, create a template recipe with the Package Importer tool.	See the *Wind River Linux User's Guide: Importing a Source Package from the Web* for guidelines.
Publicly available SRPM package	Package Importer tool	If possible, migrate to the tar file version of the package, and use an existing recipe. Then covert any SRPM patches to a quilt patch list.	To migrate a SRPM package, see the *Wind River Linux User's Guide: Importing a SRPM Package from the Web*. For instructions on using Quilt, see the *Wind River Linux User's Guide: Patching with Quilt*.
Wind River Linux 4.x BSP	Workbench, BSP Cloning tool	Clone the Wind River Linux 7.x BSP that is closest to your Linux 4.x BSP by using the	See *About Kernel and BSP Changes*

Current Format	Migration Tool(s)	Migration Strategy	Document Reference(s)
		export tool, and then move your custom content into the cloned BSP.	

Platform Project Migration Workflow

Use the following workflow as a guideline for migrating a platform project from Wind River Linux 4.x to 7.x.

With Wind River Linux, platform projects are used to maintain all files necessary to configure, build, and debug a distribution for a given target. As a result, this guide uses the overall platform project migration as a guideline for migrating a Wind River Linux 4.x, LDAT project to a Wind River Linux 7.x BitBake one.

To migrate a platform project:

1. Configure and build a new Wind River Linux 7.x platform project using a supported BSP.

   For additional information, see:

   - *About Migrating Platform Projects and Elements* on page 25
   - *Wind River Linux User's Guide: Supported Run-Time Boards*
   - *Wind River Linux User's Guide: Configuring a Platform Project Image*
2. If your Wind River Linux 4.x platform project has changes made to the BSP, integrate those changes into your new platform project image. For additional information on modifying a BSP, see *About Kernel and BSP Changes*.
3. Migrate any basic layers from your Wind River Linux 4.x platform project using the migration tools. For more specific information on migration strategy based on layer content, see *Migration Strategy Reference* on page 8.

   For additional information, see:

   - *Installing the Migration Tools* on page 13
   - *Migration Script Reference* on page 14
   - *Using the Migration Tools to Migrate Layers* on page 33

   When you use the migration tools to migrate a layer, the scripts will provide output information for any content that could not be automatically moved to a Wind River Linux 7.x layer. If you receive output of this manner, you will need to perform *Step 4*, below.
4. Manually migrate more complex layer contents as required. Depending on the content, you may need to reference the *Wind River Linux User's Guide* for additional information.
5. Import any source applications from your Wind River Linux 4.x platform project using the Package Importer tool. For additional information, see *About Application Development Changes* on page 35 and the *Wind River Linux User's Guide: About the Package Importer Tool*.

   Note that only importing source applications is supported for migration purposes. It is not possible to import a previously built application from a Wind River Linux 4.x platform project.

   For additional information, see:

- *About Application Development Changes* on page 35
- *Migrating Source Applications* on page 36
- *Migration Script Reference* on page 14
- *Using the Migration Tools to Migrate Layers* on page 33

6. Import any packages from your Wind River Linux 4.x platform project that were not updated with Wind River Linux 7.x. For example, because of a kernel revision upgrade, and changes to the host/target package system, it's possible that a package you previously used is included with Wind River Linux 7.x, and has been upgraded to a later revision. If this is the case, you can run the following command to add the package to your platform project image:

   ```
   $ make packageName.addpkg
   ```

   If your package is not included, the migration tools provide scripts for importing existing Debian and RPM source packages to the BitBake format. Once in that format, you can add the package to a specific layer, or as part of the available packages for your platform project image.

   For additional information, see:

   - *Migrating Source Applications* on page 36
   - *Migration Script Reference* on page 14
   - *Using the Migration Tools to Migrate Layers* on page 33
7. Once all aspect of your platform project are added and/or imported, debug the migration-specific issues until your Wind River Linux 7.x platform project image is complete.

2

Installation

Installing the Migration Tools

Use the information in this section to obtain and install the migration tool scripts necessary to update your supported Wind River Linux 4.x projects to 7.x.

To download the migration tool scripts from Wind River Online Support (OLS), you must have a valid Wind River Linux 7.x license. The list of migration scripts and their purpose is located at *Migration Script Reference* on page 14.

Step 1 Log into the Wind River Linux 5 Migration Tools web page.

In a web browser, navigate to *https://support.windriver.com/olsPortal/faces/maintenance/downloadDetails.jspx?contentId=040628*. When prompted, log in using your Wind River OLS username and password.

The Wind River Linux 5 LDAT Migration Tools page displays.

Step 2 Download the tools.

In the **DOWNLOADS** section, select the **ldat-tools_2.0.tar.gz** file and save it to a location on your development host.

Step 3 Unpack the tools.

Run the following command from the directory you downloaded the file to in the previous step to unpack the migration tool scripts:

```
$ tar zxvf ldat-tools_2.0.tar.gz
```

This will create a **ldat-tools** directory and install the migration scripts and supporting files to it. For example:

```
$ ls ldat-tools

dscparse.py     importpackage.py  makefileparser rpm2cpio
importlayer.py  ldat              README         rpmparse.py
```

For a description of this directory's contents, with examples for using the migration scripts, see *Migration Script Reference* on page 14.

Migration Script Reference

Quickly get information and cross-references on using the migration tool Python scripts.

Script Name	Arguments	Description

importlayer.py	LDAT layer to migrate and new layer name	Converts a LDAT layer into the new BitBake layer format.
		Example usage:
		`$ ./importlayer.py -h`
		`Usage: importlayer.py <Input LDAT Layer Directory> <Output Bitbake Layer Directory>`
		`$ ./importlayer path/to/wrll-mylayer path/to/meta-mylayer`

importpackage.py	LDAT layer and package name to migrate, and the BitBake output directory	Converts an individual LDAT Makefile into a BitBake recipe.
		Example usage:
		`$ ./importpackage.py -h`
		`Usage: importpackage.py <Input LDAT Layer Directory> \`
		`<Input LDAT Package Name> <Output Bitbake Layer Directory>`
		`$ ./importpackage.py path/to/wrll-mylayer mypackage path/to/meta-mypackage`

dscparse.py	Debian package name and output directory	Imports a Debian-based package into the BitBake package format
		Example usage:
		`$ apt-get source <debian package>`
		`$ ./dscparse.py -h`
		`Usage: dscparse.py <package.dsc> <output directory>`
		`$ ./dscparse.py <debian package_x.y.dsc> path/to/meta-debianpackage`

Script Name	Arguments	Description

rpmparse.py	RPM package name and output directory	Imports a RPM-based package into the BitBake package format for source RPM files only.
		Example usage:
		`$ yumdownloader --source <rpm package>`
		`$ ./rpmparse.py -h`
		`Usage: rpmparse.py <package.rpm> <output directory>`
		`$ ./rpmparse.py <rpm package-x.y.fc99.src.rpm> path/to/meta-rpmpackage`

3

Build System Changes

About Build System Changes

Understanding the build system changes between Wind River Linux 4.x and 7.0 will help prepare you to complete your migration tasks.

This section describes changes to the Wind River Linux build system for version 7.0, from version 4.x. For information on new features available with 7.x, refer to the *Wind River Linux Release Notes*.

In general, configuring, building, and other workflow procedures remain the same in Wind River Linux 7.x as they were in Wind River Linux 4.x. However, there have been significant changes in the way the product is structured and in the build system itself with the introduction of the Yocto Project BitBake build system. For an overview of the changes between the BitBake and LDAT build systems, see *Product Structure for Migration Purposes* on page 7.

This includes the location of project-specific configuration files, layer configuration, and how the build system processes these files when you configure the platform project or build the file system.

BitBake and LDAT Comparison

Learn about the differences in the build systems and how it affects platform project migration.

BitBake and LDAT: Two Very Different Build Systems

While both build systems are based around the same core principle — creating a Wind River Linux target distribution using minimal configuration information — each uses a different approach to reach that goal. Some of the main differences are explained at a high level in *Product Structure for Migration Purposes* on page 7. With LDAT, much of the platform project configuration information was maintained in specific layer and template directories. This configuration information included Makefiles, patches, and packages for user space platform project elements, and kernel configuration fragments and patches for kernel or BSP-related elements.

With BitBake, all platform project configuration follows the same model in that configuration (***.conf**), recipes (***.bb**), and recipe append (***.bbappend**) files define the platform project's configuration. This is explained in the the *Wind River Linux User's Guide: About the Development Environment Directory Structure*.

With Wind River Linux 4.x, making changes, such as adding a new layer for testing purposes to a platform project, required that the project be configured and built separately. With Wind River Linux 7.x, it's possible to add and remove layers and rebuild the platform project by editing the project's ***projectDir*/bitbake_build/conf/bblayers.conf** configuration file. In addition, when changes to a platform project element require updating a recipe file, the ability to use recipe append (***.bbappend**) files to update related project elements helps simplify development.

These differences pose a challenge to platform project migration, though. The information in a typical application's or package's Makefile is now maintained in a recipe (***.bb**) file, in a layer specific to the BitBake development environment. To reuse platform project metadata, such as layers with applications and packages, you must first create a new platform project with Wind River Linux 7.x, and then import the layers, packages, and applications to it.

With Wind River Linux 7.x, the ***projectDir*/layers/wrlcompat** layer maintains the scripts necessary to use the Wind River Linux configure script used to create a new Wind River Linux platform project.

Build Environment Changes

Wind River Linux 7.x includes the following new build environment directories that were not included in Wind River Linux 4.x:

- **bitbake**
- **bitbake_build**
- **layers**
- **packages**
- **README**S

For a description of the content of these directories, see the *Wind River Linux User's Guide: About the Build Environment Directory Structure*.

In Wind River Linux 7.x, the **build** and **build-tools** directories contain links to directories in the **bitbake_build** directory and are created for convenience for those familiar with those directories in the LDAT build system. There is no longer a **build/INSTALL_STAGE/*packageName*** directory. The equivalent directory is now **build/*packageName*/package**.

In Wind River Linux 7.x, the **do_install** rule populates the **build/*packageName*/image** directory. The **do_package** rule moves the required files to the **build/*packageName*/package** directory, then the **build/*packageName*/package-split** directory based on a default set of filters. If necessary, you can override or append this in the recipe using the **${${PN}_FILES}** variable entry. Typically, an explicit packaging install rule, like the one in an LDAT spec file, is not required. For additional information, refer to the Open Embedded manual at *http://www.embeddedlinux.org.cn/OEManual/recipes_packages.html*.

In addition, the following directories from Wind River Linux 4.x no longer exist in Wind River Linux 7.x:

- **build-***lib*

  This directory only appeared in cases where the project supports and specifies multilibs. This information is now part of the build system by default.
- **filesystem/fs**

  This directory was previously used to add applications or packages to the target file system. The process for doing this has changed in Wind River Linux 7.x. For additional information, see the *Wind River Linux User's Guide: Adding an Application to a Root File System with fs_final*.sh Scripts*.

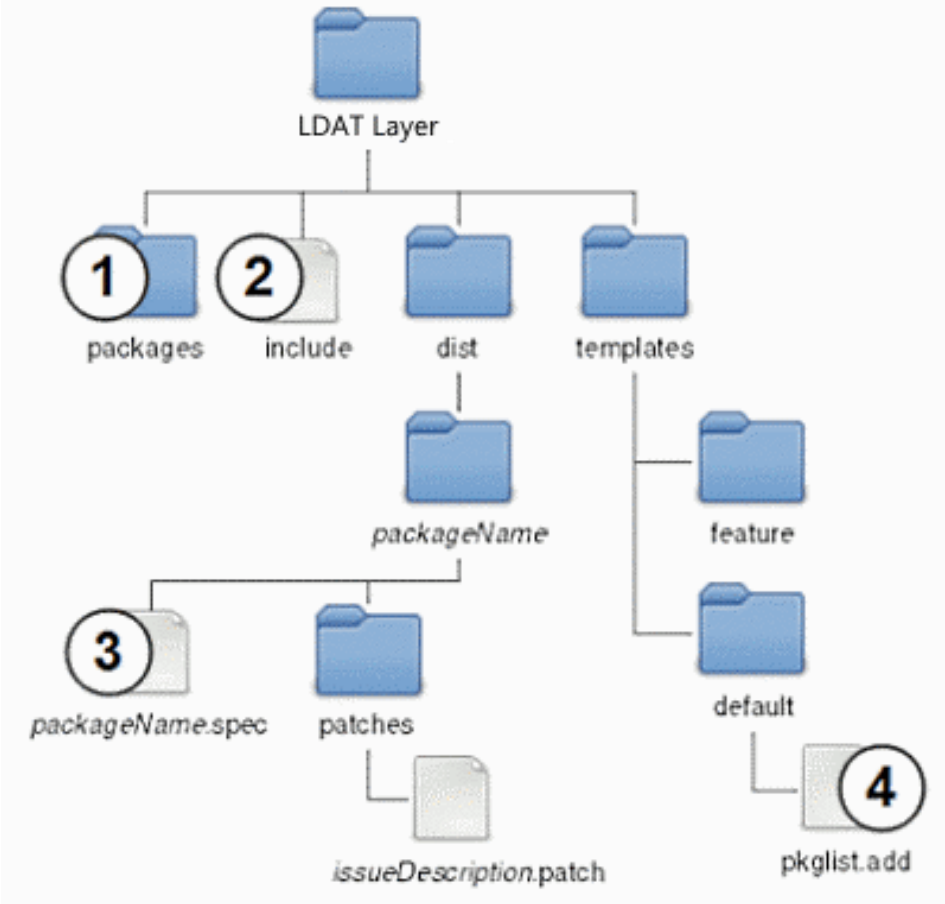
Layer Comparison

The layer requirements for BitBake are different from the requirements for LDAT. To import and/or modify a layer for use with Wind River Linux 7.x, it is important to understand these differences.

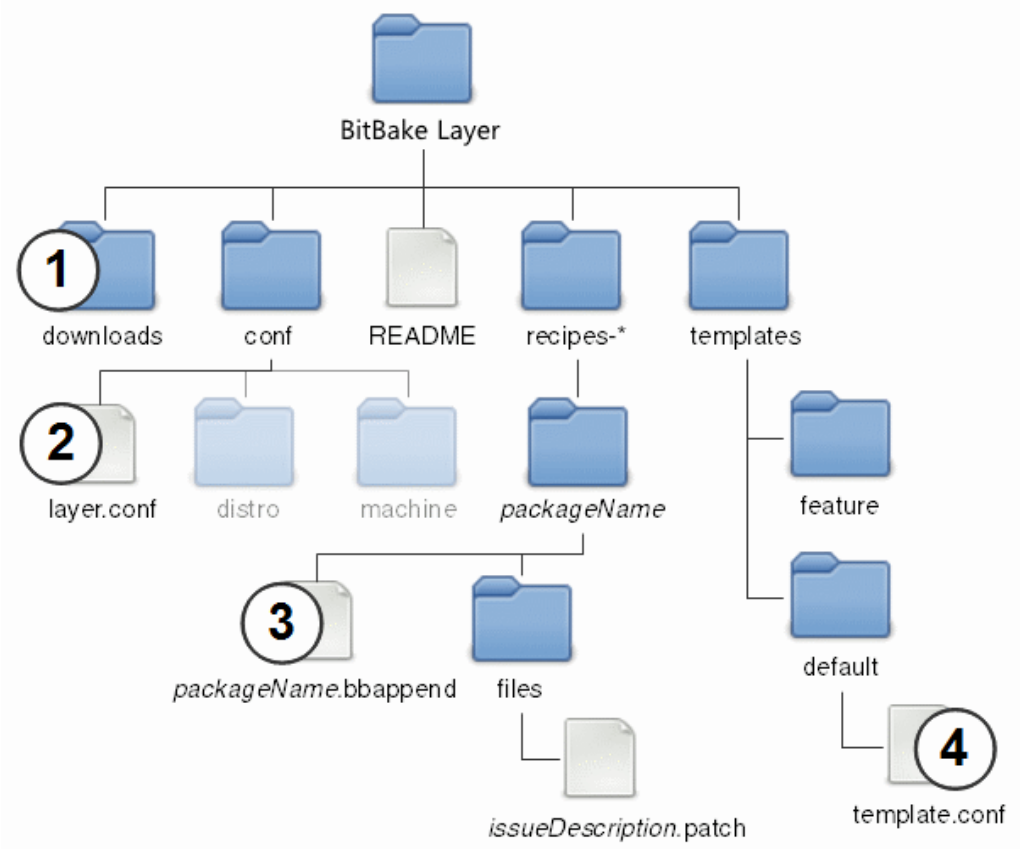
Layers in Wind River Linux 7.x are delivered as a git archive. You can not see what they contain in the install directory without a git viewing tool. When you create a platform project, the layer is checked out into the platform project's **layers** directory. This is a new subdirectory of a Wind River Linux 7.x platform project. Once in the project, the content of the layer is visible as regular files. It is this project-local version of the layer that is used to build the platform project, and it does not update when you update your install directory with an updated version of Wind River Linux 7.x. To synchronize your project's layers with updates to the install directory, navigate to the platform project's top-level directory in a console and type the following:

```
$ make reconfig
```

The Wind River Linux 4.x layer is structured in the following manner:



In comparison, the Wind River Linux 7.x layer is structured as follows:



In these diagrams, the numerical references help explain similar information that exists in a Wind River Linux 4.x layer when compared to a Wind River Linux 7.x layer. For example:

1. The **packages** directory in Wind River Linux 4.x contained the packages required by the layer to function as intended. In Wind River Linux 7.x, this directory is now named **downloads**.

   The packages in Wind River Linux were primarily in source RPM (SRPM) packages (***.src.rpm** or ***.srpm**), a build driven by ***.spec** files that is not supported in Wind River Linux 7.x. The preferred migration is to extract the tar file from the SRPM into the Wind River Linux download directory and examine the spec file, and translate it and any patches and additional build steps it provides into a BitBake recipe. Once you have identified all the changes, you can retain the ***.src.rpm** file and move that into the download directory instead, and use it as a base for the BitBake recipe to apply the changes to. For an example of examining a ***.spec** file in this manner, see *Updating a BitBake Recipe from a SRPM *.spec File* on page 29. This workflow is supported by the Import Package tool described in the *Wind River Linux User's Guide: Importing a SRPM Package from the Web*. Additionally, some translation of the spec file into a BitBake recipe is provided by the **rpmparse.py** migration script (*Migration Script Reference* on page 14).
2. The **include** file provides a sequential list of items to process in the layer. The **inlcude** file and other files with specific names placed in the root of a layer is replaced by the **layer.conf** file in Wind River Linux 7.x.
3. The `packageName`**.spec** file is used for SRPMS. It defines the build process for the package. With Wind River Linux 7.x, this information is defined in the `packageName`**.bbappend** file, and extended with files called `packageName`**.bbappend** and `packageName`**.inc**.
4. The **pkglist.add** file provides a list of packages to add to the build as part of including the layer in the build process. Typically, these packages are included in the layer's **packages** directory. With Wind River Linux 7.x, this information defined in the **pkglist.add**,

pkglist.delete file and other specially named files placed in the template's root directory are replaced by the layer's **templates/default/template.conf** file.

Understanding this information will help you create and/or customize layers for use with your Wind River Linux 7.x platform project.

4

Kernel and BSP Changes

About Kernel and BSP Changes

Understand the differences in the way that BSPs are packaged so you can use the information to customize existing BSPs as required.

Wind River Linux has always maintained the concept of the BSP as a distinct entity from the Linux kernel version. In Wind River Linux 4.x, the BSPs were packaged in a layer as an LDAT template. In Wind River Linux 7.x, BSPs are packaged as a BitBake layer, without a template.

This different approach to packaging precludes re-use from 4.x to 7.x, but so does the dramatic change in the Linux kernel version. In most cases the patches delivered in a Wind River Linux 4.x BSP are now obsolete with their functionality merged into the mainline kernel tree.

When migrating a BSP, you must first examine what your existing 4.x BSP provides in addition to the standard Wind River Linux 7.x BSP that is still relevant. Once you have that information, you can extract the delta (or difference in what is provided versus what is available) as a series of patches and configuration options and combine it with a custom Wind River Linux 7.x BSP.

5

Migrating Platform Projects

About Migrating Platform Projects and Elements

There are specific procedures and limitations for migrating Wind River Linux 4.x platform projects to 7.x, and elements of Wind River Linux 4.x platform projects to 7.x.

About 4.x to 7.x Element Migration

Because of significant changes to the build system, the kernel being updated from 2.6.34 to 3.14.22, and numerous package revision updates, a Wind River Linux 7.x platform project is quite different from a 4.x platform project. These differences are described in *Product Structure for Migration Purposes* on page 7, *About Build System Changes* on page 17, and *About Kernel and BSP Changes* on page 23.

While the end platform may provide the same functionality, the differences require a different approach to migrating platform projects. A straightforward migration from one complete platform project to another is not supported, but migrating the elements of a platform project that separate it from a default configuration and build is made possible with the migration tools provided by Wind River. These elements include project-specific layers, packages, and source applications. For information on installing these migration tools, see *Installing the Migration Tools* on page 13 and *Migration Script Reference* on page 14.

In addition to the migration scripts, for packages that are not based on a classic LDAT package built from LDAT Makefile, Wind River Linux 7.x platform projects support using the Package Importer tool. This tool imports packages into a platforms project's ***`projectDir`*/layers/local** directory layer, and adds the required BitBake recipe and directory structure necessary for integration.

After you debug the package in the local layer, you then move it into an external layer for use in subsequent projects. Using the Package Importer tool is described in *Wind River Linux User's Guide: About the Package Importer Tool (import-package)*.

If you just want to convert a series of patches to a package (that were applied by classic Makefile or spec file in Linux 4.x) and add them to a Wind River Linux 7.x layer, this process is described in the *Wind River Linux User's Guide: Introduction to Patching Userspace Packages*.

About File Systems

The file system contents of any specific build or package will not necessarily be the same in Wind River Linux 7.0, compared to the similarly named package in Wind River Linux 4 or the generally available package of the same name in a commercial Linux distribution. For example the **mke2fs** program in **e2fsprogs** utility package is not installed in target file system, the program only exits as a host build tool by default. This package is carefully partitioned in the Yocto build recipe to not install on the target to reduce its footprint. So you cannot create additional **ext2**, **ext3**, or **ext4** file systems with **mke2fs**, on the deployed target system.

You can change the behavior of this package and many others simply by adding the package to your target file system. The simplest method to do this would be to add a line to the file *`projectDir`***/layers/local/recipes-img/images/wrlinux-image-file-system.bbappend** such as

```
IMAGE_INSTALL += e2fsprogs
```

File system in the example refers to the rootfs configure option used to configure the platform project, for example, **wrlinux-image-glibc-small.bb**

Note that with this approach, other packages on which the package you are adding is dependent may also be installed on the target file system.

About the Migration Scripts for 4.3 Layers and Packages

You can use the migration tools from Wind River Online Support to assist you with migrating packages and layers created for Wind River Linux 4.x platform projects. Theses python scripts examine packages from previous versions of Wind River Linux and create a template for their inclusion in Wind River Linux 7.x. The **importlayer.py** and **importpackage.py** scripts examine the Makefile in a classic Wind River Linux 4.x package and create a translation of some of the elements found into a BitBake recipe. This is not a complete recipe, but if you are not familiar with BitBake syntax it will provide a significant time saving versus doing a manual translation. For information on recipes in general, see *Wind River Linux User's Guide: Recipes in the Development Environment*.

Currently, the script does not consider the ***.spec** file common in later Wind River Linux 4.x SRPM or spec packages; so the amount of assistance the tool provides for these packages is limited. Also the tool is intended for user packages, not kernel drivers or modules. Typical kernel packages will have patches and API dependencies on specific kernel versions, and are not suitable in most cases for an automated migration. This does not, however, mean that it cannot be done with the scripts, or that the scripts will not save you a considerable amount of time.

Both **importpackage.py** and **importlayer.py** tools create the directory structure for Wind River Linux 7.x layer and move files from the old packages into their new locations in the BitBake layer.

The tool helps with the semantics of migration; it does not check functionality or relevance of the derived recipe. For example; many Wind River Linux 4.x package Makefiles contain:

```
packagename_DEPENDS = glibc
```

This is translated into the BitBake recipe as:

```
DEPENDS = 'glibc'
```

But since there is no need to explicitly declare this dependency in Wind River Linux 7.x, and the equivalent package is now called **virtual/libc**, which represents the toolchain's EGLIBC; it could be updated to reflect that, or removed entirely. For example:

```
DEPENDS += "virtual/libc"
```

Similarly, the translation makes no attempt to supply required elements of a BitBake recipe that do not have an equivalent in the previous package format. So for example, for each package you will need to determine the package license location and enter it in the recipe:

```
LIC_FILES_CHKSUM = "file://README;beginline=42;endline=92;md5="
```

Then do a build of the package and append the displayed md5 sum in the error message to the recipe line. This process is explained in the *Wind River Linux User's Guide: Identifyng the LIC_FILES_CHKSUM Value*.

Platform Project Migration References

For a platform project-specific migration workflow, see *Platform Project Migration Workflow* on page 10.

To migrate elements of a previous Wind River Linux 4.x platform project, see *Using the Migration Tools to Migrate Layers* on page 33.

For information on migrating source applications, see *Migrating Source Applications* on page 36.

For information on updating project-specific BSP modifications, see *About Kernel and BSP Changes*.

Migrating a 6.0 Platform Project to 7.0

Migrating a platform project to 7.0 is a matter of configuring and building the platform project with the latest build system additions.

The following procedure requires a previously configured and built 6.0 platform project, updated to the latest RCPL. For additional information, see *Wind River Linux User's Guide:Updating the Product*.

Step 1 Obtain the **configure** script command used to create the 6.0 platform project.

This information is located in the ***projectDir*/config.log** file. For example:

```
configDir/configure --enable-board=qemux86-64 \
--enable-rootfs=glibc_small \
--enable-kernel=standard \
--enable-bootimage=iso
```

NOTE: In this example, *configDir* refers to the path to your Wind River Linux **configure** script; for example, **/home/user/WindRiver/wrlinux-6/wrlinux/**.

Step 2 Create a build directory for the 7.0 platform project and navigate to it.

```
$ mkdir -p qemux86-64-glibc-small-wrlx7 && cd qemux86-64-glibc-small-wrlx7
```

Step 3 Run the **configure** script from *Step 1* in the new project directory.

a) Copy the **configure** script command from *Step 1* into the terminal.

```
$ configDir/configure \
--enable-board=qemux86-64 \
--enable-rootfs=glibc_small \
--enable-kernel=standard \
--enable-bootimage=iso
```

b) Modify the path to the **configure** script to match the new Wind River Linux 7.0 installation (***configDir***) location and run the script.

```
$ new_configDir/configure \
--enable-board=qemux86-64 \
--enable-rootfs=glibc_small \
--enable-kernel=standard \
--enable-bootimage=iso
```

Typically, if you performed a default installation, all that is required is to change the *configDir* from **wrlinux-6** to **wrlinux-7**.

The **configure** script will configure the platform project to match the 6.0 project.

Step 4 Build the file system.

```
$ make
```

Once complete, the platform project will be built and symbolically linked to the new Wind River Linux 7.0 installation location. This includes updating the platform project to the latest kernel and Yocto revision released for Wind River Linux 7.0.

Step 5 Optionally verify your project is using the 7.0 installation.

Run the following command from the platform project directory:

```
$ ls -l git
```

The system should return:

```
$ lrwxrwxrwx 1 user user 40 Aug 16 17:29 git ->
/home/user/new_installDir/wrlinux-7/git
```

where *new_installDir* represents the path to the new installation location.

Step 6 Add applications from the 6.0 platform project to the 7.0 project.

This step assumes that applications for your platform project reside in the ***projectDir*/layers/local** directory. If your applications reside in a custom layer, you will have to ensure that layer is added to the ***projectDir*/layers.conf** file, and conform to the Yocto layer requirements.

a) Copy the application folders from the 6.0 ***projectDir*/layers/local** directory to the 7.0 directory.

b) Update the ***projectDir*/layers/local/recipes-img/images/wrlinux-image-*files-ystem*.bb** file.

In this recipe file, *file-system* represents the name of the root file system used to configure the platform project.

Locate the following lines at the end of the file:

```
#### END Auto Generated by configure ####
```

Add a line for each application using the **IMAGE_INSTALL** macro. The following example adds the Hello Linux (**hello**) application from the sample applications.

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

c) Save the file.

Step 7 Build the 7.0 platform project.

```
$ make
```

Step 8 Optionally launch the platform project on a simulator to run and test the application(s).

```
$ make start-qemu
```

NOTE: Depending on your application's requirements, you may need to test it on an actual hardware target. Refer to your BSP README for information on launching a hardware target.

Updating a BitBake Recipe from a SRPM *.spec File

Use this procedure as a guideline to update configuration and build information from a Wind River Linux 4.x SRPM package ***.spec** file into a 7.x BitBake recipe file.

In the following procedure, the **tcl.spec** file from a Wind River Linux 4.3 platform project is examined to use its contents as a basis for updating a recipe file for use with a Wind River Linux 7.x platform project.

Step 1 Import the package.

Use the Package Importer tool to import the **tcl_8.5.0** package into your Wind River Linux 7.x platform project. See the *Wind River Linux User's Guide: Importing a SRPM Package from the Web* for guidelines.

The result will be a new ***projectDir*/layers/local/recipes-local/tcl/tcl_8.5.0.bb** recipe file.

Step 2 Extract all of the files from the RPM, and move the tar file to the ***projectDir*/layers/local/downloads** directory.

Step 3 Open the **tcl.spec** file in an editor to examine it.

A copy of the file is provided at the end of this procedure for your convenience. In this file, you will see that four patches are supplied in the RPM file, and an additional five patches are supplied by Wind River in the **dist** directory.

Step 4 Move all of these patches to the ***projectDir*/layers/local/recipes-local/tcl/files** directory.

Step 5 Add a reference to these patch files to the ***projectDir*/layers/local/recipes-local/tcl/tcl_8.5.0.bb** file, using the URL from the spec file.

For example, you would add the following to the **SRC_URI** variable:

```
SRC_URI = "
http://downloads.sourceforge.net/sourceforge/tcl/tcl8.5.0-src.tar.gz; \
   file://tcl-8.5.0-autopath.patch \
   file://tcl-8.5.0-conf.patch \
   file://tcl-8.5.0-make.patch  \
   file://tcl-8.5.0-hidden.patch \
   file://tcl-8.5.0-fix-tclConfig.patch \
   file://tcl-8.5.0-rm-exec-binaries.patch \
   file://tcl8.4.15-lib-perm.patch \
   file://tcl-8.5.0-add-platform.patch \
   file://tcl-no-ld-lib-path.patch \
   file://tcl8.4.15-lib-perm.patch \
   file://tcl-fix-stack-direction-check.patch \
"
```

Step 6 Notice that the spec file includes: **Buildrequires: autoconf**.

If this is present in a spec file, add **inherit autotools** to the ***projectDir*/layers/local/recipes-local/tcl/tcl_8.5.0.bb** recipe file.

Step 7 Copy and/or modify the **Summary** and description from the spec file and add it to your recipe file.

```
SUMMARY = "Tcl scripting language development environment"
```

Step 8 Optionally, copy the **URL** in the spec file and use it for the **HOMEPAGE** variable in your recipe.

Step 9 Use the **License** value in the spec file to update the **LICENSE** variable in your recipe file.

Typically, the value in the spec file will translate into a valid BitBake license value, but not in this case. You will want to add a valid BitBake license.

Step 10 Examine the **%build** macro in the spec file to see if it is doing anything unusual.

In this case, it is moving to the **unix** directory to build, so you will need custom build rules in the BitBake recipe to accomplish the same thing. For example:

```
EXTRA_OECONF = "--enable-threads"
do_configure () {
    cd unix
     autotools_do_configure
}

EXTRA_OEMAKE = "-C unix "
```

Step 11 Examine the **%install** macro in the spec file to duplicate any non-standard steps in this stage of the build.

In this example, we use a BitBake **_append** modifier for our rule, so that the **autotools** version of **do_install** we inherited will execute first.

```
do_install_append() {
    ln -s tclsh8.5 ${D}/usr/bin/tclsh
    # for linking with -lib${PN}
    ln -s lib${PN}8.5.so ${D}/usr/lib/lib${PN}.so

    mkdir -p ${D}/usr/lib/${PN}8.5
    mkdir -p ${D}/usr/include/${PN}-private/generic
    mkdir -p ${D}/usr/include/${PN}-private/unix
```

```
    find generic unix -name "*.h" -exec cp -p '{}' ${D} usr/include/${PN}-private/'{}'
';'
    ( cd ${D}/usr/include
        for i in *.h ; do
            [ -f ${D}/usr/include/${PN}-private/generic/$i ] && ln -sf ../../$i $
{D}/usr/include/${PN}-private/generic;
        done
     )
}
```

Step 12 Examine the **%files** macro in the spec file to see if there are any files that are not included in the default mask for installation, and append them to the equivalent macro in your recipe.

```
FILES_${PN} += "\
    ${datadir}/* \
    ${datadir}/tcl8/8.5/* "

FILES_${PN}-dev += "\
    ${libdir}/${PN}Config.sh "
```

This completes the comparison of a ***.spec** file and the migration of information from it to a BitBake recipe file.

The **tcl.spec** file used for this procedure is provided below:

```
%define majorver 8.5

Summary: Tcl scripting language development environment
Name: tcl
Version: %{majorver}.0
Release: 6_WR%{?_ldat_rel}
Epoch: 1
License: Attribution-style
Group: Development/Languages
URL: http://tcl.sourceforge.net/
Source0: http://downloads.sourceforge.net/sourceforge/tcl/tcl%{version}-src.tar.gz
BuildRoot: %{_tmppath}/%{name}-%{version}-%{release}-root-%(%{__id_u} -n)
Buildrequires: autoconf
Provides: tcl(abi) = %{majorver}
Obsoletes: tcl-tcldict <= 8.5.2
Provides: tcl-tcldict = 8.5.2
Patch0: tcl-8.5.0-autopath.patch
Patch1: tcl-8.5.0-conf.patch
Patch2: tcl-8.5.0-make.patch
Patch3: tcl-8.5.0-hidden.patch

# WRLinux patches
Patch500: tcl-8.5.0-fix-tclConfig.patch
Patch501: tcl-8.5.0-rm-exec-binaries.patch
Patch502: tcl8.4.15-lib-perm.patch
Patch503: tcl-8.5.0-add-platform.patch
Patch504: tcl-no-ld-lib-path.patch
Patch505: tcl-fix-stack-direction-check.patch

%description
The Tcl (Tool Command Language) provides a powerful platform for
creating integration applications that tie together diverse
applications, protocols, devices, and frameworks. When paired with the
Tk toolkit, Tcl provides a fastest and powerful way to create
cross-platform GUI applications. Tcl can also be used for a variety
of web-related tasks and for creating powerful command languages for
applications.

%package devel
Summary: Tcl scripting language development environment
Group: Development/Languages
Requires: %{name} = %{epoch}:%{version}-%{release}

%description devel
The Tcl (Tool Command Language) provides a powerful platform for
creating integration applications that tie together diverse
applications, protocols, devices, and frameworks. When paired with the
```

```
Tk toolkit, Tcl provides a fastest and powerful way to create
cross-platform GUI applications. Tcl can also be used for a variety
of web-related tasks and for creating powerful command languages for
applications.

The package contains the development files and man pages for tcl.

%prep
%setup -q -n %{name}%{version}
chmod -x generic/tclThreadAlloc.c

%patch0 -p1 -b .autopath
%patch1 -p1 -b .conf
%patch2 -p1 -b .make
%patch3 -p1 -b .hidden

# WRLinux patches
%patch500 -p1 -b .fix
%patch501 -p1 -b .exec-binaries
%patch502 -p1 -b .lib-perm
%patch503 -p1 -b .add-platform
%patch504 -p1 -b .no-ld-lib-path
%patch505 -p1 -b .tcl-fix-stack-direction-check

%build
# WRLinux set values for configure and compilation
%configure_target
pushd unix
autoconf
# WRLinux avoid to check if strtod is ok because of cross-compile
tcl_cv_strtod_buggy=ok; export tcl_cv_strtod_buggy
%configure --enable-threads
make %{?_smp_mflags} TCL_LIBRARY=%{_datadir}/%{name}%{majorver}

# don't run "make test" by default
#  make test

%install
# WRLinux set values for configure and compilation
%configure_target
rm -rf $RPM_BUILD_ROOT
make install -C unix INSTALL_ROOT=$RPM_BUILD_ROOT TCL_LIBRARY=%{_datadir}/%{name}%
{majorver}

ln -s tclsh%{majorver} $RPM_BUILD_ROOT%{_bindir}/tclsh

# for linking with -lib%{name}
ln -s lib%{name}%{majorver}.so $RPM_BUILD_ROOT%{_libdir}/lib%{name}.so

mkdir -p $RPM_BUILD_ROOT/%{_libdir}/%{name}%{majorver}

mkdir -p $RPM_BUILD_ROOT/%{_includedir}/%{name}-private/{generic,unix}
find generic unix -name "*.h" -exec cp -p '{}' $RPM_BUILD_ROOT/%{_includedir}/%{name}-
private/'{}' ';'
( cd $RPM_BUILD_ROOT/%{_includedir}
    for i in *.h ; do
            [ -f $RPM_BUILD_ROOT/%{_includedir}/%{name}-private/generic/$i ] && ln -
sf ../../$i $RPM_BUILD_ROOT/%{_includedir}/%{name}-private/generic ;
    done
)

# remove buildroot traces
sed -i -e "s|$PWD/unix|%{_libdir}|; s|$PWD|%{_includedir}/%{name}-private|"
$RPM_BUILD_ROOT/%{_libdir}/%{name}Config.sh
rm -rf $RPM_BUILD_ROOT/%{_datadir}/%{name}%{majorver}/tclAppInit.c
rm -rf $RPM_BUILD_ROOT/%{_datadir}/%{name}%{majorver}/ldAix

%files
%defattr(-,root,root,-)
%{_bindir}/tclsh*
%{_datadir}/%{name}%{majorver}
%{_datadir}/%{name}8/%{majorver}/*
%{_libdir}/lib%{name}%{majorver}.so
%{_mandir}/man1/*
%{_mandir}/man3/*
%{_mandir}/mann/*
```

```
%dir %{_libdir}/%{name}%{majorver}
%doc README changes
%doc license.terms

%files devel
%defattr(-,root,root,-)
%{_includedir}/*
%{_libdir}/lib%{name}stub%{majorver}.a
%{_libdir}/lib%{name}.so
%{_libdir}/%{name}Config.sh
```

Using the Migration Tools to Migrate Layers

The migration tools help you migrate specific elements of your platform projects to the Wind River Linux 7.x project format, as described in this section.

This procedure requires a layer from a Wind River Linux 4.x platform project to migrate, and that you have downloaded and installed the migration tool scripts. See *Installing the Migration Tools* on page 13.

All commands assume you are working in the home directory and assume a functional installation of Wind River Linux 4.x and Wind River Linux 7.x.

Step 1 Migrate the layer.

Run the following command to use the **importlayer.py** script to migrate an existing 4.x layer to a new layer directory.

```
$ ./ldat-tools/importlayer.py \
installDir/wrlinux-4/layers/wrll-userspace/lsb new_layer
```

The script will run and convert the layer. Any conversion errors will display to provide information on any issues that may require manual conversion.

Step 2 Display the contents of the layer:

```
$ ls new_layer

conf  recipes  templates
```

Notice that the main folders required by a BitBake layer are present.

Step 3 Explore the structure of the recipe subdirectory:

```
$ tree new_layer/recipes
```

and explore the contents:

```
new_layer/recipes/
├── jpeg-6b
│   ├── files
│   │   └── libjpeg.spec
│   ├── importlog.txt
│   └── jpeg-6b_6b.bb
├── lsb-dist-checker
│   ├── files
│   │   └── lsb-dist-checker.spec
│   ├── importlog.txt
│   └── lsb-dist-checker_4.0.1.9-2.bb
├── lsb-setup
│   ├── files
│   ├── importlog.txt
│   └── lsb-setup_4.0.1-1.bb
└── wrs-lsb
    ├── files
    │   ├── redhat-lsb.spec
    │   └── rpm_patches.list
    ├── importlog.txt
    └── wrs-lsb_4.0-2.fc13.bb
```

Notice that recipe files (***.bb**) are created for each application that comprises the **lsb** layer. We do not expect this example to be functional, as we have chosen several packages that include ***.spec** files. But, the migrated layer has enough content that it can be included in Wind River Linux 7.x platform project, and the migration debugged.

6

Migrating User Space

About Application Development Changes

You should be aware of and understand the differences in the way that the toolchain and sysroots are managed so you can use the information to customize existing BSPs as required.

About the Toolchain

With Wind River Linux 4.x, the toolchain, and support for 32- and 64-bit libraries (multilibs), was maintained using the templates located in ***installDir*/wrlinux-4/layers/wrll-toolchain-version/arch/templates** directory. This included templates for each supported architecture, cpu, and multilib support. This setup made it possible for you to specify a toolchain version as part of using the platform project **configure** command.

With Wind River Linux 7.x, changes to the build, to add multilib support for example, are made by editing the platform project's **local.conf** file. For additional information, see the *Wind River Linux User's Guide: About Sysroots and Multilibs*.

SDK and Sysroots Comparison

With Wind River Linux 4.x, you had to export the location of the platform project's sysroots to your development host to facilitate application development. In a standard installation, exported sysroots were located in the ***installDir*/wrlinux-4/sysroots** directory, with source sysroots located in the platform project's ***projectDir*/host-cross/arch** directory. Whenever you needed to export sysroots for application development, you ran the **make export-sysroot** command from the *projectDir*.

With Wind River Linux 7.x, you have the choice of exporting the sysroot, or a software development kit (SDK). The SDK includes the exported toolchain and supports application

development on a Windows host. For additional information, see the *Wind River Linux User's Guide: Exporting the SDK*.

Migrating Source Applications

To include your Wind River Linux 4.x applications in a Wind River Linux 7.x platform project, you can import the source using the Package Importer tool.

Because of the differences between build systems, and the requirements of the BitBake build configuration and recipe definitions, simply importing a previously built application project is not possible. To include your application, you must import the source into your Wind River Linux 7.x platform project.

This process is explained in the *Wind River Linux User's Guide: Importing a SRPM Package from the Web*. Once you launch the tool as explained in the procedure, you will select **Source Package** or **Application Source Tree**, depending on your application's source. Once the import is completed, your application package will be added to the ***projectDir*/layers/local/recipes-local** directory, and will include a BitBake recipe to define application configuration and build requirements.

7

Leveraging Workbench

Workbench Migration Tools

You can use Workbench to help simplify your platform project migration process.

New Platform Project Configure Tool

As explained throughout this guide, the migration process begins when you create a new Wind River Linux 7.x platform project, based on the configuration used for your Wind River Linux 4.x platform project. To help simplify this process, you can use the Workbench New Platform Project Configure tool.

To launch the tool, right-click in the Project Explorer view and select **New** > **Project** > **Wind River Linux** > **Wind River Linux Platform Project**. For specific information on configuring platform projects in Workbench, see the *Wind River Workbench by Example Guide (Linux Version): Developing Platform Project Images*.

Package Importer Tool

The Package Importer tool takes a source package and converts it into a BitBake-ready package, placing the contents in the ***projectDir*/layers/local** directory for inclusion in the platform project build and further development. For an example of this process, see *Migrating Source Applications* on page 36 and the *Wind River Linux User's Guide: About the Package Importer Tool (import-package)*.

8

Reference

Wind River Linux Documentation

Understanding the full range of documentation available is invaluable to helping you find the information you need to work effectively in Wind River Linux.

All Wind River Systems documentation is available in the Knowledge Library *https://knowledge.windriver.com*.

Additionally, much of the documentation is available through the start menu of the installation host, for example under **Applications** > **Wind River** > **Documentation**in the Gnome desktop for Linux.

Most of the documentation is available online as PDFs or HTML accessible through Wind River Workbench online help. Links to the PDF files are available by selecting **Wind River** > **Documentation** from your operating system start menu.

From a Workbench installation you can view the documentation in a Web browser locally (**Help** > **Help Contents**).

The documentation is also available below your installation directory (called `installDir`) through the command line as follows:

- PDF Versions—To access the PDF, point your PDF reader to the ***.pdf** file, for example:

  ***installDir*/docs/extensions/eclipse/plugins/com.windriver.ide.doc.wr_linux_7/wind_river_linux_users_guide_70/wind_river_linux_users_guide_70.pdf**
- HTML Versions—To access the HTML, point your web browser to the **index.html** file, for example:

  ***installDir*/docs/extensions/eclipse/plugins/com.windriver.ide.doc.wr_linux_7/wind_river_linux_users_guide_70/html/index.html**

External Documentation

Use external documentation to enhance your developer knowledge and capabilities. The following table lists open source documentation related to the Yocto Project and Wind River Linux:

Table 1 **Wind River Linux External Documentation**

External Information source	Location
The Yocto Project	Online: *http://www.yoctoproject.org*
BitBake User Manual	Online: *http://www.yoctoproject.org/docs/1.7/bitbake-user-manual/bitbake-user-manual.html*
OpenEmbedded Core (OE-Core)	Online: *http://www.openembedded.org/wiki/OpenEmbedded-Core*
QEMU	Online: *http://wiki.qemu.org*
GNU Toolchain Documentation	***installDir*/wrlinux-7/layers/binary-toolchain-4.9-3/share/doc/wrs-linux-arm-wrs-linux-gnueabi**—for ARM-based target systems ***installDir*/wrlinux-7/layers/binary-toolchain-4.9-3/share/doc/wrs-linux-aarch64-wrs-linux-gnu**—for 64-bit ARM-based target systems ***installDir*/wrlinux-7/layers/binary-toolchain-4.9-3/share/doc/wrs-linux-i686-wrs-linux-gnu**—for x86-based target systems ***installDir*/wrlinux-7/layers/binary-toolchain-4.9-3/share/doc/wrs-linux-mips-wrs-linux-gnu**—for MIPS-based target systems
GNU Toolchain Documentation	***installDir*/wrlinux-7/layers/binary-toolchain-4.9-3/share/doc/wrs-linux-arm-wrs-linux-gnueabi**—for ARM-based target systems ***installDir*/wrlinux-7/layers/binary-toolchain-4.9-3/share/doc/wrs-linux-aarch64-wrs-linux-gnu**—for 64-bit ARM-based target systems ***installDir*/wrlinux-7/layers/binary-toolchain-4.9-3/share/doc/wrs-linux-i686-wrs-linux-gnu**—for x86-based target systems ***installDir*/wrlinux-7/layers/binary-toolchain-4.9-3/share/doc/wrs-linux-mips-wrs-linux-gnu**—for MIPS-based target systems

Index