PetaLinux Tools Documentation

PetaLinux Command Line Reference

UG1157 (v2014.4) November 25, 2014

XILINX®

Revision History

Date	Version	Notes
11/25/2014	2014.4	Initial public release for PetaLinux Tools 2014.4

Online Updates

Please refer to the PetaLinux v2014.4 Master Answer Record (Xilinx Answer Record #55776) for the latest updates on PetaLinux Tools usage and documentation.

XILINX

Table of Contents

Introduction

PetaLinux is a development and build environment which automates many of the tasks required to successfully boot embedded Linux on Xilinx AP SoC's and FPGA's. This document contains detailed information about the various tools that comprise the PetaLinux environment. Each chapter of this document corresponds to a specific tool.

PetaLinux Tools Overview

There are six independent tools that make up the PetaLinux design flow. These tools are:

- petalinux-boot
- petalinux-build
- petalinux-config
- petalinux-create
- petalinux-package
- petalinux-util

In most cases, the individual PetaLinux tools are flexible such that the specific options passed to the tools present the user with a unique usage model compared to other options for the same tool. This document refers to these usage models as "workflows."

For the purposes of this document, command line arguments that behave as a modifier for a workflow are referred to as "options." When options can accept user-specified values, these values are shown in italics. In some cases, omitting the user-specified value may result in a built-in default behavior. See the "Default Value" column in the tables for details about relevant default values.

If an option is mandatory for the tool to operate properly, it is denoted as "REQUIRED." If an option is entirely optional or optional for a specific workflow, it is denoted as "OPTIONAL."

Since examples are the key to understanding the PetaLinux workflows, usage examples are provided for each PetaLinux workflow.

Design Flow Overview

In general, the PetaLinux tools follow a sequential workflow model. The table below provides an example design workflow demonstrating the order in which tasks should be completed and the corresponding tool or workflow for that task.

Design Flow Step	Tool / Workflow
Hardware Platform Creation	Vivado
Create PetaLinux Project	petalinux-create -t project

Initialize PetaLinux Project	petalinux-config --get-hw-description
Configure System-Level Options	petalinux-config
Create User Components	petalinux-create -t COMPONENT
Configure the Linux Kernel	petalinux-config -c kernel
Configure the Root Filesystem	petalinux-config -c rootfs
Build the System	petalinux-build
Test the System	petalinux-boot --qemu
Deploy the System	petalinux-package

The petalinux-boot Tool

The `petalinux-boot` tool boots the specified Linux system image files. This tool provides two distinct workflows. In the `petalinux-boot --jtag` workflow, the system image files are downloaded and booted on a physical board via a JTAG cable connection. In the `petalinux-boot --qemu` workflow, the system image files are loaded and booted via the QEMU software emulator. Specifying either the `--jtag` or `--qemu` option for the `petalinux-boot` tool is mandatory.

By default, the `petalinux-boot` tool loads files from the `<PROJECT>/images/linux/` directory.

The table below details the command line options that are common to all `petalinux-boot` workflows.

Option	Functional Description	Value Range	Default Value
`--jtag`	REQUIRED. Use the JTAG workflow. Mutually exclusive with the QEMU workflow.	none	none
`--qemu`	REQUIRED. Use the QEMU workflow. Mutually exclusive with the JTAG workflow.	none	none
`--prebuilt`	OPTIONAL. Boot a prebuilt image.	1 (bitstream/FSBL) 2 (U-Boot) 3 (Linux Kernel)	none
`--boot-addr` *BOOTADDR*	OPTIONAL. Boot address.	user-specified	none
`-i,--image` *IMAGEPATH*	OPTIONAL. Linux kernel image.	user-specified	`vmlinux`(Zynq) `image.elf`(MicroBlaze)
`--uboot`	OPTIONAL. Specify U-Boot elf binary.	user-specified	`u-boot.elf`
`--kernel`	OPTIONAL. Specify Linux kernel binary.	user-specified	`zImage`(Zynq) `image.elf`(MicroBlaze)
`-v,--verbose`	OPTIONAL. Displays additional output messages.	none	none
`-h,--help`	OPTIONAL. Displays tool usage information.	none	none

NOTE: `--prebuilt 1` is not valid for the QEMU workflow.

Details for the petalinux-boot --jtag Workflow

This workflow boots the MicroBlaze/Zynq system with a PetaLinux image via a JTAG connection.

The following table contains details of options specific to the JTAG boot workflow.

Option	Functional Description	Value Range	Default Value
`--extra-xmd` *COMMAND*	OPTIONAL. Additional XMD commands to run during boot.	user-specified	none
`--xmd-conn` *COMMAND*	OPTIONAL. Special XMD connection command to run prior to boot. May be repeated.	user-specified	none
`--load-addr` *LOADADDR*	OPTIONAL. Address to load the image.	user-specified	none
`--regdata` *REGDATA*	OPTIONAL. Additional register data.	user-specified	none
`--tcl` *OUTPUTFILE*	OPTIONAL. Log JTAG Tcl commands used for boot.	user-specified	none
`--targetcpu` *CPUID*	OPTIONAL. Run on specified CPU.	0 to n-1	0
`--fpga`	OPTIONAL. Program pre-built bitstream. Assumes the presence of the `pre-built` directory. See petalinux-package for more details.	user-specified	`download.bit`
`--bitstream` *BITSTREAM*	OPTIONAL. Program specified bitstream.	user-specified	none

NOTE: This workflow may not work as expected when executed within a virtual machine.
NOTE: The `--fpga` option looks for `download.bit` in `<PROJECT>/pre-built/linux/implementation` by default.

Example Usage of the petalinux-boot --jtag Workflow

The following examples demonstrate proper usage of the `petalinux-boot --jtag` workflow.

- Download and boot a pre-built bitstream (and FSBL for Zynq) via JTAG to a physical board.
  ```
  $ petalinux-boot --jtag --prebuilt 1
  ```
- Download and boot a pre-built U-Boot elf via JTAG to a physical board.
  ```
  $ petalinux-boot --jtag --prebuilt 2
  ```
- Download and boot a built kernel image via JTAG to a physical board.
  ```
  $ petalinux-boot --jtag --kernel
  ```
  For MicroBlaze, it will download `image.elf`
  For Zynq, it will download `zImage` and `system.dtb`

Details for the petalinux-boot --qemu Workflow

This workflow boots the MicroBlaze/Zynq system with a PetaLinux image via the QEMU emulator. Many QEMU options require superuser (root) access to operate properly. The `--root` option enables ROOT MODE and will prompt the user for sudo credentials.

The following table contains details of options specific to the QEMU boot workflow.

Option	Functional Description	Value Range	Default Value
`--dtb` *DTBFILE*	OPTIONAL. Use a specified device tree file	user-specified	`system.dtb`
`--dhcpd` *TOGGLE*	OPTIONAL. Enable or disable dhcpd daemon. ROOT MODE only.	enable disable	enable
`--iptables-allowed`	OPTIONAL. Allow Linux iptables commands. ROOT MODE only.	none	disabled
`--net-intf` *INTERFACE*	OPTIONAL. Network interface on the host to bridge with the QEMU subnet. ROOT MODE only.	user-specified	eth0
`--subnet` *SUBNET*	OPTIONAL. Specify a specific sub-net for the networking interface. ROOT MODE ONLY.	user-specified	192.168.10.1/24
`--root`	OPTIONAL. Enable QEMU ROOT MODE.	none	disabled
`--qemu-args`	OPTIONAL. Pass additional arguments to the QEMU engine.	user-specified	none

Example Usage of the petalinux-boot --qemu Workflow

The following examples demonstrate proper usage of the `petalinux-boot --qemu` workflow.

- Load and boot a pre-built U-Boot elf via QEMU.
  ```
  $ petalinux-boot --qemu --prebuilt 2
  ```
- Load and boot a pre-built U-Boot elf via QEMU in root mode.
  ```
  $ petalinux-boot --qemu --root --prebuilt 2
  ```

The petalinux-build Tool

The `petalinux-build` tool builds either the entire embedded Linux system or a specified component of the Linux system. While the tool provides a single workflow, the specifics of its operation can be dictated via the `petalinux-build -c` and `petalinux-build -x` options.

The table below outlines the valid options for the `petalinux-build` tool.

Option	Functional Description	Value Range	Default Value
`-p,--project` *PROJECT*	OPTIONAL. PetaLinux project directory path.	user-specified	current directory
`-c, --component` *COMPONENT*	OPTIONAL. Build specified component. "all" is the implied default.	all bootloader kernel u-boot rootfs	all
`-x, --execute` *MAKE-TARGET*	OPTIONAL. Execute specified GNU `Makefile` command.	build clean distclean all install package	all
`--makeenv` *ENVARS*	OPTIONAL. Additional GNU `make` environment variables.	none	none
`-v,--verbose`	OPTIONAL. Displays additional output messages.	none	none
`-h,--help`	OPTIONAL. Displays tool usage information.	none	none

Details for the -c Option

The `-c` option builds the specified component of the embedded system. When no components are specified, the `petalinux-build` tool operates on the project as a whole. User-created components for the root filesystem can be built by targeting those components by name (e.g., with `-c rootfs/<COMPONENT-NAME>`).

The list below summarizes the available components that can be targeted with this workflow.

- **all** - Build all components of the project and copy them into `<PROJECT>/images/`. This is the default behavior with no options.
- **bootloader** - Build only the bootloader elf image and copy it into `<PROJECT>/images/`. For Zynq devices, this is FSBL. For MicroBlaze CPUs, this is FS-BOOT.
- **device-tree** - Build only the device-tree DTB file and copy it into `<PROJECT>/images/`.
- **kernel** - Build only the Linux kernel image and copy it into `<PROJECT>/images/`.
- **rootfs** - Build only the root filesystem image and copy it into `<PROJECT>/images/`.
- **u-boot** - Build only the U-Boot elf image and copy it into `<PROJECT>/images/`.

Details for the -x Option

The `-x` option allows the user to specify standard GNU Makefile options to the `petalinux-build` tool to control how the specified components are manipulated.

The list below summarizes the available Makefile commands that can be used with this option.

- **clean** - Clean build data for the target component. Must be used with the `-c` option. Not valid with `-c all`.
- **distclean** - Clean the build area. This removes the `<PROJECT>/build/` directory.
- **all** - Build the entire PetaLinux project and copy output into `<PROJECT>/images/`. This is the same as `-c all`.
- **build** - Build the entire PetaLinux project but do not copy into `<PROJECT>/images/`.
- **install** - Build (if needed) and copy device tree, U-Boot, and Linux kernel binaries into the staging location in `<PROJECT>/build/`.
- **package** - Generate FIT image `image.ub` from the current contents of build area and copy into `<PROJECT>/images/`.

Example Usage of the petalinux-build Workflow

The following examples demonstrate proper usage of the `petalinux-build` workflow.

- Clear the build area of the PetaLinux project for archiving as a BSP or for revision control. This example retains the `images` directory of the project.
  ```
  $ petalinux-build -x distclean
  ```
- Clean all build collateral from the U-Boot component of the PetaLinux project.
  ```
  $ petalinux-build -c u-boot -x clean
  ```
- Create an updated FIT image from the current contents of the build area.
  ```
  $ petalinux-build -x package
  ```
- Build the entire PetaLinux project.
  ```
  $ petalinux-build -c all
  ```

The petalinux-config Tool

The `petalinux-config` tool allows the user to customize the specified project. This tool provides two seperate workflows. In the `petalinux-config --get-hw-description` workflow, a project is initialized or updated to reflect the specified hardware configuration. In the `petalinux-config -c COMPONENT` workflow, the specified component is customized using a menuconfig interface.

The table below details the available options for the `petalinux-config` tool.

Option	Functional Description	Value Range	Default Value
`--get-hw-description=`*PATH*	REQUIRED. Initialize or update the hardware configuration for the PetaLinux project. Mutually exclusive with `-c`.	user-specified	none
`-c,--component` *COMPONENT*	REQUIRED. Configured the specified system component. Mutually exclusive with `--get-hw-description`.	none kernel rootfs	none
`--searchpath`	OPTIONAL. Modify the search environment used by PetaLinux.	--prepend --append --replace --print --delete	none
`--defconfig=`*DEFCONFIG*	OPTIONAL. Linux kernel only. Use the specified defconfig file to initialize the Linux kernel configuration.	user-specified	none
`--oldconfig`	OPTIONAL. Restore the configuration for the specified component to a prior version. Without `-c`, restores system-level configuration.	none	none
`-v,--verbose`	OPTIONAL. Displays additional output messages.	none	none
`-h,--help`	OPTIONAL. Displays tool usage information.	none	none

Details for the --searchpath Option

The `petalinux-config --searchpath` option allows the user to control how the `--searchpath` option manipulates the SEARCHPATH environment PetaLinux uses for referencing components. This option must be used with one of the modifiers detailed below. Multiple modifiers may be used simultaneously. The table below details the available modifiers and their impact on the SEARCHPATH used by PetaLinux. The path `<PROJECT>/components` is always the highest priority while `<PETALINUX-INSTALL-DIR>/components` is always the lowest priority.

- **--prepend *PATH*** - prepend *PATH* to project external search path
- **--append *PATH*** - append *PATH* to project external search path
- **--replace *PATH*** - replace the project external search path (if any) with *PATH*
- **--print** - print full project search path
- **--delete** - delete project external search path

Details for the petalinux-config --get-hw-description Workflow

The `petalinux-config --get-hw-description` workflow allows the user to initialize or update a PetaLinux project with hardware-specific information from the specified Vivado hardware project. The components affected by this process may include FSBL configuration, U-Boot options, Linux kernel options, and the Linux device tree configuration. This workflow should be used carefully to prevent accidental and/or unintended changes to the hardware configuration for the PetaLinux project. The path used with this workflow is the directory that contains the HDF file rather than the full path to the HDF file itself. This entire option can be omitted if run from the directory that contains the HDF file.

Example Usage of the petalinux-config --get-hw-description Workflow

The following examples demonstrate proper usage of the `petalinux-config --get-hw-description` workflow.

- Initialize a PetaLinux project using an externally sourced device tree generator tool.
  ```
  $ petalinux-config --get-hw-description=<PATH-TO-HDF> --searchpath --prepend
  <PATH-TO-DTG>
  ```
- Initialize a PetaLinux project from within the directory containing an HDF.
  ```
  $ petalinux-config --get-hw-description -p <PATH-TO-PETALINUX-PROJECT>
  ```
- Initialize a PetaLinux project from a neutral location.
  ```
  $ petalinux-config --get-hw-description=<PATH-TO-HDF> -p <PATH-TO-PETALINUX-
  PROJECT>
  ```

Details for the petalinux-config -c COMPONENT Workflows

The `petalinux-config -c COMPONENT` workflows allow the user to use a standard menuconfig interface to control how the embedded Linux system is built. When `petalinux-config` is executed with no other options, it launches the system-level or "generic" menuconfig. This interface allows the user to specify information such as the desired boot device or metadata about the system such as default hostname. The `petalinux-config -c kernel` and `petalinux-config -c rootfs` workflows launch the menuconfig interfaces for customizing the Linux kernel and root filesystem, respectively.

The `--oldconfig` option allows the user to restore a prior configuration. Old configurations have the filename `CONFIG.old` within the directory containing the specified component.

NOTE: Xilinx technical support supports Xilinx-specific options and/or customizations in the Linux kernel rather than general Linux kernel configuration.

Example Usage of the petalinux-config -c COMPONENT Workflow

The following examples demonstrate proper usage of the `petalinux-config -c COMPONENT` workflow.

- Start the menuconfig for the system-level configuration.
  ```
  $ petalinux-config
  ```
- Load the previous configuration for the root filesystem.
  ```
  $ petalinux-config -c rootfs --oldconfig
  ```
- Load the Linux kernel configuration with a specific default configuration.
  ```
  $ petalinux-config -c kernel --defconfig=xilinx_zynq_base_trd_defconfig
  ```

The petalinux-create Tool

The `petalinux-create` tool creates objects that are part of a PetaLinux project. This tool provides two separate workflows. In the `petalinux-create -t project` workflow, the tool creates a new PetaLinux project directory structure. In the `petalinux-create -t COMPONENT` workflow, the tool creates a component within the specified project.

These workflows are executed with `petalinux-create -t project` or `petalinux-create -t COMPONENT`, respectively.

The table below details the command line options that are common to all `petalinux-create` workflows.

Option	Functional Description	Value Range	Default Value
`-t,--type` *TYPE*	REQUIRED. Specify the *TYPE* of object to create.	project apps libs modules generic	none
`-n,--name` *NAME*	REQUIRED. Create object with the specified *NAME*. This is optional when creating a project from a BSP source.	user-specified	none
`-p,--project` *PROJECT*	OPTIONAL. PetaLinux project directory path.	user-specified	current directory
`--force`	OPTIONAL. Overwrite existing files on disk.	none	none
`-h,--help`	OPTIONAL. Display usage information.	none	none

Details for the petalinux-create -t project Workflow

The `petalinux-create -t project` workflow creates a new PetaLinux project at the specified location with the specified name. By default, this workflow creates a new Zynq AP SoC-based project in the current working directory. To override these defaults, use the options found below. By default, the directory structure created by this command is minimal and is not useful for building a complete system until initialized using the `petalinux-config --get-hw-description` workflow. Projects created using a BSP file as their source are suitable for building immediately.

The following table details options used specifically when creating a project.

Option	Functional Description	Value Range	Default Value
`--template` *TEMPLATE*	OPTIONAL. Assume specified CPU architecture.	zynq microblaze	zynq
`-s,--source` *SOURCE*	OPTIONAL. Create project based on specified BSP file. *SOURCE* is the full path on disk to the BSP file.	user-specified	none
`--out` *OUTPUTDIR*	OPTIONAL. Create a project in the specified directory.	none	current directory

Example Usage of the petalinux-create -t project Workflow

The following examples demonstrate proper usage of the `petalinux-create -t project` workflow.

- Create a new project from a reference BSP file.
  ```
  $ petalinux-create -t project -s <PATH-TO-BSP>
  ```
- Create a new project from based on the MicroBlaze template.
  ```
  $ petalinux-create -t project -n <NAME> --template microblaze
  ```
- Create a new project from a neutral location.
  ```
  $ petalinux-create -t project -n <NAME> --template zynq --out <PATH-TO-CREATE>
  ```

Details for the petalinux-create -t COMPONENT Workflows

The `petalinux-create -t COMPONENT` workflows allow the user to create various components within the specified PetaLinux project. These components can then be selectively included or excluded from the final system by toggling them using the `petalinux-config -c rootfs` workflow. There are no component-specific options for the `petalinux-create -t generic` or `petalinux-create -t modules` workflows.

The `petalinux-create -t apps` workflow allows the user to customize how application components are initialized during creation. The following table details options common when creating applications within a PetaLinux project.

Option	Functional Description	Value Range	Default Value
`-s,--source` *SOURCE*	OPTIONAL. Create the component from pre-existing content on disk. Valid formats are .tar.gz, .tar.bz2, .tar, .zip, and source directory (uncompressed).	user-specified	none

`--template` *TEMPLATE*	OPTIONAL. Create the component using a pre-defined project template.	c c++ autoconfig install	c
`--enable`	OPTIONAL. Upon creating the component, automatically enable it in the project's root filesystem. Else, enable using the `petalinux-config -c rootfs`.	none	disabled

The `petalinux-create -t libs` workflow allows the user to customize how library components are initialized during creation. These options allow the user to ensure that the created libraries are compiled and included in the final root filesystem properly. The following table details options specific to library components within a PetaLinux project.

Option	Functional Description	Value Range	Default Value
`--priority`	OPTIONAL. Specify the build priority for the library. Denoted by an integer suffix on the `Kconfig` file in the library directory. e.g., `Kconfig.`*n*	1 - 11	7

Example Usage of the petalinux-create -t COMPONENT Workflow

The following examples demonstrate proper usage of the `petalinux-create -t COMPONENT` workflow.

- Create an application component that is enabled in the root filesystem.
  ```
  $ petalinux-create -t apps -n <NAME> --enable
  ```
- Create a new library component that has a compile priority of 1.
  ```
  $ petalinux-create -t libs -n <NAME> --priority 1
  ```
- Create a new install-only application component. In this flow, nothing is compiled.
  ```
  $ petalinux-create -t apps -n <NAME> --template install
  ```

The petalinux-package Tool

The `petalinux-package` tool packages a PetaLinux project into a format suitable for deployment. The tool provides several workflows whose operation varies depending on the target package format. The supported formats/workflows are *boot*, *bsp*, *firmware*, and *pre-built*.

The `petalinux-package` tool is executed using the package type name to specify a specific workflow in the format `petalinux-package --`*PACKAGETYPE*.

- The *boot* package type creates a file (`.BIN` or `.MCS`)that allows the target device to boot.
- The *bsp* package type creates a `.bsp` file which includes the entire contents of the target PetaLinux project.
- The *firmware* package type creates a `.tar.gz` file which includes the needed files to update a PROM device on a board which has already been configured. This package format is only compatible with the `upgrade-firmware` PetaLinux demonstration application.
- The *pre-built* package type creates a new directory within the target PetaLinux project called "pre-built" and contains pre-built content that is useful for booting directly on a physical board. This package type is commonly used as a precursor for creating a *bsp* package type.

By default, the `petalinux-package` tool loads default files from the `<PROJECT>/images/linux/` directory.

The table below details the command line options that are common to all of the `petalinux-package` workflows.

Option	Functional Description	Value Range	Default Value
`-p,--project` *PROJECT*	OPTIONAL. PetaLinux project directory path.	user-specified	current directory
`-h,--help`	OPTIONAL. Display usage information.	none	none

Details for the petalinux-package --boot Workflow

The `petalinux-package --boot` workflow generates a bootable image that can be used directly with a Zynq AP SoC device or MicroBlaze-based FPGA design. For Zynq AP SoC devices, the default bootable format is *BOOT.BIN* which can be booted from an SD card. For MicroBlaze-based designs, the default format is an MCS PROM file suitable for programming via Vivado or other PROM programmer.

For Zynq AP SoC devices, this workflow is a wrapper around the `bootgen` utility provided with Xilinx SDK. For MicroBlaze-based FPGA designs, this workflow is a wrapper around the corresponding Vivado Tcl commands and generates an MCS formatted programming file. This MCS file can be programmed directly to a target board and then booted.

The table below details the options that are valid when creating a bootable image with the `petalinux-package --boot` workflow.

Option	Functional Description	Value Range	Default Value
`--format` *FORMAT*	REQUIRED. Image file format to generate.	BIN MCS	BIN
`--fsbl` *FSBL*	REQUIRED. Path on disk to FSBL elf binary.	user-specified	`zynq_fsbl.elf`(Zynq) `fs-boot.elf`(MicroBlaze)
`--force`	OPTIONAL. Overwrite existing files on disk.	none	none
`--fpga` *BITSTREAM*	OPTIONAL. Path on disk to bitstream file.	user-specified	none
`--uboot` *=UBOOT-IMG*	OPTIONAL. Path on disk to U-Boot elf binary.	user-specified.	`u-boot.elf`
`--kernel` *=KERNEL-IMG*	OPTIONAL. Path on disk to Linux Kernel image. Used to include FIT image in `BOOT.BIN`.	user-specified	`image.ub`
`--add` *DATAFILE*	OPTIONAL. Path on disk to arbitrary data to include.	user-specified	none
`--offset` *OFFSET*	OPTIONAL. Offset at which to load the prior data file.	user-specified	none
`--bmm` *BMMFILE*	OPTIONAL. Valid for MicroBlaze only.	user-specified	BMM in directory with FPGA bitstream
`--flash-size` *SIZE*	OPTIONAL. Must be a power-of-2. Valid for MicroBlaze only. Not needed for parallel flash types.	user-specified	16MB (SPI)
`--flash-intf` *INTERFACE*	OPTIONAL. Valid for MicroBlaze only.	SERIALx1 SPIx1 SPIx2 SPIx4 BPIx8 BPIx16 SMAPx8 SMAPx16 SMAPx32	auto-detect

`-o, --output` *OUTPUTFILE*	OPTIONAL. Path on disk to write output image.	user-specified	current directory

Example Usage of the petalinux-package --boot Workflow

The following examples demonstrate proper usage of the `petalinux-package --boot` workflow.

- Create a BOOT.BIN file for a Zynq AP SoC device.
  ```
  $ petalinux-package --boot --format BIN --fsbl --u-boot -o <PATH-TO-OUTPUT>
  ```
- Create a BOOT.BIN file for a Zynq AP SoC device that includes a PL bitstream and FIT image.
  ```
  $ petalinux-package --boot --format BIN --fsbl --u-boot --fpga <PATH-TO-BITSTREAM>
  --kernel -o <PATH-TO-OUTPUT>
  ```
- Create a x8 SMAP PROM MCS file for a MicroBlaze design.
  ```
  $ petalinux-package --boot --format MCS --fsbl --u-boot --fpga <PATH-TO-BITSTREAM>
  --flash-size <SIZE> --flash-intf SMAPx8 -o <PATH-TO-OUTPUT>
  ```

Details for the petalinux-package --bsp Workflow

The `petalinux-package --bsp` workflow compiles all contents of the specified PetaLinux project directory into a BSP file with the provided file name. This `.bsp` file can be distributed and later used as a source for creating a new PetaLinux project. This workflow is generally used as the last step in producing a project image that can be distributed to other users. All Xilinx reference BSP's for PetaLinux are packaged using this workflow.

The table below details the options that are valid when packaging a PetaLinux BSP file with the `petalinux-package --bsp` workflow.

Option	Functional Description	Value Range	Default Value
`-o, --output` *BSPNAME*	REQUIRED. Path on disk to store the BSP file. File name will be of the form *BSPNAME*.bsp.	user-specified	current directory
`-p,--project` *PROJECT*	OPTIONAL. PetaLinux project directory path. In the BSP context, multiple project areas can be referenced and included in the output BSP file.	user-specified	current directory
`--force`	OPTIONAL. Overwrite existing files on disk.	none	none
`--clean`	OPTIONAL. Clean the hardware implementation results to reduce package size.	none	none

`--hwsource` *HWPROJECT*	OPTIONAL. Path to a Vivado project to include in the BSP file.	none	none
`--no-extern`	OPTIONAL. Exclude components external to the project referenced using the `--searchpath` option. This may prevent the BSP from building for other users.	none	none
`--no-local`	OPTIONAL. Exclude components referenced in the local PetaLinux project. This may prevent the BSP from building for other users.	none	none

Example Usage of the petalinux-package --bsp Workflow

The following examples demonstrate proper usage of the `petalinux-package --bsp` workflow.

- Clean the project and then build the BSP image.
  ```
  $ petalinux-package --bsp --clean -o <PATH-TO-BSP>
  ```
- Build a BSP image that includes a reference hardware definition.
  ```
  $ petalinux-package --bsp --hwsource <PATH-TO-HW-EXPORT> -o <PATH-TO-BSP>
  ```
- Build a BSP image from a neutral location.
  ```
  $ petalinux-package --bsp -p <PATH-TO-PROJECT> -o <PATH-TO-BSP>
  ```

Details for the petalinux-package --firmware Workflow

The `petalinux-package --firmware` workflow creates a firmware update package based on the specified PetaLinux project. The firmware package allows the user to selectively update components of a deployed system. This package may contain components such as U-Boot, the Linux kernel, a Linux device tree, or a Linux root filesystem.

The table below details the options that are valid when packaging a PetaLinux firmware image with the `petalinux-package --firmware` workflow.

Option	Functional Description	Value Range	Default Value
`-o, --output` *PACKAGENAME*	OPTIONAL. Full path and name on disk to store the firmware image. Default location is current directory.	user-specified	firmware.tar.gz
`-p,--project` *PROJECT*	OPTIONAL. PetaLinux project directory path.	user-specified	current directory

`--linux=`*UBIMAGE*	OPTIONAL. Update the Linux kernel partition with the specified *UBIMAGE*.	none	`image.ub`
`--dtb=`*DTBFILE*	OPTIONAL. Update the device tree DTB partition with the specified *DTBFILE*.	none	`system.dtb`
`--fpga` *BITSTREAM*	OPTIONAL. Update the FPGA bitstream partition with the specified *BITSTREAM*.	none	none
`--u-boot=`*UBOOT-S*	OPTIONAL. Update the U-Boot binary partition with the specified *UBOOT-S* binary.	none	`u-boot-s.bin`
`--bootbin=`*BOOT.BIN*	OPTIONAL. Update the boot partition with the specified *BOOT.BIN* binary. Zynq only.	none	`BOOT.bin`
`--jffs2=`*JFFS2IMAGE*	OPTIONAL. Update the user's JFFS2 partition with the specified *JFFS2IMAGE* image.	none	`jffs2.img`
`-a,--add` *dev:file*	OPTIONAL. Update the flash partition named *dev* with the file specified by *file*. This option can be repeated multiple times.	user-specified	none
`--flash` *FLASHTYPE*	OPTIONAL. Specify the type of flash device with which the image is compatible.	spi parallel	parallel
`--little-endian`	OPTIONAL. Specify that the target system is little endian. AXI systems are little-endian.	none	none
`--big-endian`	OPTIONAL. Specify that the target system is big endian. PLB systems are big-endian.	none	none
`--data-width` *SIZE*	OPTIONAL. Specify the bit width of the data bus for the target flash device.	8 16 32	none
`--product` *STRING*	OPTIONAL. Specify a product compatible string used to validate firmware image.	user-specified	none

`--pre` *SCRIPT*	OPTIONAL. Specify a *SCRIPT* that should be run on the target platform prior to updating the flash partitions.	user-specified	none
`-v,--verbose`	OPTIONAL. Displays additional output messages.	none	none

The `--image`, `--dtb`, `--uboot` and `--jffs2` options allow the user to override the default filenames and partitions using the partition:file syntax of the `--add` option.

Example Usage of the petalinux-package --firmware Workflow

The following examples demonstrate proper usage of the `petalinux-package --firmware` workflow.

- Install the FIT image (image.ub) into the flash partition called safe-image.
  `$ petalinux-package --firmware -a /dev/flash/safe-image:<PATH-TO-FIT-IMAGE>`
- Package firmware image with bitstream, U-Boot and Linux kernel for MicroBlaze.
  `$ petalinux-package --firmware --fpga <BITSTREAM> --u-boot --linux -o <FILE-TO-CREATE>`

Details for the petalinux-package --prebuilt Workflow

The `petalinux-package --prebuit` workflow creates a new directory named "pre-built" inside the directory hierarchy of the specified PetaLinux project. This directory contains the required files to facilitate booting a board immediately without completely rebuilding the project. This workflow is intended for users who will later create a PetaLinux BSP file for distribution using the `petalinux-package --bsp` workflow. All Xilinx reference PetaLinux BSP's contain a pre-built directory.

The table below details the options that are valid when including pre-built data in the project with the `petalinux-package --prebuilt` workflow.

Option	Functional Description	Value Range	Default Value
`-p,--project` *PROJECT*	OPTIONAL. PetaLinux project directory path.	user-specified	current directory
`--force`	OPTIONAL. Overwrite existing files on disk.	none	none
`--clean`	OPTIONAL. Remove all files from the `<PROJECT>/prebuilt` directory.	none	none
`--fpga` *BITSTREAM*	OPTIONAL. Include the *BITSTREAM* file in the prebuilt directory.	user-specified	none

`-a,--add` *src:dest*	OPTIONAL. Add the file/directory specifed by *src* to the directory specifed by by *dest* in the pre-built directory.	user-specified	none

Example Usage of the petalinux-package --prebuilt Workflow

The following examples demonstrate proper usage of the `petalinux-package --prebuilt` workflow.

- Include a specific bitstream in the pre-built area.
  `$ petalinux-package --prebuilt --fpga <BITSTREAM>`
- Include a specific data file in the pre-built area.
  `$ petalinux-package --prebuilt -a <APP>:images/<APP>`

The petalinux-util Tool

The `petalinux-util` tool provides various support services to the other PetaLinux workflows. The tool itself provides several workflows depending on the support function needed.

Details for the petalinux-util --gdb Workflow

The `petalinux-util --gdb` workflow is a wrapper around the standard GNU GDB debugger and simply launches the GDB debugger in the current terminal. Executing `petalinux-util --gdb --help` at the terminal prompt provides verbose GDB options that can be used.

Example Usage of the petalinux-util --gdb Workflow

The following example demonstrates proper usage of the `petalinux-util --gdb` workflow.

- Launch the GNU GDB debugger.
  ```
  $ petalinux-util --gdb
  ```

Details for the petalinux-util --jtag-logbuf Workflow

This workflow logs the Linux kernel printk output buffer that occurs when booting a Linux kernel image via JTAG. This workflow is intended for debugging the Linux kernel for review and debug. This workflow may be useful for users when the Linux kernel is not producing output via a serial terminal. For details on how to boot a system via JTAG, see the `petalinux-boot --jtag` workflow. For MicroBlaze, the image is `<PROJECT>/image/linux/image.elf`. For ARM, the image is `<PROJECT>/image/linux/vmlinux`.

The table below details the options that are valid when using the `petalinux-util --jtag-logbuf` workflow.

Option	Functional Description	Value Range	Default Value
`-i,--image` *IMAGEPATH*	REQUIRED. Linux kernel ELF image.	user-specified	none
`-p,--project` *PROJECT*	OPTIONAL. PetaLinux project directory path.	user-specified	current directory
`--noless`	OPTIONAL. Do not pipe output to the `less` command.	none	none
`-v,--verbose`	OPTIONAL. Displays additional output messages.	none	none
`-h,--help`	OPTIONAL. Displays tool usage information.	none	none

Example Usage of the petalinux-util --jtag-logbuf Workflow

The following examples demonstrate proper usage of the `petalinux-util --jtag-logbuf` workflow.

- Launch a specific Linux kernel image.
  `$ petalinux-util --jtag-logbuf -i <PATH-TO-IMAGE>`
- Launch the JTAG logger from a neutral location. This workflow is for Zynq devices only.
  `$ petalinux-util --jtag-logbuf -i <PATH-TO-IMAGE> -p <PATH-TO-PROJECT>`

Details for the petalinux-util --update-sdcard Workflow

This workflow automates the loading of a root filesystem to a physical partition that exists on an SD card. It is only valid for Zynq AP SoC devices. This workflow is flexible such that it can load the VFAT partition (for `BOOT.BIN`), the Linux ext partition (for the root filesystem), or both. The `petalinux-util --update-sdcard` workflow copies the `BOOT.BIN` file from `<PROJECT>/images/linux` to a parition specified by *boot:*. If this file does not exist in this location, the tool fails.

When the *:rootfs* element is omitted, the tool copies the `<PROJECT>/images/linux/image.ub` file to the VFAT partition as well. When the *:rootfs* component is present, the tool copies the contents of `<PROJECT>/build/linux/rootfs/targetroot/` to the location specified by *:rootfs*. Superuser access is required for this tool to complete successfully unless the target directories for *boot* and *rootfs* are mounted with appropriate access permissions. The table below details the options that are valid when using the `petalinux-util --update-sdcard` workflow.

Option	Functional Description	Value Range	Default Value
`-d,--dir` *boot:rootfs*	REQUIRED. Copy files to the SD card.	user-specified	none
`-p,--project` *PROJECT*	OPTIONAL. PetaLinux project directory path.	user-specified	current directory
`-h,--help`	OPTIONAL. Display usage information.	none	none

Example Usage of the petalinux-util --update-sdcard Workflow

The following examples demonstrate proper usage of the `petalinux-util --update-sdcard` workflow.

- Copy the BOOT.BIN and image.ub files for a Zynq AP SoC device to the VFAT partition of the SD card. This requires a privileged user or appropriate access permissions for the specified path.
  `$ petalinux-util --update-sdcard --dir <VFAT-PATH>`
- Copy the BOOT.BIN (and image.ub) and contents of the root filesystem to the ext file system on the SD card. This requires a privileged user or appropriate access permissions for the specified paths.
  `$ petalinux-util --update-sdcard --dir <VFAT-PATH>:<EXT-PATH>`

Details for the petalinux-util --webtalk Workflow

This workflow toggles the Xilinx WebTalk feature *ON* or *OFF*. Xilinx WebTalk provides anonymous usage data about the various PetaLinux tools to Xilinx. A working Internet connection is required for this feature.

Option	Functional Description	Value Range	Default Value
`--webtalk`	REQUIRED. Toggle WebTalk.	on off	on
`-h,--help`	OPTIONAL. Display usage information.	none	none

Example Usage of the petalinux-util --webtalk Workflow

The following examples demonstrate proper usage of the `petalinux-util --webtalk` workflow.

- Toggle the WebTalk feature off.
  ```
  $ petalinux-util --webtalk off
  ```
- Toggle the WebTalk feature on.
  ```
  $ petalinux-util --webtalk on
  ```

Additional Resources

References

PetaLinux Tools Documentation is available at http://www.xilinx.com/petalinux.