WIND RIVER

Wind River® Intelligent Device Platform XT

PROGRAMMER'S GUIDE

2.0

EDITION 5

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

10 Dec 2014

Contents

PART I: INTRODUCTION TO IDP

PART II: KEY-RELATED TASKS

PART III: SYSTEM OWNER TASKS

PART V: APPLICATION DEVELOPMENT VENDOR TASKS

PART VI: REFERENCES

PART I

Introduction to IDP

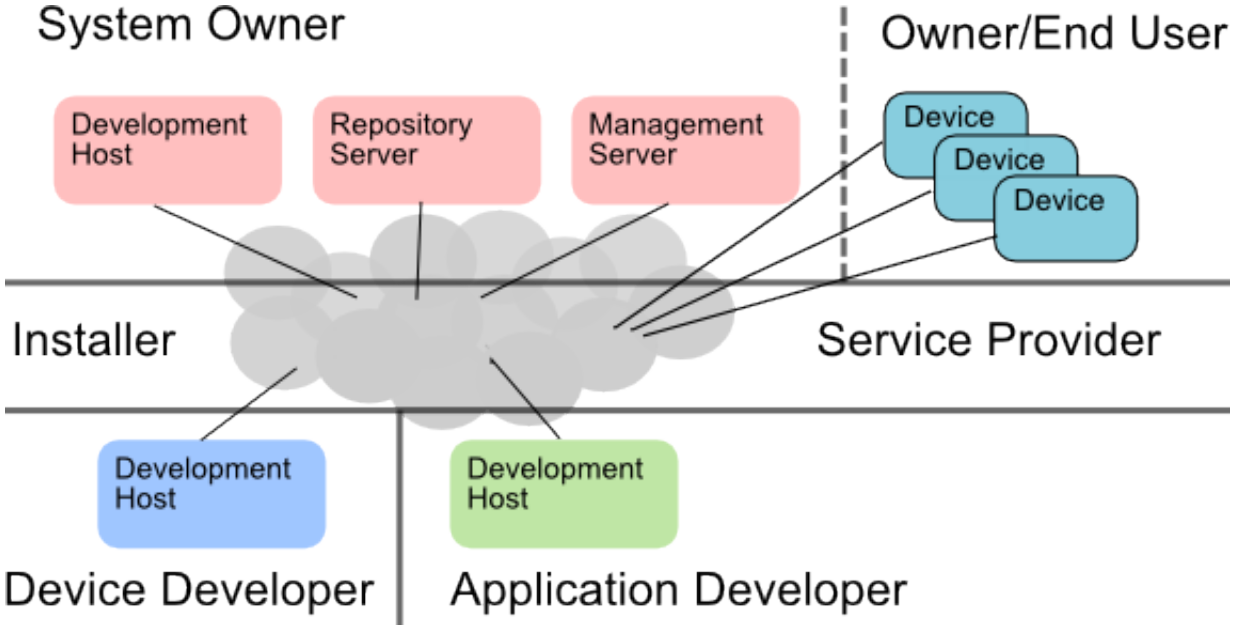
1

Introduction and Overview

Wind River Intelligent Device Platform Overview

The Wind River Intelligent Device Platform XT (IDP XT) packages the Wind River commercial-grade Linux development platform with security and management tools for gateways.

IDP XT provides integrated development and management support for distributed systems that utilize smart services with cloud computing. It includes secure remote management layer for cloud-based smart services, including automated customer interaction and support.



Included in IDP XT

- Wind River Linux
- Wind River Workbench
- Wind River Intelligent Device Platform XT
- McAfee Embedded Control
- BSPs for the following boards:

  Cross Hill
  Clanton Hill
  Galileo
  Advantech UTX-3115

Related Links

IDP User Roles on page 14
IDP XT supports development, deployment, and management of cloud-based systems including servers and distributed devices. For this reason, users of IDP XT may have different roles and be employed by different organizations.

Where to Find Information on page 16
The Wind River Intelligent Device Platform XT (IDP XT) provides documentation for Wind River IDP XT capabilities. It also utilizes Wind River Linux documentation and third-party hardware and software documentation.

Accessing Documentation on page 18
You can access IDP XT documentation through online support.

IDP User Roles

IDP XT supports development, deployment, and management of cloud-based systems including servers and distributed devices. For this reason, users of IDP XT may have different roles and be employed by different organizations.

Roles Associated with Cloud-Based Systems

The following table shows roles that are commonly identified in developing and managing cloud-based systems.

Role	Responsibility
OEM	The original equipment manufacturer who manufactures products or components that are sold to another company and retailed under that company's brand. Provides device hardware and sometimes also firmware and base software.
Integrator	Combines hardware, firmware, system software, and applications provided by other roles and initializes the target device.
Software Provider	Designs and builds the application and provides updated RPMs.
Owner	Owns the device and specifies which functions it performs. Manages the device throughout its life cycle.
Service Provider	Owns the SIM card or provides the data connection to the device.

Role	Responsibility
Operator	Provides ongoing management of the device after it is deployed.
Installer	Deploys the device in the field. Installs the device and performs activation as required.
End User	Interacts with the device application level or consumes output from the device.

Roles Used in IDP XT Documentation

While each of the roles described in the table play significant parts in the creation and deployment of the system, they are rarely all performed by employees of different companies. A few groupings of roles are common, as shown in the scenarios.

For the IDP XT documentation, Wind River has chosen to group the roles into the following groups:

Wind River Role Name	Role Description
System Owner	owner operator service provider end user
Device Development Vendor	OEM integrator
Application Development Vendor	software provider

At the present time the IDP XT documentation does not include topics relevant to the service provider or installer roles.

Smart Meter Deployment Scenario

HydroCo, an electrical utility company, is running a smart metering project. HardwareCo will supply the meters, which will communicate to the head office using 3G. SWVendorCo will write the software application. MobileCo will provide the SIM card and network connection. CableGuy will install the meters at the end user locations. HydroCo will manage data collection and device management.

Company or Employee	Roles
HydroCo	owner, operator, and end user
HardwareCo	OEM and integrator
SWVendorCo	software vendor
MobileCo	service provider
CableGuy	installer

Set Top Box Deployment Scenario

CableCo provides television, telephone, and internet services to its customers. CableCo wants to deploy a new set top box with advanced capabilities. CableCo will specify functionality, select hardware, and use its existing retail outlets and installation technicians. DRMCo will provide conditional access software and smart cards for the box and the video broadcast servers. STBCo will provide the hardware and base operating system, which they will purchase from an OEM. AppCo will provide the application software and middleware.

Company or Employee	Roles
CableCo	owner, operator, service provider, and installer
STBCo	OEM and integrator
DRMCo	software provider
AppCo	software provider
CableCo customer	end user

Related Links

Wind River Intelligent Device Platform Overview on page 13
The Wind River Intelligent Device Platform XT (IDP XT) packages the Wind River commercial-grade Linux development platform with security and management tools for gateways.

Where to Find Information on page 16
The Wind River Intelligent Device Platform XT (IDP XT) provides documentation for Wind River IDP XT capabilities. It also utilizes Wind River Linux documentation and third-party hardware and software documentation.

Accessing Documentation on page 18
You can access IDP XT documentation through online support.

Where to Find Information

The Wind River Intelligent Device Platform XT (IDP XT) provides documentation for Wind River IDP XT capabilities. It also utilizes Wind River Linux documentation and third-party hardware and software documentation.

The following documentation is available in the Wind River help system and on the Wind River Online Support Web site.

Wind River Documentation

Wind River Intelligent Device Platform XT Programmer's Guide

Provides instructions for installing, configuring the Intelligent Device Platform and modifying it for your specific requirements (this document).

Wind River Intelligent Device Platform XT Security Guide

Provides guidance on performing a security analysis and matching IDP XT capabilities with assessed needs.

Wind River Intelligent Device Platform XT Release Notes

Provides general product information, changes in this release, usage caveats, and known problems.

Wind River OPC for IDP Programmer's Guide

Provides guidance on using Wind River OPC with IDP XT.

Wind River Linux Getting Started Guide, 5.0.1

Provides instructions for creating, modifying, deploying, and debugging platform and application projects using the command-line and Workbench.

Wind River Linux User's Guide, 5.0.1

Provides command-line instructions for configuring, building, and developing platform projects as well as detailed information on the development environment and build system.

Wind River Workbench by Example Guide (Linux Version), 3.3

Provides procedures and examples for using Workbench to configure, build, and debug Wind River Linux application, platform, and kernel module projects.

NOTE: This list represents the primary documents for developing an Intelligent Device platform target system and is not complete. For the full set of documents that come with Wind River Linux, see the *Wind River Linux User's Guide, 5.0.1* .

McAfee Documentation

McAfee Embedded Control User Guide

provides an overview of McAfee Embedded Control as well as installation and configuration information and examples for getting started.

McAfee Application Control Product Guide

provides details of McAfee Application Control including installation and licensing, capabilities, and troubleshooting.

McAfee Application Control Command Line Interface Guide

provides details of McAfee Application Control commands and arguments.

McAfee Change Control Product Guide

provides details of McAfee Change Control including installation and licensing, capabilities, and troubleshooting.

McAfee Change Control Command Line Interface Guide

provides details of McAfee Change Control basic and advanced commands.

Related Links

Wind River Intelligent Device Platform Overview on page 13
The Wind River Intelligent Device Platform XT (IDP XT) packages the Wind River commercial-grade Linux development platform with security and management tools for gateways.

IDP User Roles on page 14
IDP XT supports development, deployment, and management of cloud-based systems including servers and distributed devices. For this reason, users of IDP XT may have different roles and be employed by different organizations.

Accessing Documentation on page 18

You can access IDP XT documentation through online support.

Accessing Documentation

You can access IDP XT documentation through online support.

- Access documentation through the Workbench main menu.

  Select **Help** > **Help Contents** > **Wind River Documentation**.
- Access documentation through the independent help browser.

  After you install Workbench, run the following command:

  ```
  $ <installDir>/workbench-3.3/x86-linux2/bin/wrhelp.sh
  ```
- Access documentation through the file system in the installation directory (*installDir*).

Options	Description
PDF Versions	Point your PDF reader to the ***.pdf** file, for example: *installDir* **/docs/extensions/eclipse/plugins/com.windriver.ide.doc.wr_intelligent_device_platform_XT_2.0/mc_afee_documents/mec_ug_en-us.pdf**
HTML Versions	Point your Web browser to the index.html file, for example: ***installDir*/docs/extensions/eclipse/plugins/com.windriver.ide.doc.wr_intelligent_device_platform_XT_2.0/wr_idp_programmers_guide/index.html**

- Access documentation on the *Wind River Online Support Web* site.

  Log on to Wind River Online Support and select

  Products > **Wind River Intelligent Device Platform XT** > **Manuals**.

Related Links

Wind River Intelligent Device Platform Overview on page 13
The Wind River Intelligent Device Platform XT (IDP XT) packages the Wind River commercial-grade Linux development platform with security and management tools for gateways.

IDP User Roles on page 14
IDP XT supports development, deployment, and management of cloud-based systems including servers and distributed devices. For this reason, users of IDP XT may have different roles and be employed by different organizations.

Where to Find Information on page 16
The Wind River Intelligent Device Platform XT (IDP XT) provides documentation for Wind River IDP XT capabilities. It also utilizes Wind River Linux documentation and third-party hardware and software documentation.

2

Architecture

Intelligent Device Platform Architecture

The architecture defines the location of installed products layers, profiles, and templates. IDP XT builds on the standard Linux architecture.

The IDP XT installation contains layers that include configuration files, templates, and code that extend your development possibilities. Using the **--enable-addons=wr-idp** option in your configuration includes the layers that add or extend platform project capabilities. Different capabilities are available depending on the BSP and board you choose.

For more information about the Linux architecture, see the Wind River Linux documentation.

For detailed information on IDP XT layers and features, see *Layers and Features* on page 151.

Related Links

Systems in Development on page 20
IDP XT leverages the Wind River Linux development tools and adds security, connectivity, and management support

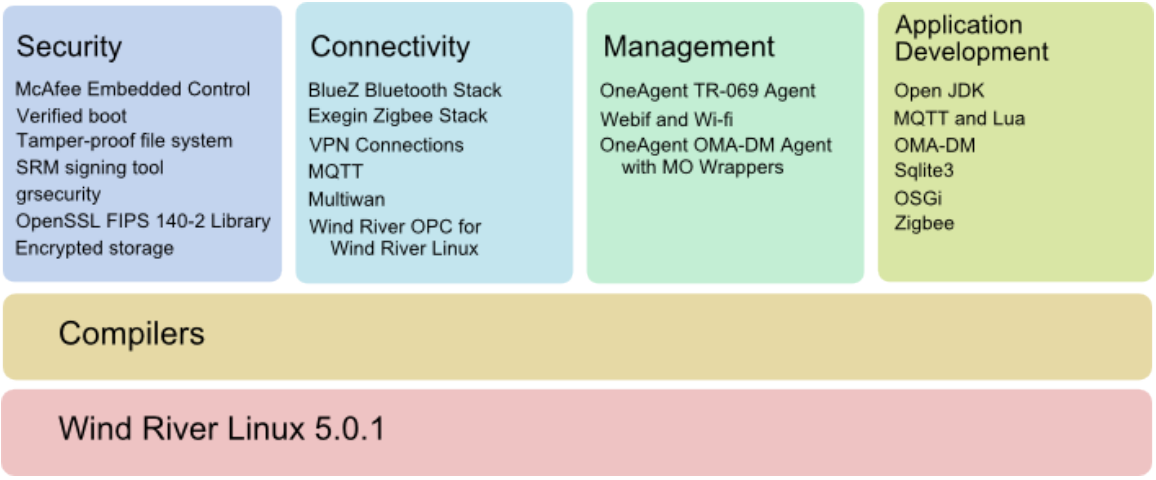
Deployed Devices on page 21
Devices deployed in IDP XT systems provide secure connections, secure boot and software updates for devices, and Web interfaces for system management.

IDP Features on page 21

This topic lists the IDP XT layers and features with their descriptions.

Systems in Development

IDP XT leverages the Wind River Linux development tools and adds security, connectivity, and management support



Related Links

Intelligent Device Platform Architecture on page 19
The architecture defines the location of installed products layers, profiles, and templates. IDP XT builds on the standard Linux architecture.

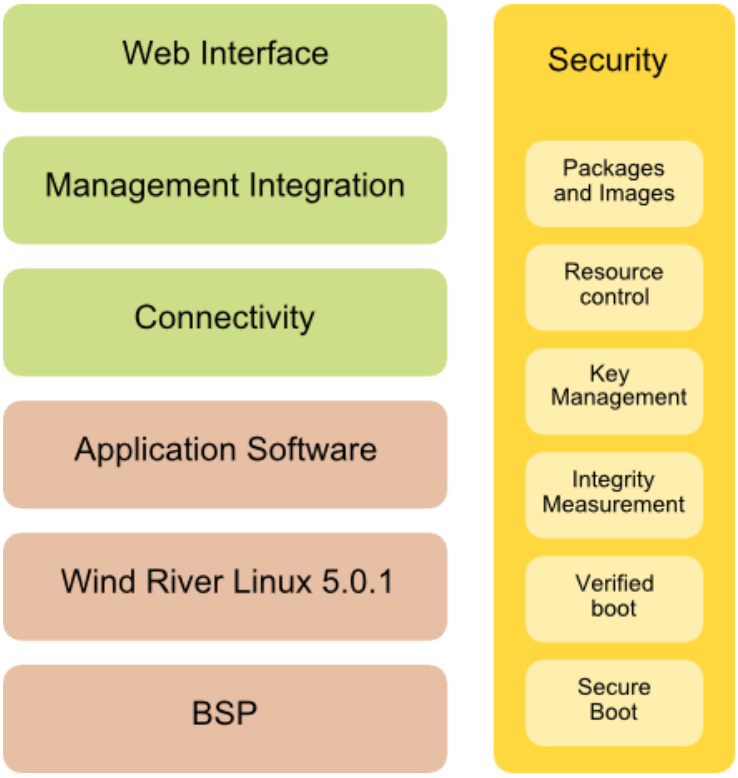
Deployed Devices on page 21
Devices deployed in IDP XT systems provide secure connections, secure boot and software updates for devices, and Web interfaces for system management.

IDP Features on page 21

This topic lists the IDP XT layers and features with their descriptions.

Deployed Devices

Devices deployed in IDP XT systems provide secure connections, secure boot and software updates for devices, and Web interfaces for system management.



Related Links

Intelligent Device Platform Architecture on page 19
The architecture defines the location of installed products layers, profiles, and templates. IDP XT builds on the standard Linux architecture.

Systems in Development on page 20
IDP XT leverages the Wind River Linux development tools and adds security, connectivity, and management support

IDP Features on page 21
This topic lists the IDP XT layers and features with their descriptions.

IDP Features

This topic lists the IDP XT layers and features with their descriptions.

For information on which boards support which capabilities, see *Layers and Features* on page 151.

Layers	Description	Features	Description
meta-java-dl	Virtual layer which provides copies of oe-core referenced components so that users do not need to download from a network.	N/A	N/A
wr-digi-idigiconnector	ISV layer that provides software for users to connect to the iDigi cloud server.	default	Provides APIs to connect to the iDigi device cloud to manage the connected device using SCI.
wr-wks-oneagent-oma-dm-ia	ISV layer that provides the OMA-DM protocol and framework necessary to add the OMA-DM client/agent, to facilitate communication with a remote OMA-DM server on Intel Architecture boards.	default	Adds the OMA-DM client/ agent for communicating with a remote OMA-DM server.
wr-wks-oneagent-tr069	ISV layer for Works Systems OneAgent TR-069 agent. TR-069 supports a variety of management APIs.	default	Adds the TR-069 agent for communication between a client and a TR-069-enabled autoconfiguration.
wr-prosyst-mbs-smarthome-sdk-ia	ISV layer for ProSyst mBS Smart Home SDK binaries for Intel Architecture boards. Requires additional setup prior to use.	default	Provides configuration and required packages for the ProSyst mBS Smart Home capabilities. Requires additional setup prior to use; see the README for additional information.
wr-exegin-zigbee-ia	ISV layer for the Exegin Zigbee Stack, a communications stack for managing wireless network connections on Intel Architecture boards. This is an optional, add-on product.	default	Provides the Exegin Zigbee Stack, a communications stack for managing wireless network connections, and utilities and firmware binary files to update firmware on an Exegin Q58 zigbee module.

Layers	Description	Features	Description
wr-mcafee	ISV layer for McAfee Embedded Control, which uses dynamic whitelisting to ensure that only trusted applications run on servers and clients.	default	Adds McAfee Application Control, McAfee Change Control, and McAfee Integrity Monitor to IDP XT projects.
wr-srm	The main SRM layer. Implements secure package management in your platform project.	default	Provides infrastructure to boot to a trusted software stack and to securely manage devices remotely.
		Encrypted Storage	Provides **dm-crypt** kernel capability and **cryptsetup** front end tool to implement secure storage. No separate feature template.
		Secure Package Management	Prevents installing RPM packages without authorized signatures. No separate feature template.
		grsec_std	Provides grsecurity and related tools
		secure-boot	Provides infrastructure and demo files to perform secure boot on Quark boards. Included by default with wr-srm for Cross Hill and Clanton Hill BSPs.
		non_grsec	A virtual feature to remove the default grsec_std feature.
wr-idp-devkit	The main Intelligent Device Platform layer. Most features and packages are located here.	default	Provides default system configuration for each board

Layers	Description	Features	Description
		backports	Backports kernel modules in a newer version to the current version of Wind River Linux in order to work with a newer version of device firmware. This feature is specifically for the Intel Dual Band Wireless-AC 7260 Wi-Fi Card. Use of this feature with other wi-fi/ bluetooth modules is not supported hence not recommended.
		bluetooth	Provides the BlueZ Bluetooth software implementation.
		boot_delay_network	Decreases boot time by delaying the configuration of networking and networking-related services for 10 seconds after the login prompt appears.
		firewall	Provides Linux Firewall Software
		graphics_qt	Adds basic packages and features that are needed by QT and for starting IDP XT as a graphical work station.
		idp_devkit_full	Provides a convenient way to include all board-independent features into a project.
		ipsec_vpn	Adds the strongSwan IPsec VPN implementation to your platform project.
		l2tp	Adds the L2TP VPN implementation to your platform project.
		mqtt	Provides client/server tools for the MQTT protocol and utiltiies based on LUA for publishing and subscribing to MQTT topics.

Layers	Description	Features	Description
		online_updates	Provides the ability to update a target system's binary RPMs from an online RPM repository server.
		opc	Specifies the communication of real-time plant data between control devices from different manufacturers. OPC is OLE for process control. OLE is object linking and embedding.
		opc_demo	Adds the Wind River OPC Demo. For more information, see the *Wind River OPC for IDP Programmer's Guide*.
		openjdk-bin	Provides the OpenJDK binary.
		pppoe	Provides point-to-point connectivity over Ethernet.
		pptp_vpn	Provides the point-to-point tunneling protocol (PPTP) for VPN connections.
		recovery	Adds the reset_media script to your target file system. This script, along with kernel and file system images, is used to create bootable recovery media.
		upnp	Provides UPnP support for your platform project.
		vlan	Adds 802.1Q protocol and configuration support for your platform project
		webif	Adds the Webif interface, a web browser-based interface for configuring network connections and checking the services running on the target.

Layers	Description	Features	Description
		wrs_qt_demo	Adds the Wind River QT Demo, which demonstrates QT toolkit development capability.
wr-ima-appraise	ISV layer for application Integrity Measurement	default	Uses IMA Appraisal to prevent loading applications and libraries without authorized signatures.
		wrs_qt_demo	Adds the Wind River QT Demo, which demonstrates QT toolkit development capability.

Related Links

Intelligent Device Platform Architecture on page 19
The architecture defines the location of installed products layers, profiles, and templates. IDP XT builds on the standard Linux architecture.

Systems in Development on page 20
IDP XT leverages the Wind River Linux development tools and adds security, connectivity, and management support

Deployed Devices on page 21
Devices deployed in IDP XT systems provide secure connections, secure boot and software updates for devices, and Web interfaces for system management.

3

IDP Security

Application Integrity Measurement (Tamper-proof File System)

The tamper-proof file system, also known as Application Integrity Measurement (AIM), includes an Integrity Measurement Architecture (IMA) Appriase layer.

Embedded devices deployed in the field usually have multiple stakeholders and each stakeholder has different needs for access to the device, applications, and data. The tamper-proof file system capability allows the device owner to prevent end users from making arbitrary modifications to the IDP XT software system deployed in the field. Only authorized users can make modifications to the system once it has been securely deployed, for example, by using the remote management capability of IDP XT.

For more information, see *Enabling IMA Appraise* on page 162.

Secure Boot

IDP XT offers hardware-based secure boot for Cross Hill, Clanton Hill, and Advantech UTX-3115 boards.

NOTE: Secure boot for the Galileo board is not supported.

Secure boot is provided by the IDP XT Secure Remote Management capability (SRM), which is available with the **wr-srm** layer.

Secure boot uses a security table and keys plus a signed kernel image and rootfs image to verify that the kernel image and file system have not been tampered with before allowing the boot to proceed.

McAfee Embedded Control for Wind River

McAfee Embedded Control for Wind River includes a McAfee layer that allows you to configure McAfee embedded products for use with the Wind River target platform.

McAfee Embedded Control provides the following capabilities to the Wind River Linux target platforms:

- McAfee Application Control
- McAfee Change Control

To ensure these capabilities work correctly, you must perform some extra configuration tasks. For more information, see:

- *McAfee Embedded Control* on page 69
- *McAfee Embeded Control Users Guide*—this is a McAfee PDF document supplied with IDP XT.

Standalone SRM Signing Tool

You can use the SRM Signing Tool (SST) to sign boot loader, kernel, and rootfs files and RPM packages. The tool can be used on any Linux host, whether or not IDP XT is installed.

SST is provided by the IDP XT Secure Remote Management Feature (SRM), which is available for Intel Architecture boards by configuring the **wr-srm** layer in your platform project.

SST provides **imtool** to assist the deployed SRM intelligent system with verifying packages before installing them.

The grsecurity Tool

The **grsecurity** tool allows you to create and manage security policy rules.

Grsecurity is supported on the following boards:

Cross Hill
Clanton Hill
Galileo
Advantech UTX-3115

The IDP XT SRM layer includes **grsecurity** and related tools by default. For more information on how to use grsecurity, see *grsecurity and Related Tools* on page 227.

Encrypted Storage

Encrypted storage, also known as secure storage, is used to store sensitive information on the target device

When encrypted storage is combined with other SRM capabilities, the device owner can make sure that the encrypted storage can only be accessed on a device that is running the trusted

software. Encrypted storage is not part of IDP XT's SRM but complements SRM by providing additional security capabilities. Encrypted storage utilizes the SRM infrastructure. For more information, see *Encrypted Storage* on page 219.

4

IDP Connectivity

BlueZ Bluetooth Stack

Add Bluetooth support to your platform project with the BlueZ Bluetooth software implementation.

The BlueZ stack supports core Bluetooth layers and protocols with a modular implementation, including the following capabilities:

- Symmetric multi processing safe
- Multithreaded data processing
- Support for multiple Bluetooth devices
- Real hardware abstraction
- Standard socket interface to all layers
- Device and service level security support

BlueZ is fully documented online at *http://www.bluez.org*

Note that this adds the software capability and functionality to support Bluetooth communications. An external Bluetooth adapter is required. For information on adding Bluetooth support to your platform project, see:

Including Bluetooth in a Platform Project on page 159

Exegin Zigbee Stack

The Exegin Zigbee Stack is an optional, add-on product for managing wireless communications.

The Exegin ZigBee Stack implementation uses the same four-layer architecture defined in the ZigBee specification:

Physical (PHY)

based on the IEEE 802.15.4 specification.

Media access (MAC)

based on the IEEE 802.15.4 specification.

Network (NWK)

includes functionality for mesh networking, allowing any node in a ZigBee network to take on the role of ZigBee Coordinator (ZC) or a ZigBee Router (ZR); this means any node can act as a coordinator, a router, or an end-device.

Application (APL)

consists of a layer that sits at the top of the ZigBee stack. It provides ZigBee Cluster Libraries (ZCL) and a framework for developers to add their own application-specific functionality. In addition, it provides common application functionality that is offered by every ZigBee device (embodied in the ZigBee Device Object, or ZDO).

Configuring and building your platform project using the **--with-layer=wr-exegin-zigbee-ia** option adds the Exegin ZigBee stack to your project.

For more information on how to test the Exegin ZigBee stack and form a ZigBee network, see the **README** file located in the following directory:

***projDir*/layers/wr-idp/wr-exegin-zigbee-ia/recipes-exegin/zbstack-exegin/files**

For more information on how to develop ZigBee applications on IDP XT using the Exegin ZigBee stack, see the documents in the **doc** directory of the ZibBee SDK contained in:

***projDir*/layers/wr-idp/wr-exegin-zigbee-ia/ downloads/exegin-zb-sdk-clanton-linux-exmac-1.6.51.tar.gz**.

To learn more about Exegin and the ZigBee stack in general, see:

- *Setting Up a Zigbee Network* on page 199
- The *Exegin Zigbee Stack home page*.

VPN Connections

Wind River provides different options for managing VPN connections with Intelligent Device Platform target systems.

A VPN uses the Internet to provide remote offices or individual users with secure access to their organization's network. The following VPN solutions are available for your target system:

IPSec VPN

Use this option to add VPN connectivity to your embedded device. This open source solution is fully documented online at *http://www.strongswan.org/* . Refer to this online documentation, and the **README** located at:

***projDir*/layers/wr-idp/wr-idp-devkit/templates/feature/ipsec_vpn**

You can add this to any platform project using the **--with-template=feature/ipsec_vpn** configuration option.

L2TP VPN

Use this option to add the OpenL2TP VPN solution. OpenL2TP is an open source solution L2TP client and server designed for use as an enterprise L2TP VPN server or in embedded networking products. It is designed to support hundreds of sessions, each with a different configuration. It is fully documented online at *http://www.openl2tp.org/* . Refer to this online documentation, and the README located at:

***projDir*/layers/wr-idp/wr-idp-devkit/templates/feature/l2tp**

You can add this to any platform project using the **--with-template=feature/l2tp** configuration option.

PPTP VPN

Use this option to add PPTP VPN connectivity to your embedded device. PPTP VPN is an open source solution providing free tunnel access across the Internet. It is documented online at *http://www.pptpvpn.org/*. Also see the **README** file located at:

***projDir*/layers/wr-idp/wr-idp-devkit/templates/feature/pptp_vpn**

You can add this to any platform project using the **--with-template=feature/pptp_vpn** configuration option.

MQTT

Use Message Queue Telemetry Transport (MQTT) in small footprint systems located remotely where internet/network bandwidth can be expensive.

MQTT is a lightweight (low power, low network bandwidth) publish-and-subscribe messaging protocol. It is open source and an important protocol of the M2M/Internet of Things (IoT) revolution. Sensors, mobile phones, and embedded systems are some examples where MQTT is used.

IDP XT provides Mosquitto which is an open source server implementation for version 3.1 of the MQTT protocol. You can include the MQTT in your platform projects using the **--with-template=feature/mqtt** option.

For details on how to install and use this feature, see:

Installing MQTT and Lua on page 173
About MQTT and Lua on page 213

Multiwan

Enable the multiwan utility to facilitate Internet communications for your target platform.

The multiwan monitors the status of the networking interfaces. When the primary interface is down, this daemon automatically connects the secondary interface. The utility is part of the **wr-idp-devkit layer** and is included on your IDP XT target automatically.

Enable multiwan using the Webif interface. On the **Network** tab, select **Multiwan**. Change the value of **Multi WAN Monitor Enable** to **Enable**. Select **Save** and **Apply Changes**.

You can increase the polling interval of the **multiwan** process to reduce CPU usage. Modify the value of **health_interval** in the following configuration file:

/etc/config/multiwan

The default value of **health_interval** is **3**; increasing it to **50** will noticeably reduce CPU usage.

Wind River OPC for Wind River Linux

Wind River OPC is Wind River's implementation of the OLE for Process Control (OPC) specifications for Linux.

Wind River OPC includes implementations of the Data Access (DA) servers as well as interactive client tools. Wind River OPC is tightly integrated with Wind River Linux, and Wind River DCOM.

Using Wind River OPC, you can quickly and efficiently develop applications for process control, robotics, machine builders, semiconductor manufacturing, distributed control systems, discrete controllers, test and measure equipment, and other industrial devices.

For more information, see:

- *Installing Wind River OPC* on page 184
- *Wind River OPC User's Guide (Wind River Linux Version)*

5

IDP Management

OneAgent TR-069 Agent

The OneAgent TR-069 agent provides a protocol and API stack for communication between a TR-069-enabled client and server.

The OneAgent bundle provides a TR-069-compliant (Technical Report 069) protocol and API stack for communication between a TR-069-enabled client and server. The TR-069 technical specification is titled *CPE WAN Management Protocol (CWMP)*. It defines an application layer protocol for remote management of end-user devices. CPE, or customer premises equipment, acts as the client. In the Intelligent Device Platform system, this client communication is managed by the OneAgent implementation. ACS, the auto-configuration server, provides access to the WAN as the TR-069 server.

When used as part of a network system, implementing TR-069 provides the following functionality for your device platform:

- auto-configuration and dynamic service provisioning
- software/firmware image management
- status and performance monitoring
- diagnostics

You can include this agent with the **--with-layer= wr-wks-oneagent-tr069** option. For more information, see *OneAgent OMA-DM Agent and MO Wrappers* on page 36.

Related Links

Webif and Wi-Fi Connections on page 36
Wind River provides a web-based interface called Webif for managing Wi-Fi connections with IDP XT target systems.

OneAgent OMA-DM Agent and MO Wrappers on page 36

SRM utilizes the Works System OneAgent OMA Device Management Communications (DMC) agent. The agent supports several OMA DM management objects (MO) through extensible wrappers called MO Wrappers.

Webif and Wi-Fi Connections

Wind River provides a web-based interface called Webif for managing Wi-Fi connections with IDP XT target systems.

Use the Webif Interface to add a web-based, customizable solution for configuring a wireless gateway router. The Webif interface provides configuration options for Ethernet (wired), Wi-Fi (802.11), and 3G connections. The Webif interface is hosted on an Appweb Web server running on the IDP XT target.

Configuring and building your platform project with the **--enable-addons=wr-idp** option and using **glib-idp** or **glib-idp-small** for the rootfs installs the binaries, scripts, and configuration files on the target file system at the following locations:

/etc/
/www/
/usr/lib/webif

View the Webif debug messages with the following commands:

```
# tail -f /var/log/appweb/error.log
```

Confirm the **appweb** service is running with the following command:

```
# ps -ef | grep appweb
```

For additional information on Webif, see *About Webif* on page 81 and the *X-Wrt* home page.

Related Links

OneAgent TR-069 Agent on page 35
The OneAgent TR-069 agent provides a protocol and API stack for communication between a TR-069-enabled client and server.

OneAgent OMA-DM Agent and MO Wrappers on page 36
SRM utilizes the Works System OneAgent OMA Device Management Communications (DMC) agent. The agent supports several OMA DM management objects (MO) through extensible wrappers called MO Wrappers.

OneAgent OMA-DM Agent and MO Wrappers

SRM utilizes the Works System OneAgent OMA Device Management Communications (DMC) agent. The agent supports several OMA DM management objects (MO) through extensible wrappers called MO Wrappers.

The DMA agent reports device information and executes commands using the OMA-DM protocol to a remote OMA server. The agent supports OMA DM management objects (MO) through extensible wrappers called MO Wrappers.

The MO Wrappers are a layer between the OMA DMC agent and the target device. The layer collects all of the incoming information and maps commands from the DMC to the device. The information consists of device properties and other system-related information. MO Wrappers

are designed to be extensible, making it possible to create new wrappers without making modifications to the DMC. Currently the following objects are supported: **DevInfo**, **DMAcc**, **ConnMO**, and **SCOMO**.

For more information, see:

Installing OMA-DM on page 182
About the OMA-DM Agent on page 223
About MO Wrappers on page 224

Related Links

OneAgent TR-069 Agent on page 35
The OneAgent TR-069 agent provides a protocol and API stack for communication between a TR-069-enabled client and server.

Webif and Wi-Fi Connections on page 36
Wind River provides a web-based interface called Webif for managing Wi-Fi connections with IDP XT target systems.

PART II

Key-Related Tasks

6

About Key Management

The system owner, the vendors for system hardware, software, and applications, and the end user (often the owner) have specific roles to play in maintaining a chain of trust for deployed devices.

System Owner

The system owner controls the security of the system and its devices. Vendors typically provide system images and application software in the form of installable packages (for example, RPMs). The owner serves as the authority that provides the vendors with a vendor CA certificate which allows them to sign their images and packages.

The system owner can use any tool to generate their own keys and certificates for vendors.

Vendor

Vendors sign the system image, file system, and application packages for secure remote management using a vendor certificate received from the system owner. During development they can simulate the owner role using the SRM Signing Tool (SST) until the actual CA certificate is available. Before deployment, they must replace the simulated signature with a real signature based on the CA certificate when it is available from the owner.

End User

The end user validates all software before installing it on a device. In common scenarios, the system owner also has the role of the end user.

7

Key Management for Vendors

Key Management for Vendors

Vendors sign the system image, file system, and application packages for secure remote management to prevent unauthorized changes to the device. IDP XT provides the SRM Signing Tool (SST) to sign boot loader and kernel binaries and RPM packages.

Device developers provide system software, including the system build. Application developers provide software in the form of signed RPMs. As vendors, both receive vendor certificates from the authority. The vendor certificate allows them to sign the system image and application packages so devices can tell if the authority has approved the software for installation on the device.

SST provides key management for vendors, whether they are creating devices and system hardware or applications. SST can run on your development host or can run independently on any Linux server without IDP XT installed. SST allows vendors to simulate the owner role during the development stage, before they receive owner certificates.

NOTE: Before final production, you must obtain keys and certificates directly from the owner and install them on the device. This is the only way to protect the deployed device from loading unauthorized software.

SST is located in the following folder:

***projDir*/layers/wr-idp/wr-srm/recipes-devtools/sst/files**

Usage Notes for SST

- SST supports **intel_atom_baytrail** and **intel_quark** devices.
- Using the **sign-all** subcommand requires root privileges.
- The target boots if the boot loader is not signed. However, if the bootloader is signed by an incorrect SST owner certificate, the system does not boot.
- The target does not boot if the boot loader is signed but the kernel is not signed correctly by SST.
- The target *does* boot if neither the boot loader nor the kernel image is signed.
- The validity period of certificate produced by SST is ten years starting at the local time on producing machine when the certificate was created. The certificate time must be consistent with that on the target.

Related Links

Installing Required Packages for SST on page 45
SST is a stand-alone utility which runs on a Linux machine. It uses standard Linux packages and does not require installing the IDP XT product.

Generating a New Owner Key and Certificates on page 46
Use the Wind River SST to generate owner keys and certificates during the development cycle before you receive actual keys and certificates from the owner.

Generating a New Vendor Key and Certificate on page 48
Like the owner, the vendor needs to generate private keys and a X.509v3 certificate. In production, the certificate will be issued by the device owner. During development, the SST tool allows you to create development keys and certificates yourself.

Signing Boot Loaders on page 50
Use the **SST sign-bootloader** command to sign your boot loader image.

Signing Kernels on page 51
Use the **SST sign-kernel** command to sign your kernel image.

Signing Application Folders on page 53
Use the **SST sign-app-folder** command to sign all the files in a given folder.

Signing a Single RPM on page 54
Use the **SST sign-rpm --mode=rpm** command to sign a single RPM.

Signing Multiple RPMs on page 55
Use the **SST sign-rpm --mode=dir** command to sign multiple RPM packages in a directory in batch mode.

Signing the rootfs File on page 57
Use the **SST sign-all** command to sign the rootfs file with the owner root certificate and the vendor certificate and private key .

SST Reference on page 59

SST uses a set of subcommands with additional options.

Installing Required Packages for SST

SST is a stand-alone utility which runs on a Linux machine. It uses standard Linux packages and does not require installing the IDP XT product.

For a complete list of required packages, see *Packages Required for SST* on page 243

- Use **apt-get** or **yum**, depending on your Linux distribution, to install any packages that are missing.

```
$ sudo apt-get install <pkg1> <pkg2> <pkg3> ...
$ sudo yum install <pkg1> <pkg2> <pkg3> ...
```

- (Optional) Propagate the tools to other hosts.
  a) Tar the file.

```
$ cd <projDir>/layers/wr-idp/wr-srm/recipes-devtools/sst/files
$ tar czvf SST.tgz *
```

  b) Copy the tar file to the desired location and untar.

```
$ cp SST.tgz <newLocation>
$ cd <newLocation>
$ tar xzvf SST.tgz
```

Related Links

Key Management for Vendors on page 43
Vendors sign the system image, file system, and application packages for secure remote management to prevent unauthorized changes to the device. IDP XT provides the SRM Signing Tool (SST) to sign boot loader and kernel binaries and RPM packages.

Generating a New Owner Key and Certificates on page 46
Use the Wind River SST to generate owner keys and certificates during the development cycle before you receive actual keys and certificates from the owner.

Generating a New Vendor Key and Certificate on page 48
Like the owner, the vendor needs to generate private keys and a X.509v3 certificate. In production, the certificate will be issued by the device owner. During development, the SST tool allows you to create development keys and certificates yourself.

Signing Boot Loaders on page 50
Use the **SST sign-bootloader** command to sign your boot loader image.

Signing Kernels on page 51
Use the **SST sign-kernel** command to sign your kernel image.

Signing Application Folders on page 53
Use the **SST sign-app-folder** command to sign all the files in a given folder.

Signing a Single RPM on page 54
Use the **SST sign-rpm --mode=rpm** command to sign a single RPM.

Signing Multiple RPMs on page 55
Use the **SST sign-rpm --mode=dir** command to sign multiple RPM packages in a directory in batch mode.

Signing the rootfs File on page 57

Use the **SST sign-all** command to sign the rootfs file with the owner root certificate and the vendor certificate and private key .

SST Reference on page 59
SST uses a set of subcommands with additional options.

Generating a New Owner Key and Certificates

Use the Wind River SST to generate owner keys and certificates during the development cycle before you receive actual keys and certificates from the owner.

SST allows vendors to simulate owner keys and certificates during development. It generates a simulated owner private key and the vendor certificate provided by the owner.

Step 1 Create a new **outputE** directory.

Step 2 Generate the private key and certificate.

The command syntax for generating keys and certificate is:

```
$ ./SST create-key --role=owner [--verbose=no] --machine=intel_atom_baytrail|
intel_quark \
[--name=<owner>] [--output-dir=.]
```

The options are as follows:

Option	Description	Default Value
verbose	Open or close the signing trace. Value can be **yes** or **no**.	no
name	User defined name for the role.	Name of the role
output-dir	The output directory where you can find your private key and CA certificate	SST current directory (“**.**”)
machine	Target architecture. Can be **intel_quark** or **intel_atom_baytrail**.	**intel_atom**
role	Trust role in SRM. Can only be **vendor** or **owner**.	N/A

This example uses the **intel_atom_baytrail** machine type for the **intel-atom-baytrail** BSP.

```
$ ./SST create-key --role=owner --verbose=no --name=ownerE --output-dir=./outputE \
--machine=intel_atom_baytrail
```

NOTE: If you already have a private key, the command to create the certificates uses the following syntax:

```
./SST create-key --role=owner --priv-key=owner-private.pem [--name=owner]
[--verbose=no] [--output-dir=.] --machine=intel_atom_baytrail|intel_quark
```

The options are as follows:

Option	Description	Default Value
verbose	Open or close the signing trace. Value can be **yes** or **no**.	no
name	Owner's name.	The value of the role.
output-dir	The output directory where you can find your private key and CA certificate	SST current directory (“**.**”)
machine	The target architecture can be **intel_quark** or **intel_atom_baytrail**.	**intel-atom**
priv-key	Your existing private key.	N/A
role	Trust role in SRM. Can only be **vendor** or **owner**.	N/A

Step 3 Verify the output files.

Locate the following files in the **./outputE** directory.

ownerE-private.pem
ownerE-cert.pem

Related Links

Key Management for Vendors on page 43
Vendors sign the system image, file system, and application packages for secure remote management to prevent unauthorized changes to the device. IDP XT provides the SRM Signing Tool (SST) to sign boot loader and kernel binaries and RPM packages.

Installing Required Packages for SST on page 45
SST is a stand-alone utility which runs on a Linux machine. It uses standard Linux packages and does not require installing the IDP XT product.

Generating a New Vendor Key and Certificate on page 48
Like the owner, the vendor needs to generate private keys and a X.509v3 certificate. In production, the certificate will be issued by the device owner. During development, the SST tool allows you to create development keys and certificates yourself.

Signing Boot Loaders on page 50
Use the **SST sign-bootloader** command to sign your boot loader image.

Signing Kernels on page 51
Use the **SST sign-kernel** command to sign your kernel image.

Signing Application Folders on page 53
Use the **SST sign-app-folder** command to sign all the files in a given folder.

Signing a Single RPM on page 54
Use the **SST sign-rpm --mode=rpm** command to sign a single RPM.

Signing Multiple RPMs on page 55
Use the **SST sign-rpm --mode=dir** command to sign multiple RPM packages in a directory in batch mode.

Signing the rootfs File on page 57

Use the **SST sign-all** command to sign the rootfs file with the owner root certificate and the vendor certificate and private key .

SST Reference on page 59
SST uses a set of subcommands with additional options.

Generating a New Vendor Key and Certificate

Like the owner, the vendor needs to generate private keys and a X.509v3 certificate. In production, the certificate will be issued by the device owner. During development, the SST tool allows you to create development keys and certificates yourself.

A vendor certificate allows the vendor to create signed images and packages for the device users. Users can then validate any packages they receive from vendors before they install them.

Step 1 Generate the private key and certificate.

The command syntax for generating keys and certificate is:

```
$ ./SST create-key --role=vendor [--verbose=no] [--name=<vendor>] \
[--output-dir=.][--issuer=owner]
```

The options are as follows:

Option	Description	Default Value
verbose	Open or close the signing trace. Value can be **yes** or **no**.	no
role	Trust role in SRM. Can only be **vendor** or **owner**.	N/A
name	Vendor name.	The value of role.
output-dir	The output directory where you can find your private key and CA certificate	SST current directory (“**.**”)
issuer	The name of the issuer who delegates to this vendor.	**owner**

```
$ ./SST create-key --role=vendor --verbose=no --name=vendorE --issuer=ownerE \
--output-dir=./outputE
```

NOTE: If you already have a private key, use the following command to create the certificate:

```
$ ./SST create-key --role=vendor [--verbose=no] --priv-key=vendor-private.pem \
[--name=vendor] [--output-dir=.] [--issuer=owner]
```

The options are as follows:

Option	Description	Default Value
verbose	Open or close the signing trace. Value can be **yes** or **no**.	no
name	Vendor name.	The value of role.
output-dir	The output directory where you can find your private key and CA certificate	SST current directory (“**.**”)
issuer	The name of the issuer who delegates to this vendor.	**owner**
priv-key	The vendor's existing private key.	N/A
role	Trust role in SRM. Can only be **vendor** or **owner**.	N/A

Step 2 Verify the output files.

Locate the following files in the **./outputE** directory.

vendorE-private.pem
vendorE-cert.pem

Related Links

Key Management for Vendors on page 43
Vendors sign the system image, file system, and application packages for secure remote management to prevent unauthorized changes to the device. IDP XT provides the SRM Signing Tool (SST) to sign boot loader and kernel binaries and RPM packages.

Installing Required Packages for SST on page 45
SST is a stand-alone utility which runs on a Linux machine. It uses standard Linux packages and does not require installing the IDP XT product.

Generating a New Owner Key and Certificates on page 46
Use the Wind River SST to generate owner keys and certificates during the development cycle before you receive actual keys and certificates from the owner.

Signing Boot Loaders on page 50
Use the **SST sign-bootloader** command to sign your boot loader image.

Signing Kernels on page 51
Use the **SST sign-kernel** command to sign your kernel image.

Signing Application Folders on page 53
Use the **SST sign-app-folder** command to sign all the files in a given folder.

Signing a Single RPM on page 54
Use the **SST sign-rpm --mode=rpm** command to sign a single RPM.

Signing Multiple RPMs on page 55
Use the **SST sign-rpm --mode=dir** command to sign multiple RPM packages in a directory in batch mode.

Signing the rootfs File on page 57

Use the **SST sign-all** command to sign the rootfs file with the owner root certificate and the vendor certificate and private key .

SST Reference on page 59
SST uses a set of subcommands with additional options.

Signing Boot Loaders

Use the **SST sign-bootloader** command to sign your boot loader image.

- Sign the boot loader using the command for your BSP.

BSP	Command

intel-atom-baytrail	This example uses the **intel_atom_baytrail** machine type for the **intel-atom-baytrail** BSP.
	`$ ./SST sign-bootloader --machine=intel_atom_baytrail --verbose=no \`
	`--owner-cert=./ownerE-cert.pem \`
	`--vendor-cert=./vendorE-cert.pem \`
	`--priv-key=./vendorE-private.pem\`
	`./grub.efi`
	When the command completes successfully, your **grub.efi** file is updated.

intel-quark	This example uses the **intel_quark** machine type for the **intel-quark** BSP.
	`$ ./SST sign-bootloader --machine=intel_quark --verbose=no \`
	`--owner-cert=<keysDir>/ownerE-cert.pem \`
	`--romkey-dir=<keysDir>/quark-romkey \`
	`<path-to-grub>/grub-idp.efi`
	When the command completes successfully, the **grub.efi** and **grub.conf** files are signed. They are included in **Flash-*xxx*.bin**, which is burned to the flash when you build the project.

The options are as follows:

Option	Description	Default Value
verbose	Open or close the signing trace. Value can be **yes** or **no**.	no
machine	The target architecture can be **intel_quark** or **intel_atom_baytrail**.	**intel_atom**
romkey-dir	The directory that includes the rom key files for verification of the secure ROM. Required for **intel_quark**; not used for **intel_atom_baytrail**.	Current directory

Option	Description	Default Value
owner-cert	The root certificate should be injected into the bootloader.	The **owner-cert.pem** file in the SST current directory .
vendor-cert	The device vendor certificate which was used to sign **grub.efi** for the BIOS.	The v**endor-cert.pem** file in the SST current directory.
priv-key	The private key which was used to sign **grub.efi** to for the BIOS.	**vendor-private.pem** the in current directory.

Related Links

Key Management for Vendors on page 43
Vendors sign the system image, file system, and application packages for secure remote management to prevent unauthorized changes to the device. IDP XT provides the SRM Signing Tool (SST) to sign boot loader and kernel binaries and RPM packages.

Installing Required Packages for SST on page 45
SST is a stand-alone utility which runs on a Linux machine. It uses standard Linux packages and does not require installing the IDP XT product.

Generating a New Owner Key and Certificates on page 46
Use the Wind River SST to generate owner keys and certificates during the development cycle before you receive actual keys and certificates from the owner.

Generating a New Vendor Key and Certificate on page 48
Like the owner, the vendor needs to generate private keys and a X.509v3 certificate. In production, the certificate will be issued by the device owner. During development, the SST tool allows you to create development keys and certificates yourself.

Signing Kernels on page 51
Use the **SST sign-kernel** command to sign your kernel image.

Signing Application Folders on page 53
Use the **SST sign-app-folder** command to sign all the files in a given folder.

Signing a Single RPM on page 54
Use the **SST sign-rpm --mode=rpm** command to sign a single RPM.

Signing Multiple RPMs on page 55
Use the **SST sign-rpm --mode=dir** command to sign multiple RPM packages in a directory in batch mode.

Signing the rootfs File on page 57
Use the **SST sign-all** command to sign the rootfs file with the owner root certificate and the vendor certificate and private key .

SST Reference on page 59
SST uses a set of subcommands with additional options.

Signing Kernels

Use the **SST sign-kernel** command to sign your kernel image.

- Sign the kernel using the command for your BSP.

BSP	Command

intel-atom-baytrail	This example uses the **intel_atom_baytrail** machine type for the **intel-atom-baytrail** BSP.
	`$ ./SST sign-kernel --machine=intel_atom_baytrail --verbose=no \`
	`--priv-key=./vendorE-private.pem \`
	`--vendor-cert=./vendorE-cert.pem ./bzImage-intel-atom-baytrail.bin`
	When the command completes successfully, your kernel image **bzImage-intel-atom.bin** file is updated.

intel-quark	This example uses the **intel_quark** machine type for the **intel-quark** BSP.
	`$ ./SST sign-kernel --machine=intel_quark --verbose=no \`
	`--priv-key=./vendorE-private.pem \`
	`--vendor-cert=./vendorE-cert.pem \`
	`--romkey-dir=./quark-romkey ./bzImage-intel-quark.bin`
	When the command completes successfully, your kernel image **bzImage-intel-quark.bin** file is updated and **bzImage.csbh** is generated..

The options are as follows:

Option	Description	Default Value
verbose	Open or close the signing trace. Value can be **yes** or **no**.	no
machine	The target architecture can be **intel_quark** or **intel_atom_baytrail**.	**intel_atom**
priv-key	The private key which was used to sign the kernel image.	**vendor-private.pem** the in current directory.
vendor-cert	The device vendor certificate should be injected into the kernel file.	The **vendor-cert.pem** file in the SST current directory .
romkey-dir	The directory that includes the rom key files for verification of the secure ROM. Required for **intel_quark**; not used for **intel_atom_baytrail**.	Current directory

Related Links

Key Management for Vendors on page 43
Vendors sign the system image, file system, and application packages for secure remote management to prevent unauthorized changes to the device. IDP XT provides the SRM Signing Tool (SST) to sign boot loader and kernel binaries and RPM packages.

Installing Required Packages for SST on page 45

SST is a stand-alone utility which runs on a Linux machine. It uses standard Linux packages and does not require installing the IDP XT product.

Generating a New Owner Key and Certificates on page 46
Use the Wind River SST to generate owner keys and certificates during the development cycle before you receive actual keys and certificates from the owner.

Generating a New Vendor Key and Certificate on page 48
Like the owner, the vendor needs to generate private keys and a X.509v3 certificate. In production, the certificate will be issued by the device owner. During development, the SST tool allows you to create development keys and certificates yourself.

Signing Boot Loaders on page 50
Use the **SST sign-bootloader** command to sign your boot loader image.

Signing Application Folders on page 53
Use the **SST sign-app-folder** command to sign all the files in a given folder.

Signing a Single RPM on page 54
Use the **SST sign-rpm --mode=rpm** command to sign a single RPM.

Signing Multiple RPMs on page 55
Use the **SST sign-rpm --mode=dir** command to sign multiple RPM packages in a directory in batch mode.

Signing the rootfs File on page 57
Use the **SST sign-all** command to sign the rootfs file with the owner root certificate and the vendor certificate and private key .

SST Reference on page 59
SST uses a set of subcommands with additional options.

Signing Application Folders

Use the **SST sign-app-folder** command to sign all the files in a given folder.

- Generate a signature-list file for all the binaries residing inside a folder.

```
$ ./SST sign-app-folder --verbose=no --priv-key=./vendorE-private.pem \
--output-list=./signature_listE apps_folder
```

The options are as follows:

Option	Description	Default Value
verbose	Open or close the signing trace. Value can be **yes** or **no**.	no
output-list	The signature list of the application binary in the *app_folder* directory.	the **signature_list** file in the current directory
priv-key	The private key which was used to get the application signature list.	**vendor-private.pem** in the SST current directory.

When the command completes successfully, a signature-list file named **signature_listE** is created.

Related Links

Key Management for Vendors on page 43
Vendors sign the system image, file system, and application packages for secure remote management to prevent unauthorized changes to the device. IDP XT provides the SRM Signing Tool (SST) to sign boot loader and kernel binaries and RPM packages.

Installing Required Packages for SST on page 45
SST is a stand-alone utility which runs on a Linux machine. It uses standard Linux packages and does not require installing the IDP XT product.

Generating a New Owner Key and Certificates on page 46
Use the Wind River SST to generate owner keys and certificates during the development cycle before you receive actual keys and certificates from the owner.

Generating a New Vendor Key and Certificate on page 48
Like the owner, the vendor needs to generate private keys and a X.509v3 certificate. In production, the certificate will be issued by the device owner. During development, the SST tool allows you to create development keys and certificates yourself.

Signing Boot Loaders on page 50
Use the **SST sign-bootloader** command to sign your boot loader image.

Signing Kernels on page 51
Use the **SST sign-kernel** command to sign your kernel image.

Signing a Single RPM on page 54
Use the **SST sign-rpm --mode=rpm** command to sign a single RPM.

Signing Multiple RPMs on page 55
Use the **SST sign-rpm --mode=dir** command to sign multiple RPM packages in a directory in batch mode.

Signing the rootfs File on page 57
Use the **SST sign-all** command to sign the rootfs file with the owner root certificate and the vendor certificate and private key .

SST Reference on page 59
SST uses a set of subcommands with additional options.

Signing a Single RPM

Use the **SST sign-rpm --mode=rpm** command to sign a single RPM.

- Sign the RPM.

```
$ ./SST  sign-rpm --verbose=no --mode=rpm --priv-key=./vendorE-private.pem \
./example.rpm
```

The options are as follows:

Option	Description	Default Value
verbose	Open or close the signing trace. Value can be **yes** or **no**.	no
mode	The RPM sign mode, either **rpm** or **dir**.	**rpm**

Option	Description	Default Value
priv-key	The private key which was used to sign the rpm package.	**vendor-private.pem** in the current directory.

When the command completes successfully, the RPM **example.rpm** is signed with the vendor private key **vendorE-private.pem**.

Related Links

Key Management for Vendors on page 43
Vendors sign the system image, file system, and application packages for secure remote management to prevent unauthorized changes to the device. IDP XT provides the SRM Signing Tool (SST) to sign boot loader and kernel binaries and RPM packages.

Installing Required Packages for SST on page 45
SST is a stand-alone utility which runs on a Linux machine. It uses standard Linux packages and does not require installing the IDP XT product.

Generating a New Owner Key and Certificates on page 46
Use the Wind River SST to generate owner keys and certificates during the development cycle before you receive actual keys and certificates from the owner.

Generating a New Vendor Key and Certificate on page 48
Like the owner, the vendor needs to generate private keys and a X.509v3 certificate. In production, the certificate will be issued by the device owner. During development, the SST tool allows you to create development keys and certificates yourself.

Signing Boot Loaders on page 50
Use the **SST sign-bootloader** command to sign your boot loader image.

Signing Kernels on page 51
Use the **SST sign-kernel** command to sign your kernel image.

Signing Application Folders on page 53
Use the **SST sign-app-folder** command to sign all the files in a given folder.

Signing Multiple RPMs on page 55
Use the **SST sign-rpm --mode=dir** command to sign multiple RPM packages in a directory in batch mode.

Signing the rootfs File on page 57
Use the **SST sign-all** command to sign the rootfs file with the owner root certificate and the vendor certificate and private key .

SST Reference on page 59
SST uses a set of subcommands with additional options.

Signing Multiple RPMs

Use the **SST sign-rpm --mode=dir** command to sign multiple RPM packages in a directory in batch mode.

- Sign all the RPMs in a directory.

```
$ ./SST sign-rpm --mode=dir --verbose=no --priv-key=./vendorE-private.pem rpmDir
```

The options are as follows:

Option	Description	Default Value
verbose	Open or close the signing trace. Value can be **yes** or **no**.	no
mode	The RPM sign mode, either **rpm** or **dir**.	**dir**
priv-key	The private key which was used to sign the rpm package.	**vendor-private.pem** in the current directory. The default key, for testing purposes, is located in ***projDir*/layers/wr-idp/wr-srm/files/keys/vendor-private.pem**

When the command completes successfully, all RPM packages inside the ***rpmDir*** directory are signed with the vendor private key **vendorE-private.pem**.

Related Links

Key Management for Vendors on page 43
Vendors sign the system image, file system, and application packages for secure remote management to prevent unauthorized changes to the device. IDP XT provides the SRM Signing Tool (SST) to sign boot loader and kernel binaries and RPM packages.

Installing Required Packages for SST on page 45
SST is a stand-alone utility which runs on a Linux machine. It uses standard Linux packages and does not require installing the IDP XT product.

Generating a New Owner Key and Certificates on page 46
Use the Wind River SST to generate owner keys and certificates during the development cycle before you receive actual keys and certificates from the owner.

Generating a New Vendor Key and Certificate on page 48
Like the owner, the vendor needs to generate private keys and a X.509v3 certificate. In production, the certificate will be issued by the device owner. During development, the SST tool allows you to create development keys and certificates yourself.

Signing Boot Loaders on page 50
Use the **SST sign-bootloader** command to sign your boot loader image.

Signing Kernels on page 51
Use the **SST sign-kernel** command to sign your kernel image.

Signing Application Folders on page 53
Use the **SST sign-app-folder** command to sign all the files in a given folder.

Signing a Single RPM on page 54
Use the **SST sign-rpm --mode=rpm** command to sign a single RPM.

Signing the rootfs File on page 57
Use the **SST sign-all** command to sign the rootfs file with the owner root certificate and the vendor certificate and private key .

SST Reference on page 59

SST uses a set of subcommands with additional options.

Signing the rootfs File

Use the **SST sign-all** command to sign the rootfs file with the owner root certificate and the vendor certificate and private key .

- Sign the rootfs file using the command for your BSP.

BSP	Command

intel-atom-baytrail	The example uses the **intel_atom_baytrail** machine type for the **intel-atom-baytrail** BSP.
	1. Sign the tar file.
	`$ sudo ./SST sign-all --mode=tarball --machine=intel_atom_baytrail \ --owner-cert=./ownerE-cert.pem --verbose=no\ --vendor-cert=./vendorE-cert.pem \ --priv-key=./vendorE-private.pem \ --output=./signed-images.tar.bz2 \ ./wrlinux-image-glibc-std-intel-atom-baytrail.tar.bz2`
	When the command completes successfully, the rootfs tar file **wrlinux-image-glibc-std-intel-atom-baytrail.tar.bz2** is signed and the resulting output file is stored as **signed-images.tar.bz2**.
	2. Sign the block file.
	`$ sudo ./SST sign-all --mode=image --machine=intel_atom_baytrail \ --owner-cert=./ownerE-cert.pem \ --vendor-cert=./vendorE-cert.pem \ --priv-key=./vendorE-private.pem \ --output=./signed-images.ext3 \ ./wrlinux-image-glibc-std-intel-atom-baytrail.ext3`
	When the command completes successfully, the rootfs block file **wrlinux-image-glibc-std-intel-atom-baytrail.ext3** is signed and the resulting output file is stored as **signed-images.ext3**.

BSP	Command

intel-quark	The example uses the **intel_quark** machine type for the **intel-quark** BSP.
	1. Sign the tar file.
	`$ sudo ./SST sign-all --mode=tarball --machine=intel_quark \ --owner-cert=./ownerE-cert.pem --verbose=no\ --romkey-dir=./quark-romkey \ --vendor-cert=./vendorE-cert.pem \ --priv-key=./vendorE-private.pem \ --output=./signed-images.tar.bz2 \ ./wrlinux-image-glibc-idp-intel-quark.tar.bz2`
	When the command completes successfully, the rootfs tar file **wrlinux-image-glibc-std-intel-quark.tar.bz2** is signed and the resulting output file is stored as **signed-images.tar.bz2**.
	2. Sign the block file.
	`$ sudo ./SST sign-all --mode=image --machine=intel_quark \ --owner-cert=./ownerE-cert.pem \ --romkey-dir=./quark-romkey \ --vendor-cert=./vendorE-cert.pem \ --priv-key=./vendorE-private.pem \ --output=./signed-images.ext3 \ ./wrlinux-image-glibc-idp-intel-quark.ext3`
	When the command completes successfully, the rootfs block file **wrlinux-image-glibc-std-intel-quark.ext3** is signed and the resulting output file is stored as **signed-images.ext3**.

The options are as follows:

Option	Description	Default Value
verbose	Open or close the signing trace. Value can be **yes** or **no**.	no
output	Signed rootfs file.	The srm-enabled-images.tar.bz2 file in current directory.
machine	The target architecture can be **intel_quark** or **intel_atom_baytrail**.	**intel_atom**
romkey-dir	The directory that includes the rom key files for verification of the secure ROM. Required for **intel_quark**; not used for **intel_atom_baytrail**.	Current directory
priv-key	The private key which was used to sign the kernel image.	**vendor-private.pem** the in current directory.
owner-cert	The root certificate should be injected into the bootloader.	The **owner-cert.pem** file in the SST current directory .

Option	Description	Default Value
vendor-cert	The device vendor certificate which should be injected into the kernel file.	The **vendor-cert.pem** file in the SST current directory .

Related Links

Key Management for Vendors on page 43
Vendors sign the system image, file system, and application packages for secure remote management to prevent unauthorized changes to the device. IDP XT provides the SRM Signing Tool (SST) to sign boot loader and kernel binaries and RPM packages.

Installing Required Packages for SST on page 45
SST is a stand-alone utility which runs on a Linux machine. It uses standard Linux packages and does not require installing the IDP XT product.

Generating a New Owner Key and Certificates on page 46
Use the Wind River SST to generate owner keys and certificates during the development cycle before you receive actual keys and certificates from the owner.

Generating a New Vendor Key and Certificate on page 48
Like the owner, the vendor needs to generate private keys and a X.509v3 certificate. In production, the certificate will be issued by the device owner. During development, the SST tool allows you to create development keys and certificates yourself.

Signing Boot Loaders on page 50
Use the **SST sign-bootloader** command to sign your boot loader image.

Signing Kernels on page 51
Use the **SST sign-kernel** command to sign your kernel image.

Signing Application Folders on page 53
Use the **SST sign-app-folder** command to sign all the files in a given folder.

Signing a Single RPM on page 54
Use the **SST sign-rpm --mode=rpm** command to sign a single RPM.

Signing Multiple RPMs on page 55
Use the **SST sign-rpm --mode=dir** command to sign multiple RPM packages in a directory in batch mode.

SST Reference on page 59
SST uses a set of subcommands with additional options.

SST Reference

SST uses a set of subcommands with additional options.

```
$ ./SST <sub-command> <options> [<target>]
```

Subcommands

Sub-command	Description
create-key	create private keys and X.509v3 certificates
sign-bootloader	process boot-loader images (U-Boot or GRUB)

Sub-command	Description
sign-kernel	process Linux kernel images (uImage or bzImage)
sign-app-folder	process folder and generate signature list
sign-rpm	process RPM packages
sign-all	process Wind River target rootfs tarball

Options and Targets

The `options` and the optional `target` depend on the sub-command. See *Related Links* for command examples.

Related Links

Key Management for Vendors on page 43
Vendors sign the system image, file system, and application packages for secure remote management to prevent unauthorized changes to the device. IDP XT provides the SRM Signing Tool (SST) to sign boot loader and kernel binaries and RPM packages.

Installing Required Packages for SST on page 45
SST is a stand-alone utility which runs on a Linux machine. It uses standard Linux packages and does not require installing the IDP XT product.

Generating a New Owner Key and Certificates on page 46
Use the Wind River SST to generate owner keys and certificates during the development cycle before you receive actual keys and certificates from the owner.

Generating a New Vendor Key and Certificate on page 48
Like the owner, the vendor needs to generate private keys and a X.509v3 certificate. In production, the certificate will be issued by the device owner. During development, the SST tool allows you to create development keys and certificates yourself.

Signing Boot Loaders on page 50
Use the **SST sign-bootloader** command to sign your boot loader image.

Signing Kernels on page 51
Use the **SST sign-kernel** command to sign your kernel image.

Signing Application Folders on page 53
Use the **SST sign-app-folder** command to sign all the files in a given folder.

Signing a Single RPM on page 54
Use the **SST sign-rpm --mode=rpm** command to sign a single RPM.

Signing Multiple RPMs on page 55
Use the **SST sign-rpm --mode=dir** command to sign multiple RPM packages in a directory in batch mode.

Signing the rootfs File on page 57
Use the **SST sign-all** command to sign the rootfs file with the owner root certificate and the vendor certificate and private key .

8

Validating Keys and Certificates

About Validating Device Keys and Certificates

Device users, whether customers or system owners, receive the signed RPM packages and associated certificates from vendors which they use to maintain the security of their devices.

The device manager performs the following tasks to maintain device security:

1. *Verify and install* the vendor CA certificate and keys provided by the owner or vendor.
2. *Verify packages* before installing them on the device.

Use the **imtools** interface to *perform installation and verification tasks*.

Related Links

Verifying the Vendor CA Certificate on page 62
Before installing packages, you must receive a vendor CA certificate from the owner to authorize a specific vendor to install packages on your device.

Verifying and Installing Packages on page 62
You can verify the authenticity of all packages when you install them on the target using the **rpm -i** command.

imtool Reference on page 63

This reference lists **imtool** options for operating on certificates and packages.

Verifying the Vendor CA Certificate

Before installing packages, you must receive a vendor CA certificate from the owner to authorize a specific vendor to install packages on your device.

You may also receive vendor CA certificates from authorized vendors to authorize their sub-vendors.

Step 1 Boot the IDP XT target device.

Step 2 Copy the vendor CA certificate to the device.

```
$ scp vendor-cert.pem root@<ip-address-of-IDP-target>
```

Step 3 Verify the vendor CA certificate.

```
# imtools --verifycert vendor-cert.pem
Certificate vendor-cert.pem has been installed.
```

If the CA certificate passes the verification test, the vendor certificate and the associated public key are automatically saved on the target. If the test fails, the vendor's public key is not saved.

Step 4 Verify that the certificate was added.

```
# imtools --listcert
vendor-cert.pem
```

Step 5 (Optional) Remove a particular certificate.

```
# imtools --removecert vendor-cert.pem
```

Related Links

About Validating Device Keys and Certificates on page 61
Device users, whether customers or system owners, receive the signed RPM packages and associated certificates from vendors which they use to maintain the security of their devices.

Verifying and Installing Packages on page 62
You can verify the authenticity of all packages when you install them on the target using the **rpm -i** command.

imtool Reference on page 63
This reference lists **imtool** options for operating on certificates and packages.

Verifying and Installing Packages

You can verify the authenticity of all packages when you install them on the target using the **rpm -i** command.

Before proceeding, you must verify the vendor CA certificate and install the vendor public keys.

- Verify and install an RPM package.

  When you use the **rpm** command to install an RPM package, it verifies the package automatically and imports the IMA signature after installation.

This example verifies and installs **rpma-1.0-r2.atom.rpm**.

```
# rpm -i rpma-1.0-r2.atom.rpm
MD5 Code:52166863377948d3b1acee6c62927277
Find right certificate: vendor-cert.pem
Ceritficate vendor-cert.pem is verified successfully
RPM package rpma-1.0-r2.atom.rpm is verified successfully
[ /usr/bin/a2 ]
[ /usr/bin/a1 ]
Import IMA signature successfully
```

Related Links

About Validating Device Keys and Certificates on page 61
Device users, whether customers or system owners, receive the signed RPM packages and associated certificates from vendors which they use to maintain the security of their devices.

Verifying the Vendor CA Certificate on page 62
Before installing packages, you must receive a vendor CA certificate from the owner to authorize a specific vendor to install packages on your device.

imtool Reference on page 63
This reference lists **imtool** options for operating on certificates and packages.

imtool Reference

This reference lists **imtool** options for operating on certificates and packages.

Action	Command
Verify the vendor CA certificate	`# imtools --verifycert vendor-cert.pem`
Verify the vendor CA certificate was added to the device	`# imtools --listcert` `vendor-cert.pem`
Remove a particular certificate	`# imtools --removecert vendor-cert.pem`
Verify an RPM package before installation	`# imtools --verifyrpm <package_name>`

Related Links

About Validating Device Keys and Certificates on page 61
Device users, whether customers or system owners, receive the signed RPM packages and associated certificates from vendors which they use to maintain the security of their devices.

Verifying the Vendor CA Certificate on page 62
Before installing packages, you must receive a vendor CA certificate from the owner to authorize a specific vendor to install packages on your device.

Verifying and Installing Packages on page 62
You can verify the authenticity of all packages when you install them on the target using the **rpm -i** command.

PART III

System Owner Tasks

9

Introduction to System Owner Tasks

The system owner role may perform tasks that are sometimes assigned to the system operator, the service provider, and the end user as well as typical owner tasks.

The system owner designs and specifies the system in order to achieve business objectives. The owner usually obtains device hardware, system software, and application software from device development vendors and application software vendors. The owner may obtain management software from a third-party or contract with a service provider for cloud services, but the owner manages the system and oversees system security.

Management Tools

WebIf system management

About Webif on page 81

Security Tools

Software repository

RPM Repository Server on page 95

Tamper-proof file system

Application Integrity Measurement (Tamper-proof File System) on page 27

10

McAfee Embedded Control

Full documentation of McAfee Embedded Control is provided by McAfee.

The *McAfee Embedded Control User Guide* is the primary document for McAfee Embedded Control. It describes key tasks required to install, configure, and run the product. It also points you to other documents that contain more detail or background information. This document is available as part of the Wind River Help installation.

The following tasks may be required when managing devices with McAfee Embedded Control:

- Enabling the product on client machines
- Verifying that only authorized code or programs can run
- Verifying that an application is tamper proof
- Verifying that binaries are tamper proof
- Performing emergency changes

The following reference information is available:

- *McAfee Application Control Product Guide*
- *McAfee Application Control Command Line Interface Guide*
- *McAfee Change Control Product Guide*
- *McAfee Change Control Command Line Interface Guide*

These documents are available as part of the Wind River Help installation.

11

Integrating OpenSSL and TPM

About TPM and Key Protection

You can use TPM and associated tools on devices that have TPM hardware. You can generate keys, store RSA key pairs, protect your private key, and perform encryption and decryption on the chip.

TPM is supported on Cross Hill and Advantech UTX-3115 boards.

When you generate a key using the TPM hardware, the private part of the key is stored in the TPM chip itself rather than in a private (permission protected) directory on your machine. All the encryption and decryption steps that require the private key must be done by the TPM chip.

Even if you generate the private key on a host machine using OpenSSL tools, you can still use the TPM engine to protect the private key. The key wrapping capability of TPM wraps the private key so that only TPM can parse and use it.

Related Links

Preparing to Use TPM on page 72
Enable TPM for generating and storing keys in the CMOS. You must also include the **openssl-tpm-engine** package in your platform project.

Creating a Key Using TPM Hardware on page 73

Wrapping a Software Key Into TPM Storage on page 74

The term `wrapping` means encryption which stores the base-64 PEM-formatted software key into the TPM, wraps it with the SRK key, and creates the output index file **rootkey.pem**.

Testing the OpenSSL TPM Engine Integration on page 75
Wind River IDP XT provides a set of scripts (a demo application) that you can use to confirm your OpenSSL TPM engine integration is working properly.

About the Open Source Toolkit for SSL/TLS (OpenSSL) on page 78
The OpenSSL package included in IDP XT is configured for enhanced security.

SSL API Functions on page 78
This reference lists the APIs used in the demo application.

http://trousers.sourceforge.net/
http://sourceforge.net/projects/trousers/files/?source=navbar
https://wiki.emulab.net/wiki/Tpm
http://www.openssl.org/docs/ssl/ssl.html#API_FUNCTIONS
https://wiki.emulab.net/wiki/Tpmopenssl_tpm_engine-0.4.2/README

Preparing to Use TPM

Enable TPM for generating and storing keys in the CMOS. You must also include the **openssl-tpm-engine** package in your platform project.

TPM is supported on Intel Atom boards.

For information on enabling TPM in the CMOS, see *Enabling Encrypted Storage* on page 174.

The **openssl-tpm-engine** package provides the **create_tpm_key** tool and the **libtpm.so** dynamic library which act as an engine. The **create_tpm_key** tool generates the TPM hardware key or wraps a software key (typically generated by the **openssl genrsa** command) into the TPM hardware. The **libtpm.so** library is dynamically loaded by the **openssl** command with the **-engine tpm** option.

The TPM hardware engine supports two passwords:

a well-known password (20 bytes of zero)
a user supplied password

For more information, see Step 3 on page 73.

Step 1 Build a platform project and boot your board in the standard way.

You must including at least the following option in your configure command:

- **--enable-addons=wr-idp**

For more information see:

- *Building Platform Projects for Quark Boards* on page 122
- *Building Platform Projects for Advantech UTX-3115 Boards* on page 126

Step 2 Log into the target and check the ownership of the TPM.

```
# tpm_statistic
```

If the value in the **Owned Status** field is **Owned**, this task is complete and you are ready to generate keys.

If the value in the **Owned Status** field is **Not Owned**, proceed with the next step.

Step 3 Take ownership of TPM.

Option	Command
Set the Owner and SRK secret to a well-known password, which in this case is 20 bytes of zeros.	`# tpm_takeownership -y -z`
Set new Owner password and SRK secret.	`# tpm_changeownerauth -z -s -o`

NOTE: If the SRK has already seen set to something other than the well-known password (20 bytes of zero), you can reset it to the well-known password first with the **tpm_changeownerauth -r -s -o** command.

Related Links

About TPM and Key Protection on page 71
You can use TPM and associated tools on devices that have TPM hardware. You can generate keys, store RSA key pairs, protect your private key, and perform encryption and decryption on the chip.

Creating a Key Using TPM Hardware on page 73

Wrapping a Software Key Into TPM Storage on page 74
The term *wrapping* means encryption which stores the base-64 PEM-formatted software key into the TPM, wraps it with the SRK key, and creates the output index file **rootkey.pem**.

Testing the OpenSSL TPM Engine Integration on page 75
Wind River IDP XT provides a set of scripts (a demo application) that you can use to confirm your OpenSSL TPM engine integration is working properly.

About the Open Source Toolkit for SSL/TLS (OpenSSL) on page 78
The OpenSSL package included in IDP XT is configured for enhanced security.

SSL API Functions on page 78
This reference lists the APIs used in the demo application.

Creating a Key Using TPM Hardware

You can create a key based on the well-known password for use during development. However, for deployment you must create a unique key.

- Generate the hardware key.

Option	Command
SRK is set to a well-known password.	`# create_tpm_key rootkey.pem -z` `Success`

You have already set the SRK password.	`# create_tpm_key rootkey.pem`
	`SRK Password:enter_your_password`
	`Success`

Both commands store the keys in the TPM hardware and generate an ouput index file **rootkey.pem**. If the following tags occur in the file, the key has been generated and stored in the TPM chip.

```
-----BEGIN/END TSS KEY BLOB-----
```

- (Optional) View usage options for the script.

```
# create_tpm_key -h
create_tpm_key: create a TPM key and write it to disk
usage: create_tpm_key [options] <filename>
Options:
-e|--enc-scheme encryption scheme to use [PKCSV15] or OAEP
-q|--sig-scheme signature scheme to use [DER] or SHA1
-s|--key-size key size in bits [2048]
-z|--zerokey use well known 20 bytes zero as SRK password.
-a|--auth require a password for the key [NO]
-p|--popup use TSS GUI popup dialogs to get the password for the key [NO] (implies
--auth)
-w|--wrap [file] wrap an existing openssl PEM key
-h|--help print this help message
```

Related Links

About TPM and Key Protection on page 71
You can use TPM and associated tools on devices that have TPM hardware. You can generate keys, store RSA key pairs, protect your private key, and perform encryption and decryption on the chip.

Preparing to Use TPM on page 72
Enable TPM for generating and storing keys in the CMOS. You must also include the **openssl-tpm-engine** package in your platform project.

Wrapping a Software Key Into TPM Storage on page 74
The term *wrapping* means encryption which stores the base-64 PEM-formatted software key into the TPM, wraps it with the SRK key, and creates the output index file **rootkey.pem**.

Testing the OpenSSL TPM Engine Integration on page 75
Wind River IDP XT provides a set of scripts (a demo application) that you can use to confirm your OpenSSL TPM engine integration is working properly.

About the Open Source Toolkit for SSL/TLS (OpenSSL) on page 78
The OpenSSL package included in IDP XT is configured for enhanced security.

SSL API Functions on page 78
This reference lists the APIs used in the demo application.

Wrapping a Software Key Into TPM Storage

The term *wrapping* means encryption which stores the base-64 PEM-formatted software key into the TPM, wraps it with the SRK key, and creates the output index file **rootkey.pem**.

Step 1 Create a key on a Linux host using OpenSSL tools.

```
$ openssl genrsa -out softkey.pem 1024
```

Step 2 Transfer the key from the host to the IDP XT target.

```
$ scp softkey.pem root@IP-Address-of-IDP-Target
```

Step 3 Wrap the key on the IDP XT target.

Option	Command
SRK is set to a well-known password.	`# create_tpm_key -z -w softkey.pem -s 1024 rootkey.pem` `Success`

You have already set the SRK password.	`# create_tpm_key -w softkey.pem -s 1024 rootkey.pem`
	`SRK Password:enter_your_password`
	`Success`

Related Links

About TPM and Key Protection on page 71
You can use TPM and associated tools on devices that have TPM hardware. You can generate keys, store RSA key pairs, protect your private key, and perform encryption and decryption on the chip.

Preparing to Use TPM on page 72
Enable TPM for generating and storing keys in the CMOS. You must also include the **openssl-tpm-engine** package in your platform project.

Creating a Key Using TPM Hardware on page 73

Testing the OpenSSL TPM Engine Integration on page 75
Wind River IDP XT provides a set of scripts (a demo application) that you can use to confirm your OpenSSL TPM engine integration is working properly.

About the Open Source Toolkit for SSL/TLS (OpenSSL) on page 78
The OpenSSL package included in IDP XT is configured for enhanced security.

SSL API Functions on page 78
This reference lists the APIs used in the demo application.

Testing the OpenSSL TPM Engine Integration

Wind River IDP XT provides a set of scripts (a demo application) that you can use to confirm your OpenSSL TPM engine integration is working properly.

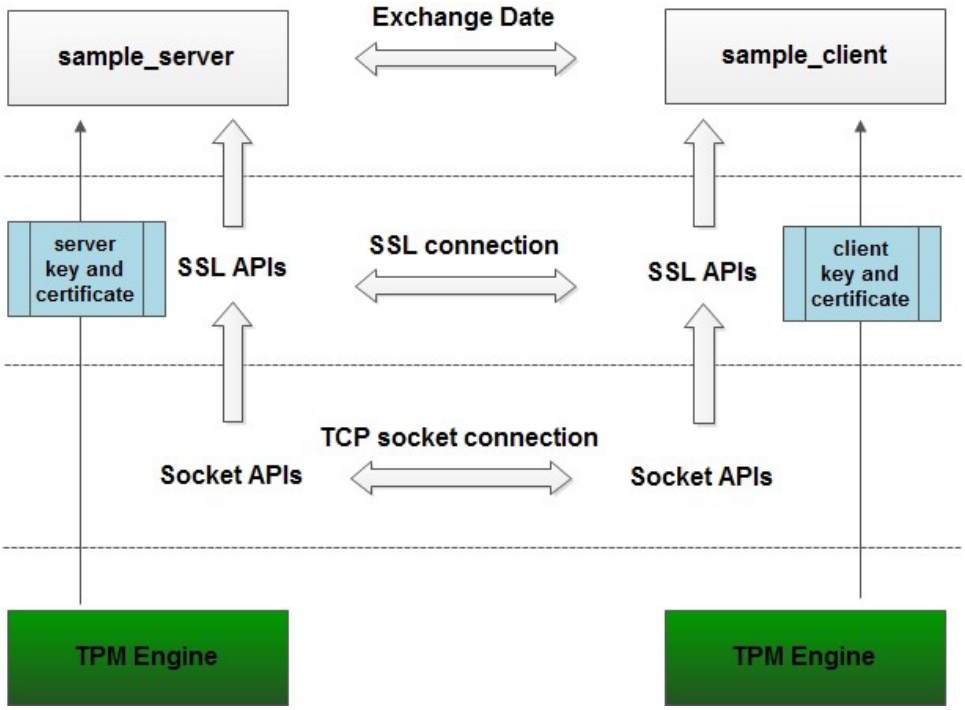
The source for the demo application is located in the following directory:

***projDir*/layers/wr-idp/wr-srm/recipes-samples/sample-openssl-tpmengine/files/src**

The test scripts generate client and server keys and start a test server and client. You can then send data securely from the client to the server. For a detailed understanding of how OpenSSL works with the TPM engine, view the test script **test-openssl-tpm-engine** located in the following directory:

/root/examples/openssl-tpm-engine

The following is the block diagram for the demo application:



NOTE: You can set the SRK password directly by adding the **-k** option to the **test-openssl-tpm-engine** command. For example:

```
# ./test-openssl-tpm-engine genkeys -k "your-password"
```

The well-known key consists of non-ASCII characters (20 bytes of zeros) and cannot be typed on the terminal. There are two methods of setting the well-known key:

Using the **-k** "**#WELLKNOWN#**":

```
# ./test-openssl-tpm-engine genkeys -k "#WELLKNOWN#"
```

Using the **-z** option:

```
# ./test-openssl-tpm-engine genkeys -z
```

This example uses the well-known key. To use the SRK password, set it and remove the **-z** option from the example commands.

Step 1 Build the **sample-openssl-tpm-engine** package on the host.

```
$ make -C build sample-openssl-tpm-engine
```

Step 2 Copy the RPM to the target.

The file is located in the following directory:

***projDir*/build/sample-openssl-tpm-engine-1.0-r0/deploy-rpms/atom/sample-openssl-tpm-engine-1.0-r0.atom.rpm**

Step 3 Change to the test directory.

```
# cd /root/examples/openssl-tpm-engine
```

Step 4 Generate the keys and certificates for CA/Server/Client using the sample test script.

```
# ./test-openssl-tpm-engine genkeys -z -c
```

Step 5 Start the OpenSSL TLS server.

```
# ./test-openssl-tpm-engine server -z
```

Step 6 Open another terminal by pressing **CTRL+ALT+F2** and log in.

Step 7 Start the OpenSSL TLS client.

```
# ./test-openssl-tpm-engine client -z
```

When you connect successfully, the following messages appear:

- **Hello World!** on the server console
- **I could hear you** on the client console

```
SEND MESSAGE  [12]: Hello World!
SERVER REPLY [17]: I could hear you.
```

```
GET MESSAGE [12]: Hello World!
SEND REPLY  [17]: I could hear you.
```

NOTE: The default client/server used in the example are from the demo application. To test the s_server/s_client demo from OpenSSL, add the **–s** option to the **test-openssl-tpm-engine** command. This demo implements a TlSv1 connection and can send and receive characters from each side. Press **CTRL+C** to terminate the server and client program on both the terminals.

Related Links

About TPM and Key Protection on page 71
You can use TPM and associated tools on devices that have TPM hardware. You can generate keys, store RSA key pairs, protect your private key, and perform encryption and decryption on the chip.

Preparing to Use TPM on page 72
Enable TPM for generating and storing keys in the CMOS. You must also include the **openssl-tpm-engine** package in your platform project.

Creating a Key Using TPM Hardware on page 73

Wrapping a Software Key Into TPM Storage on page 74
The term *wrapping* means encryption which stores the base-64 PEM-formatted software key into the TPM, wraps it with the SRK key, and creates the output index file **rootkey.pem**.

About the Open Source Toolkit for SSL/TLS (OpenSSL) on page 78
The OpenSSL package included in IDP XT is configured for enhanced security.

SSL API Functions on page 78

This reference lists the APIs used in the demo application.

About the Open Source Toolkit for SSL/TLS (OpenSSL)

The OpenSSL package included in IDP XT is configured for enhanced security.

OpenSSL is a cryptography toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) network protocols and related cryptography standards required by them.

The OpenSSL package included in IDP XT is configured with the following security enhancements:

- Support for 40/56 bit keys is deprecated. The minimum key length is 128 bits.
- DES support is disabled.
- Elliptic Curve Diffie-Hellman (ECDH) support is disabled.
- Elliptic Curve Digital Signature Algorithm GF support is disabled.
- MD2 support is disabled.
- TLS v1.1 support is enabled.
- TLS v1.2 support is enabled.

Related Links

About TPM and Key Protection on page 71
You can use TPM and associated tools on devices that have TPM hardware. You can generate keys, store RSA key pairs, protect your private key, and perform encryption and decryption on the chip.

Preparing to Use TPM on page 72
Enable TPM for generating and storing keys in the CMOS. You must also include the **openssl-tpm-engine** package in your platform project.

Creating a Key Using TPM Hardware on page 73

Wrapping a Software Key Into TPM Storage on page 74
The term `wrapping` means encryption which stores the base-64 PEM-formatted software key into the TPM, wraps it with the SRK key, and creates the output index file **rootkey.pem**.

Testing the OpenSSL TPM Engine Integration on page 75
Wind River IDP XT provides a set of scripts (a demo application) that you can use to confirm your OpenSSL TPM engine integration is working properly.

SSL API Functions on page 78
This reference lists the APIs used in the demo application.

SSL API Functions

This reference lists the APIs used in the demo application.

A comprehensive list of OpenSSL APIs can be found at:

http://www.openssl.org/docs/ssl/ssl.html#API_FUNCTIONS.

Initialization

SSL_load_error_strings( )
OpenSSL_add_all_algorithms( )

TPM Engine

SSL_CTX_use_PrivateKey_tpm( )
ENGINE_load_tpm( ) (load and initialize the TPM engine)
ENGINE_by_id( ) (get the TPM engine handler)
ENGINE_load_private_key( ) (load the private key into the TPM)
SSL_CTX_use_PrivateKey( ) (bind the private key to SSL ctx)

Server

TLSv1_server_method( )
SSL_CTX_new( )
SSL_CTX_use_certificate_file( )
SSL_CTX_use_PrivateKey_file( )
SSL_CTX_check_private_key( )
SSL_CTX_set_verify( )
SSL_CTX_set_client_CA_list( )
SSL_new();SSL_set_fd( )
SSL_accept( )
SSL_get_cipher( )
SSL_get_version( )
SSL_read();SSL_write( )

Client

TLSv1_client_method( )
SSL_CTX_new( )
SSL_CTX_use_certificate_file( )
SSL_CTX_use_PrivateKey_file( )
SSL_CTX_check_private_key( )
SSL_CTX_set_verify( )
SSL_new( )
SSL_set_fd( )
SSL_connect( )
SSL_get_cipher( )
SSL_get_version( )
SSL_read( )
SSL_write( )
SSL_shutdown( )

Related Links

About TPM and Key Protection on page 71
You can use TPM and associated tools on devices that have TPM hardware. You can generate keys, store RSA key pairs, protect your private key, and perform encryption and decryption on the chip.

Preparing to Use TPM on page 72
Enable TPM for generating and storing keys in the CMOS. You must also include the **openssl-tpm-engine** package in your platform project.

Creating a Key Using TPM Hardware on page 73

Wrapping a Software Key Into TPM Storage on page 74

The term `wrapping` means encryption which stores the base-64 PEM-formatted software key into the TPM, wraps it with the SRK key, and creates the output index file **rootkey.pem**.

Testing the OpenSSL TPM Engine Integration on page 75
Wind River IDP XT provides a set of scripts (a demo application) that you can use to confirm your OpenSSL TPM engine integration is working properly.

About the Open Source Toolkit for SSL/TLS (OpenSSL) on page 78
The OpenSSL package included in IDP XT is configured for enhanced security.

12

Webif Router Configuration

About Webif

You can use the Webif interface to configure your gateway or router (your IDP XT device) in the same way you would configure your home Wi-Fi router.

When you create a default SRM platform project, the **wr-idp-devkit** layer and glibc-idp are automatically configured. This causes Webif to be included in the project and sets up your IDP XT target to act as a gateway by default.

Related Links

Webif Interface Main Tabs on page 82
The Main tabs provide the bases for making configuration changes to your Wi-Fi gateway.

Webif Interface Default Settings on page 83
Refer to these default settings when you need to setup or modify your Wi-Fi router.

Webif Interface Prerequisites on page 84
Before starting the Webif interface workflow, you must have the correct hardware and software.

Launching and Accessing the Webif Interface on page 85

You can use the Webif interface to launch and access your network rather than the command line interface.

Saving Changes in the Webif Interface on page 88
Once you make changes to your router configuration, you must save them.

Network Tab on page 89
Use the Network tab to view and change basic networking parameters, including LAN, WAN, and WWAN for your gateway router.

Changing WiFi Mode from AP to Client via Webif on page 90
Change the operating mode of your WiFi module from access point mode to client mode.

Changing WiFi Mode from Client to AP via Webif on page 92
Change the operating mode of your WiFi module from client mode to access point mode.

Starting a Zigbee Network on page 94
If you use Webif, you can use it join and leave a network instead of using the command line.

Webif Interface Main Tabs

The Main tabs provide the bases for making configuration changes to your Wi-Fi gateway.

The default Webif interface provides the following tabs and their relevant settings to make configuration changes to your residential home gateway router:

Info Tab

Use to get system information.

Graphs Tab

Use to get information on CPU usage and traffic on various interfaces.

Status Tab

Use to view the status of routers, modules, the system, and so on.

Log Tab

Use to view the **syslog** and **dmesg** logs.

System Tab

Use to set system-specific setting, such as time, theme and language, access control, password, and backup and restore, and to upgrade and reset the router.

Includes Startup, Crontabs, File Editor, Mountpoints, and TPM subtabs.

Network Tab

Use to view and set detailed networking parameters. These include WAN, LAN, WWAN, Wireless, Bluetooth, Firewall, DHCP, Hosts, Routes, UPnP, Zigbee, MultiWAN, and Tweaks. For more information, see *Network Tab* on page 89.

VPN Tab

Use to add a new IPsec configuration rule for your own IPsec-based VPN network.

Device Agent Tab

Use to manage repositories and agents. Includes RPM Repository, WKS OMA DMC, and OneAgent TR069 subtabs.

In addition, you can create your own pages (tabs) and include them with your Intelligent Device Platform system. See *Adding a New Webif Page* on page 188.

Related Links

About Webif on page 81
You can use the Webif interface to configure your gateway or router (your IDP XT device) in the same way you would configure your home Wi-Fi router.

Webif Interface Default Settings on page 83
Refer to these default settings when you need to setup or modify your Wi-Fi router.

Webif Interface Prerequisites on page 84
Before starting the Webif interface workflow, you must have the correct hardware and software.

Launching and Accessing the Webif Interface on page 85
You can use the Webif interface to launch and access your network rather than the command line interface.

Saving Changes in the Webif Interface on page 88
Once you make changes to your router configuration, you must save them.

Network Tab on page 89
Use the Network tab to view and change basic networking parameters, including LAN, WAN, and WWAN for your gateway router.

Changing WiFi Mode from AP to Client via Webif on page 90
Change the operating mode of your WiFi module from access point mode to client mode.

Changing WiFi Mode from Client to AP via Webif on page 92
Change the operating mode of your WiFi module from client mode to access point mode.

Starting a Zigbee Network on page 94
If you use Webif, you can use it join and leave a network instead of using the command line.

Webif Interface Default Settings

Refer to these default settings when you need to setup or modify your Wi-Fi router.

When working with the Webif interface, you will need the following information:

- **Web login username/password**: admin/admin
- **WAN port**: eth0 (DHCP to get IP address)
- **WLAN** (Cross Hill, Clanton Hill, and Galileo boards only):
  - wlan0
  - Radio on
  - 802.11N/G mode
  - ESSID:IDPDK-XXXX (where XXXX is the last 4 hexadecimal digits of the IDP XT WLAN MAC address)
  - Authentication: WPA2(PSK)
  - Password: (**windriveridp**)
- **Bridge**: br-bridge (including wlan0 and other Ethernet interfaces, STATIC IP: 192.168.1.1, with DHCP server running on it)

You can use the Webif interface to change the default configuration. See *Launching and Accessing the Webif Interface* on page 85 and *Saving Changes in the Webif Interface* on page 88.

Related Links

About Webif on page 81

You can use the Webif interface to configure your gateway or router (your IDP XT device) in the same way you would configure your home Wi-Fi router.

Webif Interface Main Tabs on page 82
The Main tabs provide the bases for making configuration changes to your Wi-Fi gateway.

Webif Interface Prerequisites on page 84
Before starting the Webif interface workflow, you must have the correct hardware and software.

Launching and Accessing the Webif Interface on page 85
You can use the Webif interface to launch and access your network rather than the command line interface.

Saving Changes in the Webif Interface on page 88
Once you make changes to your router configuration, you must save them.

Network Tab on page 89
Use the Network tab to view and change basic networking parameters, including LAN, WAN, and WWAN for your gateway router.

Changing WiFi Mode from AP to Client via Webif on page 90
Change the operating mode of your WiFi module from access point mode to client mode.

Changing WiFi Mode from Client to AP via Webif on page 92
Change the operating mode of your WiFi module from client mode to access point mode.

Starting a Zigbee Network on page 94
If you use Webif, you can use it join and leave a network instead of using the command line.

Webif Interface Prerequisites

Before starting the Webif interface workflow, you must have the correct hardware and software.

The following hardware and software are required for using the Webif interface:

- Wind River Intelligent Device Platform XT 2.0 installation on top of a Wind River Linux 5.0.1 installation with RCPL updates on a supported host.
- Any Webif-supported board:
  - Cross Hill
  - Clanton Hill
  - Galileo
  - Advantech UTX-3115
- Ethernet or Wi-Fi (recommended) connection.
- A connection from the device to the host the host.
  - A serial connection for Cross Hill, Clanton Hill, and Galileo boards.
  - An HDMI to DVI connection for Advantech UTX-3115 boards.
- A host that connects to your IDP XT device using Ethernet or Wi-Fi. If you use Ethernet, make sure your host is connected to the IDP XT target using an Ethernet cable. If you use Wi-Fi, make sure your wireless radio on your host machine is switched ON. You may use your Linux development host if it meets these requirements.

Before proceeding, you must connect the hardware board to the host and power the board on.

Related Links

About Webif on page 81

You can use the Webif interface to configure your gateway or router (your IDP XT device) in the same way you would configure your home Wi-Fi router.

Webif Interface Main Tabs on page 82
The Main tabs provide the bases for making configuration changes to your Wi-Fi gateway.

Webif Interface Default Settings on page 83
Refer to these default settings when you need to setup or modify your Wi-Fi router.

Launching and Accessing the Webif Interface on page 85
You can use the Webif interface to launch and access your network rather than the command line interface.

Saving Changes in the Webif Interface on page 88
Once you make changes to your router configuration, you must save them.

Network Tab on page 89
Use the Network tab to view and change basic networking parameters, including LAN, WAN, and WWAN for your gateway router.

Changing WiFi Mode from AP to Client via Webif on page 90
Change the operating mode of your WiFi module from access point mode to client mode.

Changing WiFi Mode from Client to AP via Webif on page 92
Change the operating mode of your WiFi module from client mode to access point mode.

Starting a Zigbee Network on page 94
If you use Webif, you can use it join and leave a network instead of using the command line.

Launching and Accessing the Webif Interface

You can use the Webif interface to launch and access your network rather than the command line interface.

This example uses a Quark board in most steps.

Step 1 Modify the configure command to include the Webif component.

Use the following configure command:

```
$ $WIND_LINUX_CONFIGURE --enable-board=intel-quark --enable-kernel=standard \
--enable-rootfs=glibc-idp --enable-addons=wr-idp \
--enable-jobs=4 --enable-parallel-pkgbuilds=4
```

It is unnecessary to include **feature/webif** in the configure line to include in the Webif component; Webif is included automatically when you use **--enable-rootfs=glibc-idp**.

Complete the remaining steps for secure boot to confirm the boot is working correctly.

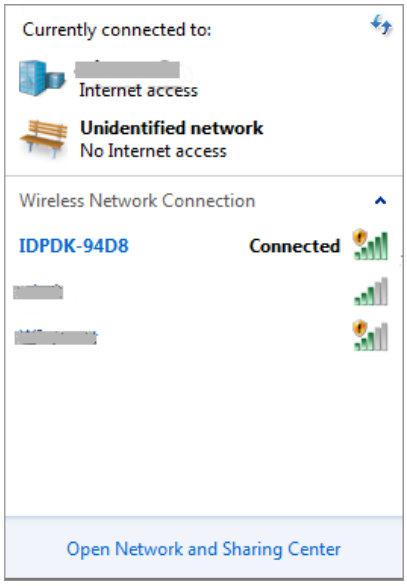
Optionally, you can perform a secure or verified boot on the IDP XT target using a modified configure command.

For board-specific information, see:

Performing a Secure Boot on Cross Hill and Clanton Hill Boards on page 145
Performing a Secure Boot on Advantech UTX-3115 Using UEFI on page 146
Performing a Verified Boot on page 148

NOTE: Wind River recommends that you perform a secure boot of your IDP XT target before using Webif to configure your IDP device. The instructions that follow assume that you will be performing a secure boot.

Step 2 Configure your host/target connection.

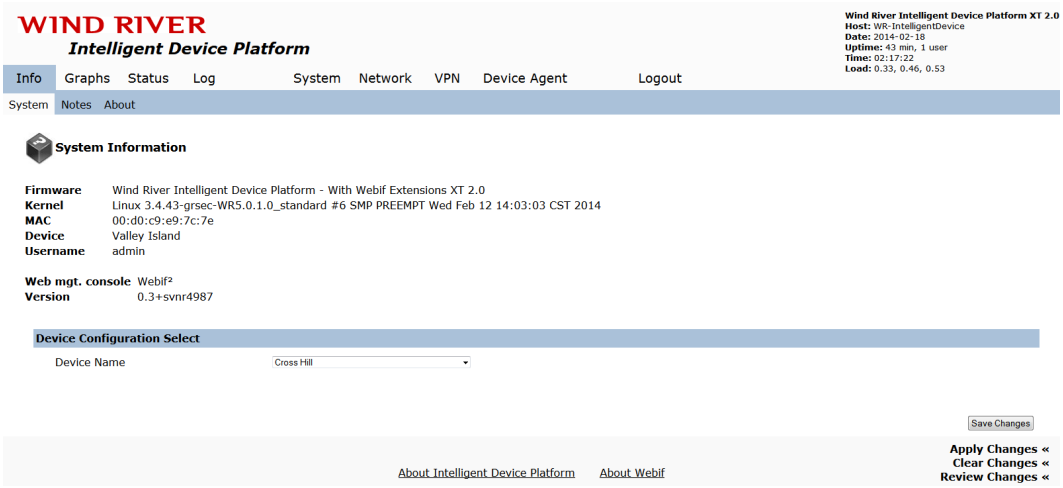
Connection Type	Procedure
If your IDP XT target has a WiFi module:	On your host machine, search for the Wireless network (**IDPDK-XXXX**) started by the IDP target and connect to it. The default password is **windriveridp** and is stored in the **/etc/config/wireless** file on the IDP XT target. 
If your IDP XT target does not have a WiFi module:	Connect the IDP XT target to your host machine using an Ethernet cable. Find out the IP address assigned to **eth0** of the IDP XT target. Both systems should be able to ping to each other.

Step 3 Start the Webif interface in a Web browser on your host.

Connection Type	Procedure
If your IDP XT target has a WiFi module:	Type **https://192.168.1.1** and press **ENTER**.
If your IDP XT target does not have a WiFi module:	Type **https://*ipAddrOfEth0*** and press **ENTER**.

Step 4 At the prompt, enter **admin** for both the username and password, then click **OK**.

The Webif interface displays the System Information page.



Step 5 To change the default configuration setting, see *Saving Changes in the Webif Interface* on page 88.

Step 6 To add a new Webif page in the interface, see *Adding a New Webif Page* on page 188.

You can also start, create, and join a ZigBee network from the Webif interface. For more information, see:

- *Starting a Zigbee Network* on page 94

Related Links

About Webif on page 81
You can use the Webif interface to configure your gateway or router (your IDP XT device) in the same way you would configure your home Wi-Fi router.

Webif Interface Main Tabs on page 82
The Main tabs provide the bases for making configuration changes to your Wi-Fi gateway.

Webif Interface Default Settings on page 83
Refer to these default settings when you need to setup or modify your Wi-Fi router.

Webif Interface Prerequisites on page 84
Before starting the Webif interface workflow, you must have the correct hardware and software.

Saving Changes in the Webif Interface on page 88
Once you make changes to your router configuration, you must save them.

Network Tab on page 89
Use the Network tab to view and change basic networking parameters, including LAN, WAN, and WWAN for your gateway router.

Changing WiFi Mode from AP to Client via Webif on page 90
Change the operating mode of your WiFi module from access point mode to client mode.

Changing WiFi Mode from Client to AP via Webif on page 92
Change the operating mode of your WiFi module from client mode to access point mode.

Starting a Zigbee Network on page 94

If you use Webif, you can use it join and leave a network instead of using the command line.

Saving Changes in the Webif Interface

Once you make changes to your router configuration, you must save them.

When you make changes to your configuration, consider the following information to ensure your changes are saved properly.

Step 1 Edit your configuration in Webif. See *Webif Interface Main Tabs* on page 82.

Step 2 Click the **Save Changes** button located at the bottom right of the configuration page.

The **Save Changes** action calls **/www/cgi-bin/webif/config.sh** to save changes to **/tmp/.uci/${package}** or **/tmp/.webif/***. These changes are saved to a temporary location. You must click **Apply Changes** to make them permanent.

Step 3 Click **Apply Changes**.

The **Apply Changes** action calls **/usr/lib/webif/apply.sh** to save and apply your changes.

Related Links

About Webif on page 81
You can use the Webif interface to configure your gateway or router (your IDP XT device) in the same way you would configure your home Wi-Fi router.

Webif Interface Main Tabs on page 82
The Main tabs provide the bases for making configuration changes to your Wi-Fi gateway.

Webif Interface Default Settings on page 83
Refer to these default settings when you need to setup or modify your Wi-Fi router.

Webif Interface Prerequisites on page 84
Before starting the Webif interface workflow, you must have the correct hardware and software.

Launching and Accessing the Webif Interface on page 85
You can use the Webif interface to launch and access your network rather than the command line interface.

Network Tab on page 89
Use the Network tab to view and change basic networking parameters, including LAN, WAN, and WWAN for your gateway router.

Changing WiFi Mode from AP to Client via Webif on page 90
Change the operating mode of your WiFi module from access point mode to client mode.

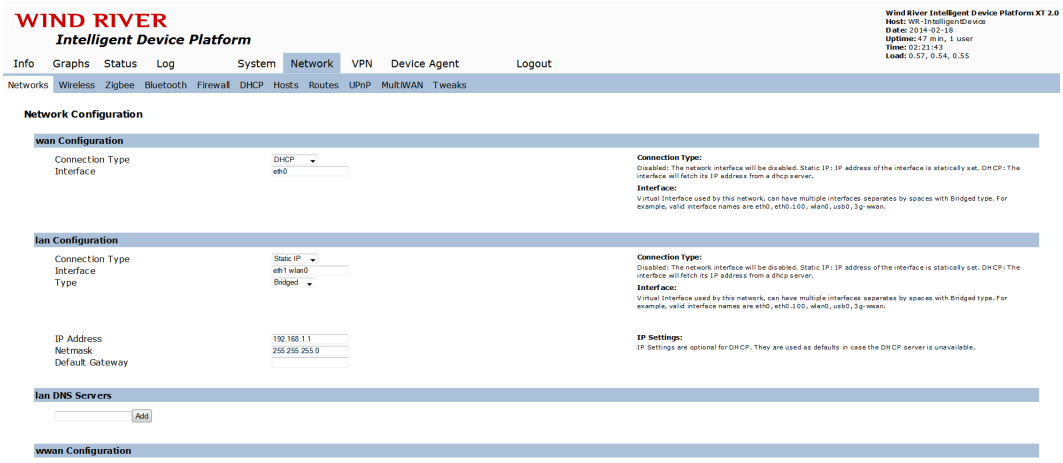
Changing WiFi Mode from Client to AP via Webif on page 92
Change the operating mode of your WiFi module from client mode to access point mode.

Starting a Zigbee Network on page 94

If you use Webif, you can use it join and leave a network instead of using the command line.

Network Tab

Use the Network tab to view and change basic networking parameters, including LAN, WAN, and WWAN for your gateway router.



Setting	Description
Network Configuration	Use this section to modify your Network settings. Some key options include: • WAN Configuration - Connection Type: Default is **DHCP**. - Interface: Default is **eth0**. • LAN Configuration: - Connection Type: Default is Static IP - Interface: Default is **wlan0** - IP Address: Default is **192.168.1.1** • WWAN Configuration: - Connection Type: Default is **WWAN** - Interface: Default is **3g-wwan** - Device: Default is **/dev/ttyACM0**. - Service Type: Select a network that matches your SIM card. Default is **UMTS**. - APN Name: Matches the access point name of the network the 3G modem is connected to.
Wireless Configuration	Use this section to modify your WLAN settings. Some key options include: • Radio: Use to turn wireless radio on or off. • Mode: Use to select the wireless mode. • Channel: Use to select a channel for the wireless radio. • ESSID: Default is **IDPDK-** *XXXX*

Setting	Description
	• IP Address: Defaults to **192.168.1.1**. • Encryption Type: Default is **WPA2 (PSK)**

Once you make changes, click **Save Changes** before navigating away from this page. See *Saving Changes in the Webif Interface* on page 88.

Related Links

About Webif on page 81
You can use the Webif interface to configure your gateway or router (your IDP XT device) in the same way you would configure your home Wi-Fi router.

Webif Interface Main Tabs on page 82
The Main tabs provide the bases for making configuration changes to your Wi-Fi gateway.

Webif Interface Default Settings on page 83
Refer to these default settings when you need to setup or modify your Wi-Fi router.

Webif Interface Prerequisites on page 84
Before starting the Webif interface workflow, you must have the correct hardware and software.

Launching and Accessing the Webif Interface on page 85
You can use the Webif interface to launch and access your network rather than the command line interface.

Saving Changes in the Webif Interface on page 88
Once you make changes to your router configuration, you must save them.

Changing WiFi Mode from AP to Client via Webif on page 90
Change the operating mode of your WiFi module from access point mode to client mode.

Changing WiFi Mode from Client to AP via Webif on page 92
Change the operating mode of your WiFi module from client mode to access point mode.

Starting a Zigbee Network on page 94
If you use Webif, you can use it join and leave a network instead of using the command line.

Changing WiFi Mode from AP to Client via Webif

Change the operating mode of your WiFi module from access point mode to client mode.

If your IDP XT target has a WiFi module, its default working mode is access point (AP) mode. In this mode, the module acts as an access point to which wireless devices can connect. In client mode, the module becomes a wireless client and, as such, connects to another access point.

Perform all steps in the following procedure in the IDP Webif interface.

To change from AP mode to client mode, do the following:

Step 1 On the **Network** tab, click **Networks**.

Step 2 Under **lan Configuration**, in the **Interface** field, remove **wlan0**.

When the WiFi module is in AP mode, it connects to a wireless LAN, **wlan0**, in bridged mode. This step removes that connection.

Step 3 Under **Add Network**, in the text field, type a name for a new network, **lan1** for example; then click **Add Network**.

Step 4 Under **lan1 Configuration**, in the **Connection Type** drop-down list, click **DHCP**.

Step 5 In the **Interface** field, type **wlan0**; then click **Save Changes**.

Step 6 On the **Network** tab, click **Wireless**.

Step 7 Under **Wireless Virtual Adaptor Configuration for Wireless Card wlan0**, in the **Mode** drop-down list, click **Client**.

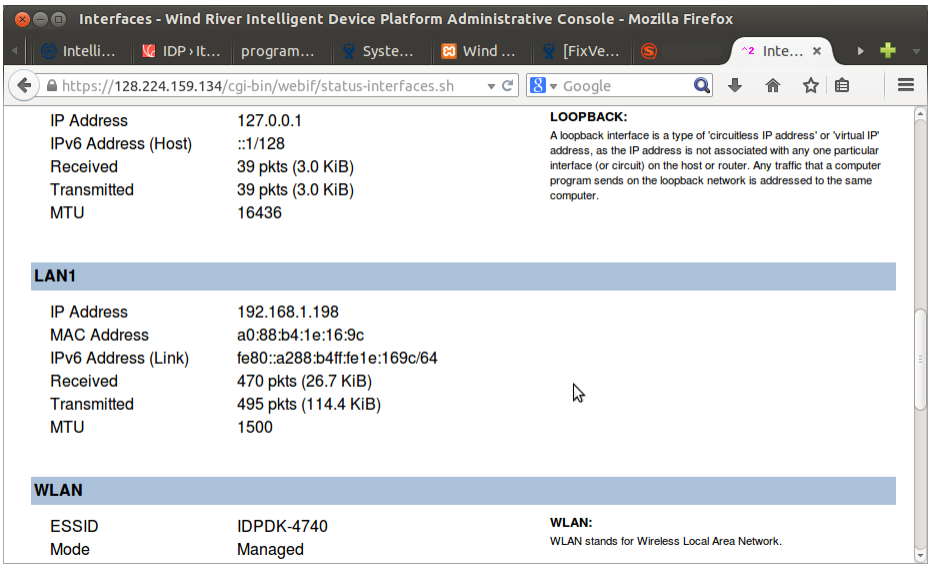
Step 8 In the **ESSID** field, type the wireless network server ESSID.

Step 9 In the **Encryption Type** drop-down list, select the encryption type; then type the associated password.

Step 10 Click **Save Changes**; then click **Apply Changes**.

Step 11 Check the **lan1** status to confirm that you have successfully set up client mode.

On the **Status** tab, under **Interfaces**, **wlan0** shows the IP address fetched from the WiFi server, for example 192.168.1.xx



Related Links

About Webif on page 81
You can use the Webif interface to configure your gateway or router (your IDP XT device) in the same way you would configure your home Wi-Fi router.

Webif Interface Main Tabs on page 82
The Main tabs provide the bases for making configuration changes to your Wi-Fi gateway.

Webif Interface Default Settings on page 83
Refer to these default settings when you need to setup or modify your Wi-Fi router.

Webif Interface Prerequisites on page 84
Before starting the Webif interface workflow, you must have the correct hardware and software.

Launching and Accessing the Webif Interface on page 85
You can use the Webif interface to launch and access your network rather than the command line interface.

Saving Changes in the Webif Interface on page 88

Once you make changes to your router configuration, you must save them.

Network Tab on page 89
Use the Network tab to view and change basic networking parameters, including LAN, WAN, and WWAN for your gateway router.

Changing WiFi Mode from Client to AP via Webif on page 92
Change the operating mode of your WiFi module from client mode to access point mode.

Starting a Zigbee Network on page 94
If you use Webif, you can use it join and leave a network instead of using the command line.

Changing WiFi Mode from Client to AP via Webif

Change the operating mode of your WiFi module from client mode to access point mode.

If your IDP XT target has a WiFi module, you can run it in client mode to make it operate as a wireless client that can associate with an access point. The default working mode of the module is access point (AP) mode. In this mode, the module acts as an access point to which wireless networks can connect.

Perform all steps in the following procedure in the IDP Webif interface.

To change from client mode to AP mode, do the following:

Step 1 On the **Network** tab, click **Wireless**.

Step 2 Under **Wireless Virtual Adaptor Configuration for Wireless Card wlan0**, in the **Mode** drop-down list, click **Access Point**.

Step 3 In the **ESSID** field, type the wireless network server ESSID.

Step 4 In the **IP Address** field, type the default WiFi module IP address, 192.168.1.1.

Step 5 In the **Netmask** field, type the subnet mask 255.255.255.0.

Step 6 In the **Encryption Type** drop-down list, select the encryption type, type the associated password; then click **Save Changes**.

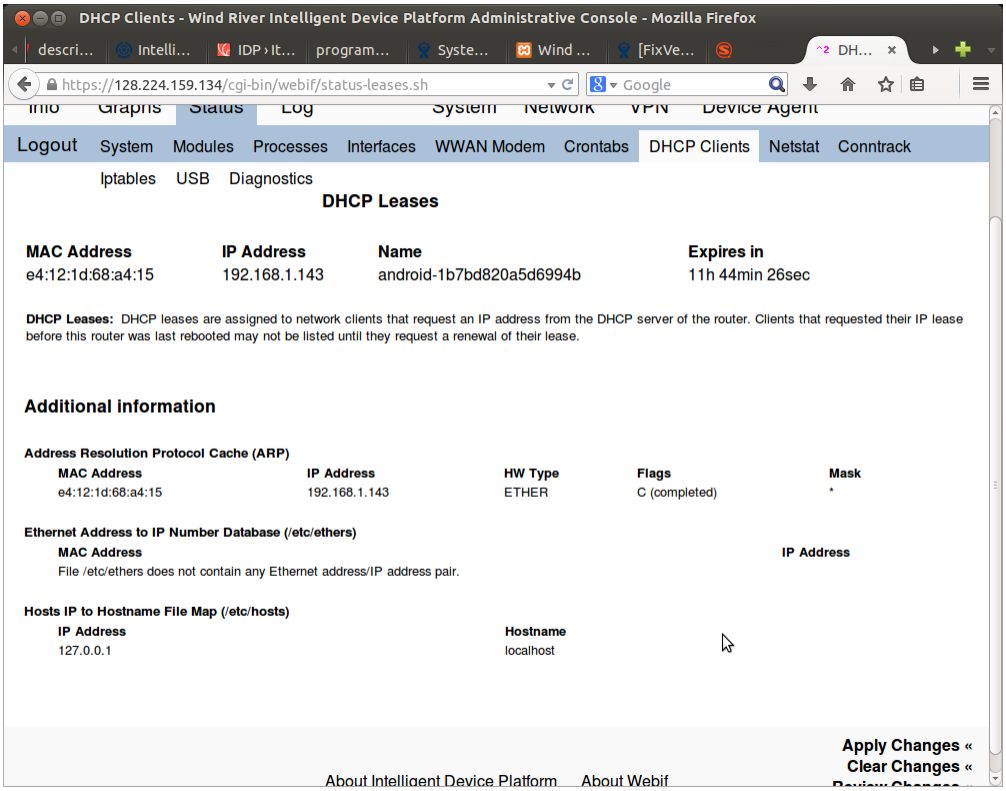
Step 7 On the **Network** tab, click **Networks**.

Step 8 Under **Remove Network**, in the **Remove Network** drop-down list, select the network associated with the current client mode, for example **lan1**.

Step 9 Under **lan Configuration**, in the **Interface** field, type the wireless LAN that you want to use to connect to AP mode, for example **wlan0**—the LAN configuration **Type**, should be **Bridged**.

Step 10 Click **Save Changes**; then click **Apply Changes**.

Step 11 After a wireless client connects to your WiFi module, now in AP mode, check the **DHCP Clients** status to confirm that you have successfully set up AP mode: on the **Status** tab, review the information shown under **DHCP Clients**.



Related Links

About Webif on page 81
You can use the Webif interface to configure your gateway or router (your IDP XT device) in the same way you would configure your home Wi-Fi router.

Webif Interface Main Tabs on page 82
The Main tabs provide the bases for making configuration changes to your Wi-Fi gateway.

Webif Interface Default Settings on page 83
Refer to these default settings when you need to setup or modify your Wi-Fi router.

Webif Interface Prerequisites on page 84
Before starting the Webif interface workflow, you must have the correct hardware and software.

Launching and Accessing the Webif Interface on page 85
You can use the Webif interface to launch and access your network rather than the command line interface.

Saving Changes in the Webif Interface on page 88
Once you make changes to your router configuration, you must save them.

Network Tab on page 89
Use the Network tab to view and change basic networking parameters, including LAN, WAN, and WWAN for your gateway router.

Changing WiFi Mode from AP to Client via Webif on page 90
Change the operating mode of your WiFi module from access point mode to client mode.

Starting a Zigbee Network on page 94

If you use Webif, you can use it join and leave a network instead of using the command line.

Starting a Zigbee Network

If you use Webif, you can use it join and leave a network instead of using the command line.

- On the Webif page, select **Network** > **Zigbee**.

Options	Description
To become the Coordinator:	Click on the **Start and Create a network** button.
To join as a Router:	Click on the **Start and Join the network** button.

- To stop the Zigbee stack and leave the network, click on the **Leave the network** button.

Related Links

About Webif on page 81
You can use the Webif interface to configure your gateway or router (your IDP XT device) in the same way you would configure your home Wi-Fi router.

Webif Interface Main Tabs on page 82
The Main tabs provide the bases for making configuration changes to your Wi-Fi gateway.

Webif Interface Default Settings on page 83
Refer to these default settings when you need to setup or modify your Wi-Fi router.

Webif Interface Prerequisites on page 84
Before starting the Webif interface workflow, you must have the correct hardware and software.

Launching and Accessing the Webif Interface on page 85
You can use the Webif interface to launch and access your network rather than the command line interface.

Saving Changes in the Webif Interface on page 88
Once you make changes to your router configuration, you must save them.

Network Tab on page 89
Use the Network tab to view and change basic networking parameters, including LAN, WAN, and WWAN for your gateway router.

Changing WiFi Mode from AP to Client via Webif on page 90
Change the operating mode of your WiFi module from access point mode to client mode.

Changing WiFi Mode from Client to AP via Webif on page 92
Change the operating mode of your WiFi module from client mode to access point mode.

13

Secure Repository

RPM Repository Server

The RPM repository server maintains the customized packages for your SRM solution as part of IDP XT integrity management.

Wind River recommends that you set up the RPM server on a separate host.

For information on using the WebIf interface to manage the repository, see *Webif Interface Main Tabs* on page 82.

Adding a Local Repository

Software packages will be downloaded from the repository server to the local repository.

You must build **spm-repo** on the host and copy it to the device on which you will be adding the repository.

Step 1 Build a platform project and boot your board in the standard way.

For more information see:

- *Building Platform Projects for Advantech UTX-3115 Boards* on page 126
- *Building Platform Projects for Quark Boards* on page 122

Step 2 Add the repository to the device.

```
# spm_repo --addlocal <local_rpm_directory>
```

The parameter *local_rpm_directory* is the local directory where the rpm packages are stored. For example:

```
# spm_repo --addlocal /opt/rpm_repo
```

The following messages may occur:

Error message	Description
`Directory <target_local_directory> is not exist`	The RPM repository does not exist on the device.
`Adding local repository <localrpmrepo> error`	The RPM repository exists but the command failed to add it.
`Adding local repository <localrpmrepo> successfully`	The RPM repository has been successfully added.

Step 3 Confirm the contents of the repository.

```
# ls -l /opt/rpm_repo/
total 3200
-rw-r--r-- 1 root root 2513039 Nov  7 16:16 cups-1.4.1-1_WR4.3.0.0.1.atom-3d6a02d.rpm
-rw-r--r-- 1 root root  536333 Nov  7 16:16 cups-
libs-1.4.1-1_WR4.3.0.0.1.atom-3d6a02d.rpm
-rw-r--r-- 1 root root   15236 Nov  7 16:16 cups-
lpd-1.4.1-1_WR4.3.0.0.1.atom-3d6a02d.rpm
-rw-r--r-- 1 root root  180338 Nov  7 16:16 dbus-glib-0.82-2_WR4.3.0.0.atom-75365e0.rpm
-rw-r--r-- 1 root root    5967 Nov  7 16:16 memtest-1.0-1_WR4.3.0.0.atom-f92430e.rpm
drwxr-xr-x 2 root root    4096 Apr 26 17:50 repodata
```

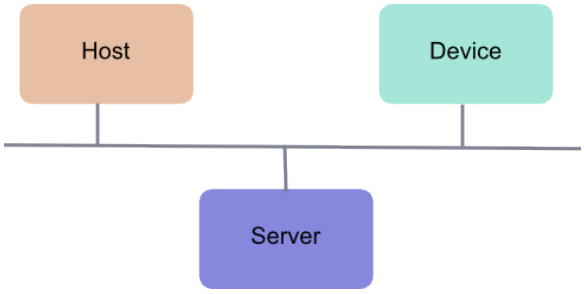
Remote Repositories

The remote repository is a server where software packages are stored for download to the device.

In order to download software packages securely to a deployed device, you must install a Web server and create a remote repository to hold the packages. You will create and package the software as RPMs on a development host and transfer them to the server. You can then connect the device to the repository and download the software using the Web server.

The example assumes the following:

- The software is packaged using RPM. RPM is the default Wind River Linux build system packaging method.
- You will transfer the packages to the server using your preferred method. For example, you might use FTP or a USB drive.
- You will provide the packages to the device over the internet using the Apache Web server. Apache is the default Web server for Red Hat Linux.
- The device, server, and development host are on the same network so both host and device have access to the server.



Host

your development system.

Device

the system receiving the software updates.

Server

the system providing those updates to any requesting device.

Related Links

Installing Server Software on page 97
You must install Web server software and tools for managing the repository on your server.

Setting Up the Web Server on page 98
You must set up directories on the server, place the RPMs in the directories, identify the directories to the Web server, and set up the RPM infrastructure for the directories.

Starting the Web Server on page 99
Start Apache on the Web server machine; verify that it is running and that you can see the repository in your host browser.

https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Deployment_Guide/sec-Yum_Repository.html

http://www.webmo.net/support/yum_repository.html

http://www.techrepublic.com/blog/linux-and-open-source/create-your-own-yum-repository/

Installing Server Software

You must install Web server software and tools for managing the repository on your server.

These examples use a Fedora host.

Step 1 Install the Apache Web server.

On the server, install the apache2 Web server if it is not already installed.

```
$ sudo yum install httpd
```

Step 2 Install the **createrepo** tool.

On the server, install the **createrepo** tool, if it is not already installed.

```
$ sudo yum install createrepo
```

Related Links

Remote Repositories on page 96

The remote repository is a server where software packages are stored for download to the device.

Setting Up the Web Server on page 98
You must set up directories on the server, place the RPMs in the directories, identify the directories to the Web server, and set up the RPM infrastructure for the directories.

Setting Up the Web Server

You must set up directories on the server, place the RPMs in the directories, identify the directories to the Web server, and set up the RPM infrastructure for the directories.

All these steps are performed on the server.

Step 1 Find the value for the **DocumentRoot** system variable.

DocumentRoot is the directory where Apache looks for files.

```
$ grep "^DocumentRoot" /etc/httpd/conf/httpd.conf
DocumentRoot "/var/www/html"
```

Record this value for later use.

Step 2 Create a directory for your RPM in **DocumentRoot**.

```
$ sudo mkdir /var/www/html/rpm
```

Step 3 Place the RPMs in a temporary location on the server.

Use a method such as FTP or a USB stick. The example assumes you place the files in ***projDir*/export/RPMS**.

Step 4 Copy the RPMs from the temporary location to your **rpm** directory.

```
$ cd projectdir/export/RPMS
$ sudo cp -Rv * /var/www/html/rpm
```

Step 5 Configure Apache to use this directory.

Modify the **/etc/httpd/conf/httpd.conf** file by adding the following lines:

```
<Directory /var/www/html/rpm>
    Options +Indexes
</Directory>
```

Step 6 (Optional) Add your server name to the Apache configuration file.

Modify the **/etc/httpd/conf/httpd.conf** file by uncommenting the line containing **ServerName** and adding your server name or IP address.

Original line:

```
#ServerName www.example.com:80
```

Example modified line:

```
ServerName 192.168.2.250
```

Step 7 Create the RPM infrastructure for the directories.

Use **createrepo** to create the necessary files used by the **smart** tool to transfer RPMs from the repository to the device. Run the command in each subdirectory of the **rpm** directory.

```
$ cd /var/www/html/rpm/all
$ sudo createrepo .
$ cd /var/www/html/rpm/i586
$ sudo createrepo .
$ cd /var/www/html/rpm/intel-quark
$ sudo createrepo .
```

Related Links

Remote Repositories on page 96
The remote repository is a server where software packages are stored for download to the device.

Installing Server Software on page 97
You must install Web server software and tools for managing the repository on your server.

Starting the Web Server on page 99
Start Apache on the Web server machine; verify that it is running and that you can see the repository in your host browser.

Starting the Web Server

Start Apache on the Web server machine; verify that it is running and that you can see the repository in your host browser.

Step 1 Start the Web server.

```
$ sudo apachectl restart
httpd not running, trying to start
```

Step 2 Verify that the Web server is running.

The **httpd** processes are the apache2 processes. Your output will be similar to the example.

```
$ ps ax | grep httpd
 4241 ?        Ss     0:00 /usr/sbin/httpd -k restart
 4242 ?        S      0:00 /usr/sbin/httpd -k restart
 4243 ?        S      0:00 /usr/sbin/httpd -k restart
 4244 ?        S      0:00 /usr/sbin/httpd -k restart
 . . .
 4257 pts/1    S+     0:00 grep --color=auto httpd
```

Step 3 Confirm that you can see the RPMs from your Web browser on your development host or management server.

```
$ firefox <MyServer>/rpm
```

where *MyServer* is either the name or the IP address of your server.

You should get a directory listing of the **/var/www/html/rpm** directory with the label **/rpm**.

Step 4 Exit the browser.

Related Links

Remote Repositories on page 96
The remote repository is a server where software packages are stored for download to the device.

Setting Up the Web Server on page 98

You must set up directories on the server, place the RPMs in the directories, identify the directories to the Web server, and set up the RPM infrastructure for the directories.

Managing Repositories

Adding a Remote Repository

Identify the repository to the device. In most installations, the repository is stored on a remote server that can download packages to devices.

Step 1 Sign the remote repository.

For more information, see:

Signing a Single RPM on page 54
Signing Multiple RPMs on page 55

Step 2 Add the remote repository to the device.

```
# spm_repo --addremote <remote_repository_name> <remote_repository_URL>
```

remote_repository_name

The name to use for the remote repository.

remote_repository_URL

The URL of the remote repository.

For example, to add the signed repository **/var/www/html/rpm/i586**, use the following command:

```
$ spm_repo --addremote myrepo http://192.0.2.0/rpm_repo/rpm/i586
```

The following messages may occur:

Error message	Description
`Updating remote repository <remote_repository_name> error when adding repository`	The RPM repository does not exist on the machine.
`Adding remote repository <remote_repository_name> error`	The RPM repository exists but the command failed to add it.
`Adding remote repository <remote_repository_name> successfully`	The RPM repository has been successfully added

Related Links

Removing a Repository on page 101
Remove a repository from the list of repositories available to the device.

Listing Repositories on page 101

You can list all repositories available to the device using **spm_repo --listrepo**.

Removing a Repository

Remove a repository from the list of repositories available to the device.

Adding a remote repository maps the repository name to the URL. For this reason, the commands for operating on existing repositories are the same for local and remote repositories.

- Use the **spm_repo** command to delete a repository from the device.

```
# spm_repo --deleterepo rpm_repo myrepo
```

The following messages may occur:

Error message	Description
`Removing repository <repository_name> error`	The command failed to remove the repository.
`Removing repository <repository_name> successfully`	The RPM repository was successfully removed.

Related Links

Adding a Remote Repository on page 100
Identify the repository to the device. In most installations, the repository is stored on a remote server that can download packages to devices.

Listing Repositories on page 101
You can list all repositories available to the device using **spm_repo --listrepo**.

Listing Repositories

You can list all repositories available to the device using **spm_repo --listrepo**.

Adding a remote repository maps the repository name to the URL. For this reason, the commands for operating on existing repositories are the same for local and remote repositories.

- Use the **spm_repo** command to list all repositories on the device.

```
# spm_repo --listrepo
Below is all repositories on the target:
localrpmrepo
rpmsys
myrepo
```

Related Links

Adding a Remote Repository on page 100
Identify the repository to the device. In most installations, the repository is stored on a remote server that can download packages to devices.

Removing a Repository on page 101

Remove a repository from the list of repositories available to the device.

Managing RPM Packages

Adding an RPM Package to the Device

Once you have access to a repository, you can download and install RPM packages to the device.

Adding a remote repository maps the repository name to the URL. For this reason, the commands for operating on existing repositories are the same for local and remote repositories.

- Use the **spm_repo** command to install an RPM package on the device.

  In this examples, the package is **memtest-1.0-1_WR4.3.0.0.atom-f92430e.rpm**.

```
# spm_repo --installrpm memtest-1.0-1_WR4.3.0.0.atom-f92430e.rpm
```

Related Links

Listing the RPM Packages Installed on the Device on page 102
Use the **spm_repo --listrpm** command to list RPM packages installed on the device.

Removing an RPM Package from the Device on page 102
Use the **spm_repo --deleterpm** command to remove an RPM package from the device.

Listing the RPM Packages Installed on the Device

Use the **spm_repo --listrpm** command to list RPM packages installed on the device.

Adding a remote repository maps the repository name to the URL. For this reason, the commands for operating on existing repositories are the same for local and remote repositories.

- Use the **spm_repo** command to list all RPM packages installed on the device.

```
# spm_repo --listrpm
```

Related Links

Adding an RPM Package to the Device on page 102
Once you have access to a repository, you can download and install RPM packages to the device.

Removing an RPM Package from the Device on page 102
Use the **spm_repo --deleterpm** command to remove an RPM package from the device.

Removing an RPM Package from the Device

Use the **spm_repo --deleterpm** command to remove an RPM package from the device.

Adding a remote repository maps the repository name to the URL. For this reason, the commands for operating on existing repositories are the same for local and remote repositories.

- Use the **spm_repo** command to remove an installed RPM package from the the device.

  In this example, the package is **memtest-1.0-1_WR4.3.0.0.atom-f92430e.rpm**.

```
# spm_repo --deleterpm memtest-1.0-1_WR4.3.0.0.atom-f92430e.rpm
```

Related Links

Adding an RPM Package to the Device on page 102
Once you have access to a repository, you can download and install RPM packages to the device.

Listing the RPM Packages Installed on the Device on page 102
Use the **spm_repo --listrpm** command to list RPM packages installed on the device.

14

Tamper-Proof File System

Application Integrity Measurement

The Intelligent Device Platform Integrity Measurement Architecture (IMA), a part of Secure Remote Management (SRM), tests that the application has not been tampered with before allowing the device to load and run it.

IMA ensures application integrity through a tamper-proof file system. The tamper-proof file system prevents end users from making modifications to the device software and from executing unauthorized applications on the device. The device software can only be updated using the authorized approaches provided by SRM.

NOTE: The tamper-proof file system does not support NFS; you cannot install a signed RPM package on a remote NFS server and run it from the device.

If you want to enable the tamper-proof file system using **--enable-addons=wr-idp**, add the IMA layer with **–with-layer=wr-ima-appraise** and boot the device. This includes the tamper-proof file system capability.

To activate the tamper-proof file system, use the **tar** file called ***-dist-srm.tar.bz2**. For example:

***projDir*/export/images/wrlinux-image-glibc-idp-intel-quark-dist-srm.tar.bz2**

The tamper-proof file system capability is available for the following BSPs:

intel-quark
intel-atom-baytrail

Related Links

Using the Tamper-Proof File System on page 106

The tamper-proof file system is part of the Integrity Measurement Architecture (IMA); including this capability on an embedded device provides device owners with strict control over the software deployed on the device.

Using the Tamper-Proof File System

The tamper-proof file system is part of the Integrity Measurement Architecture (IMA); including this capability on an embedded device provides device owners with strict control over the software deployed on the device.

The purpose of application integrity measurement is to assure that the run results of text-based scripts can be trusted when the system invokes them with a *controlled approach*.

NOTE: For compiled executable files and text-based plain scripts, the tamper-proof capability always prevents them from running if they cannot provide a verified signature. Text-based plain scripts are bash, perl, or python scripts that are invoked from an absolute or relative path.

Examples of controlled invocation:

```
$ ./certain-script.sh
$ /root/certain-perl-script.pl
```

However, when these scripts are executed directly by the interpreter, the tamper-proof capability does not prevent them from running; running from the interpreter is not a *controlled approach*.

Examples of uncontrolled invocation:

```
$ bash ./certain-script.sh
$ perl /root/certain-perl-script.pl
```

The tamper-proof file system is not enabled by default when you enable the **wr-idp** addon. If you want to enable the tamper-proof file system, use **--enable-addons=wr-idp, --with-layer=wr-ima-appr** and boot the device. This includes the tamper-proof file system capability. To activate the tamper-proof file system, use the **tar** file called ***-dist-srm.tar.bz2**, for example:

***projDir*/export/images/wrlinux-image-glibc-idp-intel-quark-dist-srm.tar.bz2**

Step 1 Build a platform project and boot your board in the standard way.

For more information see:

- *Building Platform Projects for Quark Boards* on page 122
- *Building Platform Projects for Advantech UTX-3115 Boards* on page 126

The following configure command uses the intel-quark BSP as an example:

```
$ $WIND_LINUX_CONFIGURE --enable-rootfs=glibc-idp --enable-addons=wr-idp \
--enable-kernel=standard --enable-board=intel-quark --with-layer=wr-ima-appraise
```

Building this configuration creates two tar files:

wrlinux-image-glibc-idp-intel-quark.tar.bz2

If it is configured using **--with-layer=wr-ima-appraise**, this image contains all SRM capabilities that enable the tamper-proof file system — they are not enabled by default.

wrlinux-image-glibc-idp-intel-quark-dist-srm.tar.bz2

This image contains all the SRM capabilities. They are enabled by default with default keys. Use this image to demonstrate SRM capabilities. You can find the default keys at:

***projDir*/layers/wr-idp/wr-srm/files/keys/owner-cert.pem**
***projDir*/layers/wr-idp/wr-srm/files/keys/owner-private.pem**
***projDir*/layers/wr-idp/wr-srm/files/keys/vendor-cert.pem**
***projDir*/layers/wr-idp/wr-srm/files/keys/vendor-private.pem**

NOTE: Use the SST commands to sign either set of images with custom keys. For more information, see *Key Management for Vendors* on page 43.

Step 2 Verify that an unauthorized application cannot run on the system.

```
# ls
examples
# cp `which ls` ./ls-copied
# ls
examples  ls-copied
# ./ls-copied
-sh: ./ls-copied: Permission denied
# echo $?
126
```

The exit status value **126** indicates that the command did not run successfully. In this case, the RSA signature for **ls-copied** was not found on the system.

Step 3 Modify a script or executable and verify that it will not run.

This example makes arbitrary modifications to **imtools** and then tries to run it.

```
# imtools -h
Usage:imtool      --verifycert  <CA Certificate>
                  --listcert
                  --removecert  <CA Certificate>
                  --verifyrpm   <RPM Package>
# vi /usr/bin/imtools
<Make some modifications to the script, for example, by changing some help text, and
save the file>

# imtools -h
-sh: /usr/bin/imtools: /bin/sh: bad interpreter: Permission denied
```

The script cannot run because the RSA signature for **im-tools** does not match the one stored on the system.

Step 4 Verify that an authorized application can run successfully.

The **ls** command has an RSA signature and has not been modified.

```
# ls
examples  ls-copied
# echo $?
0
```

The exit status value **0** indicates that the command ran successfully. In this case, the RSA signature for **ls** matched the one stored on the system.

Step 5 (Optional) If you do not need the tamper-proof file system, when you build the project do not add the **wr-ima-appraise** layer to your **$WIND_LINUX_CONFIGURE** command.

```
$ $WIND_LINUX_CONFIGURE --enable-rootfs=glibc-idp --enable-addons=wr-idp \
--enable-kernel=standard --enable-board=intel-quark
```

NOTE: The **wr-ima-appraise** layer depends on **wr-srm**; do not use **--with-layer=wr-ima-appraise** in your **$WIND_LINUX_CONFIGURE** command when you use **--without-layer=wr-srm**.

Related Links

Application Integrity Measurement on page 105
The Intelligent Device Platform Integrity Measurement Architecture (IMA), a part of Secure Remote Management (SRM), tests that the application has not been tampered with before allowing the device to load and run it.

15

Sign and Update RPM Packages, Kernel Images, and GRUB Bootloader Images

Generating and Installing a Signed RPM package

Generate a signed RPM package for IDP XT and install it on the target.

The Secure Remote Management (SRM) feature is enabled in IDP XT, which prevents the installation of unauthorized RPM packages—you can successfully install only properly signed RPM packages.

To generate a signed RPM package and install it on the target, do the following:

Step 1 Configure your IDP XT project to build an RPM package.

Assuming you are using an intel-quark BSP, the **configure** command is:

```
${product_dir}/wrlinux-x/wrlinux/configure --enable-board=intel-quark --enable-
addons=wr-idp\
--enable-kernel=standard --enable-rootfs=glibc-idp
```

For more information about how to configure IDP XT, see *Preparing to Build and Boot IDP* on page 121.

Step 2 Build your RPM package.

To build an RPM package called **example-1.0**, for example, run the following commands:

```
$ cd ${project_dir}
```

```
$ make -C build example
```

This results in deploying the RPM package in **${project_dir}/build/${package}/deploy-rpms/{arch}/**. For example:

```
$ ls build/example-1.0-r2/deploy-rpms/i586/
example-1.0-r2.i586.rpm
```

Step 3 Sign the RPM package with SST.

```
$ ./SST sign-rpm --verbose=no --mode=rpm --priv-key=./vendor-private.pem ./
example-1.0-r2.i586.rpm
```

This step updates the RPM package with the properly signed package, using the vendor's private key, **vendor-private.pem**, to sign the package.

For more information about signing RPMs, see *Signing a Single RPM* on page 54 and *Signing Multiple RPMs* on page 55.

Step 4 Copy the vendor's certificate to the target.

Step 5 Install the vendor's certificate on the target.

```
# imtools --verifycert vendor-cert.pem
Certificate vendor-cert.pem is verified successfully
Certificate vendor-cert.pem is installed successfully
```

The vendor's certificate is used to verify the signed RPM package—it must be installed on the target. The certificate must also have been delegated by the owner, otherwise it cannot be successfully installed.

Step 6 Copy the RPM package to the target.

Step 7 Install the RPM package on the target.

```
# rpm -ivh example-1.0-r2.i586.rpm
Find right certificate: vendor-cert.pem
Certificate vendor-cert.pem is verified successfully
RPM package example-1.0-r2.i586.rpm is verified successfully
Preparing...                ########################################### [100%]
1:example                  ########################################### [100%]
```

This step results in the successful installation of the RPM package **example-1.0-r2.i586.rpm**.

If you try to install an unsigned RPM package, you will see the following error message:

```
# rpm -ivh example-1.0-r2.i586.rpm
RPM example-1.0-r2.i586.rpm is not signed with extend openssl signature
```

Related Links

Generating and Updating a Signed Kernel Image on page 111
Sign and update a kernel image.

Generating and Updating a Signed Bootloader Image on page 113
A signed bootloader supports verified and secure boot, essential processes in the Secure Remote Management (SRM) feature in IDP XT.

Generating and Updating a Signed Kernel Image

Sign and update a kernel image.

The Secure Remote Management (SRM) feature is enabled in IDP XT, which prevents the loading of a kernel image that is not properly signed.

To sign and update a kernel image, do the following:

Step 1 Configure your IDP XT project to build the kernel.

Assuming you are using an Intel Quark BSP, the **configure** command is:

```
${Product_dir}/wrlinux-x/wrlinux/configure --enable-board=intel-quark --enable-
addons=wr-idp\
--enable-kernel=standard --enable-rootfs=glibc-idp
```

For more information about how to configure IDP XT, see *Preparing to Build and Boot IDP* on page 121.

Step 2 Build the kernel image.

```
$ cd ${project_dir}
$ make -C build linux-windrvier
```

This puts the kernel image in **${project_dir}/build/linux-windriver-3.4-r0/image/boot**

Step 3 Sign the kernel image with SST.

If your target is Cross Hill, for example, run the following command:

```
$ ./SST sign-kernel --machine=intel_quark --verbose=no \
--priv-key=./vendor-private.pem \
--vendor-cert=./vendor-cert.pem \
--romkey-dir=./quark-romkey ./bzImage
```

Where **vendor-private.pem** is the vendor's private key, **vendor-cert.pem** is the vendor's certificate, and **quark-romkey** is the directory that includes the ROM key for Cross Hill's secure ROM.

The command above updates the kernel image file **bzImage** and generates a new signature file **bzImage.csbh**.

For more information, see *Signing Kernels* on page 51.

Step 4 Update the kernel image on your boot device.

a) If your boot method is UEFI, format the boot device into two partitions: VFAT and EXT3.

For information about how to deploy a boot device, see *Preparing to Build and Boot IDP* on page 121.

To update the kernel image if your boot device is U-KEY, for example, do the following:

1. Connect your USB storage device to the host machine.
2. Mount the VFAT partition.

   Assuming your USB storage device is **/dev/sdc1** on your host:

```
$ mount /dev/sdc1 ./vfat
```

3. Replace the **bzImage** file in the VFAT partition with the updated **bzImage** file signed by SST.

```
$ cp bzImage  ./vfat
```

NOTE: If you are using an Intel Quark board with a secure ROM, replace the files in the VFAT partition with both the **bzImage** and **bzImage.csbh** files, otherwise you will not be able to boot your system.

```
$ cp bzImage  ./vfat
$ cp bzImage.csbh  ./vfat
```

4. Unmount your boot device.

```
$ umount /dev/sdc1
```

b) If you are using the legacy boot method, format your boot device as an EXT3 file system.

For information about how to deploy a boot device, see *Preparing to Build and Boot IDP* on page 121.

To update the kernel image if your boot device is U-KEY, for example, do the following:

1. Connect your USB storage device to the host machine.
2. Mount the U-KEY, assuming your U-KEY is **/dev/sdc1** on your host machine.

```
$ mount /dev/sdc1 ./ext3
```

3. Replace the **bzImage** file in **./ext3/boot** with **bzImage**, updated using SST.
4. Umount your boot device.

```
$ umount /dev/sdc1
```

Related Links

Generating and Installing a Signed RPM package on page 109
Generate a signed RPM package for IDP XT and install it on the target.

Generating and Updating a Signed Bootloader Image on page 113

A signed bootloader supports verified and secure boot, essential processes in the Secure Remote Management (SRM) feature in IDP XT.

Generating and Updating a Signed Bootloader Image

A signed bootloader supports verified and secure boot, essential processes in the Secure Remote Management (SRM) feature in IDP XT.

To generate and update a signed bootloader image, do the following:

Step 1 Configure your IDP XT project to build a bootloader.

Assuming you are using an intel-atom-baytrail BSP, the **configure** command is:

```
${Product_dir}/wrlinux-x/wrlinux/configure --enable-board=intel-atom-baytrail --enable-
addons=wr-idp\
--enable-kernel=standard --enable-rootfs=glibc-idp
```

For more information about how to configure IDP XT, see *Preparing to Build and Boot IDP* on page 121.

Step 2 Build the bootloader.

- If your system boot method is UEFI, run the following command to build **grub-efi**:

  ```
  $ cd ${project_dir}
  $ make -C build grub-efi
  ```

  This puts the **grub-efi** image and GRUB configuration file in **${Project_dir}/build/ ${grub-efi package}/image/boot/grub**.
- If you booth your system using the legacy boot method, run the following command to build GRUB or **grub-ima**:

  ```
  $ cd ${project_dir}
  $ make -C build grub
  ```

  or

  ```
  $ make -C build grub-ima
  ```

  This puts the bootloader and GRUB configuration file in **${Project_dir}/build/{grub package}/ image/boot/grub**

Step 3 Sign the bootloader with SST.

- If you are using a UTX-3115 board as your target, sign the bootloader with the following command:

  ```
  $ ./SST sign-bootloader --machine=intel_atom_baytrail --verbose=no \
  --owner-cert=./owner-cert.pem \
  --vendor-cert=./vendor-cert.pem \
  --priv-key=./vendor-private.pem\
  ```

```
./grub.efi
```

Where **owner-cert.pem** is the owner's certificate, **vendor-cert.pem** is the vendor's certificate, delegated by the owner, and **vendor-private.pem** is the vendor's private key.

This command updates the **grub.efi** file.

- If your target is an Intel Quark board, Cross Hill for example, run the following command to sign the **grub.conf** file using the SST tool:

```
$ ./SST sign-bootloader --machine=intel_quark --verbose=no \
--owner-cert=./ownerE-cert.pem \
--romkey-dir=./quark-romkey \
./grub.conf
```

This command generates a new **grub.conf.csbh** file for the secure ROM.

- If you are using the legacy boot method, sign the the bootloader file, **stage2**, with the following command:

```
$ ./SST sign-bootloader --machine=intel_atom --verbose=no \
--owner-cert=./owner-cert.pem \
./stage2
```

This command updates the bootloader image, **stage2**, to a signed version of the file.

For more information about signing bootloader images, see *Signing Boot Loaders* on page 50.

Step 4 Update the bootloader image on your boot device.

If your boot method is UEFI, format the boot device into two partitions: VFAT and EXT3.

For information about how to deploy a boot device, see *Preparing to Build and Boot IDP* on page 121.

To update the bootloader image if your device is U-KEY, do the following:

a) If you are using an Intel Quark board with secure ROM, do the following:

1. Run **make** to build an SPI flash image:

```
$ make -C build spi-layout-tools-native
```

For Intel Quark boards, you should burn the bootloader image to the SPI flas as part of the flash image.

Running **make** as shown above puts the flash image in **${project_dir}/build-tools/spi-layout-tools-native-1.0.1-r1/deploy-spi-layout-tools-native/**. This location contains both the bootloader image and the associated **csbh** file.

2. Update the SPI flash image.

   You can update the SPI flash image using the SF-100 programmer or using a capsule update binary file. For more information, see *Updating Flash Firmware for Quark Boards* on page 132.
3. Connect your U-KEY to the host machine.
4. Mount the VFAT partition.

The following command assumes that the VFAT partition on your USB storage device is **/dev/sdc1** on your host machine.

```
$ mount /dev/sdc1 ./vfat
```

5. Replace the **grub.conf** configuration file and the associated **csbh** file in **./vfat/boot/** with the new bootloader configuration and **cdbh** filesgenerated by SST.

```
$ cp grub.conf  ./vfat/boot/grub/grub.conf
$ cp grub.conf.csbh  ./vfat/boot/grub/grub.conf.csbh
```

6. Unmout the boot device.

```
$ umount /dev/sdc1
```

b) If you are using an Intel Bay Trail board, do the following:

1. Connect your boot device to the host machine.
2. Mount the VFAT partition.

   The following command assumes that the VFAT partition on your USB storage device is **/dev/sdc1** on your host machine.

```
$ mount /dev/sdc1 ./vfat
```

3. Replace the **BOOTIA32.efi** and **BOOTIA32.conf** files in **./vfat/EFI/BOOT** with the updated **grub.efi** and **grub.conf**:

```
$ cp grub.efi  ./vfat/EFI/BOOT/BOOTIA32.efi
$ cp grub.conf ./vfat/EFI/BOOT/BOOTIA32.conf
```

4. Unmount your boot device.

```
$ umount /dev/sdc1
```

Step 5 If you are using the legacy boot method, format your boot device with the EXT3 file system.

For information about how to deploy a boot device, see *Preparing to Build and Boot IDP* on page 121.

To update the kernel image, if your boot device is U-KEY for example, do the following:

a) Connect your boot device to the host machine.
b) Mount your device on the host system.

The following command assumes that your USB storage device is **/dev/sdc1** on your host machine.

```
$ mount /dev/sdc1 ext3
```

c) Replace the GRUB file **stage2** in **ext3/boot/grub** with the updated version.

```
$ cp stage2 ext3/boot/grub
```

d) Update **ext3/boot/grub/menu.lst**.

e) Install GRUB.

```
$cd ${project_dir}
$ sudo ./grub-0.97  << EOF
device (hd0) /dev/sdc
root (hd0, 0)
setup --stage2=./ext3/boot/grub/stage2 (hd0)
quit
EOF
```

Related Links

Generating and Installing a Signed RPM package on page 109
Generate a signed RPM package for IDP XT and install it on the target.

Generating and Updating a Signed Kernel Image on page 111
Sign and update a kernel image.

PART IV

Device Development Vendor Tasks

16

Introduction to Device Development Tasks

Device developers create hardware, firmware, and system software for devices.

The Wind River Intelligent Device Platform XT (IDP XT) provides enhancements to Wind River Linux 5 to support security and remote management for cloud-enabled devices. Device developers configure Wind River Linux and IDP XT features including security, build kernel images and root file systems, load and test the devices, and deliver the devices and signed software packages to the owner.

17

Building and Booting

Preparing to Build and Boot IDP

Before building and booting an IDP XT image on your target, you must install Wind River Linux and IDP XT, obtain a supported board and peripherals, and boot the board.

Step 1 Obtain the required hardware and software.

- A Wind River IDP XT 2.0 installation with Wind River Linux 5.0.1 on a supported host.
- Any IDP XT 2.0 supported board:
  - Cross Hill
  - Clanton Hill
  - Galileo
  - Advantech UTX-3115
- Ability to create a flash image (Quark boards) or a boot medium with 8 GB capacity (Baytrail boards).

- A display device compatible with the display port on the target. For Quark boards, see your BSP documentation for information on the serial console 3.5MM audio to DB9 cable.
- A connection between the development host and the board.

Step 2 Install Wind River Linux 5.0.1 on your development host and update it with the latest RCPL version.

Step 3 Install Wind River Intelligent Device Platform XT 2.0 on your development host.

Step 4 Connect the board to the host.

About the wrenv.sh Script

The **wrenv.sh** script sets all the WRL related environment variables including the path to the configure command for platform projects.

The command to run the script is:

```
$ installDir/wrenv.sh -p wrlinux-5
```

If you choose not to run **wrenv.sh**, you must explicitly specify the full path to the configure command instead of just specifying **$WIND_LINUX_CONFIGURE** in your configure line. The full path is similar to ***installDir*/wrlinux-5/wrlinux/configure**.

All the examples in this guide assume that you have executed **wrenv.sh**.

About the deploy.sh Script

NOTE: The **deploy.sh** script is intended for use only with USB storage devices. The script does not support SD card deployment.

For a detailed list of options for **deploy.sh**, execute the script with the **-h** option:

```
$ ./deploy.sh -h
```

For detailed information on using the **deploy.sh** script, see the **README** file located at:

***projDir*/layers/wr-idp/wr-idp-devkit/recipes-devtools/deploy-tool/files**.

Building Platform Projects for Quark Boards

Configure and build a platform project configured for a Quark board.

IDP XT supports the following Intel Quark boards:

- Clanton Hill
- Cross Hill
- Galileo

There are several differences between Quark boards and Intel boards supported for previous versions of IDP XT:

- Quark boards require that the system files in Flash on the device match the system files on the boot media.
- Quark boards do not have video output capability.
- Quark boards use a special 3.5MM audio to DB9 serial cable.
- Quark boards can boot from either a USB Flash drive or from an MMC card.

For more information, see your BSP documentation.

NOTE: Wind River recommends that you exclude the **grsecurity** feature from your project during development. **grsecurity** is automatically included; to exclude **grsecurity**, add the following option to your configure line:

--with-template=non_grsec

Intel Galileo boards do not support security features. When you configure a project for a Galileo board, add the following option:

--without-layer=wr-srm

Step 1 Set the Wind River Linux environment variables on your host machine.

```
$ <installDir>/wrenv.sh -p wrlinux-5
```

Step 2 Create a platform project directory, *projDir*.

```
$ mkdir <projDir>
$ cd <projDir>
```

Step 3 Configure the platform project.

The following configure command uses **intel-quark** as the board type:

```
$ $WIND_LINUX_CONFIGURE --enable-board=intel-quark --enable-addons=wr-idp \
--enable-kernel=standard --enable-rootfs=glibc-idp --enable-bootimage=ext3,hdd \
--enable-parallel-pkgbuilds=4 --enable-jobs=4
```

NOTE: $WIND_LINUX_CONFIGURE is an environment variable set by **wrenv.sh** for the location of the **configure** command.

NOTE: The **openjdk-bin** layer is not installed by default. If you want to add the capabilities of Wind River OpenJDK to your build, add the **feature/openjdk-bin** option to your **configure** command. For example:

```
$ $WIND_LINUX_CONFIGURE --enable-board=intel-quark --enable-addons=wr-idp \
--enable-kernel=standard --enable-rootfs=glibc-idp \
--with-template=feature/openjdk-bin --enable-bootimage=ext3,hdd \
--enable-parallel-pkgbuilds=4 --enable-jobs=4
```

Step 4 Build the project.

```
$ make fs
```

Building the project generates the following items in the ***projDir*/export/images** folder:

bzImage-initramfs-intel-quark.bin

The kernel image with **initramfs** bundled inside it. Use this image if you are using the IDP XT SRM capabilities. (Wind River recommends this image.)

bzImage-intel-quark.bin

The kernel image without **initramfs**. SRM capabilities will not work if you use this kernel image.

wrlinux-image-glibc-idp-intel-quark-dist-srm.tar.bz2

A tar file containing the root file system image with the SRM capabilities enabled by default. The image is signed with default keys, which cannot be used for production systems. (Wind River recommends this image for development.)

wrlinux-image-glibc-idp-intel-quark.tar.bz2

A tar file containing the root file system image with the SRM capabilities disabled by default. The image is not signed.

You are now ready to deploy your platform project to the device.

Related Links

Deploying Quark Boards Using a vfat-Formatted USB Drive on page 124
Manually format a USB drive with vfat and tar the system files to it. Flash additional system files to the device and boot the device from the USB drive.

Deploying Quark Boards Using a Script on page 125
Use the **deploy.sh** script to place the system files on the boot media. Flash additional system files to the device and boot the device.

Deploying Quark Boards Using a vfat-Formatted USB Drive

Manually format a USB drive with vfat and tar the system files to it. Flash additional system files to the device and boot the device from the USB drive.

After configuring and building your platform project with IDP XT, you must tar the system files to a USB drive and install the bios and boot loader from the same project to Flash. The files in Flash and on the USB drive must be from the same project for the boot to succeed.

Step 1 Format the USB drive to VFAT.

Ubuntu host:

Use **fdisk** and **mkfs.vfat** to format the USB drive; mount the USB drive to **/mnt**

Windows host:

Format the USB drive to **vfat32**.

Step 2 Tar ***projDir*/export/images/intel-quark-idp-srm-bundle.tar.bz2** or *projDir*/**export/images/intel-quark-idp-bundle.tar.bz2** to the USB drive.

Step 3 Unmount the USB drive.

```
# umount /dev/sdb1
```

Step 4 Insert the USB drive in the board USB port on the target.

Step 5 Update the target's Flash firmware.

For more information, see: *Updating Flash Firmware for Quark Boards* on page 132

Step 6 Boot the target.

The boot behavior is as follows:

- When the Quark board has a USB Flash drive attached, but no MMC card, the board boots from USB and uses USB Flash drive as the root device.
- When the Quark board has an MMC card attached, but no USB Flash drive, the board boots from the MMC card and uses it as root device.
- When the Quark board has both a USB Flash drive and an MMC card connected at boot time, the default is to boot from USB and use the USB Flash drive as the root device.

NOTE: Only one USB Flash drive is supported. Do not attach more than one USB Flash drive to the target at boot time.

Related Links

Building Platform Projects for Quark Boards on page 122
Configure and build a platform project configured for a Quark board.

Deploying Quark Boards Using a Script

Use the **deploy.sh** script to place the system files on the boot media. Flash additional system files to the device and boot the device.

After configuring and building your platform project with IDP XT, you must deploy the system files to the boot media and install the BIOS and boot loader from the same project to Flash. The files in Flash and on the USB drive must be from the same project for the boot to succeed.

NOTE: The **deploy.sh** script is intended for use only with USB storage devices. The script does not support SD card deployment.

Step 1 Deploy the file to the USB drive using **deploy.sh**.

You can use either of the following files:

- ***projDir*/export/intel-quark-glibc-idp-standard-dist-srm.tar.bz2**
- ***projDir*/export/intel-quark-idp-standard-dist.tar.bz2**

NOTE: Using the **-y** option on **deploy.sh** deletes existing partitions and creates new partitions. If you do not use the **-y** option, you must create VFAT and ext3/ext4 partitions manually before running **deploy.sh**. For information on creating partitions, see *Updating the Target System* on page 136.

For more information on the **deploy.sh** script, see *About the deploy.sh Script* on page 122.

For example, using a Cross Hill board and assuming your device is **/dev/sdb**:

```
$ sudo ./deploy.sh -f \
projDir/export/intel-quark-glibc-idp-standard-dist-srm.tar.bz2 \
-d /dev/sdb -b cross-hill -u -y
```

After deployment your USB device is formatted with two partitions, a VFAT and an EXT3 file system.

You can set the rootfs partition size in bytes with the **-p** option. For example:

```
$ sudo ./deploy.sh -f \
projDir/export/intel-quark-glibc-idp-standard-dist-srm.tar.bz2 \
-d /dev/sdb -b cross-hill -u -y -p "3G"
```

Step 2 Insert the USB drive in the board USB port on the target.

Step 3 Update the target's Flash firmware.

For more information, see: *Updating Flash Firmware for Quark Boards* on page 132

Step 4 Boot the target.

The boot behavior is as follows:

- When the Quark board has a USB Flash drive attached, but no MMC card, the board boots from USB and uses USB Flash drive as the root device.
- When the Quark board has an MMC card attached, but no USB Flash drive, the board boots from the MMC card and uses it as root device.
- When the Quark board has both a USB Flash drive and an MMC card connected at boot time, the default is to boot from USB and use the USB Flash drive as the root device.

NOTE: Only one USB Flash drive is supported. Do not attach more than one USB Flash drive to the target at boot time.

Related Links

Building Platform Projects for Quark Boards on page 122
Configure and build a platform project configured for a Quark board.

Building Platform Projects for Advantech UTX-3115 Boards

Configure and build a platform project for an Advantech UTX-3115 board.

IDP XT supports the Advantech UTX-3115 board.

NOTE: Wind River recommends that you exclude the **grsecurity** feature from your project during development. **grsecurity** is automatically included; to exclude **grsecurity**, add **--with-template=non_grsec** to your configure line.

Step 1 Set the Wind River Linux environment variables on your host machine.

```
$ <installDir>/wrenv.sh -p wrlinux-5
```

Step 2 Create a platform project directory, *projDir*.

```
$ mkdir <projDir>
$ cd <projDir>
```

Step 3 Configure the platform project.

Use the following configure command:

```
$ $WIND_LINUX_CONFIGURE --enable-board=intel-atom-baytrail --enable-addons=wr-idp \
--enable-kernel=standard --enable-rootfs=glibc-idp \
--enable-parallel-pkgbuilds=4 --enable-jobs=4
```

NOTE: $WIND_LINUX_CONFIGURE is an environment variable set by **wrenv.sh** for the location of the **configure** command.

NOTE: The **openjdk-bin** layer is not installed by default. If you want to add the capabilities of Wind River OpenJDK to your build, add the **feature/openjdk-bin** option to your **configure** command. For example:

```
$ $WIND_LINUX_CONFIGURE --enable-board=intel-quark --enable-addons=wr-idp \
--enable-kernel=standard --enable-rootfs=glibc-idp \
--with-template=feature/openjdk-bin --enable-bootimage=ext3,hdd \
--enable-parallel-pkgbuilds=4 --enable-jobs=4
```

Step 4 Build the project.

```
$ make fs
```

Building the project generates the following items in the ***projDir*/export/images** folder:

bzImage-initramfs-intel-atom-baytrail.bin

The kernel image with **initramfs** bundled inside it. Use this image if you are using the IDP XT SRM capabilities. (Wind River recommends this image.)

bzImage-intel-atom-baytrail.bin

The kernel image without **initramfs**. SRM capabilities will not work if you use this kernel image.

wrlinux-image-glibc-idp-intel-atom-baytrail-dist-srm.tar.bz2

A tar file containing the root file system image with the SRM capabilities enabled by default. The image is signed with default keys, which cannot be used for production systems. (Wind River recommends this image for development.)

wrlinux-image-glibc-idp-intel-atom-baytrail.tar.bz2

A tar file containing the root file system image with the SRM capabilities disabled by default. The image is not signed.

Step 5 Deploy the kernel image and root file system on the USB drive.

Follow the procedure appropriate to your configuration.

Related Links

Deploying Advantech UTX-3115 Boards Using a Script on page 128
Deploy your image to a USB storage device using a Wind River script. Use it to boot your board.

Deploying Advantech UTX-3115 Boards Manually on page 129
Deploy your image to a USB storage device manually. Use it to boot your board.

Configuring the BIOS for UTX-3115 and other Bay Trail Boards on page 130

Configure the default 32-bit or 64-bit BIOS in UTX-3115 and other Bay Trail boards.

Deploying Advantech UTX-3115 Boards Using a Script

Deploy your image to a USB storage device using a Wind River script. Use it to boot your board.

NOTE: The **deploy.sh** script is intended for use only with USB storage devices. The script does not support SD card deployment.

Step 1 Insert the USB storage device in the host machine.

Step 2 Execute the **deploy.sh** script from your project directory.

This command deploys the image on your USB storage device (**/dev/sdb**):

```
$ sudo ./deploy.sh -f projDir/export/images/<fileName>.tar.bz2 -d \
/dev/sdb -y -u
```

where ***fileName*.tar.bz2** is one of the following:

- **wrlinux-image-glibc-idp-intel-atom-baytrail-dist-srm.tar.bz2**
- **wrlinux-image-glibc-idp-intel-atom-baytrail.tar.bz2**

NOTE: Occasionally, you might need to boot from a legacy GRUB boot loader. For example, if your board has a 64-bit UEFI BIOS and you do not want to update the BIOS image to 32-bits, you have to use a legacy GRUB boot loader to boot your board.

The following command depolys the image to your USB storage device (**/dev/sdb**) for both UEFI and legacy GRUB boot operations. (Select UEFI boot or legacy GRUB boot in the BIOS. For more information see *Configuring the BIOS for UTX-3115 and other Bay Trail Boards* on page 130.)

```
$ sudo ./deploy.sh -f \
projDir/export/images/fileName.tar.bz2 -d \
/dev/sdb -y -u -g ./grub-0.97
```

You can set the rootfs partition size in bytes with the **-p** option. For example:

```
$ sudo ./deploy.sh -f \
projDir/export/images/fileName.tar.bz2 -d \
/dev/sdb -y -u -p "3G"
```

For more information on the **deploy.sh** script, see *About the deploy.sh Script* on page 122.

Step 3 Unplug the USB storage device from your host machine.

Step 4 Plug the USB storage device into the target and boot from it.

NOTE: The default user ID and password for Wind River targets are:

User ID: **root**
Password: **root**

Related Links

Building Platform Projects for Advantech UTX-3115 Boards on page 126

Configure and build a platform project for an Advantech UTX-3115 board.

Deploying Advantech UTX-3115 Boards Manually on page 129
Deploy your image to a USB storage device manually. Use it to boot your board.

Configuring the BIOS for UTX-3115 and other Bay Trail Boards on page 130
Configure the default 32-bit or 64-bit BIOS in UTX-3115 and other Bay Trail boards.

Deploying Advantech UTX-3115 Boards Manually

Deploy your image to a USB storage device manually. Use it to boot your board.

To deploy your image to a USB storage device manually:

Step 1 Format the USB drive if you have not already done so.

Create two partitions, one formatted with vfat and one with ext3. For more information, see *Preparing USB Boot Media* on page 247

Step 2 Mount the USB drive.

```
$ su
# mkdir /mnt/sdb_vfat
# mount /dev/sdb1 /mnt/sdb_vfat
# mkdir /mnt/sdb_ext3
# mount /dev/sdb2 /mnt/sdb_ext3
```

Step 3 Extract the rootfs tar file.

```
# tar xjvf projDir/export/images/ \
wrlinux-image-glibc-idp-intel-atom-baytrail-dist-srm.tar.bz2 -C /mnt/sdb_ext3
```

Step 4 Update the vfat partition.

```
# mkdir -p /mnt/sdb_vfat/EFI/BOOT

# cp /mnt/sdb_ext3/boot/bzImage /mnt/sdb_vfat/
# cp /mnt/sdb_ext3/boot/grub/grub.efi /mnt/sdb_vfat/EFI/BOOT/BOOTIA32.EFI
# cp /mnt/sdb_ext3/boot/grub/grub.conf /mnt/sdb_vfat/EFI/BOOT/BOOTIA32.conf
```

Modify **/mnt/sdb_vfat/EFI/BOOT/BOOTIA32.conf** as follows:

```
root (hd0, 0)
kernel    /bzImage root=/dev/sdb2 rootdelay=5
```

Step 5 Optional (if you want to boot from legacy GRUB): Modify **/mnt/sdb_ext3/boot/grub/menu.lst** as follows:

```
title       Wind River Intelligent Device Platform
root        (hd0,1)
kernel      /boot/bzImage root=/dev/sdb2 rw,noatime rootwait reboot=bios
```

Step 6 Optional (if you want to boot from legacy GRUB): Install GRUB

$ ./grub-0.97 --batch

```
Probing devices to guess BIOS drives. This may take a long time.
```

```
    GNU GRUB  version 0.97  (640K lower / 3072K upper memory)

 [ Minimal BASH-like line editing is supported.  For the first word, TAB
   lists possible command completions.  Anywhere else TAB lists the possible
   completions of a device/filename. ]
grub> device (hd0) /dev/sdb
grub> root (hd0,1)
grub> setup (hd0)
grub> quit
```

Step 7 Unmount the USB drive.

```
# umount /dev/sdb1
# umount /dev/sdb2
# exit
```

Step 8 Unplug the USB drive from your host machine and plug it into the target.

Step 9 Connect the HDMI cable to a display monitor and to the target.

Step 10 Connect the power adaptor and power on the target.

Step 11 Check the BIOS configuration (for more information see *Configuring the BIOS for UTX-3115 and other Bay Trail Boards* on page 130) and boot the target.

NOTE: Do not use the traditional Wind River Linux approach of issuing the command **make usb-image-burn** to prepare the USB image because **make usb-image-burn** uses **syslinux** as the boot loader instead of GRUB; GRUB 0.97 is required for SRM to function properly.

NOTE: The default user ID and password for Wind River targets are:

User ID: **root**
Password: **root**

Related Links

Building Platform Projects for Advantech UTX-3115 Boards on page 126
Configure and build a platform project for an Advantech UTX-3115 board.

Deploying Advantech UTX-3115 Boards Using a Script on page 128
Deploy your image to a USB storage device using a Wind River script. Use it to boot your board.

Configuring the BIOS for UTX-3115 and other Bay Trail Boards on page 130
Configure the default 32-bit or 64-bit BIOS in UTX-3115 and other Bay Trail boards.

Configuring the BIOS for UTX-3115 and other Bay Trail Boards

Configure the default 32-bit or 64-bit BIOS in UTX-3115 and other Bay Trail boards.

The default BIOS in UTX-3115 and other Bay Trail boards can be 32-bit or 64-bit, and the BIOS configuration might not match. When the board has a 32-bit BIOS, IDP XT uses UEFI to boot the board (this is the recommended procedure); when the board has a 64-bit BIOS, it must be booted using a legacy GRUB boot loader. (A 64-bit BIOS cannot support 32-bit EFI files, so for Bay Trail boards IDP XT can boot only a 32-bit rootfs using a legacy GRUB boot loader.)

Step 1 Power on and press **F7** or **Del** to enter the BOIS setup menu.

Step 2 Ensure CSM is enabled:

Advanced > **CSM Configuration** > **CSM Support** > **Enabled**

Step 3 Ensure the boot type is supported in the BIOS:

- To boot from legacy GRUB in a 64-bit BIOS, ensure legacy boot is enabled:

  Advanced > **CSM Configuration** > **Boot option filter** > **UEFI and legacy**

  or

  Advanced > **CSM Configuration** > **Boot option filter** > **Legacy only**
- To boot from UEFI in a 32-bit BIOS, ensure UEFI boot is enabled:

  Advanced > **CSM Configuration** > **Boot option filter** > **UEFI and legacy**

  or

  Advanced > **CSM Configuration** > **Boot option filter** > **UEFI only**

NOTE: Boot option filter might be under **Boot** on some boards: **Boot** > **Boot option filter**

Step 4 Ensure that only the EHCI or XHCI USB controller is supported:

Chipset > **South Bridge** > **USB Configuration** > **XHCI** > **Disabled**

Chipset > **South Bridge** > **USB Configuration** > **EHCI** > **Enabled**

or

Chipset > **South Bridge** > **USB Configuration** > **XHCI** > **Enabled**

Chipset > **South Bridge** > **USB Configuration** > **EHCI** > **Disabled**

Step 5 Ensure the Boot Option #1 points to the device you want to use.

- To boot from a USB storage device in UEFI mode:

  Boot > **Hard Drive BBS Priorities** <USB storage device name>

  Boot > **Boot Option #1** UEFT:<USB storage device name>
- To boot from a USB storage device in GRUB mode (no UEFI):

  Boot > **Hard Drive BBS Priorities** <USB storage device name>

  Boot > **Boot Option #1** <USB storage device name>

Related Links

Building Platform Projects for Advantech UTX-3115 Boards on page 126
Configure and build a platform project for an Advantech UTX-3115 board.

Deploying Advantech UTX-3115 Boards Using a Script on page 128
Deploy your image to a USB storage device using a Wind River script. Use it to boot your board.

Deploying Advantech UTX-3115 Boards Manually on page 129

Deploy your image to a USB storage device manually. Use it to boot your board.

Updating BIOS Images on Advantech UTX-3115 Boards Using an SF-100 Programmer

Update a UEFI BIOS image on an Intel Baytrail board using an SF-100 flash programmer.

⚠ **CAUTION:** Ensure that the BIOS image you intend to use is for the Advantech UTX-3115 board (see *Step 2*). If you use a BIOS image file for the UTX-3110 board, for example, you will damage your UTX-3115 board beyond repair.

To update a UEFI BIOS image on an Intel Baytrail board using an SF-100 flash programmer:

Step 1 Install the DediProg software driver on your host computer.

Step 2 Obtain the Advantech UTX-3115 BIOS image file, for example **A115X013X32.bin**, and copy it to your host.

Step 3 Disconnect your Advantech UTX-3115 board's power supply.

Step 4 Connect the SF-100 to your host computer and target board.

Step 5 Update the BIOS image with the following command:

dpcmd -t 450 -u A115X013X32.bin

When you apply power to your target board, the BIOS image will be updated.

Updating Flash Firmware for Quark Boards

Intel Quark BSPs allow for the generation of SPI Flash images (firmware) for a range of supported boards with different profiles.

You can update the UEFI BIOS on Intel Quark boards using the following methods:

- An SF-100 Programmer
- Capsule update in Linux
- Capsule update in a UEFI shell

SPI Flash images are named according to the platform name, flash size, and feature to which they correspond:

Flash-<platform>-<flash_size>-<feature>.bin

In this context, these terms have the following meanings:

<platform>

This is the Intel Quark SoC platform name: **clantonpeak**, **kipsbay**, **galileo**, **crosshill**, or **clantonhill**. Note that the flash image you are using must correspond to the name of the board. If that is not the case, the board will not boot.

<flash_size>

This is the size in bytes of the on-board SPI Flash device you are using. Note that the image size can be smaller than the capacity of the flash chip. For example, you can use a 4MB flash image on an Intel Galileo board that has an 8MB flash chip.

An 8MB flash image typically contains a kernel and an **initramfs** with limited functions, while a 4MB flash image typically contains only UEFI firmware. In either case, we recommend that you use an SD memory card or a USB storage device to deploy a full rootfs.

<feature>

This corresponds to the templates you specify in your project configuration.

NOTE: When you have built your project, the following location contains all the flash images and other files mentioned in this section:

<project-dir>/export/images (symlink to **<project-dir>/bitbake_build/tmp/deploy/images**)

Related Links

Programming SPI Flash Memory Using an SF-100 Programmer on page 133
Program an SPI flash chip on an Intel Quark board with an SF-100 flash programmer.

Updating Flash Using Capsule Update in an EFI Shell on page 134
Update flash firmware on an Intel Quark board using the capsule update method in a UEFI shell.

Updating Flash Using Capsule Update in Linux on page 135
Update flash firmware using the capsule update method in Linux.

Programming SPI Flash Memory Using an SF-100 Programmer

Program an SPI flash chip on an Intel Quark board with an SF-100 flash programmer.

If you choose not to use one of the capsule methods for updating flash firmware, we recommend using the *DediProg SF-100 Flash Programmer* to burn your the flash chip. Refer to the DediProg documentation for full details on installing and using the SF-100 Flash Programmer.

The flash chip is programmed at the factory with a valid MAC address. The default MAC address in the generated flash image (**<Flash-<platform>-<flash_size>-<feature>.bin**) is specified in the **<board>-platform-data.ini** file (**<path_to_intel-quark_BSP_layer>/recipes-support/spi-layout-tools/files/platform-data/<board>-platform-data.ini**).

To avoid overriding the valid MAC address perform the following steps before you build the flash image:

1. Open the **<board>-platform-data.ini** file.
2. Under the MAC address section, find the line `data.value=xxxxxxxxxxxx` and set a real MAC address.
3. Build your project and generate a flash image that contains a valid MAC address.

NOTE: The flash capsule method of updating flash firmware does not override the MAC address in the flash chip.

To program an SPI flash chip on an Intel Quark board with an SF-100 flash programmer:

Step 1 Install the DediProg USB driver.

Follow the DediProg installation instructions.

Step 2 Disconnect the power supply from your target board.

Step 3 Burn the flash chip. At the command line, run the following command:

dpcmd -t 450 -u Flash-<platform>-<flash_size>-<feature>.bin

To burn a chip with a 4MB flash image for example, run the following command:

dpcmd -t 450 -u Flash-<platform>-4M-<feature>.bin -a 0x400000

If the DediProg software does not detect the flash chip correctly, it will display the following error:

```
Error: chip not identified.
By reading the chip ID, the chip applies to [ ]
parameters to be applied by default
chip size is 0 bytes.
```

To resolve this issue, disconnect and then reconnect the cable for your USB storage device. If the issue persists, in the DediProg software UI click **Device Manager** > **Universal Serial Bus controllers** > **DediProg SF Programmer driver**; then disable and re-enable the driver.

Related Links

Updating Flash Firmware for Quark Boards on page 132
Intel Quark BSPs allow for the generation of SPI Flash images (firmware) for a range of supported boards with different profiles.

Updating Flash Using Capsule Update in an EFI Shell on page 134
Update flash firmware on an Intel Quark board using the capsule update method in a UEFI shell.

Updating Flash Using Capsule Update in Linux on page 135
Update flash firmware using the capsule update method in Linux.

Updating Flash Using Capsule Update in an EFI Shell

Update flash firmware on an Intel Quark board using the capsule update method in a UEFI shell.

NOTE: If you want to use this method to update the UEFI BIOS firmware on your target board, add **--with-template=feature/devl-debug** to the configure command for your project. This ensures that when you generate a flash image, it will include a UEFI shell.

To update flash on an Intel Quark board using capsule update in a UEFI shell:

Step 1 Format a microSD or USB storage device to create a **w95 FAT32** file system.

- On a Linux host, use **fdisk** and **mkfs.vfat** to format the storage device.
- On a Windows host, format the storage device to **vfat32**.

Step 2 Copy **CapsuleApp.efi** and **Flash-<platform>-<flash_size>-<feature>.cap** to your microSD card or USB storage device.

Step 3 Connect the storage device to your board and apply power to the board.

Step 4 Type **DEL** or **F7** to open the boot manager menu.

Step 5 Select the **UEFI Internal Shell** boot item.

Step 6 Press **ESC** to open the UEFI shell.

Step 7 Type the following command:

```
fs0:
CapsuleApp.efi <capsule_file_name.cap>
```

For example, the **capsule_file_name.cap** file on a Cross Hill board is **Flash-crosshill-8M-secured.cap**.

The CapsuleApp updates your SPI flash firmware, a process that takes about three minutes. If the update succeeds, the system boots normally.

Related Links

Updating Flash Firmware for Quark Boards on page 132
Intel Quark BSPs allow for the generation of SPI Flash images (firmware) for a range of supported boards with different profiles.

Programming SPI Flash Memory Using an SF-100 Programmer on page 133
Program an SPI flash chip on an Intel Quark board with an SF-100 flash programmer.

Updating Flash Using Capsule Update in Linux on page 135
Update flash firmware using the capsule update method in Linux.

Updating Flash Using Capsule Update in Linux

Update flash firmware using the capsule update method in Linux.

When you choose to perform a capsule update of flash firmware, you use a board-specific capsule file **Flash-<platform>-<flash_size>-<feature>.cap**, located in the **<project-dir>/export/images** directory.

To update flash firmware on an Intel Quark board in Linux:

Step 1 Copy the board-specific capsule file **Flash-<platform>-<flash_size>-<feature>.cap** to **/lib/firmware** on the target file system.

Step 2 On the target system, run the following commands:

```
$ modprobe efi_capsule_update
$ echo -n <capsule_file_name.cap>  > /sys/firmware/efi_capsule/capsule_path
$ echo 1 > /sys/firmware/efi_capsule/capsule_update
$ reboot
```

For example, the **capsule_file_name.cap** file on a Cross Hill board is **Flash-crosshill-8M-secured.cap**.

IMPORTANT: Do not disconnect power from the board or reboot during this procedure, otherwise your board will no longer function.

The flash capsule update takes place when you reboot the board.

Related Links

Updating Flash Firmware for Quark Boards on page 132
Intel Quark BSPs allow for the generation of SPI Flash images (firmware) for a range of supported boards with different profiles.

Programming SPI Flash Memory Using an SF-100 Programmer on page 133

Program an SPI flash chip on an Intel Quark board with an SF-100 flash programmer.

Updating Flash Using Capsule Update in an EFI Shell on page 134
Update flash firmware on an Intel Quark board using the capsule update method in a UEFI shell.

Updating the Target System

Update the file system, kernel image, boot loader, and firmware on the target device.

At some point you may need to make changes to a component in your securely deployed system. All components of your system (file system, kernel image, and boot loader) must be updated whenever you change any component.

Boot Limitations for Baytrail Boards

If you are accessing the target remotely, and cannot access the BIOS, you must deploy the new boot loader and kernel to the existing first VFAT partition of the current boot media, because the target always fetches the boot loader from in the first VFAT partition of the current boot media.

If you can access the BIOS and change the boot sequence, then you can deploy the new boot loader and kernel to the first VFAT partition of any boot media.

Boot Limitations for Quark Boards

If a U-Disk is plugged into the target, the target always fetches the boot loader from the first VFAT partition of the U-Disk. If no U-Disk is plugged into the target, the target fetches the boot loader from the first VFAT partition on the MMC card.

If you do not remove the U-Disk after updating the system, then you must update your system by putting a VFAT partition on the U-Disk. If you remove the U-Disk after updating system, you must put the new boot loader and kernel in the first VFAT partition on the MMC card.

On boot media such as U-Disks or SD cards, only the first VFAT partition is recognized by the target.

Step 1 Boot the target, log in, and start the **sshd** service.

```
# /etc/init.d/sshd start
```

Step 2 On the target console, create a new VFAT partition for the new boot loader, if needed (based on the boot limitations outlined above), and a new ext3/ext4 partition for the new rootfs, on a U-Disk or MMC card.

If you want to deploy your new boot loader and rootfs on existing VFAT and ext3/ext4 partitions on the target's U-Disk or MMC card, skip this step.

NOTE: You must confirm that there is sufficient unpartitioned space on the U-Disk or MMC card. For the VFAT partition, 128 MB is needed. For the ext3/ext4 partition, a minimum of 1 GB is needed.

a) Log into the target.
b) Create a 128 MB primary partition for the new boot loader.

The following example uses a storage device at **/dev/sda**.

```
# fdisk /dev/sda
Welcome to fdisk (util-linux 2.21.1).
```

```
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.


Command (m for help): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p):
Using default response p
Partition number (1-4, default 1):
Using default value 1
First sector (2048-62533295, default 2048):
Using default value 2048
Last sector, +sectors or +size{K,M,G} (2048-62533295, default 62533295): +128M

Partition 1 of type Linux and of size 128 MiB is set

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
#
```

There is now a new primary partition at device node **/dev/sda1**.

c) Create a primary partition greater than 1 GB for the new rootfs.

The following example uses a storage device at **/dev/sda**.

```
# fdisk /dev/sda
Welcome to fdisk (util-linux 2.21.1).

Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.


Command (m for help): n
Partition type:
   p   primary (1 primary, 0 extended, 3 free)
   e   extended
Select (default p):
Using default response p
Partition number (1-4, default 2):
Using default value 2
First sector (264192-62533295, default 264192):
Using default value 264192
Last sector, +sectors or +size{K,M,G} (264192-62533295, default 62533295): +3G

Partition 2 of type Linux and of size 3 GiB is set

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: Re-reading the partition table failed with error 16: Device or resource
busy.
The kernel still uses the old table. The new table will be used at
the next reboot or after you run partprobe(8) or kpartx(8)
Syncing disks.
#
```

There is now a new primary ext3/ext4 partition at device node **/dev/sda2**.

You will reference the two newly created device nodes, (**/dev/sda1** and **/dev/sda2** in the examples above), when you invoke the **update.sh** script, in Step 7 on page 139.

d) Notify the kernel of partition table changes.

```
# partprobe
```

e) Unmount the new partitions.

```
# umount /dev/sda1
# umount /dev/sda2
```

f) Format the new 128 MB primary partition with a VFAT file system.

```
# mkfs.vfat /dev/sda1
```

g) Format the new 3 GB+ primary partition with an ext3 or ext4 file system.

- To create an ext3 file system:

```
# mkfs.ext3 -I 128 /dev/sda2
```

- To create an ext4 file system:

```
# mkfs.ext4  -I 128 /dev/sda2
```

Step 3 On the development host, prepare the new rootfs tarball.

For information on generating a new rootfs tarball, see *Preparing to Build and Boot IDP* on page 121.

Step 4 On the target, copy ***boot_media_vfat_partition*/boot/grub/grub.conf** to **/root/grub.conf**, then modify it to point to the new boot loader and rootfs. Refer to the boot limitations above to determine the value of *boot_media_vfat_partition*.

Edit **/root/grub.conf** as follows:

a) Change **root=*old_path*** to **root=*new_path***.

Replace *new_path* with the device node which will contain your new rootfs. In the examples above, this is /dev/sda2.

b) Change **kernel /boot/bzImage** to **kernel /bzImage**.

TIP: The kernel image is deployed under the root, /, in a VFAT file system.

Step 5 For Intel Quark boards except Galileo, sign the modified **grub.conf**.

a) Copy the modified **/root/grub.conf** from the target to the development host.

```
# scp /root/grub.conf  username@hostIP:~/grub.conf
```

b) Sign the modified **grub.conf** on the development host.

For example:

```
$ cd projDir/layers/wr-idp/wr-srm/recipes-devtools/sst/files/
$ ./SST sign-bootloader --machine=intel_quark --verbose=no \
--owner-cert=keysDir/ownerE-cert.pem \
--romkey-dir=keysDir/quark-romkey \
~/grub.conf
```

Step 6 Transfer the files from the development host to the running target.

a) Transfer the new rootfs tarball from the host machine to the running target.

```
$ scp projDir/export/intel-quark-glibc-idp-standard-dist-srm.tar.bz2 \
root@ip_address:/root/update
```

b) For Intel Quark boards except Galileo, transfer the signed **grub.conf** and **grub.conf.csbh** from the development host to the running target.

```
$ scp ~/grub.conf ~/grub.conf.csbh root@ip_address:/root/update/conf
```

c) For Galileo and Baytrail boards, move the modified **grub.conf** on the target to **/root/update/conf**.

```
# mv /root/grub.conf /root/update/conf/
```

Step 7 On the target, update the whole system.

a) Log into the target.

b) Execute the **update.sh** script to deploy the new rootfs and boot loader.

The following example deploys the new boot loader to **/dev/sdb1** and the new rootfs to **/dev/sda2**. It assumes the U-Disk is still connected to the target after you update the system since the target only recognizes the existing first VFAT partition of the current boot media as explained in the boot limitations above.

```
# cd /root/update
# bash ./update.sh -d /dev/sdb1 -k /dev/sda2 -f ./inte*.tar.gz \
-b cross-hill -u
```

NOTE:

- For securely deployed systems, you must use the *uncontrolled invocation* of **update.sh**—forcing the interpreter to execute it directly—as shown in the example above.
- **-d** specifies the device node to put the new boot loader on. In the example above, it is **/dev/sdb1**.
- **-k** specifies the device node to put the new rootfs on. In the example above, it is **/dev/sda2**.
- **-b** is only needed for Quark boards
- **-f** specifies the rootfs tarball name
- **-u** installs the UEFI boot loader

For more information on the **update.sh** script, see *About the deploy.sh Script* on page 122.

Step 8 For Intel Quark boards, update the firmware now, if needed. For more information, see *Updating Flash Firmware for Quark Boards* on page 132.

Step 9 Reboot the target.

Related Links

Building Platform Projects for Quark Boards on page 122
Configure and build a platform project configured for a Quark board.

Building Platform Projects for Advantech UTX-3115 Boards on page 126
Configure and build a platform project for an Advantech UTX-3115 board.

Updating Flash Firmware for Quark Boards on page 132

Intel Quark BSPs allow for the generation of SPI Flash images (firmware) for a range of supported boards with different profiles.

Using the IA Recovery Image

You can create your own customized recovery image for Intel Architecture boards. In addition, the IDP XT installation provides one prebuilt recovery image.

To use the recovery image, you must have the following minimum-size media:

A USB Flash drive of at least 1GB
An MMC card or SSD of at least 2GB

Creating a Recovery Image

Follow the instructions starting with Step *1* on page 140.

NOTE: Updating IDP XT by installing a new RCPL does not update the recovery image. To include particular fixes in your recovery image, you must create a new recovery image.

1. Edit the **config.log** file to use the new RCPL.
2. Rebuild the project.

```
$ make reconfig
```

3. Build the rootfs and the image and place them on a USB stick.
4. Boot the target from the recovery image.

Using the Prebuilt Image

The prebuilt image is:

recovery.img

To use this image, follow the instructions starting with Step *4* on page 141. Replace **export/usb.img** with the full path to **recovery.img**.

Step 1 Configure a platform project using **--with-template=feature/recovery**.

```
$ $WIND_LINUX_CONFIGURE --enable-board=intel-atom-baytrail --enable-addons=wr-idp \
--enable-kernel=standard --enable-rootfs=glibc-idp \
--with-template=feature/recovery \
--enable-parallel-pkgbuilds=4 --enable-jobs=4
```

Step 2 Make the rootfs.

```
$ make fs
```

Step 3 Build the USB image.

This example uses a tool in the project directory that builds a USB image.

NOTE: A root account is required to run this tool.

NOTE: For Cross Hill and Clanton Hill boards, you must add **-b cross-hill** or **-b clanton-hill** to the configure line. For more information, see the help menu for the **create-usbimage** tool.

NOTE: Occasionally, you might need to boot from a legacy GRUB boot loader. For example, if your board has a 64-bit UEFI BIOS and you do not want to update the BIOS image to 32-bits, you have to use a legacy GRUB boot loader to boot your board.

To boot using a legacy GRUB boot loader, add the following to your command line **-g ./grub-0.97**

```
$ sudo ./create-usbimg \
-f export/intel-atom-baytrail-glibc-idp-standard-dist-srm-tar.bz2 \
-o export/usb.img -r
```

Step 4 Copy the image to a USB stick.

```
$ sudo dd if=export/usb.img of=/dev/sdb bs=4M
$ sudo sync
```

Change **sdb** to match your target device.

Step 5 Insert the USB drive into the board and select **boot from this USB device** in GRUB.

Step 6 Copy the recovery image to another device on the target system.

This example copies the rootfs to the solid state drive (SSD) on a Baytrail board:

```
# tgt=/dev/sda /sbin/reset_media
```

This example copies the rootfs to the SD card:

```
# tgt=/dev/mmcblk0 /sbin/reset_media
```

IDP Preconfigured Profiles

IDP XT 2.0 comes with two preconfigured profiles providing two footprint configuration options: **glibc-idp-small** and **glibc-idp**. Each profile adds specific layers, features, and packages to your platform project.

You will typically use the small-footprint profile for end-point-type devices and the standard-footprint profile for gateway-type devices. These pre-validated configurations can save development time even if you choose to customize them later in the development cycle.

Preconfigured Profile Customization

You can customize the preconfigured profiles by modifying the bitbake recipes.

Small Footprint

***projDir*/layers/wr-idp/wr-idp-devkit/recipes-base/images/wrlinux-image-glibc-idp-small.bb**

Standard Footprint

***projDir*/layers/wr-idp/wr-idp-devkit/recipes-base/images/wrlinux-image-glibc-idp.bb**

Layers, Features, and Packages Included in Preconfigured Profiles

The following table shows which layers and features provide the packages for the preconfigured profiles. All layers and features are included in the standard profile. The layers and features included in the small profile are marked.

Layer	Included in Small Profile	Features and Packages
wr-idp-devkit	yes	**feature/webif** (including **wifi**, **webif**, and **ntp**)
	yes	**feature/firewall**
	yes	**mobile ipv6** (package)
	no	**feature/pppoe** **feature/upnp** **feature/vlan** **feature/mqtt**
wr-srm	yes	**srm** (default)
wr-mcafee	yes	**solidcores3** (package)
oe-core	yes	**dhcp** (package) (including **dhcpv6**)
meta-networking (from Wind River Linux 5.0.x)	no	**radvd** (package)
oe-core (from Wind River Linux 5.0.x)	no	**perl** (package) **python** (package)

Related Links

Using the Small-Footprint Profile on page 142
The small-footprint profile shipped in the rootfs **glibc-idp-small** is designed for end-point-type devices.

Using the Standard Profile on page 143
The standard-footprint profile shipped in the rootfs **glibc-idp** is designed for gateway-type devices.

Using the Small-Footprint Profile

The small-footprint profile shipped in the rootfs **glibc-idp-small** is designed for end-point-type devices.

- Configure your project with **--enable-rootfs=glibc-idp-small**.

  The small-footprint profile includes:

  Layers

  wr-idp-devkit wr-srm meta-java meta-java-networking meta-java-dl wr-mcafee

Features/Packages

dhcpv6 mobile ipv6 netifd webif wifi ppp firewall dnsmasq srm solidcores3

NOTE: You may be able to remove or disable additional kernel configuration options manually.

The following example uses the configure line for a Quark board:

```
$ $WIND_LINUX_CONFIGURE --enable-board=intel-quark \
--enable-rootfs=glibc-idp-small \
--enable-kernel=standard --enable-addons=wr-idp
```

Related Links

IDP Preconfigured Profiles on page 141
IDP XT 2.0 comes with two preconfigured profiles providing two footprint configuration options: **glibc-idp-small** and **glibc-idp**. Each profile adds specific layers, features, and packages to your platform project.

Using the Standard Profile

The standard-footprint profile shipped in the rootfs **glibc-idp** is designed for gateway-type devices.

- Configure your project with **--enable-rootfs=glibc-idp**.

  The standard-footprint profile includes:

  Layers

  wr-idp-devkit wr-srm meta-java meta-java-networking meta-java-dl wr-mcafee

  Features/Packages

  dhcpv6 mobile ipv6 netifd webif wifi ppp firewall dnsmasq ntp perl python pppoe upnp mqtt lua vlan srm solidcores3

  The following example uses the configure line for a Quark board:

```
$ $WIND_LINUX_CONFIGURE --enable-rootfs=glibc-idp --enable-addons=wr-idp \
--enable-kernel=standard --enable-board=intel-quark
```

Related Links

IDP Preconfigured Profiles on page 141
IDP XT 2.0 comes with two preconfigured profiles providing two footprint configuration options: **glibc-idp-small** and **glibc-idp**. Each profile adds specific layers, features, and packages to your platform project.

Platform Boot Time Optimizations

There are configuration or build options available that can speed up target boot time.

The default configuration of IDP XT runs initialization scripts in parallel to speed up boot time and utilize the multiprocessing capabilities of Linux. However, implementing any of the following strategies can provide even faster target boot times. Improvements vary for each board type.

Using a Minimal Image

The default IDP XT rootfs image that is created for the **glibc-idp** preconfigured profile includes many features that your platform project might not need. Replacing the **glibc-idp** rootfs with the **core-image-minimal** rootfs results in a smaller footprint and faster boot time; for example, in general, a target booting the **core-image-minimal** rootfs takes less than 20 seconds to display the login prompt after the boot loader starts the Linux kernel.

To build the **core-image-minimal** rootfs, run **make -C build core-image-minimal** instead of **make fs**.

For example:

```
$ $WIND_LINUX_CONFIGURE ... --enable-rootfs=glibc-idp
...
$ make -C build core-image-minimal
$ ls export/images/core-image-minimal-*
```

For more information on IDP XT preconfigured profiles, see *IDP Preconfigured Profiles* on page 141.

Disabling Security Features

IDP XT includes two layers with security features by default: **wr-srm** and **wr-mcafee**. Excluding the **wr-srm** and **wr-mcafee** layers from your platform project results in a faster boot time for your target. For example, excluding these two layers on an Intel Quark board reduces boot time by about 30 seconds.

These layers are included by default. To exclude these layers, use the **--without-layer=wr-srm,wr-mcafee** option when configuring your platform project.

For more information on **wr-srm** and **wr-mcafee**, see *IDP Features* on page 21.

Delaying Networking Services

The **boot_delay_network** feature of **wr-idp-devkit** reduces boot time by delaying the configuration of networking and networking-related services for 10 seconds after the login prompt appears. On an Intel Quark board, it reduces boot time by about 20 seconds.

When this feature is enabled, a default IDP XT image starts six network services (network, dnsmasq, ipsec, webif, firewall, and webiffirewalllog) 10 seconds after the login prompt appears. The startup log of these services is saved in **syslog** (**/var/log/messages**), instead of being displayed on the default target console (serial port or monitor over VGA).

This feature is disabled by default. To use this feature, include the **--enable-template=feature/boot_delay_network** option when configuring your platform project.

18

Alternative Booting Methods

SRM and Alternative Booting Methods

Secure Remote Management (SRM) is an IDP XT layer provided by the **wr-srm** layer. By default all IDP XT platform projects for boards that support SRM are created with the SRM layer included, which means that you can configure them for secure or trusted boot.

SRM provides the infrastructure to boot to a trusted software stack and to securely manage devices remotely.

Secure Boot

Supported for Cross Hill, Clanton Hill, and Advantech UTX-3115 boards.

Not supported for Galileo board.

Verified Boot

Supported for Advantech UTX-3115 boards using software verified boot.

Secure Booting

Performing a Secure Boot on Cross Hill and Clanton Hill Boards

Performing a secure boot involves configuring and building a platform project, burning the flash image into the flash on the board, deploying the signed images to the board, and booting the board.

This example was developed on a board that was running in *secure open* mode.

Step 1 Configure and build a platform project.

For more information, see *Building Platform Projects for Quark Boards* on page 122.

Use the following configure command:

```
$ $WIND_LINUX_CONFIGURE --enable-board=intel-quark --enable-addons=wr-idp \
--enable-kernel=standard --enable-rootfs=glibc-idp \
--with-template=feature/secure-boot \
--enable-parallel-pkgbuilds=4 --enable-jobs=4
```

Step 2 Build the project.

```
$ make fs
```

Step 3 Confirm that the flash image and the SRM signed rootfs tar file were generated.

Step 4 Update the target's Flash firmware.

For more information, see: *Updating Flash Firmware for Quark Boards* on page 132

Step 5 Deploy the SRM signed rootfs on the USB boot disk.

Follow the instructions for deploying the image and rootfs and rebooting the board in:

- *Deploying Quark Boards Using a vfat-Formatted USB Drive* on page 124
- *Deploying Quark Boards Using a Script* on page 125

Related Links

Updating Flash Firmware for Quark Boards on page 132
Intel Quark BSPs allow for the generation of SPI Flash images (firmware) for a range of supported boards with different profiles.

Performing a Secure Boot on Advantech UTX-3115 Using UEFI

The Intel Baytrail device provides a secure boot policy that allows only the signed bootloader (**grub.efi**) to run on the UEFI.

Step 1 Configure and build a platform project.

Follow the configure and build steps in *Building Platform Projects for Advantech UTX-3115 Boards* on page 126. Use the following configure command:

```
$ $WIND_LINUX_CONFIGURE --enable-board=intel-atom-baytrail --enable-addons=wr-idp \
--enable-kernel=standard --enable-rootfs=glibc-idp \
--enable-parallel-pkgbuilds=4 --enable-jobs=4
```

Step 2 Sign the rootfs file with SST.

For more information, see *Signing the rootfs File* on page 57

Step 3 Burn AMI BIOS **A115X013X32.bin** onto an Intel Advantech UTX-3115 board using the SF100 DediProg device.

For more information, see: *Updating BIOS Images on Advantech UTX-3115 Boards Using an SF-100 Programmer* on page 132

Step 4 Deploy the signed rootfs on U-Disk.

For more information see:

- *Deploying Advantech UTX-3115 Boards Using a Script* on page 128
- *Deploying Advantech UTX-3115 Boards Manually* on page 129

Step 5 Insert the U-Disk into the Advantech UTX-3115 board.

Step 6 Configure the BIOS and reboot the board.

Select the following menu items:

a) **Advanced** > **CSM Configuration** > **Video** > **UEFI only** > **F4** > **Reboot**
b) **Advanced** > **CSM Configuration** > **CSM Support** > **[Disabled]** > **F4** > **Reboot**
c) **Security** > **Secure Boot menu** > **Secure Boot** > **[Disabled]**
d) **Security** > **Secure Boot menu** > **Secure Boot** > **Key Management** > **Delete All**
e) **Secure Boot Variables** > **YES** > **F4** > **Reboot**

Step 7 Enter the EFI shell console.

Press **F7** and select **UEFI: Built-in EFI shell**.

Step 8 Enter the U-Disk VFAT partition.

Select **fs0: enter U-Disk VFAT partition**.

Step 9 Run the signed **grub.efi** file **BOOTIA32.efi**.

```
# cd EFI\BOOT
# ./BOOTIA32.efi
Platform is in Setup Mode
KEK LEN: 1068
Created KEK Cert
DB LEN: 2727
Created db Cert
PK LEN: 1086
```

Step 10 Configure the BIOS again and reboot the board.

a) Press **F7** and select **Enter Setup**.
b) **Security** > **Secure Boot menu** > **Secure Boot** > **[Enable]** > **F4** > **Reboot**

Step 11 Enter the U-Disk VFAT partition.

Select **fs0: enter U-Disk VFAT partition**.

Step 12 Run the signed **grub.efi** file **BOOTIA32.efi**.

The **BOOTIA32.efi** boots the kernel successfully.

Step 13 Confirm that the secure boot policy is working correctly on the board.

a) Copy an unsigned **grub.efi** file to **/EFI/BOOT** in the VFAT partition on the U-Disk.
b) Insert the U-Disk into the Advantech UTX-3115 board.
c) Press **F7** and select **UEFI: Built-in EFI shell**.
d) Run the unsigned **grub.efi** file **UNSIGNED_BOOTIA32.efi**.

The following message appears:

```
fs0:\EFI\BOOT> UNSIGNED_BOOTIA32.efi
Error reported: Access Denied
```

This is the correct result when the secure boot policy is working correctly.

Verified Booting

Verified Boot Prerequisites

Before starting the verified boot workflow, you must have the correct hardware and software.

In addition to the standard IDP XT prerequisites, you need the following tools and hardware to enable verified boot:

Signed Kernel

an SRM-enabled kernel containing an encrypted kernel image digest.

The signed kernel is included automatically by the **wr-srm** layer, which is included automatically when you use the **--enable-addons=wr-idp** option.

Grub-0.97

an enhanced version of the standard GRUB bootloader which supports TCG-compliant computers equipped with TPMs. Grub-0.97 confirms that the ciphers of all the components it loads match the kernel image digest.

Related Links

Performing a Verified Boot on page 148
Performing a verified boot involves configuring and building a platform project, and rebooting the device.

Performing a Verified Boot

Performing a verified boot involves configuring and building a platform project, and rebooting the device.

NOTE: Wind River recommends that you exclude the **grsecurity** feature from your project during development. **grsecurity** is automatically included; to exclude **grsecurity**, add **--with-template=non_grsec** to your configure line.

Step 1 Configure and build a platform project.

For details, see *Building Platform Projects for Advantech UTX-3115 Boards* on page 126.

The following configure command uses the **intel-atom-baytrail** BSP as an example:

```
$ $WIND_LINUX_CONFIGURE --enable-board=intel-atom-baytrail --enable-addons=wr-idp \
--enable-rootfs=glibc-idp --enable-kernel=standard \
--enable-parallel-pkgbuilds=4 --enable-jobs=4
```

When you finish, you will have done the following:

1. Configured and built the platform project.
2. Burned the kernel and rootfs to the boot media.
3. Installed Grub-0.97 on the boot media.
4. Booted the device.

Step 2 Reboot the device.

Grub-0.97 verifies the kernel image before loading and booting it. For example:

```
##########################################################
##                                                      ##
##                Verified Booting ...                  ##
##                                                      ##
##########################################################

##########################################################
##                                                      ##
##                Verification OK                       ##
##                                                      ##
##########################################################
```

Related Links

Verified Boot Prerequisites on page 148
Before starting the verified boot workflow, you must have the correct hardware and software.

19

Configuring IDP Features

Layers and Features

This reference lists the IDP XT layers and associated features and indicates which boards support the features.

For more information about any layer or feature, see the **README** in the layer or feature directory.

Layers	Features	Default (layer included)	Default (glibc-idp rootfs)	Cross Hill	Clanton Hill	Galileo	Advantech UTX-3115
meta-java-dl	N/A	N/A	N/A	Yes	Yes	Yes	Yes
wr-digi-idigiconnector	default	Yes	No	Yes	Yes	Yes	Yes
wr-wks-oneagent-oma-dm-ia	default	Yes	No	Yes	Yes	Yes	Yes
wr-wks-oneagent-tr069	default	Yes	No	Yes	Yes	Yes	Yes

Layers	Features	Default (layer included)	Default (glibc-idp rootfs)	Cross Hill	Clanton Hill	Galileo	Advantech UTX-3115
wr-prosyst-mbs-smarthome-sdk-ia	default	Yes	No	Yes	Yes	Yes	Yes
wr-exegin-zigbee-ia	default	Yes	No	Yes	No	No	No
wr-mcafee	default	Yes	Yes	Yes	Yes	Yes	Yes
wr-srm	default	Yes	Yes	Yes	Yes	Yes	Yes
	grsec_std	Yes	Yes	Yes	Yes	Yes	Yes
	Encrypted Storage (No separate feature template)	Yes	Yes	Yes	Yes	Yes	Yes
	Secure Package Management (No separate feature template)	Yes	Yes	Yes	Yes	Yes	Yes
	non_grsec	No	No	Yes	Yes	Yes	Yes
wr-idp-devkit	default	Yes	Yes	Yes	Yes	Yes	Yes
	backports	No	No	Yes	Yes	Yes	Yes
	bluetooth	No	No	Yes	Yes	Yes	Yes
	boot_delay_network	No	No	Yes	Yes	Yes	Yes
	firewall	No	Yes	Yes	Yes	Yes	Yes
	graphics_qt	No	No	No	No	No	Yes
	idp_devkit_full	No	No	No	No	No	Yes
	ipsec_vpn	No	No	Yes	Yes	Yes	Yes
	l2tp	No	No	Yes	Yes	Yes	Yes
	mqtt	No	Yes	Yes	Yes	Yes	Yes

Layers	Features	Default (layer included)	Default (glibc-idp rootfs)	Cross Hill	Clanton Hill	Galileo	Advantech UTX-3115
	online_updates	No	No	Yes	Yes	Yes	Yes
	opc	No	No	Yes	Yes	Yes	Yes
	opc_demo	No	No	Yes	Yes	Yes	Yes
	openjdk-bin	No	No	Yes	Yes	Yes	Yes
	pppoe	No	Yes	Yes	Yes	Yes	Yes
	pptp_vpn	No	No	Yes	Yes	Yes	Yes
	recovery	No	No	Yes	Yes	Yes	Yes
	upnp	No	Yes	Yes	Yes	Yes	Yes
	vlan	No	No	Yes	Yes	Yes	Yes
	webif	No	Yes	Yes	Yes	Yes	Yes
	wrs_qt_demo	No	No	No	No	No	Yes
wr-ima-appraise	default	Yes	No	Yes	Yes	Yes	Yes

About Configuring Layers

In order to include any IDP XT-related layer, you must specify **--enable-addons=wr-idp** option in your configure line.

NOTE: Specifying the **--enable-addons=wr-idp** option does not include the layers in your project. The options enables the project to include any layers you choose to specify.

NOTE: For descriptions of all IDP XT layers and features, see *IDP Features* on page 21.

IDP XT supports three different types of rootfs, specified using the **--enable-rootfs** option. The IDP XT layers that are included by default depend on the rootfs you choose. Specify one of the following types of rootfs:

glibc-idp
glibc-idp-small
glibc-std

Wind River recommends using either the **glibc-idp** or the **glibc-idp-small** rootfs type for IDP XT development; they provide a basic configuration including Java, development tools, and networking, plus SRM for boards that support it. By default, all platform projects created with **glibc-idp** or **glibc-idp-small** include the following IDP XT layers:

meta-java-dl
wr-idp-devkit
wr-srm
wr-mcafee

NOTE: Two additional Wind River Linux layers, **meta-networking** and **meta-java**, are also included with **glibc-idp** and **glibc-idp-small**. Which layers are automatically included is defined in the ***projDir*/layers/wr-idp/wr-idp-devkit/templates/rootfs.cfg** file.

Related Links

About Layers with glibc-idp and glibc-idp-small on page 154
Configuring with **glibc-idp** or **glibc-idp-small** includes the default IDP XT layers in your project.

About Layers with glibc-std on page 155
If you decide to use the **glibc-std** rootfs, you must include IDP XT layers manually, including the default IDP XT layers, using the **--with-layer=** option.

Inspecting Layer Contents on page 155
You cannot inspect IDP XT layers by listing the directory contents, as layers are distributed as compressed **git** trees. Build a platform project to view the layer contents.

About Layers with glibc-idp and glibc-idp-small

Configuring with **glibc-idp** or **glibc-idp-small** includes the default IDP XT layers in your project.

The following example includes the three IDP XT default layers in a Quark project:

```
$ $WIND_LINUX_CONFIGURE --enable-addons=wr-idp --enable-board=intel-quark \
--enable-kernel=standard --enable-rootfs=glibc-idp
```

The default layers are:

meta-java-dl
wr-idp-devkit
wr-srm
wr-mcafee

To exclude a default IDP XT layer from your platform project, use the **--without-layer** option. The following example excludes the **wr-srm** layer from a Quark project:

```
$ $WIND_LINUX_CONFIGURE --enable-addons=wr-idp --enable-board=intel-quark \
--enable-kernel=standard --enable-rootfs=glibc-idp --without-layer=wr-srm
```

NOTE: You can confirm which layers are included in your project by viewing the file ***projDir*/layer_paths** file after running the configure command.

To include additional, non-default IDP XT layers in your project, add the option **--with-layer=***layerName*. The following example includes the default layers and the **wr-exegin-zigbee-ia** layer to a Quark project:

```
$ $WIND_LINUX_CONFIGURE --enable-addons=wr-idp --enable-board=intel-quark \
--enable-kernel=standard --enable-rootfs=glibc-idp --with-layer=wr-exegin-zigbee-ia
```

TIP: When you use **glibc-idp** and **glibc-idp-small**, you do not need to include the default IDP XT layers again using the **--with-layer=** option. For example:

```
$ $WIND_LINUX_CONFIGURE --enable-addons=wr-idp --enable-board=intel-quark \
--enable-kernel=standard --enable-rootfs=glibc-idp --with-layer=wr-srm
```

This configure line attempts to include the **wr-srm** layer twice: once by specifying **--enable-rootfs=glibc-idp** and again by specifying **--with-layer=wr-srm**. This does not cause problems with the build, but is unnecessary.

For more details about **glibc-idp** and **glibc-idp-small**, see *IDP Preconfigured Profiles* on page 141.

Related Links

About Configuring Layers on page 153
In order to include any IDP XT-related layer, you must specify **--enable-addons=wr-idp** option in your configure line.

About Layers with glibc-std

If you decide to use the **glibc-std** rootfs, you must include IDP XT layers manually, including the default IDP XT layers, using the **--with-layer=** option.

The following example includes the **wr-idp-devkit**, **wr-srm**, and **meta-java-dl** layers in your Quark project:

```
$ $WIND_LINUX_CONFIGURE --enable-addons=wr-idp --enable-board=intel-quark \
--enable-kernel=standard --enable-rootfs=glibc-std \
--with-layer=wr-idp-devkit,wr-srm,meta-java-dl
```

The following example does not include any IDP XT layers in your project:

```
$ $WIND_LINUX_CONFIGURE --enable-addons=wr-idp --enable-board=intel-quark \
--enable-kernel=standard --enable-rootfs=glibc-std
```

Related Links

About Configuring Layers on page 153
In order to include any IDP XT-related layer, you must specify **--enable-addons=wr-idp** option in your configure line.

Inspecting Layer Contents

You cannot inspect IDP XT layers by listing the directory contents, as layers are distributed as compressed **git** trees. Build a platform project to view the layer contents.

NOTE: You can view all the README files for all the layers included in your project in the ***projDir*/README**S directory after running the configure command.

Layers whose contents do not appear when you list the directory include wr-idp-devkit and wr-srm. For example:

- View the layers in a platform project.

a) Create a platform project.

```
$ $WIND_LINUX_CONFIGURE --enable-addons=wr-idp --enable-board=intel-quark \
--enable-kernel=standard --enable-rootfs=glibc-idp
```

b) List the project layers directory.

```
$ ls -l <projDir>/layers/wr-idp/wr-srm/
total 56
drwxrwxr-x  2 user user 4096 Aug  1 16:48 classes
drwxrwxr-x  2 user user 4096 Aug  1 16:48 conf
drwxrwxr-x  2 user user 4096 Aug  1 16:48 downloads
drwxrwxr-x  3 user user 4096 Aug  1 16:48 files
drwxrwxr-x  3 user user 4096 Aug  1 16:48 git
-rw-rw-r--  1 user user 9702 Aug  1 16:48 README
drwxrwxr-x  4 user user 4096 Aug  1 16:48 recipes-base
drwxrwxr-x 26 user user 4096 Aug  1 16:48 recipes-core
drwxrwxr-x  6 user user 4096 Aug  1 16:48 recipes-devtools
drwxrwxr-x  3 user user 4096 Aug  1 16:48 recipes-samples
drwxrwxr-x  2 user user 4096 Aug  1 16:48 scripts
drwxrwxr-x  4 user user 4096 Aug  1 16:48 templates
```

Related Links

About Configuring Layers on page 153
In order to include any IDP XT-related layer, you must specify **--enable-addons=wr-idp** option in your configure line.

About Configuring Default Features

Most IDP XT capabilities are included in your platform project by default. Those that are not can be included using the **--with-template=** option.

Every layer has one or more features. Each layer has at least one feature named **default** which is typically represented by a folder named **default**, for example, ***projDir*/layers/wr-idp/wr-srm/templates/default**.

The contents of this default folder are always included in your project when you include the corresponding layer. In the example, the layer is **wr-srm**. You can inspect the **template.conf** file to see what features are automatically included. For example, to see what is included by the **wr-srm** layer, see ***projDir*/layers/wr-idp/wr-srm/templates/default/template.conf**.

About Configuring Non-Default Features

Some layers have non-default features in addition to the default features. In most cases, include these figures using the **--with-template** option.

For example:

```
$ $WIND_LINUX_CONFIGURE --enable-board=intel-quark --enable-kernel=standard \
--enable-rootfs=glibc_idp --enable-addons=wr-idp --with-template=feature/opc_demo
```

NOTE: To see which features are included in your project, view the ***projDir*/template_paths** file after running the configure command.

For a complete list of layers and features, see *Layers and Features* on page 151

However, certain non-default features from certain layers are included automatically; in other words, you do not have to specify the **--with-template** option to include them in your project.

The rules that include these features are typically specified in the **template.conf** file located in the default feature folder, for example, ***projDir*/layers/wr-idp/wr-srm/templates/default/template.conf.**

The following table shows all features not included in a **default** folder which get included automatically when their corresponding layer is included.

Feature	Layer
grsec_std	**wr-srm**

Quark Example

The following examples using a supported Quark board include the **wr-srm**, **wr-idp-devkit**, and **meta-java-dl** layers with their default features (***layerName*/templates/default**) plus the feature **grsec_std**.

Example with **glibc-idp**:

```
$ $WIND_LINUX_CONFIGURE --enable-addons=wr-idp --enable-board=intel-quark \
--enable-kernel=standard --enable-rootfs=glibc-idp
```

Example with **glibc-std**:

```
$ $WIND_LINUX_CONFIGURE --enable-addons=wr-idp --enable-board=intel-quark \
--enable-kernel=standard --enable-rootfs=glibc-std \
--with-layer=wr-idp-devkit,wr-srm,meta-java-dl
```

NOTE: Some additional features are included in the **glibc-idp** rootfs example; for more information, see *Non-Default Features Included by glibc-idp* on page 157.

Non-Default Features Included by glibc-idp

Some non-default features are included in your project automatically whenever you specify **glibc-idp** as the rootfs.

All features in a default folder are included when the associated layer is included. In addition, the following features are also included by default if their layer is included; you do not need to specify the **--with-template** option to include them.

Feature	Layer
firewall	**wr-idp-devkit**
mqtt	**wr-idp-devkit**
pppoe	**wr-idp-devkit**
upnp	**wr-idp-devkit**
webif	**wr-idp-devkit**

For example, to include the **wr-srm**, **wr-idp-devkit**, and **meta-java-dl** layers with their default features (***layerName*/templates/default**) plus the feature **grsec_std** and all the features listed in the table above, use the following configure line:

```
$ $WIND_LINUX_CONFIGURE --enable-addons=wr-idp --enable-board=intel-quark \
--enable-kernel=standard --enable-rootfs=glibc-idp
```

However, the following configure line using **glibc-std** has a different result:

```
$ $WIND_LINUX_CONFIGURE --enable-addons=wr-idp --enable-board=intel-quark \
--enable-kernel=standard --enable-rootfs=glibc-std \
--with-layer=wr-idp-devkit,wr-srm,meta-java-dl
```

This configure line includes **wr-srm**, **wr-idp-devkit**, and **meta-java-dl** with their default features (***layerName*/templates/default**) plus the feature **grsec_std** (located at **wr-srm/templates/feature/grsec_std**). No feature from the table above is included because we specified **glibc-std** as the rootfs.

All other non-default features can be included using the **--with-template** option. For example:

```
$ $WIND_LINUX_CONFIGURE --enable-board=intel-quark --enable-addons=wr-idp \
--enable-kernel=standard --enable-rootfs=glibc_idp --with-template=feature/opc_demo
```

NOTE: To see which features are included in your project, view the ***projDir*/template_paths** file after running the configure command.

For a complete list of layers and features, see *Layers and Features* on page 151.

About Secure Remote Management

Secure Remote Management (SRM) is an IDP XT layer provided by the **wr-srm** layer. By default all IDP XT platform projects for boards that support SRM are created with the SRM layer included unless you specify the **--without-layer=wr-srm** option on the configure line.

SRM is supported for the following BSPs:

intel-quark for Cross Hill and Clanton Hill
intel-atom-baytrail for Advantech UTX-3115

The SRM layer on Advantech UTX-3115 boards requires GRUB 0.97. In addition, the procedure for deploying and booting the image on the USB drive differs depending on whether you configure your project with or without the SRM layer. For more information, see:

Booting Intel Architecture Boards without SRM Layer
Deploying Advantech UTX-3115 Boards Using a Script on page 128
Deploying Advantech UTX-3115 Boards Manually on page 129

For more information on including the SRM layer or excluding it from a platform project refer, see *About Configuring Layers* on page 153.

20

Installing Tools for Application Development and Control

Including Bluetooth in a Platform Project

Enabling Bluetooth support requires configuring your platform project with Bluetooth support.

Step 1 Run your desired configure line.

For example, to configure for a supported Quark board, use the following:

```
$ $WIND_LINUX_CONFIGURE --enable-board=intel-quark \
--enable-kernel=standard --enable-addons=wr-idp --enable-rootfs=glibc-idp \
--with-template=feature/bluetooth
```

Step 2 Build the rootfs.

```
$ make fs
```

Enabling 3G WWAN

Enabling 3G requires configuring your board with the correct layer and template and confirming various configuration settings.

Step 1 Configure and build your platform project.

For more information, see:

Building Platform Projects for Quark Boards on page 122
Building Platform Projects for Advantech UTX-3115 Boards on page 126

This example uses the **intel-quark** BSP:

```
$ $WIND_LINUX_CONFIGURE --enable-board=intel-quark \
--enable-kernel=standard --enable-rootfs=glibc-idp --enable-addons=wr-idp \
--enable-jobs=3 --enable-parallel-pkgbuilds=3
```

Step 2 Confirm that the antenna is correctly connected to the 3G modem.

NOTE: For the Telit HE910 modem, the SIM card holder is on the underside of the mini pci-e card.

Step 3 Deploy the kernel image and rootfs to a USB Flash drive and boot the target.

For more information, see:

Deploying Quark Boards Using a vfat-Formatted USB Drive on page 124
Deploying Quark Boards Using a Script on page 125
Deploying Advantech UTX-3115 Boards Using a Script on page 128
Deploying Advantech UTX-3115 Boards Manually on page 129

3G starts automatically on boot.

Step 4 Confirm that **/etc/config/network** contains the correct information.

Alternatively, you can use the Webif interface to confirm this information. Start Webif and select **Network** > **Networks**

This example shows the values for Telit HE910 and the Rogers SIM card.

For the Rogers SIM card, you can find the value for APN name in **/etc/ppp/chat/rogers_chat**.

```
# cat /etc/config/network
config interface 'loopback'
        option ifname 'lo'
        option proto 'static'
        option ipaddr '127.0.0.1'
        option netmask '255.0.0.0'

config interface 'wan'
        option ifname 'eth0'
        option proto 'dhcp'

config interface 'lan'
        option ifname 'eth1 wlan0'
        option type 'bridge'
        option proto 'static'
```

```
        option ipaddr '192.168.1.1'
        option netmask '255.255.255.0'

config interface 'wwan'
        option ifname '3g-wwan'
        option proto '3g'
        option device '/dev/ttyACM0'
        option ppp_redial 'demand'
        option defaultroute '1'
        option peerdns '1'
        option service 'umts_first'
        option sconnservice 'UMTS'
        option dialnumber '*99***1#'
        option country 'cn'
        option apn '3gnet'
config device 'modem_cell'
        option name 'modem_cell'
        option present 'Yes'
        option protoall '3g'
        option pppddev '/dev/ttyACM0'
        option statedev '/dev/ttyACM3'
        option Manufacturer 'Telit'
        option Product 'HE910'
        option Vendor '1bc7'
        option ProdID '0021'
        option SerialNumber '357164040778055'
        option Rev '12.00.003'

config device 'sim_card'
        option name 'sim_card'
        option present 'Yes'
        option IMSI '302720406815819'
        option operator 'ROGERS'
        option operator_code '302720'
```

Step 5 Confirm that the pppoe module is present on the target.

```
# lsmod | grep "pppoe"
```

Including Exegin Zigbee in the Platform Project

NOTE: To use the **wr-exegin-zigbee-ia** layer, you must select the option when you purchase Wind River Intelligent Device Platform XT.

The following hardware and software are required for the Zigbee stack:

- A Wind River IDP XT 2.0 installation on top of a Wind River Linux 5.0.1 installation with RCPL updates on a supported host.
- A Cross Hill hardware board with Q58 chip.
- A serial connection to a host console.
- Build a platform project and boot your board in the standard way.

  You must include the **wr-exegin-zigbee-ia** layer in the configure command.

  For more information see:

  - *Building Platform Projects for Quark Boards* on page 122
  - *Building Platform Projects for Advantech UTX-3115 Boards* on page 126

```
$ $WIND_LINUX_CONFIGURE --enable-board=intel-quark --enable-kernel=standard \
--enable-rootfs=glibc-idp --enable-addons=wr-idp \
```

```
--with-layer=wr-exegin-zigbee-ia \
--enable-jobs=4 --enable-parallel-pkgbuilds=4
```

Enabling IMA Appraise

Enable Integrity Measurement Architecture (IMA) Appraise by including it in your platform project.

To enable IMA Appraise in IDP XT, you must include it in your Wind River Linux platform project.

Step 1 Configure and build a platform project for your board.

In your **$WIND_LINUX_CONFIGURE** command, include, as a minimum, the following option and layer:

- Option: **--enable-addons=wr-idp**
- Layer: **--with-layer=wr-ima-appraise**

NOTE: The **wr-ima-appraise** layer depends on **wr-srm**; do not use **--with-layer=wr-ima-appraise** in your **$WIND_LINUX_CONFIGURE** command when you use **--without-layer=wr-srm**.

For example:

```
$ $WIND_LINUX_CONFIGURE --enable-board=intel-quark --enable-kernel=standard \
--enable-rootfs=glibc-idp --enable-addons=wr-idp --with-layer=wr-ima-appraise \
--enable-parallel-pkgbuilds=4 --enable-jobs=4
```

For more information, see:

- *Building Platform Projects for Quark Boards* on page 122
- *Building Platform Projects for Advantech UTX-3115 Boards* on page 126

Configuring OpenJDK

Installing OpenJDK

Wind River OpenJDK is an open source implementation of Java Platform SE (Java SE).

Wind River OpenJDK allows you to compile Java code using only free software with your Linux distribution. Wind River OpenJDK capability provides the following resources on the IDP XT target:

JRE (Java Runtime Environment)

An open source Java virtual machine (JVM) that uses OpenJDK as its Java run-time library.

- Build a platform project and boot your board in the standard way.

  You must include at least the following option and template in your configure command:

  Option: **--enable-addons=wr-idp**

Template: **feature/openjdk-bin**

For more information see:

- *Building Platform Projects for Quark Boards* on page 122
- *Building Platform Projects for Advantech UTX-3115 Boards* on page 126

Related Links

Example: Integrating Custom Java Code Into IDP on page 163
This example shows how you can integrate your own Java source code into the IDP XT build environment and execute it on the IDP XT target.

Rebuilding the Java Run-Time Environment from Source on page 166
You can rebuild the Java Runtime Environment (JRE) from source code if the pre-built JRE included on the target by the **openjdk-bin** package does not meet your needs.

Example: Integrating Custom Java Code Into IDP

This example shows how you can integrate your own Java source code into the IDP XT build environment and execute it on the IDP XT target.

There are two ways to build Java applications:

- Using your host machine's Java development environment. For more information, see your host's Java development manual. After you compile your application on your host, you can transfer it to the IDP XT target device and execute it.
- Using the Java development environment provided by the IDP XT build system. This example explains this procedure.

Step 1 Create a Java source code directory on your host.

```
$ cd sourceDir
$ mkdir -p src
```

Step 2 Import **HelloWorld.java** into the source directory.

```
$ cp path-to-Java-Source-Code sourceDir/src/HelloWorld.java
```

For this example, **HelloWorld.java** contains the following code:

```
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

Step 3 Create a **build.xml** file that will build **HelloWorld.jar**.

For this example, **build.xml** contains the following code:

```
<project>

    <target name="clean">
        <delete dir="build"/>
    </target>

    <target name="compile">
        <mkdir dir="build/classes"/>
        <javac srcdir="src" destdir="build/classes"/>
```

```
    </target>

    <target name="jar">
        <mkdir dir="build/jar"/>
        <jar destfile="build/jar/HelloWorld.jar"
             basedir="build/classes">
            <manifest>
                <attribute name="Main-Class" value="HelloWorld"/>
            </manifest>
        </jar>
    </target>

    <target name="run">
        <java jar="build/jar/HelloWorld.jar" fork="true"/>
    </target>

</project>
```

Step 4 Create a tar file of the source.

```
$ tar zcvf HelloWorld.tar.gz sourceDir
```

Step 5 Create a new custom layer.

a) Create and populate a directory for the custom layer.

```
$ mkdir myJavaLayer
$ cd myJavaLayer
$ mkdir {conf,downloads,recipes-core,templates}
$ mkdir -p templates/feature/helloworld
$ mkdir -p recipes-core/helloworld
```

b) Create a **layer.conf** file in the **conf** directory.

For this example, **layer.conf** contains the following code:

```
BBPATH ?= ""
# We have a conf and classes directory, add to BBPATH
BBPATH := "${LAYERDIR}:${BBPATH}"

# We have a packages directory, add to BBFILES
BBFILES := "${BBFILES} ${LAYERDIR}/recipes-*/*/*.bb \
    ${LAYERDIR}/recipes-*/*/*.bbappend \
    ${LAYERDIR}/classes/*.bbclass"

BBFILE_COLLECTIONS += "myjavalayer"
BBFILE_PATTERN_myjavalayer := "^${LAYERDIR}/"
BBFILE_PRIORITY_myjavalayer = "10"

# We have a pre-populated downloads directory, add to PREMIRRORS
PREMIRRORS_append := " \
    git://.*/.* file://${LAYERDIR}/downloads/ \n \
    svn://.*/.* file://${LAYERDIR}/downloads/ \n \
    ftp://.*/.* file://${LAYERDIR}/downloads/ \n \
    http://.*/.* file://${LAYERDIR}/downloads/ \n \
    https://.*/.* file://${LAYERDIR}/downloads/ \n"
```

c) Create a **template.conf** file in the **templates/feature/helloworld** directory.

For this example, **template.conf** contains the following code:

```
IMAGE_INSTALL_append += "helloworld"
```

d) Create a BitBake recipe file **helloworld.bb** in the **recipes-core/helloworld** directory.

For this example, **helloworld.bb** contains the following:

```
DESCRIPTION = "This package contains Hello World sample program for openjdk and ant"
LICENSE = "GPL-2.0"
LIC_FILES_CHKSUM = "file://src/HelloWorld.java;md5=9862605c57636530f72656b1afb51631"
```

```
SRC_URI = "file:///home/wruser/myJavaLayer/downloads/HelloWorld.tar.gz"

S = "${WORKDIR}/HelloWorld"
DEPENDS +="ant-native"

PACKAGES = "${PN}"

FILES_${PN} += "/usr/share/java"
RDEPENDS_${PN} = "openjdk-bin"

do_compile() {
    ant clean
    ant compile
    ant jar
    ant run
}

JARFILENAME="HelloWorld.jar"

do_install() {
    install -d ${D}/usr/share/java
    install -m 0755 ${S}/build/jar/HelloWorld.jar ${D}/usr/share/java
}

SRC_URI[md5sum] = "65ff48dc49315629518b3547c9ea5214"
SRC_URI[sha256sum] =
"21ce3477be703f42ade95c558731a99c509b9759849bea9e1418f2ad877e521d"
```

NOTE: Replace the **md5sum** and **sha256sum** values with values calculated using the **md5sum** and **sha256sum** utilities on your Linux host.

The **SRC_URI[md5sum]** and **SRC_URI[sha256sum]** lines must be entered on a single line. If you are copying the file from the PDF version of this document, you must correct the file contents manually; it is not possible to show these items on a single line in PDF.

For more information on how to create a recipe file, see the *Wind River Linux User's Guide, 5.0.1: Working with Layers, Recipes, and Templates*.

e) Verify the directory structure.

```
$ tree
.
+-- conf
|   +-- layer.conf
+-- downloads
+-- recipes-core
    +-- helloword
        +-- helloworld.bb
+-- templates
    +-- feature
        +-- helloworld
            +-- template.conf
```

Step 6 Copy the tar file into the **downloads** directory of the custom layer directory.

```
$ cp HelloWorld.tar.gz \
myJavaLayer/downloads/
```

Step 7 Add the path to *myJavaLayer* to ***projDir*/bitbake_build/conf/bblayers.conf**.

Step 8 Build the Java package from your platform project directory.

```
$ cd projDir
$ make -C build helloworld
```

The built package is located in the following directory:

***projDir*/build/helloworld-1.0-r0/image/usr/share/java/HelloWorld.jar**

Step 9 Build the entire project including the file system and boot the target.

Use the standard method for your board and include the following additional layers and features in your configure command:

- Layers:
  - ***myJavaLayer***
- Templates:
  - **feature/openjdk-bin**
  - **feature/helloworld**

For more information see:

- *Building Platform Projects for Quark Boards* on page 122
- *Building Platform Projects for Advantech UTX-3115 Boards* on page 126

Step 10 Execute the HelloWorld program on the IDP XT target.

```
# java -jar /usr/share/java/HelloWorld.jar
Hello, World
```

Related Links

Installing OpenJDK on page 162
Wind River OpenJDK is an open source implementation of Java Platform SE (Java SE).

Rebuilding the Java Run-Time Environment from Source on page 166
You can rebuild the Java Runtime Environment (JRE) from source code if the pre-built JRE included on the target by the **openjdk-bin** package does not meet your needs.

Rebuilding the Java Run-Time Environment from Source

You can rebuild the Java Runtime Environment (JRE) from source code if the pre-built JRE included on the target by the **openjdk-bin** package does not meet your needs.

NOTE: When you build JRE from source code, you must have the **libstdc++.a** library and Linux kernel version 3.0 or later.

If the library is not on your development host, install it manually. For example, to install **libstdc++.a** on Fedora use the following command:

```
$ yum install libstdc++-static
```

Step 1 Build your platform project with the **openjdk-bin** feature.

Step 2 Add the rebuild option to the **local.conf** file.

```
$ echo 'REBUILD_OPENJDK = "yes"' >> local.conf
```

Step 3 Build the rootfs.

```
$ make fs
```

The command builds OpenJDK and includes it in the rootfs tar file automatically.

Related Links

Installing OpenJDK on page 162
Wind River OpenJDK is an open source implementation of Java Platform SE (Java SE).

Example: Integrating Custom Java Code Into IDP on page 163
This example shows how you can integrate your own Java source code into the IDP XT build environment and execute it on the IDP XT target.

OSGi Development Workflow

Use this workflow to setup your development environment and create an OSGi-enabled platform project image.

OSGi development requires some additional configuration for use, and follows a different procedure than that required for the development of a typical Wind River Linux platform project. OSGi development takes advantage of the Eclipse integrated development environment to create home automation applications. Before you begin developing, complete the following steps to prepare your development environment and create an OSGi-enabled platform project image.

1. Install the ProSyst Smart Home SDK.
2. Install and configure the ProSyst Smart Home Eclipse plug-ins in Eclipse.
3. Create your OSGi platform image. The platform image holds the files that comprise your bundle.
4. Export your OSGi platform image. Once the platform image is created, you must export it to a format suitable for booting on your target platform.
5. Deploy your image to the target. To test your bundle on a target, you must deploy it.

Related Links

Installing the ProSyst Smart Home SDK on page 167
To use the ProSyst Smart Home SDK to develop OSGi-enabled target platforms, you must install it.

Enabling Eclipse for ProSyst Smart Home Development on page 169
Enable Eclipse to simplify OSGi development with the mBS Smart Home SDK.

Creating an OSGi Platform Image on page 170
Use Eclipse to create an OSGi plug-in project.

Exporting an OSGi Platform Image on page 171
Export your platform image for deployment.

Deploying an OSGi Platform Image on a Target on page 172
Deploy the platform image you previously created. Two deployment methods are available: the traditional method and the rootfs integration method.

Installing the ProSyst Smart Home SDK

To use the ProSyst Smart Home SDK to develop OSGi-enabled target platforms, you must install it.

The Intelligent Device Platform uses the ProSyst Smart Home SDK to develop OSGi-enabled applications. In addition, this SDK supplies the Smart Home Eclipse plug-in to facilitate OSGi development with Eclipse.

Step 1 Obtain the ProSyst Smart Home SDK, and the Wind River Extension packages:

- **ProSyst_mBS_SH_SDK_7.5_Commercial.zip**—ProSyst mBS SDK 7.5
- **ProSyst_mBS_SH_SDK_7.5_Board_Extension_Atom.zip** or **ProSyst_mBS_SH_SDK_7.5_Board_Extension_Quark.zip**

Step 2 Extract and install **ProSyst_mBS_SH_SDK_7.5_Commercial.zip** to your host system.

a) Navigate to the location of the SDK archives, for example:

```
$ cd <projDir>/layers/wr-idp/wr-prosyst-mbs-smarthome-sdk-ia/downloads
```

b) Extract the SDK archives.

```
$ unzip ProSyst_mBS_SH_SDK_7.5_Commercial.zip
```

c) Give the installer script executable permissions.

```
$ chmod a+x startinstall
```

d) Locate the product serial number.

The product serial number is located in the extracted SDK archive in **sn.txt**.

e) Run the installation script to install the SDK.

```
$ ./startinstall
```

f) Follow the ProSyst setup tool's instructions to complete the SDK setup.

The SDK will be installed in the folder **mbssh-sdk**.

Step 3 Extract and install the runtime extensions to your host system.

BSP	ZIP File Name
intel_atom_baytrail	**ProSyst_mBS_SH_SDK_7.5_Board_Extension_Atom.zip**
intel_quark	**ProSyst_mBS_SH_SDK_7.5_Board_Extension_Quark.zip**

a) Navigate to the location of the archives, for example:

```
$ cd <projDir>/layers/wr-idp/wr-prosyst-mbs-smarthome-sdk-ia/downloads
```

b) Extract the Wind River extension:

This example is for an Advantech UTX-3115 board:

```
$ unzip ProSyst_mBS_SH_SDK_7.5_Board_Extension_Quark.zip
```

A directory named **runtime** is created.

c) Copy the extracted files to the **runtime** folder.

```
$ cp -rf runtime/* mbssh-sdk/runtime
```

The **mbssh-sdk** directory was created in 2 on page 168.

Postrequisites

Once the SDKs are installed, see *Enabling Eclipse for ProSyst Smart Home Development* on page 169, to complete the process of setting up your development environment.

Related Links

OSGi Development Workflow on page 167

Use this workflow to setup your development environment and create an OSGi-enabled platform project image.

Enabling Eclipse for ProSyst Smart Home Development on page 169
Enable Eclipse to simplify OSGi development with the mBS Smart Home SDK.

Enabling Eclipse for ProSyst Smart Home Development

Enable Eclipse to simplify OSGi development with the mBS Smart Home SDK.

This manual uses Eclipse, and more specifically, the ProSyst Smart Home Eclipse plug-ins in procedures for creating OSGi-enabled target platforms with Wind River Linux.

Step 1 Download and install Eclipse 4.2 (Juno) from the *Eclipse website*.

Step 2 Install the Eclipse plug-ins as described in the mBS SDK 7.5.0 documentation, **Getting Started** > **Eclipse Plugins** at *http://dz.prosyst.com/pdoc/*.

For more information, see the *ProSyst website*.

Step 3 Once the Eclipse and Prosyst Smart Home Eclipse plug-ins are installed, start Eclipse.

Step 4 Set the target platform.

If you clicked **Switch** when prompted in the previous step, you can skip this step.

a) In the **Window** > **Preferences** window, select **Plug-in Development** > **Target Platform**.
b) Select **mBS Smart Home SDK (Active)**, and click **OK**.

Postrequisites

Once your image creation is complete, you can export your image for deployment and testing.
See *Exporting an OSGi Platform Image* on page 171

Related Links

OSGi Development Workflow on page 167
Use this workflow to setup your development environment and create an OSGi-enabled platform project image.

Installing the ProSyst Smart Home SDK on page 167
To use the ProSyst Smart Home SDK to develop OSGi-enabled target platforms, you must install it.

Creating an OSGi Platform Image on page 170

Use Eclipse to create an OSGi plug-in project.

Creating an OSGi Platform Image

Use Eclipse to create an OSGi plug-in project.

NOTE: This example creates a project that launches OSGi with the script, **JDK/JRE: Java(TM) 2 and higher**. However, if you are using your own JVM, you can select other startup scripts in the Startup Scripts pane. For more information, see:

Help > **Help Contents** > **Image Builder** > **Image Builder** > **Tasks** > **Configure Image Content**.

Step 1 Create a platform project.

a) In Eclipse, select **File > New > Other.**
b) Select **Plug-in Development > Plug-in Project**, then click **Next**.
c) Enter a name for the project and click **Next**.

This example assumes you name the project **wrdemo**.

The Content dialog opens.

d) In the Content dialog, click **Finish**.

Step 2 Right-click on the empty project (**wrdemo**) and select **New** > **Other** from the context menu.

The Select a Wizard dialog opens.

Step 3 In the Select a Wizard dialog, select **OSGi** > **Image Description** and click **Next**.

The Image Description dialog opens.

Step 4 Complete the Image Description dialog.

a) Select the **Workspace image** radio button.
b) Select **Wind River Intelligent Device Platform** from the dropdown menu.
c) Enter a name in the **File Name** field.

These examples assume you name the file **mydemo**.

d) Click **Finish**.

Postrequisites

Once your image creation is complete, you can export your image for deployment and testing. See *Exporting an OSGi Platform Image* on page 171.

Related Links

OSGi Development Workflow on page 167
Use this workflow to setup your development environment and create an OSGi-enabled platform project image.

Enabling Eclipse for ProSyst Smart Home Development on page 169
Enable Eclipse to simplify OSGi development with the mBS Smart Home SDK.

Exporting an OSGi Platform Image on page 171

Export your platform image for deployment.

Exporting an OSGi Platform Image

Export your platform image for deployment.

Once you create an OSGi platform image (see *Creating an OSGi Platform Image* on page 170), you can use this procedure to export the image for deployment to the target.

Step 1 In the platform project configuration tab for **mydemo**, in the Platform Settings section, verify that the following items are selected:

Platform column:	For Bay Trail boards: IDP_XT_2.0.2/Atom For Quark boards: IDP_XT_2.0.2/Quark
Startup Scripts column:	**JDK/JRE: Java(TM) 2 and higher**

Step 2 In the platform project **Configuration** tab, click **Add** just to the right of the Bundles section. The Add Bundles window appears.

Step 3 Select any additional bundles you choose, then click **OK**.

Step 4 At the top right-hand side of the **Configuration** tab, click the **Export Image** link under the **Testing** tab to launch the OSGi image builder export wizard.

Step 5 In the Source section, select **Workspace image**, then verify that your project image (**mydemo**) is selected.

Step 6 Select the **Create image in directory** option.

Step 7 In the **Destination** field, browse to a folder where you want your project image to reside.

Step 8 Click **Finish** to create the OSGi image. This could take some time.

Step 9 In a file viewer/explorer, navigate to the destination location created in the previous step. Verify that two new subfolders have been created: **mbsa** and **osgi**.

Postrequisites

Once you have successfully exported your platform image, you can deploy it for testing. See *Deploying an OSGi Platform Image on a Target* on page 172.

Related Links

OSGi Development Workflow on page 167
Use this workflow to setup your development environment and create an OSGi-enabled platform project image.

Creating an OSGi Platform Image on page 170
Use Eclipse to create an OSGi plug-in project.

Deploying an OSGi Platform Image on a Target on page 172

Deploy the platform image you previously created. Two deployment methods are available: the traditional method and the rootfs integration method.

Deploying an OSGi Platform Image on a Target

Deploy the platform image you previously created. Two deployment methods are available: the traditional method and the rootfs integration method.

The rootfs Integration Method

For details on using this method, see the *Run by integrating into rootfs* section in the **README** file located in the following directory:

***projDir*/layers/wr-idp/wr-prosyst-mbs-smarthome-sdk-ia/templates/default**

The Traditional Method

Once you create and export an OSGi platform image (see *Exporting an OSGi Platform Image* on page 171), you can use this procedure to deploy it to the target.

NOTE: For additional details on exporting images to Intel Architecture boards, see the **README** file located at:

***projDir*/layers/wr-idp/wr-prosyst-mbs-smarthome-sdk-ia/templates/default**

Step 1 Build a platform project and boot your board in the standard way.

For more information see:

- *Building Platform Projects for Quark Boards* on page 122
- *Building Platform Projects for Advantech UTX-3115 Boards* on page 126

Step 2 Navigate to the **/opt** folder and create a **prosyst_osgi** directory.

```
# cd /opt
# mkdir prosyst_osgi
```

Step 3 Copy the two OSGi image project folders **mbsa** and **osgi** to the **/opt/prosyst_osgi** folder on the target.

You can use SSH/SCP to transferg the data or, if target is physically accessible, you can copy the two folders to a USB drive, attach it to the target, and copy the folders.

Step 4 Start the runtime.

NOTE: The runtime requires that the board's system time is set to the current time.

```
# cd /opt/prosyst_osgi/mbsa/bin/
# ./mbsa_start
```

Once the OSGi image completes its startup process, it displays the following message:

```
[mBSA] OSGi framework is started successfully
```

Step 5 Test OSGi by accessing the OSGi configuration pages.

Open the following URL in a Web browser on your host:

http://*targetIpAddr*/system/console

Related Links

OSGi Development Workflow on page 167
Use this workflow to setup your development environment and create an OSGi-enabled platform project image.

Exporting an OSGi Platform Image on page 171
Export your platform image for deployment.

Installing Sqlite3

Sqlite3 is a lightweight database used primarily for embedded systems.

Step 1 Build a platform project and boot your board in the standard way.

You must include at least the **--enable-addons=wr-idp** option in your configure command

For more information see:

- *Building Platform Projects for Quark Boards* on page 122
- *Building Platform Projects for Advantech UTX-3115 Boards* on page 126

Step 2 Start Sqlite3.

You can use the basic Sqlite3 commands from the command line with user-space files. Note that all operations are performed by the root user.

```
# sqlite3 <test.db>
sqlite>
```

Installing MQTT and Lua

MQTT consists of the open source server Mosquitto and a client that includes utilities for publishing and subscribing to MQTT topics. The client uses the Lua language.

- Build a platform project and boot your board in the standard way.

  You must including at least the following options in your configure command:

  - **--enable-addons=wr-idp**
  - **--with-template=feature/mqtt**

  For more information see:

  - *Building Platform Projects for Quark Boards* on page 122
  - *Building Platform Projects for Advantech UTX-3115 Boards* on page 126

Configuring Encrypted Storage

Encrypted Storage Prerequisites

Using encrypted storage requires TPM chips, several Linux kernel configuration options, as well as several packages from Linux and Wind River Intelligent Device Platform XT.

Requirements for using encrypted storage:

- A TPM chip version 1.2 or later.
- The following Linux kernel configuration options enabled in the platform project:

  CONFIG_DM_CRYPT
  CONFIG_BLK_DEV_MD
- The following software packages are required on the device:

  lvm2-2.02.95
  cryptsetup-1.6.1
  trousers-devel
- Familiarity with how TPM works. For more information see:
  - *Performing a Trusted Boot*
  - the *Wind River Intelligent Device Platform XT Security Guide*

Related Links

Enabling Encrypted Storage on page 174
Enabling encrypted storage requires building a platform project, creating a sealed key, and setting up a **dm-crypt** partition.

Enabling Encrypted Storage

Enabling encrypted storage requires building a platform project, creating a sealed key, and setting up a **dm-crypt** partition.

NOTE: The **mkfs.ext3** utility used to format the USB Flash drive is available with **glibc-idp**; it is not included in **glibc-idp-small**.

Step 1 Configure and build a platform project.

For more information, see:

- *Building Platform Projects for Quark Boards* on page 122
- *Building Platform Projects for Advantech UTX-3115 Boards* on page 126

The following configure command uses the intel-atom-baytrail BSP as an example:

```
$ $WIND_LINUX_CONFIGURE --enable-rootfs=glibc-idp --enable-addons=wr-idp \
--enable-kernel=standard --enable-board=intel-atom-baytrail \
--enable-parallel-pkgbuilds=4 --enable-jobs=4
```

When you finish, you will have done the following:

1. Configured and built the platform project.
2. Burned the kernel and rootfs to a USB drive.

3. Booted the device from the USB drive.

Step 2 Enable TPM in the BIOS settings.

a) Reboot the device and press **DEL** or **F7**.

b) Access the BIOS settings; select **Enter setup**.

c) Clear TPM.

Advanced > **Trusted Computing** > **TPM Configuration** > **Pending operation** > **TPM clear**

d) In the menu, enable TPM.

Advanced > **Trusted Computing** > **TPM Configuration** > **Security Device Support [Enabled]**
TPM State [Enabled]

e) Confirm the TPM status.

```
Current TPM Status Information
TPM Enabled Status: [Enabled]
TPM Active Status: [Activated]
```

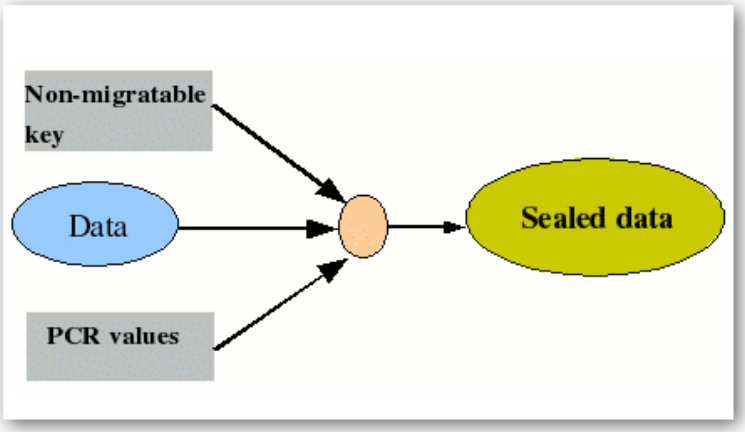
NOTE: This example was generated by BIOS version EBC MBC MF01X011, with a build date and time of 02/25/2014 16:09:55. Different BIOS versions might generate slightly different text.

f) Save and exit the BIOS settings; press **F4**.

Step 3 Log onto the device.

Step 4 Create a key.

```
# dd if=/dev/urandom of=/home/temp_plain_key bs=1 count=32
                    # cat /home/temp_plain_key | tpm_sealdata -z -p 4 -p 5 -p 8 -o /
home/sealed_key
```



NOTE: If you see an error message, perform the following steps:

1. Run the process status command to confirm that the **tcsd** deamon is running.

   # **ps aux** | **grep** `tcsd`
2. Take the ownership of the TPM chip.

   # **tpm takeownership -yz**
3. Regenerate the sealed key.
4. For Cross Hill targets only, run the following command and reboot the target.

   tpm_clear -z

Related Links

Encrypted Storage Prerequisites on page 174
Using encrypted storage requires TPM chips, several Linux kernel configuration options, as well as several packages from Linux and Wind River Intelligent Device Platform XT.

Setting Up the dm-crypt Partition with a Loop Device

If your device has no interface for an extended storage device, you can use a loopback device as a secret directory for **dm-crypt**.

Step 1 Unlock the memory block limit.

```
# ulimit -Hl unlimited
# ulimit -Sl unlimited
```

The arguments have the following meanings:

"**l**" in **Hl** or **Sl**	the maximum size that may be locked into memory
"**H**" in **Hl**	the hardware limit
"**S**" in **Sl**	the software limit
-Hl	cancel the hardware limit
-Sl	cancel the software limit

NOTE: You must modify the **Hl** value before the **Sl** value because the software limit cannot exceed the hardware limit.

Step 2 Create an unused loopback device of your choice and associate it with a secret directory.

```
# dev=`losetup -f`
# echo $dev
/dev/loop0
# dd if=/dev/urandom of=/home/secret_dir bs=1M count=10
# losetup $dev /home/secret_dir
```

Step 3 Format the loopback device as a LUKS partition.

```
# tpm_unsealdata -z -i /home/sealed_key | cryptsetup luksFormat --key-file=- $dev --
debug
```

Step 4 Mount the LUKS-encrypted device to a secret mapper device (**/dev/mapper/secret-loop**).

```
# tpm_unsealdata -z -i /home/sealed_key | cryptsetup luksOpen --key-file=- $dev secret-
loop --debug
```

A device named **/dev/mapper/secret-loop** is created.

Step 5 Confirm that the device was created.

```
# ls -1 /dev/mapper
total 0
crw-------  1  root  root   10, 236  Jun 12 15:54 control
brw-------  1  root  root  253,   0  Jun 12 16:18 secret-loop
```

Step 6 Format the mapped device as a normal block device.

```
# mkfs.ext3 -I 128 /dev/mapper/secret-loop
```

Step 7 Mount the secret device in any location.

```
# mkdir /home/mysecretdir
# mount /dev/mapper/secret-loop /home/mysecretdir
```

Related Links

Testing Encrypted Storage with a Loop Device on page 177
Test your encrypted storage installation by using an unmatched **dm-crypt** key on a loopback device.

Testing Encrypted Storage with a Loop Device

Test your encrypted storage installation by using an unmatched **dm-crypt** key on a loopback device.

Step 1 Create a file with dummy data.

```
# cat > /home/mysecretdir/secret_text_file.txt << EOF
this is a secret text
EOF
```

Step 2 Read the contents of the file.

```
# cat /home/mysecretdir/secret_text_file.txt
this is a secret text
```

Step 3 Reboot the target and wait for the GRUB menu to appear.

```
# reboot
```

Step 4 Generate the other key.

```
# dd if=/dev/urandom of=/home/otherkey bs=1 count=32
# cat /home/otherkey | tpm_sealdata -z -p 4 -p 5 -p 8 -o /home/sealed_otherkey
```

Step 5 Attempt to mount the LUKS-encrypted device using the other key.

a) Unlock the memory block limit.

```
# ulimit -Hl unlimited
# ulimit -Sl unlimited
```

b) Create an unused loopback device of your choice and associate it with a secret directory.

```
# dev=`losetup -f`
# echo $dev
/dev/loop0
# losetup $dev /home/secret_dir
```

c) Mount the LUKS-encrypted device to a secret mapper device (**/dev/mapper/secret-loop**).

```
# tpm_unsealdata -z -i /home/sealed_otherkey | cryptsetup luksOpen --key-file=-
$dev secret --debug
```

The mount command fails with the following message:

```
Command failed with code 1: No key available with this passphrase.
```

The failure is expected because the key (**/home/sealed_otherkey**) does not match the key (**/home/sealed_key**) used to generate the encrypted device.

Step 6 Reboot the system.

Step 7 Attempt to mount the LUKS-encrypted device again.

a) Unlock the memory block limit.

```
# ulimit -Hl unlimited
# ulimit -Sl unlimited
```

b) Create an unused loopback device of your choice and associate it with a secret directory.

```
# dev=`losetup -f`
# echo $dev
/dev/loop0
# losetup $dev /home/secret_dir
```

c) Mount the LUKS-encrypted device to a secret mapper device (**/dev/mapper/secret-loop**).

```
# tpm_unsealdata -z -i /home/sealed_key | cryptsetup luksOpen --key-file=- $dev
secret-loop --debug
```

d) Mount the secret device in any location.

```
# mount /dev/mapper/secret-loop /home/mysecretdir
```

The device mounts successfully.

Step 8 Confirm the mount by reading **secret_text_file.txt**.

```
# cat /home/mysecretdir/secret_text_file.txt
this is a secret text
```

You can access and read **secret_text_file.txt** because the keys match.

Related Links

Setting Up the dm-crypt Partition with a Loop Device on page 176
If your device has no interface for an extended storage device, you can use a loopback device as a secret directory for **dm-crypt**.

Setting Up the dm-crypt Partition with a USB Key

If your device has an interface for an extended storage device, you can use a real storage device, for example, a USB drive, as a secret directory for **dm-crypt**.

Step 1 Unlock the memory block limit.

```
# ulimit -Hl unlimited
# ulimit -Sl unlimited
```

The arguments have the following meanings:

"**l**" in **Hl** or **Sl**	the maximum size that may be locked into memory
"**H**" in **Hl**	the hardware limit
"**S**" in **Sl**	the software limit
-Hl	cancel the hardware limit
-Sl	cancel the software limit

NOTE: You must modify the **Hl** value before the **Sl** value because the software limit cannot exceed the hardware limit.

Step 2 Find the U-disk.

```
# fdisk -1
Device     Boot Start    End      Blocks  Id System
/dev/sdc1       62       15225897 7612918 83 Linux
```

/dev/sdc1 is the USB key device.

Step 3 Format device **/dev/sdc1** as a LUKS partition.

```
# umount /dev/sdc1
# tpm_unsealdata -z -i /home/sealed_key | cryptsetup luksFormat --key-file=- /dev/sdc1
--debug
```

Step 4 Mount device **/dev/sdc1** to a secret mapper device (**/dev/mapper/secret-usb**).

```
# tpm_unsealdata -z -i /home/sealed_key | cryptsetup luksOpen --key-file=- /dev/sdc1
secret-usb --debug
```

A device named **/dev/mapper/secret-usb** is created.

Step 5 Confirm that the device was created.

```
# ls -1 /dev/mapper
total 0
crw-------  1  root  root   10, 236  Jun 12 15:54 control
brw-------  1  root  root  253,   0  Jun 12 16:20 secret-usb
```

Step 6 Format the mapped device as a normal block device.

```
# mkfs.ext3 -I 128 /dev/mapper/secret-usb
```

Step 7 Mount the secret device in any location.

```
# mount /dev/mapper/secret-usb /home/mysecretdir
```

Related Links

Testing Encrypted Storage with a USB Key on page 180
Test your encrypted storage installation by using an unmatched **dm-crypt** key on a USB drive.

Testing Encrypted Storage with a USB Key

Test your encrypted storage installation by using an unmatched **dm-crypt** key on a USB drive.

Step 1 Create a file with dummy data.

```
# cat > /home/mysecretdir/secret_text_file.txt << EOF
this is a secret text
EOF
```

Step 2 Read the contents of the file.

```
# cat /home/mysecretdir/secret_text_file.txt
this is a secret text
```

Step 3 Reboot the target.

```
# reboot
```

Step 4 Generate the other key.

```
# dd if=/dev/urandom of=/home/otherkey bs=1 count=32
# cat /home/otherkey | tpm_sealdata -z -p 4 -p 5 -p 8 -o /home/sealed_otherkey
```

Step 5 Attempt to mount the LUKS-encrypted device.

a) Unlock the memory block limit.

```
# ulimit -Hl unlimited
# ulimit -Sl unlimited
```

b) Mount the LUKS-encrypted device to a secret mapper device (**/dev/mapper/secret-usb**).

```
# tpm_unsealdata -z -i /home/sealed_otherkey | cryptsetup luksOpen --key-
file=- /dev/sdc1 secret-usb --debug
```

The mount command fails with the following message:

```
Command failed with code 1: No key available with this passphrase.
```

The failure is expected because the key (**/home/sealed_otherkey**) does not match the key (**/home/sealed_key**) used to generate the encrypted device.

You will also receive a failure if you remove the USB drive and plug it into another machine, for example, a PC running Ubuntu 10.04. The drive cannot be mounted and a warning similar to the following is generated:

Step 6 Reboot the system.

Step 7 Attempt to mount the LUKS-encrypted device again.

a) Unlock the memory block limit.

```
# ulimit -Hl unlimited
# ulimit -Sl unlimited
```

b) Mount the LUKS-encrypted device to a secret mapper device (**/dev/mapper/secret-usb**).

```
# tpm_unsealdata -z -i /home/sealed_key | cryptsetup luksOpen --key-file=- /dev/
sdc1 secret-usb --debug
```

c) Mount the secret device in any location.

```
# mount /dev/mapper/secret-usb /home/mysecretdir
```

The device mounts successfully.

Step 8 Confirm the mount by reading **secret_text_file.txt**.

```
# cat /home/mysecretdir/secret_text_file.txt
this is a secret text
```

You can access and read **secret_text_file.txt** because the keys match.

Related Links

Setting Up the dm-crypt Partition with a USB Key on page 179

If your device has an interface for an extended storage device, you can use a real storage device, for example, a USB drive, as a secret directory for **dm-crypt**.

Installing OneAgent TR-069

Create and build a TR-069-enabled device by including the **wr-wks-oneagent-tr069** layer in you platform project configure command.

Before you can use the OneAgent TR-069 capabilities explained in *OneAgent TR-069 Agent* on page 221, you must configure and build a platform project, and add the **OneAgent_TR** package to it.

Step 1 Build a platform project and boot your board in the standard way.

For more information see:

- *Building Platform Projects for Quark Boards* on page 122
- *Building Platform Projects for Advantech UTX-3115 Boards* on page 126

```
$ $WIND_LINUX_CONFIGURE --enable-rootfs=glibc-idp --enable-addons=wr-idp \
--enable-kernel=standard --enable-board=<bspDir> --with-layer=wr-wks-oneagent-tr069 \
--enable-jobs=4 --enable-parallel-pkgbuilds=4
```

Step 2 Unpack the **oneagent-tr** package.

```
$ make -C build oneagent-tr.distclean
$ make -C build oneagent-tr.unpack
```

The contents of the package are extracted to the following folder:

***projDir*/build/oneagent-tr-5.5.17a1-r0/oneagent-tr-5.5.17a1**

Step 3 Build the package:

```
$ make -C build oneagent-tr
```

Step 4 Optionally, enter the following command to build the project and include the OneAgent TR-069 binaries on the target system:

```
$ make fs
```

For more information on how to install, configure, run, and debug TR-069 see the **README** file and the *OneAgent TR 5.5 Integration Guide* located in the following directory:

***projDir*/layers/wr-idp/wr-wks-oneagent-tr069/templates/default**

Installing OMA-DM

To use the OMA DM agent and OMA DM management objects, you must include them in your Wind River Linux platform project.

- Build a platform project and boot your board in the standard way.

  Include the following layer:

  wr-wks-oneagent-oma-dm-ia

  The following binaries and configuration files are installed on the target file system:

/usr/sbin/oma
/usr/sbin/oma_bin
/etc/oma/oma.xml
/etc/oma/*

Most of the configuration files for the OMA-DM client are placed in **/etc/oma/**.

You can view additional OMA-DM debug messages in the **/var/log/oma.log** file.

The following example uses the **intel-quark** BSP:

```
$ <installDir>/wrlinux-5/wrlinux/configure --enable-board=intel-quark \
--with-layer=wr-wks-oneagent-oma-dm-ia \
--enable-kernel=standard --enable-addons=wr-idp --enable-rootfs=glibc-idp
```

- Configure the OMA log.

  Reducing the level from **DEBUG** to **WARNING** prevents the log from growing too large.

  The OMA DM default log configuration in **/etc/oma/log.conf** is:

  filename = /var/log/oma.log
  rotate = yes
  level = DEBUG
  backup = 50M
  mode = BOTH
  limit = 5M

  Change the **level=DEBUG** to **level=WARNING**.
- (Optional) Configure the polling interval.

  You can increase the polling interval of the **oma_bin** process to reduce CPU usage. Modify the value of **PollingInterval** in the following configuration file:

  /etc/oma/oma.xml

  The default value of **PollingInterval** is **value=1**; increasing it to **1800** (for example) will noticeably reduce CPU usage.

  The default entry for **PollingInterval** is:

```
<node name='PollingInterval' dynamic='0' accesstype='63' format='int' store='0'
prop_size='1' title='polling attempts' type='text/plain' value='1'>
```

Configuring grsecurity

Configuring grsecurity in Platform Projects

The grsecurity feature **feature/grsec_std** is automatically included when you configure your platform projects with the SRM layer.

You do not need to specify **--with-template=feature/grsec_std** in your configuration command. However, you can selectively exclude the grsecurity feature from your platform project.

- Exclude grsecurity by modifying your platform project configure command.

  Add the following additional option to your configure command:

  --with-template=feature/non_grsec

Related Links

Configuring PaX in the Kernel on page 184
Modify the kernel configuration to enable the grsecurity PaX by modifying the configuration file or make a temporary modification on the **make** command line.

Configuring PaX in the Kernel

Modify the kernel configuration to enable the grsecurity PaX by modifying the configuration file or make a temporary modification on the **make** command line.

The configuration file is located at:

***projDir*/layers/wr-idp/wr-srm/templates/feature/grsec_std/meta/recipes-kernel/grsecurity/files-*bspName*/grsec_std.cfg**

- Include Pax in the kernel.

Options	Description

Modify the configuration file and build.	For example, to disable the **CONFIG_GRKERNSEC_IO** option, modify **grsec_std.cfg**:
	`-CONFIG_GRKERNSEC_IO=y`
	`+# CONFIG_GRKERNSEC_IO is not set`
	Then build the kernel:
	`$ make -C build linux-windriver.distclean && make \`
	`-C build linux-windriver`

Modify the command line:	`$ make -C build linux-windriver.menuconfig && make \` `-C build linux-windriver.rebuild`

Related Links

Configuring grsecurity in Platform Projects on page 183
The grsecurity feature **feature/grsec_std** is automatically included when you configure your platform projects with the SRM layer.

Installing Wind River OPC

In order to develop applications that use OPC, you must configure OPC in your platform project.

- Build a platform project and boot your board in the standard way.

  You must include at least the following option and template in your configure command:

  Option: **--enable-addons=wr-idp**
  Template: **feature/opc**

  For more information see:

  - *Building Platform Projects for Quark Boards* on page 122

- *Building Platform Projects for Advantech UTX-3115 Boards* on page 126

```
$ $WIND_LINUX_CONFIGURE --enable-addons=wr-idp --enable-board=intel-quark \
--enable-kernel=standard --enable-rootfs=glibc-idp \
--with-template=feature/opc
```

21

Customizing the Webif Feature

About Customizing Webif

The device owner's specification may require that you customize Webif on the device during development.

These topics describe the following:

- How to add a new page to the default Webif implementation.
- Webif page structure and page functions.
- A working example.

NOTE: The following steps can only be executed successfully on a non-SRM enabled image deployed on the target since arbitrary modifications are not permitted on a SRM enabled image. Alternatively, you can turn off IMA appraisal on SRM enabled images by adding the **ima_appraise=off** parameter to the kernel boot command line.

Related Links

Adding a New Webif Page on page 188
Use this procedure to add or modify your Webif interface configuration pages

Webif Configuration Page Structure on page 189
You can add tabs to the Webif interface using a standard page structure.

Webif Configuration Page Functions on page 190

This reference lists the Webif functions used to configure pages and gives a skeleton prototype.

Adding a New Webif Page

Use this procedure to add or modify your Webif interface configuration pages

Before you begin, see the following topics for more information:

Webif Configuration Page Structure on page 189
Webif Configuration Page Functions on page 190

Step 1 On the IDP XT target, go to the Webif directory..

```
$ cd /www/cgi-bin/webif
```

Step 2 Add a line to the **/www/cgi-bin/webif/.categories** file.

```
##WEBIF:category:NEWCATEGORY
```

Step 3 Create a Webif script named **network-lan.sh** in the **/www/cgi-bin/webif** directory.

This example is based in the example given in *Example: Working Network Webif Page* on page 191. Copy and paste the example script into the file.

Step 4 Make the script executable.

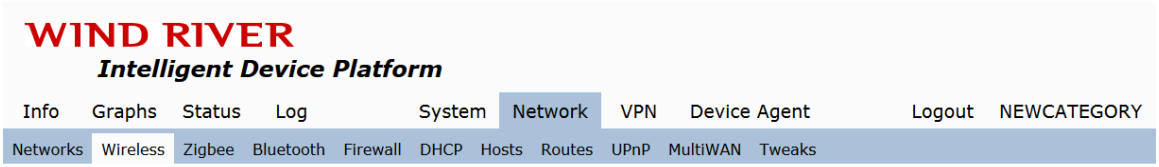
```
$ chmod 755 network-lan.sh
```

Step 5 Flush the Webif cache directory.

```
$ rm -fr /tmp/.webcache
```

Step 6 Return to your Web browser and refresh the Webif page.

You should see the new category called NEWCATEGORY at the top Webif the page.



Related Links

About Customizing Webif on page 187
The device owner's specification may require that you customize Webif on the device during development.

Webif Configuration Page Structure on page 189
You can add tabs to the Webif interface using a standard page structure.

Webif Configuration Page Functions on page 190

This reference lists the Webif functions used to configure pages and gives a skeleton prototype.

Webif Configuration Page Structure

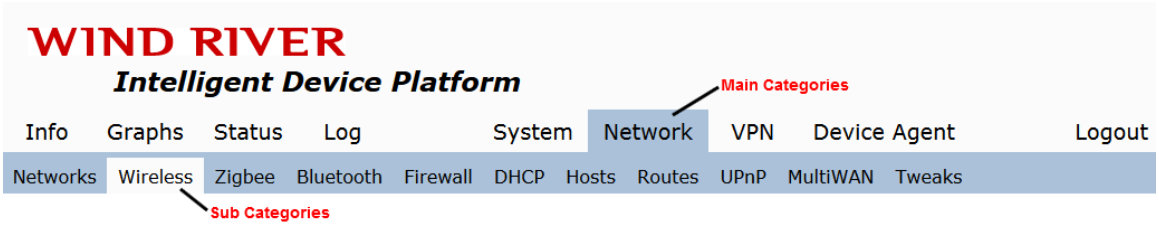
You can add tabs to the Webif interface using a standard page structure.

The Webif page is comprised of the following modifiable sections.

- Main Category — this section defines the main, top-level list of menu items. The list items are defined in the **/www/cgi-bin/webif/.categories** file. For an example of this file, see below.
- Subcategory — this section displays relevant to the main category. This section generates automatically depending on the script files located in the **/www/cgi-bin/webif** folder. Each script that displays a page on Webif must end with the following lines:

```
<!--
##WEBIF:name:${MAINCATEGORY}:${SORT_NUMBER}:${SUBCATEGORY}
-->
```

- Sort Number — The *SORT_NUMBER* variable is an integer that defines the position order of the subcategory menu items. A page with a lower sort order number displays in front of a page with a higher number. This number does not display on the page for a user to see. The following image depicts the main category and subcategories:



NOTE: If your Webif interface is missing main categories like VPN or subcategories like Zigbee, it is probably because the appropriate template or feature was not included when you configured your project.

The main categories that display on the Webif page are listed in order of appearance, with the top-most category name displaying at the left-most (first) position on the page. For example:

```
##WEBIF:category:Info
##WEBIF:category:Graphs
##WEBIF:category:Status
##WEBIF:category:Log
##WEBIF:category:-
##WEBIF:category:System
##WEBIF:category:Network
##WEBIF:category:VPN
##WEBIF:category:Device Agent
##WEBIF:category:-
##WEBIF:category:Logout
```

Related Links

About Customizing Webif on page 187
The device owner's specification may require that you customize Webif on the device during development.

Adding a New Webif Page on page 188

Use this procedure to add or modify your Webif interface configuration pages

Webif Configuration Page Functions on page 190
This reference lists the Webif functions used to configure pages and gives a skeleton prototype.

Webif Configuration Page Functions

This reference lists the Webif functions used to configure pages and gives a skeleton prototype.

Function Name	Description
uci_load	Defined in **/lib/config/uci.sh**. Transfers **/etc/config/${package}** into environment variables.
validate	Defined in **/usr/lib/webif/functions.sh**. Checks the form values that users input; sets **ERROR** if any value breaks the rules defined in **/usr/lib/webif/validate.awk**.
header	Defined in **/usr/lib/webif/webif.sh**. Outputs the HTML header.
display_form	Defined in **/usr/lib/webif/webif.sh**. Displays forms according to the syntax defined in **/usr/lib/webif/forms.awk**.
footer	Defined in **/usr/lib/webif/webif.sh**. Outputs the HTML footer.

Typical Webif Page Format

```
#!/usr/bin/webif-page
<?
. "/usr/lib/webif/webif.sh"
uci_load "${package_name}"

if empty "$FORM_submit"; then
 # Get variables from /etc/config/${package}, 2 ways:
 # 1. by 'uci get package.section.variable' function.
 # 2. by '$CONFIG_section_variable'
else
 # Variables validation
 # validate will return 0 if all validations pass, otherwise
 # return 1.
validate <<EOF
${VALIDATE_STRING}
EOF
equal "$?" 0 && {
 # Set variables to /tmp/.uci/${package}, 2 ways:
 # 1. 'uci set package.section.variable=value' function.
 # 2. 'uci_set package section variable ${value}'
 }
fi
 # Output HTML header,
 # variable "ERROR" will be set if validation fails.
header "${MAINCATEGORY}" "${SUBCATEGORY}" "${HEAD_TAB}" "${BODY_FUNCTION}"
 "${FORM_ADDRESS}"

# Define any necessary javascripts.
cat <<EOF
<script>.....</script>
EOF

# Display HTML forms
display_form <<EOF
${FORM_STRING}
EOF

# Output HTML footer
footer ? >
```

```
# The last part decides the position of the page on WEB.
# ${MAINCATEGORY}: The name of first level of Menu.
# ${SUBCATEGORY}: The name of second level of Menu.
# ${SORT_NUMBER}: The position of second level.
<!--
##WEBIF:name:${MAINCATEGORY}:${SORT_NUMBER}:${SUBCATEGORY}
-->
```

Related Links

About Customizing Webif on page 187
The device owner's specification may require that you customize Webif on the device during development.

Adding a New Webif Page on page 188
Use this procedure to add or modify your Webif interface configuration pages

Webif Configuration Page Structure on page 189
You can add tabs to the Webif interface using a standard page structure.

Example: Working Network Webif Page on page 191
This code example is based on the prototype. It is a working Webif page called **network-lan.sh**.

Example: Working Network Webif Page

This code example is based on the prototype. It is a working Webif page called **network-lan.sh**.

```
#!/usr/bin/webif-page
<?
. /usr/lib/webif/webif.sh

if empty "$FORM_submit"; then
    SAVED=0

    # Get FORM_newif_* from uci configuration file. For example:
    FORM_newif_type=$(uci get network.newif.type)
    FORM_newif_ipaddr=$(uci get network.newif.ipaddr)
    FORM_newif_netmask=$(uci get network.newif.netmask)

elif [ "$FORM_action" == "Save Changes" ] ; then
    SAVED=1

validate <<EOF
ip|FORM_newif_ipaddr|@TR<<NEWIF Address>>|required|$FORM_newif_ipaddr
netmask|FORM_newif_netmask|@TR<<NEWIF
Netmask>>|required|$FORM_newif_netmask
EOF
    # Save FORM_newif_* to uci configuration file. For example:
    equal "$?" 0 && {
        uci_set network newif type $FORM_newif_type
        uci_set network newif ipaddr $FORM_newif_ipaddr
        uci_set  network newif netmask $FORM_newif_netmask
    }   fi

header "NEWCATEGORY" "NEWIF" "@TR<<NEWIF Configuration>>" \
'onload="modechange()"' "$SCRIPT_NAME"

cat <<EOF
<script type="text/javascript" src="/webif.js"></script>
<script type="text/javascript">
<!--
function modechange()
{
    var v;

    v = isset('newif_type', 'static');
    set_visible('newif_ipaddr_field', v);
    set_visible('newif_netmask_field', v);
```

```
    hide('save');
    show('save');
}
-->
</script>
EOF

display_form <<EOF
onchange|modechange
start_form|@TR<<NEWIF IPv4 Configuration>>
field|@TR<<NEWIF IPv4 TYPE>>
select|newif_type|$FORM_newif_type
option|static|Static
option|dhcp|DHCP
helpitem|IPv4 Type
helptext|Type is Static or DHCP.
field|@TR<<NEWIF IPv4 Address>>|newif_ipaddr_field
text|newif_ipaddr|$FORM_newif_ipaddr
helpitem|IPv4 Address
helptext|This is the address.
field|@TR<<IPv4 Netmask>>|newif_netmask_field
text|newif_netmask|$FORM_newif_netmask
helpitem|IPv4 Netmask
helptext|This is the netmask.
end_form
EOF

footer ?>
<!--
##WEBIF:name:NEWCATEGORY:100:NEWIF
-->
```

NOTE: If you copy-and-paste the script to another location, confirm that any lines containing **EOF** are not indented in the output, otherwise the script will not execute from the Webif interface. Indentation is not always maintained during a copy-and-paste operation.

Related Links

Webif Configuration Page Functions on page 190
This reference lists the Webif functions used to configure pages and gives a skeleton prototype.

22

Updating WPAN Firmware for Quark Boards

Cross Hill boards provide a utility for updating WPAN firmware.

- Run the firmware update command on the device.

```
# q58_programmer_clanton_fw0.8
```

PART V

Application Development Vendor Tasks

23

Application Development

The Intelligent Device Platform provides product options and add-ons designed to meet specific development needs.

Since there are many different system applications and end-user requirements, the add-ons provide solutions to most machine-to-machine and cloud-based application needs. This includes the following technologies:

Add-on product Exegin Zigbee

provides a communications stack for managing wireless connections. See *Exegin Zigbee Stack* on page 32.

Wind River OpenJDK

provides Java-based virtual machine (VM) platform development. See *Installing OpenJDK* on page 162.

SmartHome SDK for Open Services Gateway Initiative (OSGi)

provides Java- and Linux-based development for the Eclipse integrated development environment (IDE). This includes platform and application tools to add OSGi capabilities to target platforms. See *OSGi Development with the mBS Smart Home SDK* on page 203.

Sqlite3

provides a lightweight database for embedded applications. See *Sqlite3 Command Reference* on page 207 and *Sqlite3 Data Element Reference* on page 208.

MQTT

consists of the open source server Mosquitto and a client that includes utilities for publishing and subscribing to MQTT topics. See *About MQTT and Lua* on page 213.

Lua

is an open source, powerful, light-weight programming language designed for extending applications. It is also frequently used as a general-purpose, stand-alone language. See *About MQTT and Lua* on page 213.

Encrypted storage

provides secure storage for application data on Intel Architecture devices. See *Encrypted Storage* on page 28.

OneAgent TR-069 agent

provides a protocol and API stack for communication between a TR-069-enabled client and server. See *OneAgent TR-069 Agent* on page 221.

Works System OneAgent OMA Device Management Communications (DMC) agent

supports several OMA DM management objects (MO) through extensible wrappers called MO Wrappers. The DMA agent reports device information and executes commands using the OMA-DM protocol to a remote OMA server. See *OneAgent OMA-DM Agent and MO Wrappers* on page 36.

grsecurity RBAC

provides full learning mode for generating security policy rules. See *grsecurity and Related Tools* on page 227.

The layer descriptions in *Layers and Features* on page 151 provide the layer name and installation location for each development option listed in this section.

24

Exegin Zigbee Stack

About the Zigbee Stack

ZigBee is a specification for a suite of high level communication protocols used to create personal area networks built from small, low-power digital radios.

The following hardware and software are required for the Zigbee stack:

- A Wind River IDP XT 2.0 installation on top of a Wind River Linux 5.0.1 installation with RCPL updates on a supported host.
- Two or more Cross Hill hardware boards with Q58 chip. (The example uses two boards.)
  - An image with the **wr-exegin-zigbee-ia** layer with Exegin ZigBee SDK version 1.6.51 or later on each board. IDP XT 2.0 ships with this version so no action is required.
  - A serial connection to a host console for each board.

  Before proceeding, you must have connected the hardware board to the host using a serial cable and powered it on.

Setting Up a Zigbee Network

Once you have installed the Exegin ZigBee layer and the ZigBee SDK, you can form a new network or join an existing network.

This example uses two devices in the following linear routing topology:

spidev1.0 <--> spidev1.1

where:

spidev1.0 = Coordinator
spidev1.1 = Router

Step 1 Start the coordinator (**spidev1.0**).

```
$ zapp /dev/spidev1.0 --spiHostWake 15 --stdout
SDK> config channel 11
SDK> zigbee form
SDK> zigbee pjoin 255
```

Step 2 Start the router (**spidev1.1**).

```
$ zapp /dev/spidev1.1 --spiHostWake 14 --stdout
SDK> config channel 11
SDK> zigbee join
```

Step 3 On the coordinator (**spidev1.0**), confirm that the router has joined.

```
SDK> zigbee status
-- ZigBee ZDO Status ------------------------------------------------------
                   Device Type: Coordinator
                        Status: Success

-- ZigBee APS Binding Table ----------------------------------------------
INDX  SRCADDR             SRCENDPT   CLUSTER  DSTADDR                  DSTENDPT

-- ZigBee APS Group Table ------------------------------------------------
GROUP   ENDPT

-- ZigBee Network Status -------------------------------------------------
              Logical Channels: 11
              Extended Address: 00:1c:da:00:00:00:31:6a
               Extended PAN Id: 0x001cda000000316a
                 Short Address: 0x0000
                        PAN Id: 0x33a3
              Protocol Version: 0x02
                 Stack Profile: ZigBee PRO
             Permit Join Value: 0xff
          Trust Center Address: 00:1c:da:00:00:00:31:6a

-- ZigBee Network Neighbor Table -----------------------------------------
EXTADDR                 NWKADDR  TYPE  RXIDLE  RELATION      TXFAIL   AGE   LQI
00:1c:da:00:00:00:31:6b  0x79bb   Rtr   TRUE    Child            0     0   255

-- ZigBee Network Discovery Table ----------------------------------------
NWKADDR  PANID   EXTENDED PANID      CHAN  VERS  DEPTH  PJOIN   LQI  PROFILE

-- ZigBee Network Routing Table ------------------------------------------
DSTADDR  NEXTHOP  STATUS      LASTUSED
```

The router entry in the in ZigBee Network Neighbor Table section near the bottom of the output.

25

Wind River OpenJDK

About OpenJDK

Wind River OpenJDK is an open source implementation of Java Platform SE (Java SE).

Wind River OpenJDK allows you to compile Java code using only free software with your Linux distribution. Wind River OpenJDK capability provides the following resources on the IDP XT target:

JRE (Java Runtime Environment)
An open source Java virtual machine (JVM) that uses OpenJDK as its Java run-time library.

Basic OpenJDK Command Reference

The virtual machine supplied with OpenJDK is Zero Virtual Machine.

26

OSGi Development with the MBS Smart Home SDK

OSGi Development with the mBS Smart Home SDK

The ProSyst mBS Smart Home SDK development kit included with the OSGi bundle provides a base for tailoring images for specific home device management platforms.

The ProSyst mBS Smart Home SDK development kit provided with the OSGi bundle consists of three main components:

OSGi Runtime

Contains the ProSyst implementation of the OSGi standard and a set of functional ProSyst components. The OSGi Runtime's purpose is to serve as the base for tailoring images for specific home device management platforms. It provides all components required to run OSGi on target devices also including development-driven capabilities like debugging, profiling, remote management, emulation, etc.

Eclipse Plug-ins

Offer enhanced and friendly facilities for simplified development and testing of OSGi-based applications in an emulated runtime environment or directly on the target device.

OSGi Runtime Validator

Supplies the option to validate the components of the OSGi Runtime on a specific target platform. The validation comprises functional as well as non-functional (performance and stability).

The Intelligent Device Platform provides one form of OSGi development through the implementation of the ProSyst mBS Smart Home SDK.

ProSyst mBS Smart Home SDK Packages for IA Boards

The ProSyst Smart Home SDK consists of the following packages

ProSyst_mBS_SH_SDK_7.5_Commercial.zip

the ProSyst mBS Smart Home SDK 7.5 package. This package provides modules designed to work together to help developers create robust home device management platforms.

ProSyst_mBS_SH_SDK_7.5_Board_Extension_Quark.zip

a package supplied by Wind River providing IDP XT extensions for **intel-quark**.

ProSyst_mBS_SH_SDK_7.5_Board_Extension_Atom.zip

a package supplied by Wind River providing IDP XT extensions for **intel-atom-baytrail**.

prosystOSGi-2.1.tar.gz

These files are located at: ***projDir*/layers/wr-idp/wr-prosyst-mbs-smarthome-sdk-ia/downloads**

For more information on OSGi development, see the **README** file located in the ***projDir*/layers/wr-idp/wr-prosyst-mbs-smarthome-sdk-ia/templates/default** directory.

Additional Information

For additional information on the Smart Home SDK, refer to the *ProSyst website*. This website includes information on:

- Runtime components
- OSGi platform capabilities
- The mBSA system agent that handles the runtime's native process
- Guidelines about included Java virtual machines (JVMs)
- Installing Eclipse Plugins

Developing with OSGi

In order to develop applications for OSGi, you must boot the target and configure OSGi.

In order to develop with OSGi, you need a target with an image that includes OSGi. For more information, see *OSGi Development Workflow* on page 167.

Step 1 Boot the target.

Step 2 Start the runtime.

NOTE: The runtime requires that the board's system time is set to the current time.

```
# cd /opt/prosyst_osgi/mbsa/bin/
# ./mbsa_start
```

Once the OSGi image completes its startup process, it displays the following message:

```
[mBSA] OSGi framework is started successfully
```

Step 3 Test OSGi by accessing the OSGi configuration pages.

Open the following URL in a Web browser on your host:

http://*targetIpAddr*/system/console

Step 4 Continue with application development and managing the OSGi framework and bundles.

For more information, see:

http://dz.prosyst.com/pdoc/mBS_SDK_7.3.1/common_tasks/commontasks.html

27

Sqlite3 Database

Using Sqlite3

Sqlite3 is a lightweight database used primarily for embedded systems.

You must have a board with an image that includes at least the **--enable-addons=wr-idp** option.

- Start Sqlite3.

  You can use the basic Sqlite3 commands from the command line with user-space files. Note that all operations are performed by the root user.

```
# sqlite3 <test.db>
sqlite>
```

Related Links

Sqlite3 Command Reference on page 207
Use the **sqlite3** command to enter the Sqlite3 command terminal.

Sqlite3 Data Element Reference on page 208
List of data types and data constraints with definitions.

Sqlite3 Examples on page 209

Sqlite3 Command Reference

Use the **sqlite3** command to enter the Sqlite3 command terminal.

.database	Check database file information

.schema	Show the **create** statements
.schema `table_name`	Show specified table **create** statements
.dump `table_name`	Output table content as sql statements
.help	Output help information
.quit or **.exit**	Exit Sqlite terminal

Related Links

Using Sqlite3 on page 207
Sqlite3 is a lightweight database used primarily for embedded systems.

Sqlite3 Data Element Reference on page 208
List of data types and data constraints with definitions.

Sqlite3 Examples on page 209

Sqlite3 Data Element Reference

List of data types and data constraints with definitions.

Data Types

null	Specify a NULL value
integer	Specify an integer value
real	Specify a float value
text	Specify a character string
blob	Specify binary data

Data Constraints

primary key

- The primary key value must be unique to provide a unique identifier for each record.
- The primary key also serves as an index; searching for a record is faster using the primary key.
- If the primary key type is **integer**, the column value auto-increments.

not null

Specifies that the column record cannot be empty; otherwise an error is reported.

unique

Constrains columns other than the primary key to unique values only.

check

Specifies a condition that must be met before the data can be stored.

default

Specifies a default value for a column if the value is not specified when inserting the record.

Related Links

Using Sqlite3 on page 207
Sqlite3 is a lightweight database used primarily for embedded systems.

Sqlite3 Command Reference on page 207
Use the **sqlite3** command to enter the Sqlite3 command terminal.

Sqlite3 Examples on page 209

Sqlite3 Examples

Create Table

Format:

```
create table table_name(field1 type1, field2 type2,...);
```

For example:

```
create table student_info (stu_no integer primary key, name text);
```

Insert Data

Format:

```
insert into table_name(field1, field2,...) values(val1,val2, ...);
```

For example:

```
insert into student_info(stu_no, name) values(0001, "alex");
```

Update Data Record

Format:

```
update table_name set field1=val1, field2=val2 where expression;
```

For example:

```
update student_info set stu_no=0001, name="hence" where stu_no=0001;
```

Delete Data Records

Format:

```
delete from table_name[where expression];
```

If the **where** statement is not added, the data table is cleared.

For example:

```
delete from student_info where stu_no=0001;
```

Search Data Records

Format:

```
select columns from table_name[where expression];
```

1. Output all data records

   ```
   select * from table_name;
   ```

2. Limit the number of output data records

   ```
   select * from table_name limit val;
   ```

3. Output data records ascending or descending order

   ```
   select * from table_name order by field asc;
   select * from table_name order by field desc;
   ```

4. Search for records that meet a specific condition

   ```
   select * from table_name where expression;
   select * from table_name where field in ('val1', 'val2', 'val3');
   select * from table_name where field between val1 and val2;
   ```

5. Find the number of records in the table

   ```
   select count(*) from table_name;
   ```

6. Find the number of distinct values in a column

   ```
   select distinct field from table_name;
   ```

   The **distinct** option removes any duplicates resulting in a single list of values for the column *field*.

Create Index

Format:

```
create index index_name on table_name(field);
```

Used an index when you have a large number of data tables. The index speeds up lookup of table data.

For example:

```
create index student_index on student_info(stu_no);
```

The field **stu_no** automatically uses the index when you execute a query.

Delete a Data Table or Index

Format:

```
drop table table_name;
```

For example:

```
drop table student_info;
```

Format:

```
drop index index_name;
```

For example:

```
drop index student_index;
```

Related Links

Using Sqlite3 on page 207
Sqlite3 is a lightweight database used primarily for embedded systems.

Sqlite3 Command Reference on page 207
Use the **sqlite3** command to enter the Sqlite3 command terminal.

Sqlite3 Data Element Reference on page 208
List of data types and data constraints with definitions.

28

MQTT and Lua

About MQTT and Lua

MQTT consists of the open source server Mosquitto and a client that includes utilities for publishing and subscribing to MQTT topics. The client uses the Lua language.

The MQTT server is Mosquitto, an open source implementation of a server for version 3.1 of the MQTT Protocol.

The client is an implementation of the MQTT protocol, plus command-line utilities for publishing and subscribing to MQTT topics. The implementation is based on the Lua language.

Lua is a powerful, light-weight programming language designed for extending applications. It is also frequently used as a general-purpose, stand-alone language. Lua is free software.

Related Links

The Lua Language on page 214
Lua is a powerful, light-weight programming language designed for extending applications. It is also frequently used as a general-purpose, stand-alone language. Lua is free software.

Starting the MQTT Server on page 215
Start the MQTT server using the **mosquitto** command. You can customize the Mosquitto configuration by modifying **/etc/mosquitto/conf.d**.

About Publishing and Subscribing to Messages on page 216

MQTT allows you to publish messages on the server or to subscribe to messages published by others.

The Lua Language

Lua is a powerful, light-weight programming language designed for extending applications. It is also frequently used as a general-purpose, stand-alone language. Lua is free software.

The Lua source code package contains the reference documentation. The reference documentation is the official definition of the Lua language. The documentation is located in the following file:

***Lua_sources_path*/doc/contents.html**

The Lua source package is located at:

***projDir*/layers/wr-idp/wr-idp-devkit/downloads/lua-*version*.tar.gz**

where the default *version* is **5.1.5**.

NOTE: IDP XT includes two version of Lua: 5.2.2 and 5.1.5. Wind River recommends using the 5.1.5 version. The version used by default is defined in the file ***projDir*/layers/wr-idp/wr-idp-devkit/conf/layer.conf**.

Related Links

About MQTT and Lua on page 213
MQTT consists of the open source server Mosquitto and a client that includes utilities for publishing and subscribing to MQTT topics. The client uses the Lua language.

Starting the MQTT Server on page 215
Start the MQTT server using the **mosquitto** command. You can customize the Mosquitto configuration by modifying **/etc/mosquitto/conf.d**.

About Publishing and Subscribing to Messages on page 216
MQTT allows you to publish messages on the server or to subscribe to messages published by others.

Examples: Using the Lua Program Examples on page 214
The Lua source code package contains the Lua application program examples. The application program examples directory is ***lua_source_path*/test/**.

Examples: Using the Lua Program Examples

The Lua source code package contains the Lua application program examples. The application program examples directory is ***lua_source_path*/test/**.

Step 1 Transfer the **.lua** files to your IDP XT target.

```
$ scp *.lua root@<IP-Address-of-IDP-Target>
```

Step 2 Execute the Lua example files.

The following examples show the commands and the system output for the various example files.

```
root@WR-IntelligentDevice:~# lua hello.lua
Hello world, from Lua 5.1!
root@WR-IntelligentDevice:~# luac hello.lua
root@WR-IntelligentDevice:~# lua luac.out
Hello world, from Lua 5.1!
root@WR-IntelligentDevice:~# lua echo.lua WindRiver IDP 2.1
0       echo.lua
1       WindRiver
2       IDP
3       2.1
root@WR-IntelligentDevice:~# lua sort.lua
original
        Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec
after quicksort
        Apr,Aug,Dec,Feb,Jan,Jul,Jun,Mar,May,Nov,Oct,Sep
after reverse selection sort
        Sep,Oct,Nov,May,Mar,Jun,Jul,Jan,Feb,Dec,Aug,Apr
after quicksort again
        Apr,Aug,Dec,Feb,Jan,Jul,Jun,Mar,May,Nov,Oct,Sep
root@WR-IntelligentDevice:~# lua life.lua
----------------------------------------
----------------------------------------
--OO--------------------------O---------
-OO----------------------------O-O-------
---O--------------------------O---------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
---O------------------------------------
--O-O-----------------------------------
--O-O-----------------------------------
---O------------------------------------
----------------------------------------
-------OO-------------------------------
-------OO-------------------------------
----------------------------------------
Life - generation 2000
```

Related Links

The Lua Language on page 214
Lua is a powerful, light-weight programming language designed for extending applications. It is also frequently used as a general-purpose, stand-alone language. Lua is free software.

Starting the MQTT Server

Start the MQTT server using the **mosquitto** command. You can customize the Mosquitto configuration by modifying **/etc/mosquitto/conf.d**.

Step 1 Check to see if the MQTT server is already running.

```
$ ps -ef | grep mosquitto
```

Step 2 If mosquito is not running, start or restart the server.

A script is provided for starting the server.

```
$ /etc/init.d/mqtt {start|stop|restart}
```

Step 3 (Optional) Modify the configuration of the MQTT server.

a) If you need to update some configuration items, update **/etc/mosquitto/conf.d**.
b) Run **/etc/init.d/mqtt**.
c) Restart the server.

Related Links

About MQTT and Lua on page 213
MQTT consists of the open source server Mosquitto and a client that includes utilities for publishing and subscribing to MQTT topics. The client uses the Lua language.

The Lua Language on page 214
Lua is a powerful, light-weight programming language designed for extending applications. It is also frequently used as a general-purpose, stand-alone language. Lua is free software.

About Publishing and Subscribing to Messages on page 216
MQTT allows you to publish messages on the server or to subscribe to messages published by others.

About Publishing and Subscribing to Messages

MQTT allows you to publish messages on the server or to subscribe to messages published by others.

The IDP XT MQTT implementation provides several sample files for publishing, subscribing, and testing your installation.

You can view the additional options for the commands by typing the command without specifying any options. The following commands are available:

mqtt_subscribe.lua
mqtt_publish.lua
mqtt_test.lua

Related Links

About MQTT and Lua on page 213
MQTT consists of the open source server Mosquitto and a client that includes utilities for publishing and subscribing to MQTT topics. The client uses the Lua language.

The Lua Language on page 214
Lua is a powerful, light-weight programming language designed for extending applications. It is also frequently used as a general-purpose, stand-alone language. Lua is free software.

Starting the MQTT Server on page 215
Start the MQTT server using the **mosquitto** command. You can customize the Mosquitto configuration by modifying **/etc/mosquitto/conf.d**.

Example: Multiple Messages on page 217
This example periodically publishes a message on topic **test/1** and subscribes to the message on topic **test/2**.

Example: Single Message on page 217

This example subscribes to a specific topic (**test/1**) and listens for it continuously. The message is displayed on the subscriber console when the topic is published.

Example: Multiple Messages

This example periodically publishes a message on topic **test/1** and subscribes to the message on topic **test/2**.

The example assumes the MQTT server is on **localhost**, but you can change the host value to the IP address if the server is remote.

Step 1 Set the path.

```
$ export PATH=$PATH:/root/examples/mqtt-client/
```

Step 2 Run the **mqtt_test.lua** example command.

```
$ mqtt_test.lua -d localhost
```

The command exits when the message **quit** is published on topic **test/2**.

Related Links

About Publishing and Subscribing to Messages on page 216
MQTT allows you to publish messages on the server or to subscribe to messages published by others.

Example: Single Message

This example subscribes to a specific topic (**test/1**) and listens for it continuously. The message is displayed on the subscriber console when the topic is published.

Step 1 Subscribe to a topic.

This example subscribes to a topic and listens indefinitely for messages. Use **^C** or equivalent to stop execution.

```
$ mqtt_subscribe.lua -d -t test/1
```

Step 2 Publish a topic.

This example publishes a single message and then exits.

```
$ mqtt_publish.lua -d -t test/1 -m "Test message"
```

Notice that the message “Test message” is displayed on the subscriber console as soon as you execute the **mqtt_publish.lua** command.

Related Links

About Publishing and Subscribing to Messages on page 216
MQTT allows you to publish messages on the server or to subscribe to messages published by others.

29

Encrypted Storage

Encrypted storage provides secure storage for application data on Intel Architecture devices.

Some applications need secure storage for sensitive data which must not be accessible to another device. For example, only an application with the right signature can update the data on an encrypted SD card. If you move that SD card to another device, the data cannot be either read or updated. One application of this capability is a POS application. The application keeps tax information in secure storage that cannot be modified by another device.

The Device Mapper Infrastructure

Device mapper infrastructure in Linux 2.6 and later kernels provides a generic way to create virtual layers of block devices. The **dm-crypt** subsystem of the Linux kernel implements the device mapper and provides transparent encryption of block devices using the kernel's cryptography API. You can specify a symmetric cipher, an encrption mode, a key of allowed size, and an IV generation mode to create a new block device in the **/dev** directory. Once you create the device, writes to the device are encrypted and reads are decrypted automatically. You can mount your filesystem on this block device or you can stack a **dm-crypt** device with another device such as a RAID or LVM volume.

Cryptography Tools and LUX

The **dm-crypt** module resides in kernel space and relies on user space front-end tools such as **cryptsetup** to create the encrypted volumes and do authentication. An enhanced version of **cryptsetup** provides LUKS (Linux Unified Key Setup) support for the **dm-crypt** module.

The **cryptsetup** tool provides commands for using the LUKS on-disk format. LUKS is a disk-encryption specification for Linux systems which not only facilitates the compatibility among different Linux distributions but also provides secure management of multiple user passwords. LUKS stores all the necessary setup information in the partition header enabling users to transport data seamlessly.

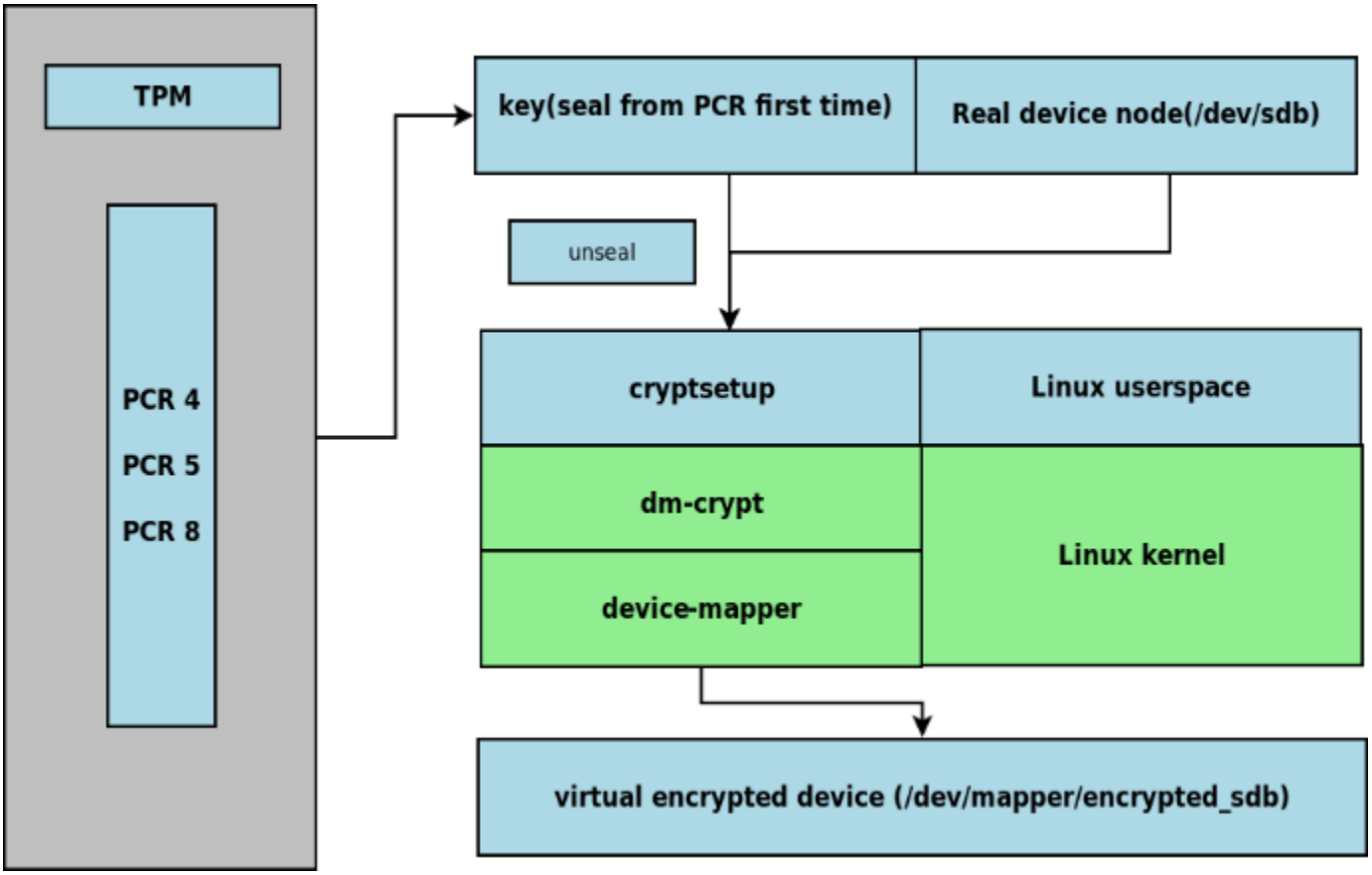
Encryption Process

When you initially create the storage device, the tool creates a sealed key from TPM PCRs. The sealed key becomes is an argument to the **cryptsetup** tool for formatting and opening the device and for creating the mapping to the real device. Once the device is prepared, reading and writing are encrypted.

The **dm-crypt** tool inserts a layer between the read/write and the real block device as shown:

Data Read/Write --> **/dev/mapper/*`virtual_device`*** --> **/dev/*`real_device`***

The following diagram shows the components involved in implementing this functionality.


TPM
PCR 4
PCR 5
PCR 8
key(seal from PCR first time)
Real device node(/dev/sdb)
unseal
cryptsetup
Linux userspace
dm-crypt
device-mapper
Linux kernel
virtual encrypted device (/dev/mapper/encrypted_sdb)

30

OneAgent TR-069 Agent

The OneAgent TR-069 agent provides a protocol and API stack for communication between a TR-069-enabled client and server.

The TR-069 technical specification is titled *CPE WAN Management Protocol (CWMP)*. It defines an application layer protocol for remote management of end-user devices.

CPE, or customer premises equipment, acts as the client. In the Intelligent Device Platform system, this client communication is managed by the OneAgent implementation. ACS, or auto-configuration server, provides access to the WAN as the TR-069 server.

When used as part of a network system, implementing TR-069 provides the following functionality for your device platform:

- Auto-configuration and dynamic service provisioning
- Software/firmware image management
- Status and performance monitoring
- Diagnostics

In order to configure the application using a remote server, you must do the following:

- implement configuration interfaces in your application

  This is a local implementation which is device related. Examples include:

  - retrieving the LAN IP address
  - getting statistical data on CPU usage
- implement an adapter layer for the TR-069 agent to call those interfaces

  All the RPC adapter methods are listed in **interface/xml/tr_lib.h**. Implement the methods you want in **interface/xml/tr_lib.c**.

NOTE: If application configuration items are not included in the existing TR-069 data model, you must define extended management nodes for your application.

For additional information on TR-069, see:

- *Broadband Forum Home*
- *Broadband Forum Technical Reports*
- *TR-069 Wiki*
- Documentation provided with IDP XT in the ***projDir*/layers/wr-wks-oneagent-tr069/templates/default** directory:

- **README**
- *OneAgent TR 5.5 Integration Guide*

31

Works Systems OneAgent OMA Agent

About the OMA-DM Agent

The DMA agent reports device information and executes commands using OMA-DM protocol to a remote OMA server.

The agent supports OMA DM management objects (MO) through extensible wrappers called MO Wrappers. Currently the following objects are supported: **DevInfo**, **DMAcc**, **ConnMO**, and **SCOMO**.

When you include the OMA-DM agent in your platform project, the following binaries and configuration files are installed on the target file system:

/usr/sbin/oma
/usr/sbin/oma_bin
/etc/oma/oma.xml
/etc/oma/*

Most of the configuration files for the OMA-DM client are placed in **/etc/oma/**.

You can view additional OMA-DM debug messages in the **/var/log/oma.log** file.

NOTE: When you install OMA-DM, the OMA-DM agent is disabled by default. Start the agent to view logs.

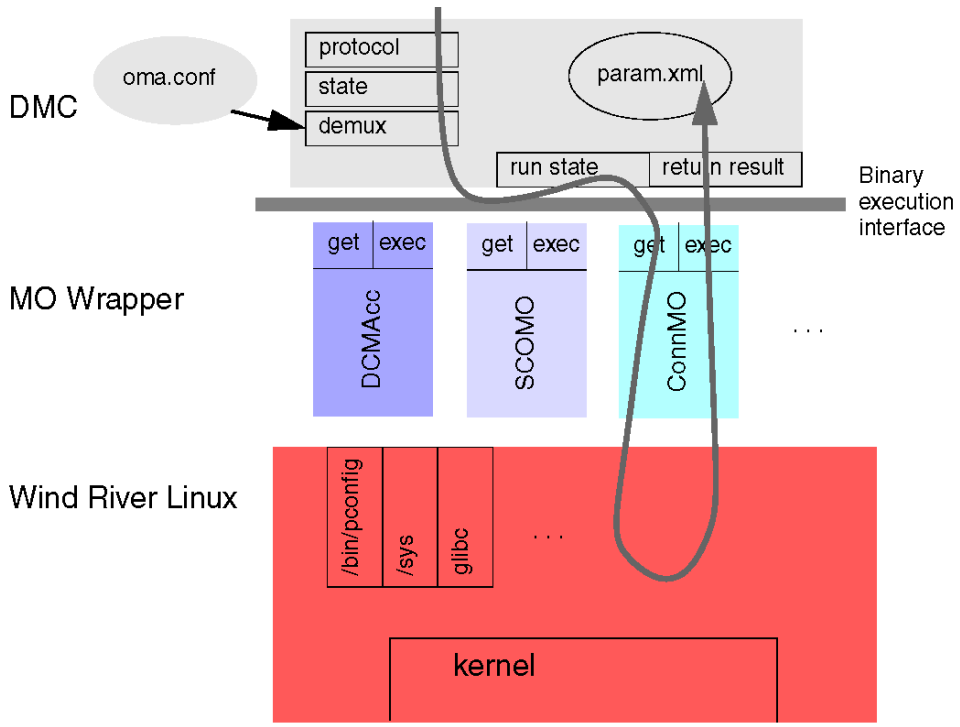


Figure 1: Integration Points Between the DMC, MO Wrappers, and the Operating System

Additional Information

For more information on the OneAgent OMA-DM DMC agent, see the **README** file contained in ***projDir*/layers/wr-idp/wr-wks-oneagent-oma-dm-ia/templates/default**.

For the *OneAgent OMA-DM 3 Integration Guide*, open the integration guide PDF after extracting the following package:

***projDir*/layers/wr-idp/wr-wks-oneagent-oma-dm-ia/downloads/oneagent_oma_dmc-3.1.tar.bz2**

Related Links

About MO Wrappers on page 224
The MO Wrappers are a layer between the OMA DMC agent and the target device.

About MO Wrappers

The MO Wrappers are a layer between the OMA DMC agent and the target device.

The layer collects all of the incoming information and maps commands from the DMC to the device. The information consists of device properties and other system-related information. MO Wrappers are designed to be extensible, making it possible to create new wrappers without making modifications to the DMC.

The following MO wrappers are installed on the target file system at **/usr/sbin/mowrappers**:

DevInfo

provides device information to the DMS that it uses to identify the device.

DMAcc

provides the authentication necessary to access the DMS.

ConnMO

provides management for connectivity settings, including Ethernet, wireless, 3G, in addition to managing other extended network connectivity.

SCOMO

manages package installation and activation. You can view SCOMO messages in the **/var/log/messages** file.

NOTE: The OMA DMC automatically uses these wrappers. The **DevInfo**, **DMAcc**, and **ConnMO** MO wrappers use the Universal Command line Configuration Tool (UCI) to set and get node values from a local database in **/etc/config/oma/** and return the node value back to the DMC. The DMC then send the results to the DMS.

Related Links

About the OMA-DM Agent on page 223
The DMA agent reports device information and executes commands using OMA-DM protocol to a remote OMA server.

32

The grsecurity Tool

grsecurity and Related Tools

IDP XT uses grsecurity to provide a default security policy and tools to customize the policy.

NOTE: In order to use grsecurity and PaX, you must have a target with an image that includes these tools.

grsecurity Overview

grsecurity (*http://en.wikipedia.org/wiki/grsecurity*) is a set of free software patches released under the GNU GPL for the Linux kernel to enhance the system security. It allows the system administrator to, among other things, define a minimum privilege policy for the system, in which every process and user have only the lowest privileges necessary to function. A typical application for grsecurity is Web servers and systems that accept remote connections from untrusted locations, such as systems offering shell access to users.

grsecurity RBAC

grsecurity's RBAC (role-based access control) provides full learning mode for generating security policy rules. The default policy is located in the **/etc/grsec/policy** file.

The Wind River IDP XT SRM solution uses the RBAC system to provide a default policy that defines the minimum privileges necessary to allow the SRM layer to work properly. Once the SRM system is up and running, you can use the grsecurity full learning mode to learn the

application policy when you customize a new package for deployment. See *Generating a Security Policy for the Package* on page 231.

grsecurity Administration Utility

The grsecurity administration utility (**gradm**) helps manage the RBAC system. **gradm** parses your access control lists (ACLs), enforces a secure base policy, and optimizes the ACLs. It also parses the learning logs (from Full Learning mode), merges them with your ACL set, and outputs the final ACLs. To use **gradm** to manage policy for packages you include on your RPM repository, see *Generating a Security Policy for the Package* on page 231. The default password for **gradm** is **windriver**.

grsecurity Full Learning Mode

SRM uses the grsecurity full learning mode to generate policy files for new packages that you want to deploy. For an example of using Full Learning mode on a package that you want to include in your package repository, see *Generating a Security Policy for the Package* on page 231.

PaX

Pax flags data memory on the stack as non-executable and program memory as non-writable. It is a set of patches applied to Linux kernel. The goal of PaX is to prevent executable memory pages from being overwritten with injected machine code and thus to prevent exploitation of common security vulnerabilities such as buffer overflows. PaX is not necessarily developed by grsecurity developers; however, it is available for *download* separately from the grsecurity Web site.

Additional Information

For more information on grsecurity, see:

- *http://en.wikibooks.org/wiki/grsecurity/Print_version*
- *http://pax.grsecurity.net/*

Related Links

grsecurity RBAC Command Reference on page 229
The grsecurity administration utility (**gradm**) manages the role-based access control (RBAC) system.

paxctl Reference on page 229
The user-space utility for controlling the PaX flags of executable files is **paxctl**.

Generating a Security Policy for the Package on page 231
Once you have booted a target with the grsecurity RBAC system enabled, you must add a security policy for your applications.

The grsecurity sysctl Interface on page 232
The sysfs in Linux provides an interface for viewing or modifying kernel parameters at runtime. However, if you have a signed SRM rootfs, you can only view, not change, parameters.

Troubleshooting grsecurity on page 232

Examples of common security issues and how to resolve them.

grsecurity RBAC Command Reference

The grsecurity administration utility (**gradm**) manages the role-based access control (RBAC) system.

The following table provides common commands for managing the grsecurity RBAC system:

Command	Description
gradm -P [rolename]	Setup RBAC administration or special role password
gradm -E	Enable the grsecurity RBAC system
gradm -D	Diable the grsecurity RBAC system
gradm -C	Check the RBAC policy for errors
gradm -S	Check the RBAC system's status
gradm -F -L /tmp/full_learning.log	Enable the grsecurity Full Learning mode

Related Links

grsecurity and Related Tools on page 227
IDP XT uses grsecurity to provide a default security policy and tools to customize the policy.

paxctl Reference on page 229
The user-space utility for controlling the PaX flags of executable files is **paxctl**.

Generating a Security Policy for the Package on page 231
Once you have booted a target with the grsecurity RBAC system enabled, you must add a security policy for your applications.

The grsecurity sysctl Interface on page 232
The sysfs in Linux provides an interface for viewing or modifying kernel parameters at runtime. However, if you have a signed SRM rootfs, you can only view, not change, parameters.

Troubleshooting grsecurity on page 232
Examples of common security issues and how to resolve them.

paxctl Reference

The user-space utility for controlling the PaX flags of executable files is **paxctl**.

The folllowing table lists some common PaX options. To view all the available command-line switches, execute **paxctl --help**.

Option	Description	Option	Description
-p	disable **PAGEEXEC**	**-P**	enable **PAGEEXEC**
-e	disable **EMUTRMAP**	**-E**	enable **EMUTRMAP**

Option	Description	Option	Description
-m	disable **MPROTECT**	**-M**	enable **MPROTECT**
-r	disable **RANDMMAP**	**-R**	enable **RANDMMAP**
-x	disable **RANDEXEC**	**-X**	enable **RANDEXEC**
-s	disable **SEGMEXEC**	**-S**	enable **SEGMEXEC**
-v	view flags	**-z**	restore default flags
-q	suppress error messages	**-Q**	report flags in short format
-c	convert **PT_GNU_STACK** into **PT_PAX_FLAGS** (see man page)	**-C**	create **PT_PAX_FLAGS** (see man page)

The following example shows how to use the paxctl commands. It is not a recommended solution to a memory protection problem; Wind River recommends that you fix the memory protection issue inside your application (modifying the source code) rather than disable the **MPROTECT** flag to get around the denial message.

If your application or a particular command is denied by grsecurity for some memory protection reason, you can disable the **MPROTECT** flag using **pxctl** to get around the grsecurity denial using the following commands:

```
$ paxctl -c <filename-which-was-denied>
$ paxctl -m <filename-which-was-denied>
$ paxctl -v <filename-which-was-denied>
```

Related Links

grsecurity and Related Tools on page 227
IDP XT uses grsecurity to provide a default security policy and tools to customize the policy.

grsecurity RBAC Command Reference on page 229
The grsecurity administration utility (**gradm**) manages the role-based access control (RBAC) system.

Generating a Security Policy for the Package on page 231
Once you have booted a target with the grsecurity RBAC system enabled, you must add a security policy for your applications.

The grsecurity sysctl Interface on page 232
The sysfs in Linux provides an interface for viewing or modifying kernel parameters at runtime. However, if you have a signed SRM rootfs, you can only view, not change, parameters.

Troubleshooting grsecurity on page 232

Examples of common security issues and how to resolve them.

Generating a Security Policy for the Package

Once you have booted a target with the grsecurity RBAC system enabled, you must add a security policy for your applications.

The **/etc/grsec/policy** file provides a default security policy for all pre-installed applications. In order to install new applications locally or remotely, you must add a new policy in the default policy file for all post-installed applications.

- Modify the default security policy.

Options	Description

If you know how to write a policy rule:	Modify **/etc/grsec/policy** and restart grsecurity RBCA system in one of the following ways:
	Method 1:
	`# echo "windriver" | gradm -D`
	`# gradm -E`
	Method 2:
	`# gradm -a admin`
	`# gradm -R`

If you have experience with the grsecurity RBAC policy:	Start full-learning mode, which can generate a reference policy file for you.
	`# gradm -F -L /etc/grsec/learning.logs`
	`# <perform the action you want the RBAC system to learn at least four times>`
	`# gradm -D`
	`# gradm -F -L /etc/grsec/learning.logs -O /etc/grsec/learning.acl`

If you need more information:	Learn how to write your own policy at the following URL: *http://en.wikibooks.org/wiki/Grsecurity/Print_version#Policy_Configuration*

Related Links

grsecurity and Related Tools on page 227
IDP XT uses grsecurity to provide a default security policy and tools to customize the policy.

grsecurity RBAC Command Reference on page 229
The grsecurity administration utility (**gradm**) manages the role-based access control (RBAC) system.

paxctl Reference on page 229
The user-space utility for controlling the PaX flags of executable files is **paxctl**.

The grsecurity sysctl Interface on page 232
The sysfs in Linux provides an interface for viewing or modifying kernel parameters at runtime. However, if you have a signed SRM rootfs, you can only view, not change, parameters.

Troubleshooting grsecurity on page 232

Examples of common security issues and how to resolve them.

The grsecurity sysctl Interface

The sysfs in Linux provides an interface for viewing or modifying kernel parameters at runtime. However, if you have a signed SRM rootfs, you can only view, not change, parameters.

With a signed SRM rootfs, the sysctl interface of grsecurity is locked for security reasons.

```
$ cat /proc/sys/kernel/grsecurity/rwxmap_logging
   1
```

Related Links

grsecurity and Related Tools on page 227
IDP XT uses grsecurity to provide a default security policy and tools to customize the policy.

grsecurity RBAC Command Reference on page 229
The grsecurity administration utility (**gradm**) manages the role-based access control (RBAC) system.

paxctl Reference on page 229
The user-space utility for controlling the PaX flags of executable files is **paxctl**.

Generating a Security Policy for the Package on page 231
Once you have booted a target with the grsecurity RBAC system enabled, you must add a security policy for your applications.

Troubleshooting grsecurity on page 232
Examples of common security issues and how to resolve them.

Troubleshooting grsecurity

Examples of common security issues and how to resolve them.

Dealing with a grsecurity RBAC Denied Issue

Enabling grsecurity's RBAC with the default policy may result in some messages regarding grsec being denied access when you execute certain applications. To eliminate these messages, either: manually add a policy for the particular application or generate a new policy for the application using the learning mode capability of grsecurity as explained above.

- Manually add a policy for the particular application.
- Generate a new policy for the application using the learning mode capability of grsecurity.

Dealing with a grsecurity PAX Denied Issue

You may encounter messages similar to the following:

```
grsec: From 192.168.0.1: denied RWX mprotect off
```

This message means that the grsecurity PaX capability has detected potentially risky code in your application. One recommended approach is to Alternatively, (see the example of disabling MPROTECT flag above) to resolve this issue. Simarlry, .

- You can modify your application code to resolve the issue. (Recommended)

- You can lower the security by disabling the related PaX protection flag. (Not recommended.) For an example, see *paxctl Reference* on page 229.
- You can disable other flags using the **paxctl** tool depending on your error message.

Modifying Files When IMA Appraisal is Active

If IMA appraisal is enabled on the target, you cannot make direct changes to the executable file. However, you can go to your IDP XT build system and add your executable to the PaX exception list. Add a line similar to **m** ***fullPathToExecutable*** (**m** is for the **MPROTECT** flag) to the following file:

***projDir*/layers/wr-idp/wr-srm/templates/feature/grsec_std/meta/files/grsec_pax_exception**

Related Links

grsecurity and Related Tools on page 227
IDP XT uses grsecurity to provide a default security policy and tools to customize the policy.

grsecurity RBAC Command Reference on page 229
The grsecurity administration utility (**gradm**) manages the role-based access control (RBAC) system.

paxctl Reference on page 229
The user-space utility for controlling the PaX flags of executable files is **paxctl**.

Generating a Security Policy for the Package on page 231
Once you have booted a target with the grsecurity RBAC system enabled, you must add a security policy for your applications.

The grsecurity sysctl Interface on page 232
The sysfs in Linux provides an interface for viewing or modifying kernel parameters at runtime. However, if you have a signed SRM rootfs, you can only view, not change, parameters.

PART VI

References

33

IDP Services Reference

This reference describes the IDP XT services available and whether or not they are started by default.

Service Name	Started By Default	Description
appweb	Yes	Starts **appweb**, which is a light weight web server.
boot	Yes	Detects various devices and load drivers for those devices.
crontab	Yes	Translates UCI config to the **crontab** configuration file **/etc/crontabs/root**.
custom-user-startup	Yes	Serves as a placeholder for user's customized startup commands. It is empty by default.
dcomd	No	Provides DCOM service used by OPC servers.
daserver1	No	Demo server that tests basic OPC function.
daserver2	No	Demo server that tests all data types supported by OPC.
daserver3	No	Demo server that stress tests OPC.
dmesgbackup	Yes	Backs up **dmesg** to a **.gz** file.
dnsmasq	Yes	Starts **dnsmasq**, which implements DNS and DHCP functions for the gateway device.
firewall	Yes	Starts **firewall** for the system with pre-configured rules.
ipsec	Yes	To initilize the **ipsec** configuration as specified by **webif**.

Service Name	Started By Default	Description
miniupnpd	No	Starts **miniupnpd** which is a deamon that implements the UPnP Internet Gateway Device (IGD) protocol. It gives a gateway the ability to do automatically port mapping.
mqtt	Yes	Starts a **mqtt** server. MQTT is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport.
multiwan	No	Starts **multiwan**, which is a deamon that manage the prioroty of upsteam interfaces according to a user configuration and determines the active upsteam interface in the system according to priority and connection status.
network	Yes	Starts **netifd** and **wifi**. **netifd** is a deamon in the system that deals with various network interface configurations.
oma_dmc	No	Starts OMA-DM agent, if the OMA-DM agent is included in image.
snmpd	Yes	Starts **snmpd**, which is a network management deamon that implements Simple Network Management Protocol.
scsrvc	Yes	McAfee Solidifier service
tcsd	Yes	Starts the **tcsd** deamon, which enables applications to use the Trousers APIs to communicate with a TPM chip.
timezone	Yes	Sets the timezone configuration for the system.
ubus	Yes	Starts **ubusd**, which is an RPC DAEMON server that enable interprocess communication.
webif	Yes	Starts **webif**, which is a Web user interface for configuring the system.
webiffirewalllog	Yes	Enables firewall logging if it is configured to be enabled.

34

IDP Packages Not Included in Any Layer

Most IDP XT packages are included in a layer. However, some packages are not; they must be included by building the package.

The following is a list of additional Wind River Linux packages provided by IDP XT but not part of any IDP XT layer/feature. To include a package into your rootfs package list, do:

```
$ make -C build <pkgName>.addpkg
```

Package	Description
conntrack-tools	a set of userspace tools for Linux that allow system administrators to interact with the Connection Tracking System, the module which provides stateful packet inspection for iptables. It includes the userspace daemon conntrackd and the commandline interface conntrack.
deltarpm	RPM tool to create rpm delta package.
ecmh	a networking daemon that acts as a full IPv6 MLDv1 and MLDv2 Multicast Router.
fuse	With FUSE it is possible to implement a fully functional filesystem in a userspace program.
igmpproxy	an IGMP (Internet Group Management Protocol) snooper/proxy daemon for routing multicast packets across networks.
iperf	a tool to measure maximum TCP bandwidth, allowing the tuning of various parameters and UDP characteristics.

Package	Description
ipset	IP Address Set, a framework inside the Linux 2.4.x and 2.6.x kernel, which can be administered by the ipset utility. Depending on the type, currently an IP set may store IP addresses, (TCP/UDP) port numbers or IP addresses with MAC addresses in a way, which ensures lightning speed when matching an entry against a set.
isakmpd	Snoops MLDv1/MLDv2 requests and forwards them onto a given interface.
keynote	Keynote tool and library.
lame	LAME Ain't an MP3 Encoder.
libcli	shared library for including a Cisco-like command-line interface into other software.
libdlna	the reference open-source implementation of DLNA (Digital Living Network Alliance) standards. (Requires **ffmpeg**, which is not included in IDP XT.)
libexosip2	High level Session Initiation Protocol (SIP) library.
libgsm	GSM Audio Library.
libosip2	Session Initiation Protocol (SIP) library.
libupnp	The portable SDK for UPnP* Devices (libupnp) provides developers with an API and open source code for building control points, devices, and bridges that are compliant with Version 1.0 of the Universal Plug and Play Device Architecture Specification.
mipv6-daemon-umip	The mobile IPv6 daemon allows nodes to remain reachable while moving around in the IPv6 Internet
mldproxy	Snoops MLDv1/MLDv2 requests and forwards them onto a given interface.
mowrappers	mowrappers is the wrapper to help to access device configurations and deploy software on it.
net-snmp	Various tools relating to the Simple Network Management Protocol.
ntfs-3g,ntfsprogs	The NTFS-3G driver is an open source, freely available NTFS driver for Linux with read and write support.
orc	The Oil Runtime Compiler.
quagga	a routing software suite, providing implementations of OSPFv2, OSPFv3, RIP v1 and v2, RIPng and BGP-4 for Unix platforms

Package	Description
schroedinger	schrodinger is a cross-platform implementation of the Dirac video compression specification as a C library.
sshfs-fuse	This is a filesystem client based on the SSH File Transfer Protocol using FUSE.
tftp-hpa, **tftp-hpa-server**	an enhanced version of the BSD TFTP client and server.
xinetd	an open-source super-server daemon which runs on many Unix-like systems and manages Internet-based connectivity. It offers a more secure extension to or version of inetd, the Internet daemon.

35

Packages Required for SST

You must install certain tools in order to use SST.

The following table lists the names of the Wind River Linux packages.

Package	Description
awk	a pattern scanning and processing language.
bash	an sh-compatible command language interpreter that executes commands read from the standard input or from a file. Bash also incorporates useful capabilities from the Korn and C shells (ksh and csh).
bc	a language that supports arbitrary precision numbers with interactive execution of statements.
bzip2	compresses files using the Burrows-Wheeler block sorting text compression algorithm and Huffman coding.
cat	concatenates FILE(s) or standard input to standard output.
cp	copies SOURCE to DEST or multiple SOURCEs to DIRECTORY.
cpio	copies files to and from archives.
cut	prints selected parts of lines from each FILE to standard output.
date	displays the current time in the given FORMAT or sets the system date.
diff	compares the contents of the two files FROM-FILE and TO-FILE.
echo	echoes STRING(s) to standard output.
expr	evaluates expressions.
file	tests each argument in an attempt to classify it. There are three sets of tests performed in this order: filesystem tests, magic number tests, and language tests.

Package	Description
find	searches the directory tree rooted at each given file name by evaluating the given expression from left to right according to the rules of precedence.
grep	searches the named input FILEs (or standard input if no files are named or the file name "**-**" is given) for lines containing a match to the given PATTERN.
head	prints the first 10 lines of each FILE to standard output. For more than one FILE, **head** precedes each with a header giving the file name.
hexdump	a filter which displays the specified files, or the standard input if no files are specified, in a user specified format.
kill	sends the specified signal to the specified process or process group. If no signal is specified, the TERM signal is sent.
md5sum	prints or checks MD5 (128-bit) checksums.
mkdir	creates one or more directories if they do not already exist.
mktemp	takes the given filename template and overwrites a portion of it to create a unique filename.
mv	renames SOURCE to DEST or moves one or more source files to DIRECTORY.
objcopy	copies and translates object files.
openssl	a cryptography toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) network protocols and the related cryptography standards required by them.
perl	a language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information. It is also a good language for many system management tasks.
printf	prints ARGUMENT(s) according to FORMAT.
rm	removes each specified file. By default it does not remove directories.
rpm2cpio	converts the **.rpm** file specified as a single argument to a cpio archive on standard out.
sed	a stream editor. A stream editor is used to perform basic text transformations on an input stream (a file or input from a pipeline).
sha1sum	prints or checks SHA1 (160-bit) checksums.
sha256sum	prints or checks SHA256 (256-bit) checksums
tar	an archiving program designed to store and extract files from an archive file known as a tar file.

Package	Description
touch	updates the access and modification times of each FILE to the current time.
tr	translates squeezes and/or deletes characters from standard input writing to standard output.
wc	prints newline, word, and byte counts for each FILE plus a total line if more than one FILE is specified.
which	takes one or more arguments. For each of its arguments, it prints to stdout the full path of the executables that would have been executed if this argument had been entered at the shell prompt.
xargs	reads items from the standard input delimited by blanks or newlines and executes the command (default is **/bin/echo**) one or more times with any initial arguments followed by items read from standard input. Blank lines on the standard input are ignored.
xxd	creates a hex dump of a given file or from standard input. It can also convert a hex dump back to its original binary form. It allows the transmission of binary data in a mail-safe ASCII representation, but has the advantage of decoding to standard output. **xxd** can also be used to perform binary file patching.

36

Preparing USB Boot Media

Preparing a USB drive to use for booting requires configuring the drive and partitions.

NOTE: The **mkfs.ext3** utility used to format the USB Flash drive is available with **glibc-idp**; it is not included in **glibc-idp-small**.

Step 1 Insert the USB Flash drive into your Linux host machine.

Step 2 Run **fdisk**.

```
$ su root
<Enter root password when prompted>
# fdisk /dev/sdb
```

The system asks you for a series of options.

Step 3 Enter the requested information.

a) Type **p** to print the partition table.
b) Type **d** to delete the old partition.
c) Type **n** to create a new partition.
d) Type **p** to make the new partition the primary partition.
e) Type **1** for the partition number.
f) Type **ENTER** to accept the defaults for the remaining options.
g) Type **w** to write the new partition table to the USB drive.

```
# fdisk /dev/sdb

WARNING: DOS-compatible mode is deprecated. It's strongly recommended to
         switch off the mode (command 'c') and change display units to
         sectors (command 'u').

Command (m for help): d
Partition number (1-4): 1

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)p
Partition number (1-4):
1
First cylinder (1-15484, default 1):
Using default value 1
Last cylinder, +cylinders or +size{K,M,G} (1-15484, default 15484):
Using default value 15484
```

```
Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
#
```

Step 4 Create an ext3 file system on the new partition and mount the USB Flash drive on your Linux host.

```
# mkfs.ext3 -I 128 /dev/sdb1
mke2fs 1.41.10 (10-Feb-2009)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
991232 inodes, 3963900 blocks
198195 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=4060086272
121 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208

Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 36 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

NOTE: When you format a USB device, you must include the **-I 128** option. Because grub-0.97 is legacy software, it cannot handle an inode size larger than 128 bytes. However, the **mkfs.ext3** tool included in most modern Linux distributions uses 256 bytes as the default value of inode. Without this option, grub-0.97 cannot recognize USB devices formatted with **mkfs.ext3**.

37

Configuring HDMI Ports for X Window Support

Learn how to configure the HDMI ports on your board to prevent a blank screen on the secondary HDMI port when running the QT Demo.

The Advantech UTX-3115 board supports HDMI.

Some displays may not be compatible with either HDMI port on the device. Incompatible displays, such as the Panasonic TC-42G25, will just show a black (blank) screen. If you are running the QT Demo, and you find that a display connected to the second HDMI port is dark and does not operate the first time that X is started, use this procedure to enable the second HDMI port:

Step 1 Configure and build your platform project in the standard way.

Include **feature/intel-emgd** in the project configure line.

For more information, see:

Building Platform Projects for Advantech UTX-3115 Boards on page 126

Step 2 Boot the target to access the file system, or press **CTRL+ALT+F2** to change to the text login terminal.

Step 3 Back up the **/etc/X11/xorg.conf** file.

Step 4 Rename the **/etc/X11/xorg.conf.dual_display** file to **/etc/X11/xorg.conf**

```
$ mv /etc/X11/xorg.conf.dual_display /etc/X11/xorg.conf
```

Step 5 Save the file and reboot the target file system or restart X.