



AN INTEL COMPANY

WIND RIVER® INTELLIGENT DEVICE PLATFORM XT

PROGRAMMER'S GUIDE

3.1



Copyright Notice

Copyright © 2016 Wind River Systems, Inc.

All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means without the prior written permission of Wind River Systems, Inc.

Wind River, Simics, Tornado, and VxWorks are registered trademarks of Wind River Systems, Inc. The Wind River logo is a trademark of Wind River Systems, Inc. Any third-party trademarks referenced are the property of their respective owners. For further information regarding Wind River trademarks, please see:

www.windriver.com/company/terms/trademark.html

This product may include software licensed to Wind River by third parties. Relevant notices (if any) are provided in your product installation at the following location:

installDir/legal-notices/

Wind River may refer to third-party documentation by listing publications or providing links to third-party websites for informational purposes. Wind River accepts no responsibility for the information provided in such third-party documentation.

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 website:

www.windriver.com

For information on how to contact Customer Support, see:

www.windriver.com/support

Intelligent Device Platform XT
Programmer's Guide, 3.1

19 February 2016

Contents

PART I: INTRODUCTION TO IDP

1 Introduction and Overview	3
Wind River Intelligent Device Platform Overview	3
IDP User Roles	4
Where to Find Information	6
Accessing Documentation	7
2 Architecture	9
Intelligent Device Platform Architecture	9
Systems in Development	10
Deployed Devices	11
IDP Features	11
3 IDP Security	17
Application Integrity Measurement (Tamper-proof File System)	17
Secure Boot	17
McAfee Embedded Control for Wind River	18
Standalone SRM Signing Tool	18
The grsecurity Tool	18
Encrypted Storage	19
4 IDP Connectivity	21
BlueZ Bluetooth Stack	21
Exegin ZigBee Stack	22
VPN Connections	22
MQTT	23
Multiwan	23
Wind River OPC for Wind River Linux	24
5 IDP Management	25
OneAgent TR-069 Agent	25
LuCI and Wi-Fi Connections	26
OneAgent OMA-DM Agent and MO Wrappers	26

PART II: KEY-RELATED TASKS

6 About Key Management	29
7 Key Management for Vendors	31
Key Management for Vendors	31
Installing Required Packages for SST	32
Generating a New Owner Key and Certificates	33
Generating a New Vendor Key and Certificate	34
Signing a Flash Image	35
Signing Shim Images	36
Signing Boot Loaders	37
Signing Kernels	38
Signing the initramfs File	38
Signing the rootfs File	39
Signing Application Folders	39
Signing a Single RPM	40
Signing Multiple RPMs	40
Embedding Keys for Quark Boards	41
SST Reference	42

PART III: SYSTEM OWNER TASKS

8 Introduction to System Owner Tasks	45
9 McAfee Embedded Control	47
10 Integrating OpenSSL and TPM	49
About TPM and Key Protection	49
Preparing to Use TPM	50
Creating a Key Using TPM Hardware	51
Wrapping a Software Key Into TPM Storage	52
Testing the OpenSSL TPM Engine Integration	52
About the Open Source Toolkit for SSL/TLS (OpenSSL)	54
11 LuCI Router Configuration	57
About LuCI	57
LuCI Interface Main Menus	57
LuCI Interface Default Settings	58

LuCI Interface Prerequisites	58
Launching and Accessing the LuCI Interface	59
Saving Changes in the LuCI Interface	61
Network Menu	62
Changing WiFi Mode from AP to Client Using LuCI	63
Changing WiFi Mode from Client to AP Using LuCI	64
12 Secure Repository	67
RPM Repository Server	67
Adding a Local Repository	67
Remote Repositories	68
Installing Server Software	69
Setting Up the Web Server	69
Starting the Web Server	71
Managing Repositories	71
Adding a Remote Repository	71
Removing a Repository	72
Listing Repositories	72
Managing RPM Packages	73
Adding an RPM Package to the Device	73
Listing the RPM Packages Installed on the Device	73
Removing an RPM Package from the Device	73
13 Tamper-Proof File System	75
Application Integrity Measurement	75
Using the Tamper-Proof File System	76
14 Sign and Update RPM Packages, Kernel Images, GRUB Boot Loader, and initramfs Images	79
Generating and Installing a Signed RPM package with SST	79
Signing an RPM Package with GPG	80
Generating and Updating a Signed Kernel Image	83
Generating and Updating a Signed Boot Loader Image	84
Generating and Updating a Signed initramfs Image	85

PART IV: DEVICE DEVELOPMENT VENDOR TASKS

15 Introduction to Device Development Tasks	89
16 Building and Booting	91
Preparing to Build and Boot IDP	91
About the wrenv.sh Script	92

About the deploytool Script	92
Building Platform Projects for Intel Quark Boards	93
Deploying Images to Intel Quark Boards	94
Building Platform Projects for Intel Bay Trail and Intel Haswell Boards	95
Configuring the BIOS for Intel Bay Trail and Intel Haswell Boards	97
Deploying Images to Intel Bay Trail and Intel Haswell Boards	98
Updating BIOS Images on GIGABYTE GB-BXBT-3825 Boards and other Intel Bay Trail Boards	99
Updating BIOS Images on ADLINK MXE-5401 Boards	99
Updating Flash Firmware for Intel Quark Boards	100
Programming SPI Flash Memory Using an SF-100 Programmer	101
Updating Flash Firmware Using Capsule Update in an EFI Shell	102
Updating Flash Firmware Using Capsule Update in Linux	102
Migrating Intel Quark Flash Firmware from IDP XT 2.0.x to IDP XT 3.1.x	103
About the GRUB Boot Menu Information	105
Updating the Target System	108
Using the IA Recovery Image	109
IDP Preconfigured Profiles	111
Platform Boot Time Optimizations	112
Configuring the Target at Boot Time	112
17 Alternative Booting Methods	115
SRM and Alternative Booting Methods	115
Secure Booting	115
Performing a Secure Boot on Cross Hill and Clanton Hill Boards	115
Performing a Secure Boot Using UEFI on Intel Baytrail Boards	116
Performing a Secure Boot Using UEFI on Intel Haswell Boards	118
Performing a Verified Boot	119
18 Configuring IDP Features	121
Layers and Features	121
About Configuring Layers	123
Inspecting Layer Contents	125
About Configuring Default Features	125
About Configuring Non-Default Features	126
Non-Default Features Included in the idp Rootfs	126
About the Secure Remote Management Layer	127
19 Installing Tools for Application Development and Control	129
Including Bluetooth in a Platform Project	129
Enabling 3G WWAN	130
Enabling IMA Appraise	131
Configuring OpenJDK	132

Installing OpenJDK	132
Example: Integrating Custom Java Code Into IDP	132
Rebuilding the Java Run-Time Environment from Source	136
OSGi Development Workflow	136
Installing the ProSyst Smart Home SDK	137
Enabling Eclipse for ProSyst Smart Home Development	138
Creating an OSGi Platform Image	139
Exporting an OSGi Platform Image	140
Deploying an OSGi Platform Image on a Target	140
Installing Sqlite3	141
Installing MQTT and Lua	142
Configuring Encrypted Storage	142
Encrypted Storage Prerequisites	142
Enabling Encrypted Storage	143
Setting Up the dm-crypt Partition with a Loop Device	144
Testing Encrypted Storage with a Loop Device	145
Setting Up the dm-crypt Partition with a USB Key	147
Testing Encrypted Storage with a USB Key	148
Installing OneAgent TR-069	149
Installing OMA-DM	150
Configuring PaX in the Kernel	151
Installing Wind River OPC	151
20 Customizing LuCI	153
About Customizing LuCI	153
21 Updating WPAN Firmware for Intel Quark Boards	155

PART V: APPLICATION DEVELOPMENT VENDOR TASKS

22 Application Development	159
23 Exegin ZigBee Stack	161
About the ZigBee Stack	161
Setting Up a ZigBee Network	161
24 Wind River OpenJDK	163
About OpenJDK	163
Basic OpenJDK Command Reference	163
25 OSGi Development with the MBS Smart Home SDK	165
OSGi Development with the mBS Smart Home SDK	165

Developing with OSGi	166
26 Sqlite3 Database	169
Using Sqlite3	169
Sqlite3 Command Reference	169
Sqlite3 Data Element Reference	170
Sqlite3 Examples	171
27 MQTT and Lua	175
About MQTT and Lua	175
The Lua Language	175
Examples: Using the Lua Program Examples	176
Starting the MQTT Broker	177
About Publishing and Subscribing to Messages	177
Example: Multiple Messages	177
Example: Single Message	178
28 Encrypted Storage	179
29 OneAgent TR-069 Agent	181
30 Works Systems OneAgent OMA Agent	183
About the OMA-DM Agent	183
About MO Wrappers	184
31 The grsecurity Tool	187
grsecurity and Related Tools	187
grsecurity RBAC Command Reference	188
paxctl Reference	188
Generating a Security Policy for the Package	190
The grsecurity sysctl Interface	190
Troubleshooting grsecurity	191
32 Authentication, Authorization, and Auditing	193
About Authentication, Authorization, and Auditing Examples	193
Example: Client Application	193
Example: Authentication with Microsoft Active Directory	195
Example: Authentication with an LDAP Server	196
Example: Authorization with a TACACS+ Server	197
Example: Authentication with a RADIUS Server	198
Example: Using LDAP as a Name Service	199

PART VI: REFERENCES

33 IDP Services Reference	203
34 IDP Packages Not Included in Any Feature	205
35 Packages Required for SST	207

PART I

Introduction to IDP

Introduction and Overview.....	3
Architecture.....	9
IDP Security.....	17
IDP Connectivity.....	21
IDP Management.....	25

1

Introduction and Overview

[Wind River Intelligent Device Platform Overview](#) 3

[IDP User Roles](#) 4

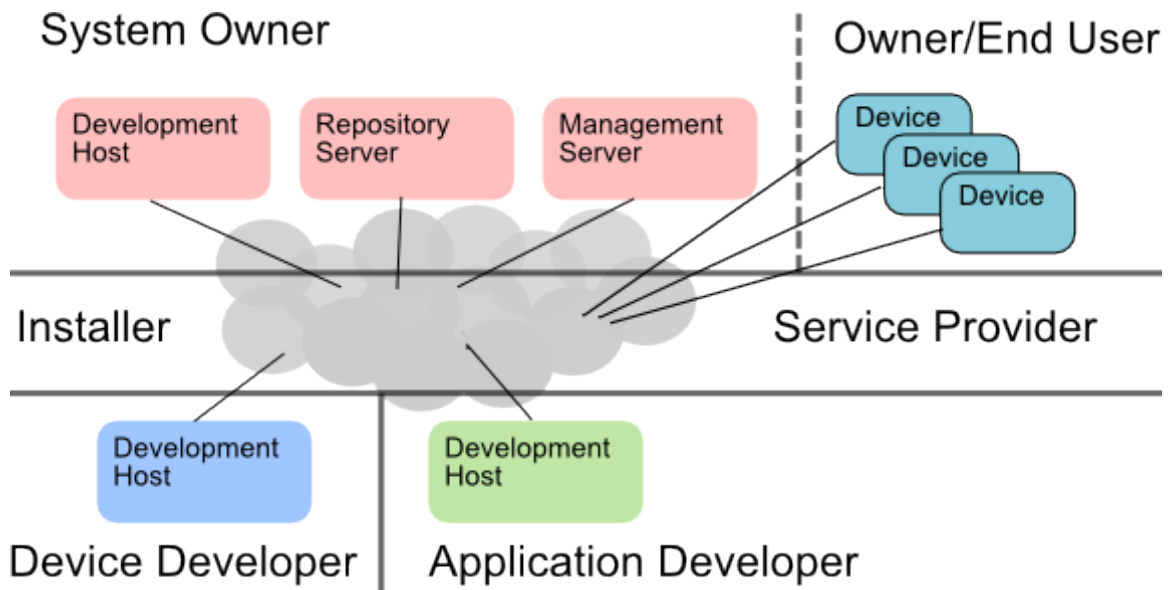
[Where to Find Information](#) 6

[Accessing Documentation](#) 7

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 with Wind River Intelligent Device Platform XT

- Wind River Linux
- Wind River Workbench
- McAfee Embedded Control
- BSPs for the following boards:
 - Cross Hill
 - Clanton Hill
 - Galileo Gen 2
 - GIGABYTE GB-BXBT-3825
 - ADLINK MXE-5401

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.
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

Company or Employee	Roles
CableCo customer	end user

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 Knowledge Library.

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, 7.0

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

Wind River Linux User's Guide, 7.0

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 7.0 Version), 4.0

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, 7.0*.

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.

Accessing Documentation

You can access IDP XT documentation through the Knowledge Library or Workbench.

- Access documentation through the Workbench main menu.
Select **Help > Help Contents > Wind River Documentation**.
- 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: <i>installDir/wrlinux-7/docs/docs/extensions/eclipse/plugins/com.windriver.ide.doc.wr_intelligent_device_platform_XT_3.1/mc_afee_documents/mec_ug_en-us.pdf</i>
HTML Versions	Point your Web browser to the index.html file, for example: <i>installDir/wrlinux-7/docs/docs/extensions/eclipse/plugins/com.windriver.ide.doc.wr_intelligent_device_platform_XT_3.1/wr_idp_xt_programmers_guide_31/index.html</i>

- Access documentation on the Wind River Knowledge Library at knowledge.windriver.com.
Log on to Knowledge Library and select:
Products > Internet of Things > Intelligent Device Platform XT.

2

Architecture

Intelligent Device Platform Architecture	9
Systems in Development	10
Deployed Devices	11
IDP Features	11

Intelligent Device Platform Architecture

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

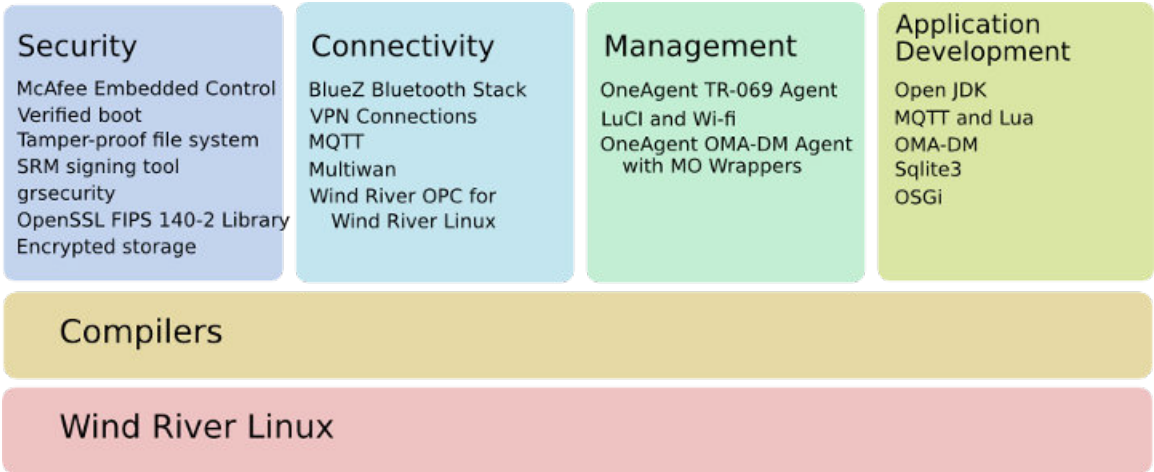
The IDP XT installation contains layers that include configuration files, templates, and code that extend your development possibilities. Using the **--enable-addons=wr-idp** option in your configuration includes the layers that add or extend platform project capabilities. Different capabilities are 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 121.

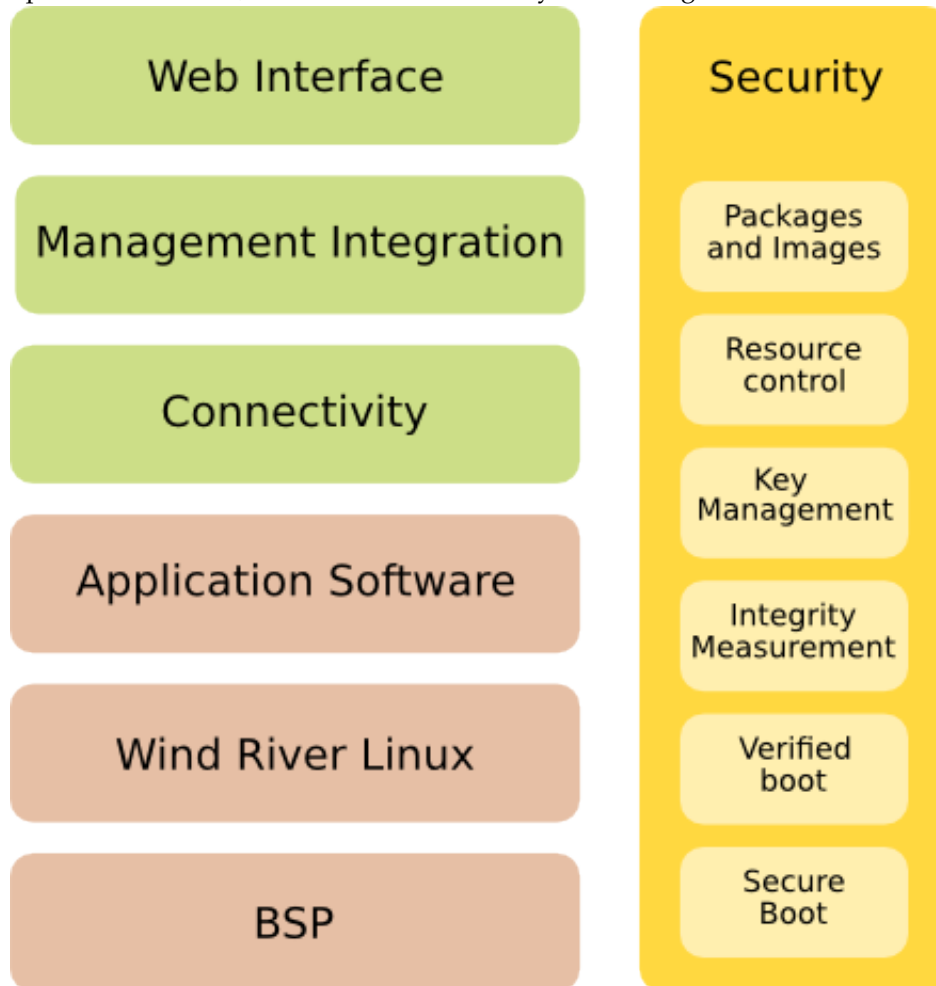
Systems in Development

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



Deployed Devices

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



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 121.

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

Layers	Description	Features	Description
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. Only supported on Cross Hill.
wr-ma	ISV layer for the McAfee Agent.	default	Adds the client-side component that enables the target to securely connect to the McAfee ePolicy Orchestrator.

Layers	Description	Features	Description
wr-mcafee	ISV layer for McAfee Embedded Control Pro, 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 security capabilities.
wr-mcafee-essential	ISV layer for McAfee Embedded Control Essential, which provides a subset of the capabilities of McAfee Embedded Control Pro.	default	Adds a subset of McAfee Application Control.
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.
		OpenSSL TPM Engine	Uses TPM hardware to provide secure key management. No separate feature template. Not supported on GIGABYTE GB-BXBT-3825 and Clanton Hill.
		Secure Package Management	Prevents installing RPM packages without authorized signatures. No separate feature template.
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 of Wind River Linux that work with a newer version of device firmware. This feature is specifically for the Intel Dual Band Wireless-AC 7260, AzureWave AW-NB159H, Marvell 88W8897 [AVASTAR] Murata, and PARKLAN WPEA-251N cards. Use of this feature with other WLAN/Bluetooth modules is not supported and therefore, not recommended.
		bluetooth	Provides the BlueZ Bluetooth software implementation.
		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. Not supported on Cross Hill and Clanton Hill.
		grsec	Provides grsecurity and related tools
		ipsec_vpn	Adds the strongSwan IPsec VPN implementation to your platform project.
		l2tp	Adds the L2TP VPN implementation to your platform project.
		luci	Adds the LuCI interface, a web browser-based interface for configuring network connections and checking the services running on the target.
		mqtt	Provides client/server tools for the MQTT protocol and utilities 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 <i>Wind River OPC for IDP Programmer's Guide</i> .
		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	Enables you to create bootable recovery media.
		vlan	Adds 802.1Q protocol and configuration support for your platform project
wr-ima-appraise	ISV layer for application Integrity Measurement	default	Uses IMA Appraisal to prevent loading applications and libraries without authorized signatures.

3

IDP Security

Application Integrity Measurement (Tamper-proof File System)	17
Secure Boot	17
McAfee Embedded Control for Wind River	18
Standalone SRM Signing Tool	18
The grsecurity Tool	18
Encrypted Storage	19

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 131.

Secure Boot

IDP XT offers hardware-based secure boot.

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 Pro and McAfee Embedded Control Essential for Wind River allow you to configure McAfee embedded products for use with the Wind River target platform.

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

- McAfee Application Control
- McAfee Change Control

McAfee Embedded Control Essential provides a subset of the capabilities of McAfee Application Control, including the following:

- Deny-exec, checksum, and integrity features
- Standalone configuration; managed configuration is not supported
- ELF binaries will be solidified; script, class, and jar files will not be solidified
- Secure Hash Algorithm 256 (SHA256) support; SHA1 support is unavailable
- ELF binary files will be automatically whitelisted during an Update window
- `/home` is added as a trusted path and excluded from solidification

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

- [McAfee Embedded Control](#) on page 47
- *McAfee Embedded 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
GIGABYTE GB-BXBT-3825
ADLINK MXE-5401

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 187.

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 179.

4

IDP Connectivity

BlueZ Bluetooth Stack	21
Exegin ZigBee Stack	22
VPN Connections	22
MQTT	23
Multiwan	23
Wind River OPC for Wind River Linux	24

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 129

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 ZigBee 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 161
- 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.pptppn.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 broker 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 142

[About MQTT and Lua](#) on page 175

Multiwan

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

The multiwan utility 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 LuCI interface. On the **Network** tab, select **Multi-WAN**. On the Multi-WAN page, select **Enable**. Click **Save & Apply**.

You can increase the polling interval of the **multiwan** process to reduce CPU usage. On the Multi-WAN page, modify the value of **Health Monitor Interval** and then click **Save & Apply**.

If you add a new interface to monitor through the LuCI interface, the default value of **Health Monitor 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 151
- *Wind River OPC User's Guide (Wind River Linux Version)*

5

IDP Management

OneAgent TR-069 Agent	25
LuCI and Wi-Fi Connections	26
OneAgent OMA-DM Agent and MO Wrappers	26

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 26.

LuCI and Wi-Fi Connections

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

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

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

```
/etc/  
/www/  
/usr/lib64/lua/5.1/luci
```

View the LuCI debug messages with the following command:

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

Confirm that the **nginx** service is running with the following command:

```
# ps -ef | grep nginx
```

For additional information on LuCI, see [About LuCI](#) on page 57.

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 150

[About the OMA-DM Agent](#) on page 183

[About MO Wrappers](#) on page 184

PART II

Key-Related Tasks

About Key Management.....	29
Key Management for Vendors.....	31

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.

Key Management for Vendors

Key Management for Vendors	31
Installing Required Packages for SST	32
Generating a New Owner Key and Certificates	33
Generating a New Vendor Key and Certificate	34
Signing a Flash Image	35
Signing Shim Images	36
Signing Boot Loaders	37
Signing Kernels	38
Signing the initramfs File	38
Signing the rootfs File	39
Signing Application Folders	39
Signing a Single RPM	40
Signing Multiple RPMs	40
Embedding Keys for Quark Boards	41
SST Reference	42

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.

After you build your platform project, a softlink to SST is available at the top level of your platform project directory:

projDir/SST

Usage Notes for SST

- SST supports the **intel-baytrail-64**, **intel-haswell-64**, and **intel-quark** BSPs.
- Using the **sign-all** subcommand requires root privileges.
- The target boots if the boot loader is not signed. However, if the boot loader 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.

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 207

- 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
```

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] \
  [--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 (".")
role	Trust role in SRM. Can only be vendor or owner .	N/A

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



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=.]
```

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 (".")
priv-key	Your existing private key.	N/A

Option	Description	Default Value
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

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

Signing a Flash Image

You can use the SST **sign-flashimage** command to update and inject the UEFI variables **PK**, **KEK**, and **DB** into the flash image for an Intel Quark board.

The variables represent the following:

DB (Authorized image signature database)

Controls what images are trusted when verifying loaded images.

KEK (Key Exchange Key)

Establishes a trust relationship between the operating system and the platform firmware.
Required to update the DB.

PK (Platform Key)

Establishes a trust relationship between the platform owner and the platform firmware
Required to update the KEK.

The command options are as follows:

Option	Description	Default Value
verbose	Open or close the signing trace. The value can be yes or no .	no
pk	The variable PK for UEFI secure boot.	owner-cert.pem file in the SST current directory
kek	The variable DB for UEFI secure boot.	vendor-cert.pem file in the SST current directory
db	The variable DB for UEFI secure boot.	vendor-private.pem file in the SST current directory

Sign the flash image.

```
$ ./SST sign-flashimage --verbose=no \
--pk=./ownerE-cert.pem \
--kek=./vendorE-cert.pem \
--db=./vendorE-cert.pem\
./flash.img
```

When the command completes successfully, the **flash.img** file is updated.

Signing Shim Images

Use the **SST sign-shim** command to sign shim images with the vendor certificate and private key.

The shim image is the first stage loader that loads the grub-efi boot loader and verifies its signature. You can sign the image with your own keys and certificates instead of the demo keys provided with IDP XT.

The options are as follows:

Option	Description	Default Value
verbose	Open or close the signing trace. Value can be yes or no .	no
vendor-cert	The device vendor certificate that was used to sign the shim image.	vendor-cert.pem file in the SST current directory
priv-key	The private key which was used to sign the shim image.	vendor-private.pem in the current directory

Sign the shim file.

```
$ ./SST sign-shim --verbose=no \
--vendor-cert=./vendorE-cert.pem \
--priv-key=./vendorE-private.pem\
./shim.img
```

When the command completes successfully, the **shim.img** file is updated.

Signing Boot Loaders

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

The options are as follows:

Option	Description	Default Value
verbose	Open or close the signing trace. Value can be yes or no .	no
grub-efi	Indicates that the target file is the grub.efi boot loader. Value can be yes or no .	yes
owner-cert	The root certificate should be injected into the boot loader file to sign. Not applicable when signing the boot loader configuration file.	owner-cert.pem file in the SST current directory
vendor-cert	The device vendor certificate that was used to sign grub.efi for the BIOS or the grub.conf configuration file.	vendor-cert.pem file in the SST current directory
priv-key	The private key that was used to sign grub.efi for the BIOS or the grub.conf configuration file.	vendor-private.pem in the current directory

- Sign the boot loader.

```
$ ./SST sign-bootloader --verbose=no --grub-efi=yes \
--owner-cert=./ownerE-cert.pem \
--vendor-cert=./vendorE-cert.pem \
--priv-key=./vendorE-private.pem\
./grub.efi
```

When the command completes successfully, the **grub.efi** file is updated.

- Sign the boot loader configuration file.

```
$ ./SST sign-bootloader --verbose=no --grub-efi=no \
--vendor-cert=./vendorE-cert.pem \
--priv-key=./vendorE-private.pem\
./grub.conf
```

When the command completes successfully, the **grub.conf.auth** file is generated.

Signing Kernels

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

Sign the kernel.

```
$ ./SST sign-kernel --verbose=no \  
--priv-key=./vendorE-private.pem \  
--vendor-cert=./vendorE-cert.pem ./bzImage
```

When the command completes successfully, your kernel image file is updated.

The options are as follows:

Option	Description	Default Value
verbose	Open or close the signing trace. Value can be yes or no .	no
priv-key	The private key which was used to sign the kernel image.	vendor-private.pem in the current directory
vendor-cert	The device vendor certificate should be injected into the kernel file.	vendor-cert.pem file in the SST current directory

Signing the initramfs File

Use the **SST sign-initramfs** command to sign the initramfs image.

The options are as follows:

Option	Description	Default Value
verbose	Open or close the signing trace. Value can be yes or no .	no
vendor-cert	The device vendor certificate that was used to sign the initramfs image.	vendor-cert.pem file in the SST current directory
priv-key	The private key that was used to sign the initramfs image.	vendor-private.pem in the current directory

Sign the initramfs.

```
$ ./SST sign-initramfs --verbose=no \  
--priv-key=./vendorE-private.pem \  
--vendor-cert=./vendorE-cert.pem ./initramfs.img
```

When the command completes successfully, the **initramfs.img.auth** signature file is created.

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.

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.	srm-enabled-images.tar.bz2 file in current directory
priv-key	The private key which was used to sign the kernel image.	vendor-private.pem in the current directory.
owner-cert	The root certificate should be injected into the bootloader.	owner-cert.pem file in the SST current directory
vendor-cert	The device vendor certificate which should be injected into the kernel file.	vendor-cert.pem file in the SST current directory

Sign the tar file.

The following command uses the image for the **intel-baytrail-64** BSP as an example:

```
$ sudo ./SST sign-all --mode=tarball \
--owner-cert=./ownerE-cert.pem --verbose=no \
--vendor-cert=./vendorE-cert.pem \
--priv-key=./vendorE-private.pem \
--output=./signed-images.tar.bz2 \
./wrlinux-image-idp-intel-baytrail-64.tar.bz2
```

When the command completes successfully, the rootfs tar file **wrlinux-image-idp-intel-baytrail-64.tar.bz2** is signed and the resulting output file is stored as **signed-images.tar.bz2**.

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

Option	Description	Default Value
output-list	The signature list of the application binary in the <i>app_folder</i> 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.

Signing a Single RPM

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

You only need to sign an RPM when your platform project is configured with the **--with-layer=wr-ima-appraise**.

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
priv-key	The private key that was used to sign the rpm package.	vendor-private.pem in the current directory.

Sign the RPM.

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

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

Signing Multiple RPMs

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

You only need to sign RPMs when your platform project is configured with the **--with-layer=wr-ima-appraise**.

The options are as follows:

Option	Description	Default Value
verbose	Open or close the signing trace. Value can be yes or no .	no

Option	Description	Default Value
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 <i>projDir/layers/wr-idp/wr-srm/files/keys/vendor-private.pem</i>

Sign all the RPMs in a directory.

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

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

Embedding Keys for Quark Boards

You can use the SST **sign-flashimage** command to implement a security policy for an Intel Quark board using keys provided by another vendor.

You can use the SST **sign-flashimage** command to embed the keys on a firmware image without the involvement of the Device Deployment Vendor.

The options are as follows:

Option	Description	Default Value
verbose	Open or close the signing trace. The value can be yes or no .	No
db	DER formatted database used to authenticate the signed image.	Optional
kek	DER formatted Key Exchange Key which is used to modify db.	Optional
pk	DER formatted Platform Key which is used to modify KEK.	Optional



NOTE: You must specify at least one of **-pk**, **-kek**, or **-db**.

For example:

Enter the following to use all three options.

```
$ ./SST sign-flashimage \
--pk=./owner-cert.pem \
--kek=./vendor-cert.pem \
```

```
--db=./vendor-cert.pem \  
projectDir/export/images/Flash-platform-8M-feature.bin
```

When the command completes successfully, the firmware image **Flash-platform-8M-feature.bin** contains the specified keys to provision the policy of UEFI Secure Boot during the first boot.

Postrequisites

The chip must then be flashed using a flash programmer. For more information, see [Programming SPI Flash Memory Using an SF-100 Programmer](#) on page 101.

Refer to [Signing Boot Loaders](#) on page 37 if you need to use SST to sign **grub-efi** with the keys.

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)
sign-kernel	process Linux kernel images (uImage or bzImage)
sign-initramfs	process initramfs images
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.

PART III

System Owner Tasks

Introduction to System Owner Tasks.....	45
McAfee Embedded Control.....	47
Integrating OpenSSL and TPM.....	49
LuCI Router Configuration.....	57
Secure Repository.....	67
Tamper-Proof File System.....	75
Sign and Update RPM Packages, Kernel Images, GRUB Boot Loader, and initramfs Images.....	79

8

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

Luci system management

[*About LuCI*](#) on page 57

Security Tools

Software repository

[*RPM Repository Server*](#) on page 67

Tamper-proof file system

[*Application Integrity Measurement \(Tamper-proof File System\)*](#) on page 17

9

McAfee Embedded Control

Full documentation for 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 Embedded Control User Guide*
- *McAfee Application Control Product Guide*
- *McAfee Application Control Command Line Interface Guide*
- *McAfee Change Control Product Guide*
- *McAfee Change Control Command Line Interface Guide*
- *McAfee Change Control and McAfee Application Control Installation Guide*

These documents are available as part of the Wind River Help installation and on the [Wind River Knowledge Library](#).

10

Integrating OpenSSL and TPM

About TPM and Key Protection	49
Preparing to Use TPM	50
Creating a Key Using TPM Hardware	51
Wrapping a Software Key Into TPM Storage	52
Testing the OpenSSL TPM Engine Integration	52
About the Open Source Toolkit for SSL/TLS (OpenSSL)	54

About TPM and Key Protection

You can use TPM (Trusted Platform Module) 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 ADLINK MXE-5401 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

<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

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.

For information on enabling TPM in the CMOS, see [Enabling Encrypted Storage](#) on page 143.

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 51.

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**

This option automatically includes the **wr-srm** layer, which includes all the TPM-related packages.

For more information see:

- [Building Platform Projects for Intel Quark Boards](#) on page 93
- [Building Platform Projects for Intel Bay Trail and Intel Haswell Boards](#) on page 95

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 Storage Root Key (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 been 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.

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: <i>enter_your_password</i> Success

Both commands store the keys in the TPM hardware and generate an output 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
```

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.	<pre># create_tpm_key -z -w softkey.pem -s 1024 rootkey.pem Success</pre>
You have already set the SRK password.	<pre># create_tpm_key -w softkey.pem -s 1024 rootkey.pem SRK Password:enter_your_password Success</pre>

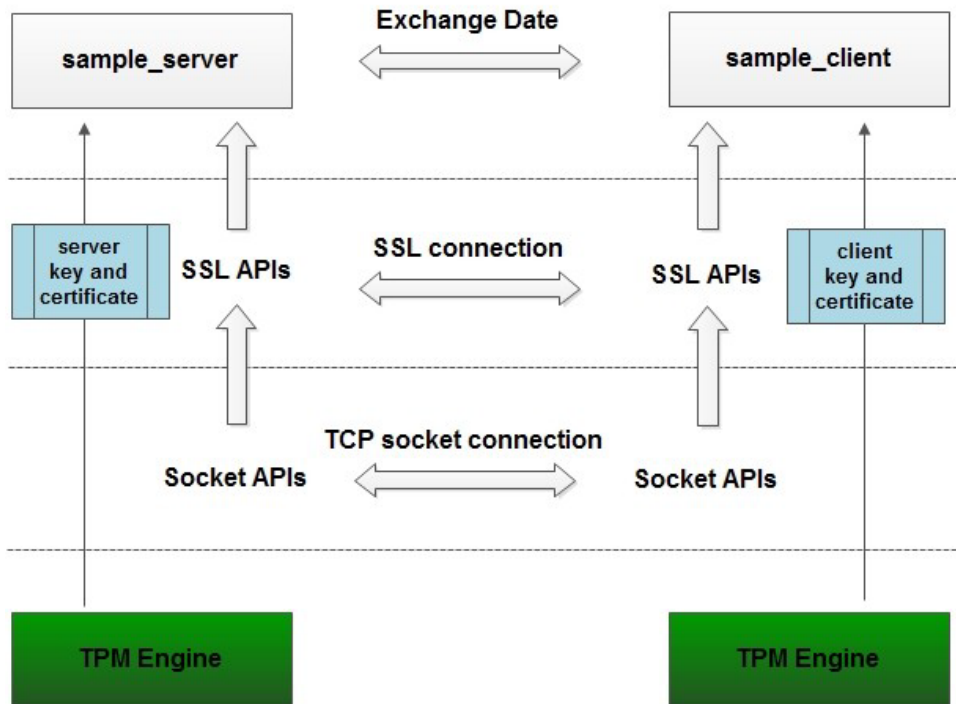
Testing the OpenSSL TPM Engine Integration

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

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 on the target after you install the sample RPM:

/root/examples/openssl-tpm-engine

The following is the block diagram for the demo application:



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.

The following steps are for an Intel Bay Trail board. If you build the sample for other boards, see the subdirectories under the *projDir/build/sample-openssl-tpm-engine/deploy-rpms* for RPM specific to your board.

Step 1 On the host, build and sign the **sample-openssl-tpm-engine** package.

```
$ make sample-openssl-tpm-engine
$ ./SST sign-rpm --priv-key=./layers/wr-idp/wr-srm/files/keys/vendor-private.pem \
  build/sample-openssl-tpm-engine/deploy-rpms/corei7_64/sample-openssl-tpm-engine-1.0-
  r0.0.corei7_64.rpm
```

Step 2 Copy the RPM to the target.

The file is located in the following directory:

```
projDir/build/sample-openssl-tpm-engine/deploy-rpms/corei7_64/sample-openssl-tpm-engine-1.0-r0.0.corei7_64.rpm
```

Step 3 Install the RPM package on the target.

```
# rpm -ivh sample-openssl-tpm-engine-1.0-r0.0.corei7_64.rpm
```

Step 4 Change to the test directory.

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

Step 5 Generate the keys and certificates for CA/Server/Client using the sample test script.

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

Step 6 Start the OpenSSL TLS server.

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

Step 7 Open another terminal by pressing **CTRL+ALT+F2** and log in.

Step 8 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.

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.

A comprehensive list of OpenSSL APIs can be found at:

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

11

LuCI Router Configuration

About LuCI	57
LuCI Interface Main Menus	57
LuCI Interface Default Settings	58
LuCI Interface Prerequisites	58
Launching and Accessing the LuCI Interface	59
Saving Changes in the LuCI Interface	61
Network Menu	62
Changing WiFi Mode from AP to Client Using LuCI	63
Changing WiFi Mode from Client to AP Using LuCI	64

About LuCI

You can use the LuCI 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 platform project with Secure Remote Management (SRM), which is enabled by default, the **wr-idp-devkit** layer and **idp** rootfs are automatically configured. This includes LuCI in the project and sets up your IDP XT target to act as a gateway by default.

LuCI Interface Main Menus

The main menus enable you to make configuration changes to your Wi-Fi gateway.

The default LuCI interface provides the following menus and their relevant settings to make configuration changes to your gateway:

Status Menu

Use to view the status of routes, modules, and the system.

System Menu

Use to specify system-specific settings, such as time, theme and language, access control, password, backup and restore, and to upgrade and reboot the router.

Network Menu

Use to view and set detailed networking parameters. These include WAN, LAN, WWAN, Wireless, Bluetooth, Firewall, DHCP, Static Routes, and MultiWAN. For more information, see [Network Menu](#) on page 62.

In addition, you can create your own pages and include them with your Intelligent Device Platform system. See [About Customizing LuCI](#) on page 153.

LuCI Interface Default Settings

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

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

- **Web login username/password:** root/root
- **WAN port:** eth0 (DHCP to get IP address)
- **WLAN:**
 - 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 LuCI interface to change the default configuration. See the following:

- [Launching and Accessing the LuCI Interface](#) on page 59
- [Saving Changes in the LuCI Interface](#) on page 61.

LuCI Interface Prerequisites

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

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

- Any LuCI-supported board:
 - Cross Hill
 - Clanton Hill
 - Galileo Gen 2

- GIGABYTE GB-BXBT-3825
- ADLINK MXE-5401
- Ethernet or Wi-Fi (recommended) connection
- a connection from the device to the host
 - A serial connection for Cross Hill, Clanton Hill, and Galileo Gen 2 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.

Ensure that you allow cookies in your browser. For example, in Internet Explorer, do the following:

- Select **Tools > Internet options**.
- Click **Privacy** and then click **Advanced**.
- Select **Override automatic cookie handling**.
- Under First-party Cookies and Third-party Cookies, choose **Accept**.
- Click **OK** and then click **OK**.

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

Launching and Accessing the LuCI Interface

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

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

This example uses a GIGABYTE GB-BXBT-3825 board in most steps.



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

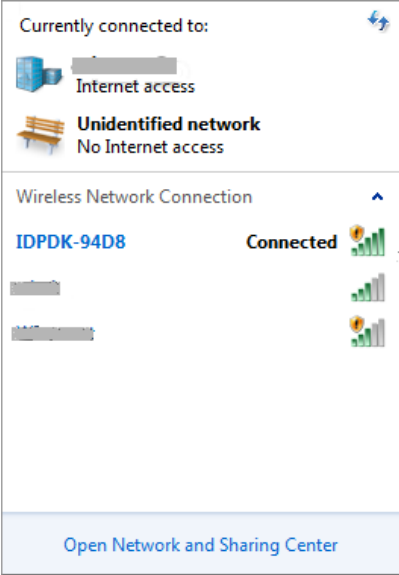
Complete the remaining steps for secure boot to confirm the boot is working correctly.

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 115
[Performing a Secure Boot Using UEFI on Intel Baytrail Boards](#) on page 116
[Performing a Secure Boot Using UEFI on Intel Haswell Boards](#) on page 118
[Performing a Verified Boot](#) on page 119

Step 1 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.  The screenshot shows the Windows Network Settings window. Under 'Currently connected to:', there is an 'Internet access' section with a globe icon and an 'Unidentified network' section with a bench icon and 'No Internet access'. Below this, the 'Wireless Network Connection' section shows a list of available networks. The network 'IDPDK-94D8' is selected and marked as 'Connected' with a signal strength indicator. At the bottom, there is a button that says 'Open Network and Sharing Center'.
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 2 Start the LuCI interface in a Web browser on your host.

Connection Type	Procedure
If your IDP XT target has a WiFi module:	In the address bar, type https://192.168.1.1 or the value configured for the wlan0 interface and press ENTER.
If your IDP XT target does not have a WiFi module:	In the address bar, type https://ipAddrOfEth0 and press ENTER.

Step 3 At the prompt, type **root** for both the username and password, then click **OK**.

The LuCI interface displays the Status page with system information.

WR-IntelligentDevice

StatusSystemNetworkLogout

AUTO REFRESH ON

Status

System

Hostname	WR-IntelligentDevice
Model	intel-baytrail-64
System Version	Intelligent Device Platform XT 3.0 / LuCI (0.12)
Kernel Version	3.14.37-XT_3.0-grsec-WR7.0.0.0_idp
Local Time	Tue Jun 2 02:23:34 2015
Uptime	16h 6m 3s
Load Average	0.00, 0.02, 0.14

Memory

Used	230332 kB / 1928820 kB (11%)
Cached	77276 kB / 1928820 kB (4%)
Buffered	46260 kB / 1928820 kB (2%)

Network

IPv4 WAN Status

Type: dhcp

eth0 Address: 128.224.159.134

Netmask: 255.255.254.0

Gateway: 128.224.158.1

DNS 1: 128.224.160.11

DNS 2: 147.11.57.128

DNS 3: 147.11.57.133

Connected: 16h 5m 45s

Active Connections

86 / 65536 (0%)

DHCP Leases

Hostname	IPv4-Address	MAC-Address	Leasetime remaining
android-11ead2aff3b09ebd	192.168.1.249	38:bc:1a:fd:b4:2b	11h 12m 40s

Step 4 To change the default configuration settings, see [Saving Changes in the LuCI Interface](#) on page 61.

Step 5 To add a new LuCI page, see [About Customizing LuCI](#) on page 153.

Saving Changes in the LuCI Interface

When you make changes to your configuration using LuCI, you must ensure that your changes are saved properly.

Step 1 Edit your configuration in LuCI. See [LuCI Interface Main Menus](#) on page 57.

Step 2 Click **Save** in the lower-right corner of the configuration page.

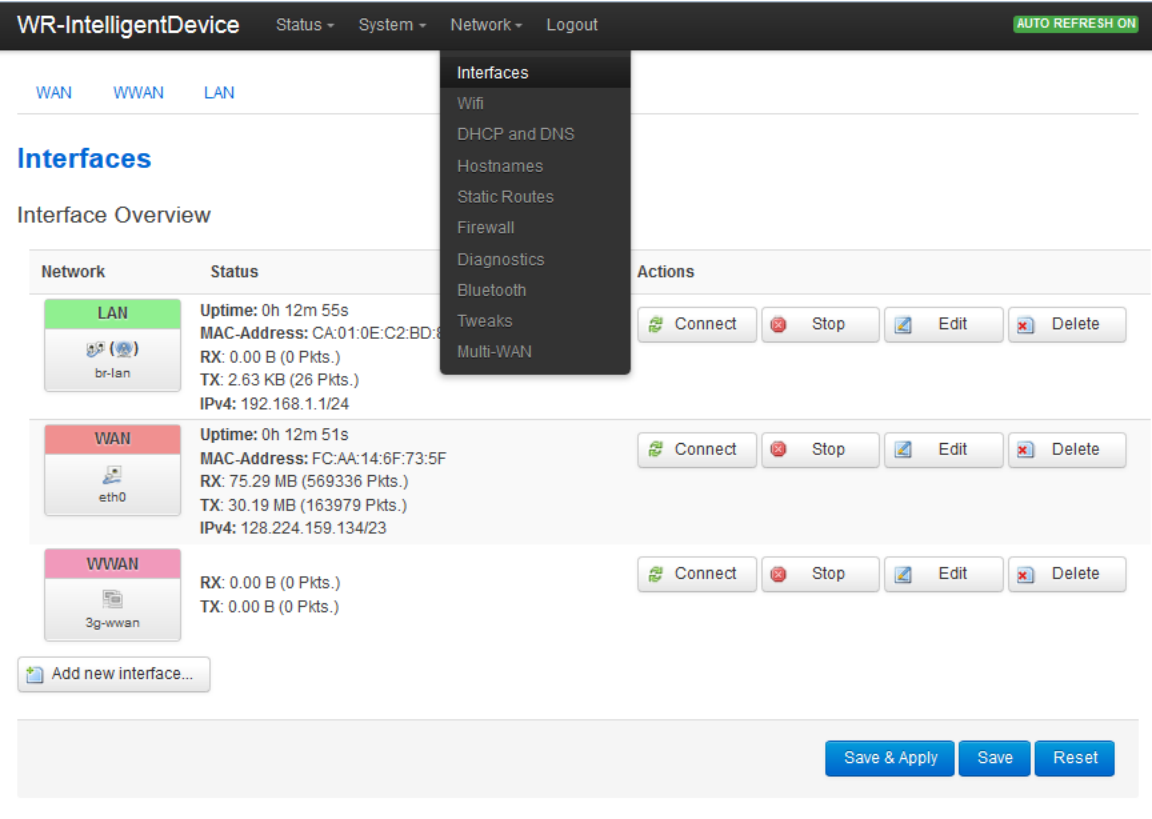
Your changes are saved only to a temporary location.

Step 3 Click Save & Apply.

Your changes are now permanently saved.

Network Menu

The Network menu enables you to view and change basic networking parameters, including LAN, WAN, and WWAN for your gateway router.



Powered by LuCI (0.12) Intelligent Device Platform XT 3.0

Menu Option	Description
Interfaces	Use this section to modify your network interface settings. Some key options include: • WAN Configuration - Connection Type: Default is DHCP. - Interface: Default is eth0. • LAN Configuration:

Menu Option	Description
	- 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.
Wifi	Use this page 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 • IP Address: Defaults to 192.168.1.1. • Encryption Type: Default is WPA2 (PSK)

Once you make changes, click **Save & Apply** before navigating away from this page. See [Saving Changes in the LuCI Interface](#) on page 61.

Changing WiFi Mode from AP to Client Using LuCI

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 LuCI interface.

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

1. Select **Network > Interfaces**.
2. In the LAN row, click **Edit**.
3. Click **Physical Settings** and then clear **Bridge interfaces**.
4. Click **General Setup** and then from the **Protocol** menu, select **DHCP client**.
5. Beside Really switch protocol?, click **Switch protocol**.

6. Click **Save & Apply**.
7. Select **Network > WiFi**.
8. In the row for the wireless network, click **Edit**.
9. From the **Mode** list, select **Client**.
10. In the **ESSID** box, type the wireless network server ESSID .
11. Click **Wireless Security** and then in the **Key** box, type the access point password.
12. Click **Save & Apply**.

Changing WiFi Mode from Client to AP Using LuCI

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 LuCI interface.

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

- Step 1** Select **Network > WiFi**.
- Step 2** In the wireless network row, click **Edit**.
- Step 3** Click **General Setup** and from the **Mode** list, select **Access Point**.
- Step 4** In the **ESSID** box, type the wireless network server ESSID.
- Step 5** Click **Wireless Security**, and then in the **Key** box, type the access point password.
- Step 6** Click **Save**.
- Step 7** Select **Network > Interfaces**.
- Step 8** In the LAN row, click **Edit**.
- Step 9** Click **Physical Settings** and then select **Bridge interfaces**.
- Step 10** Click **Save & Apply**.
- Step 11** After a wireless client connects to your WiFi module, confirm that you have successfully set up AP mode.
Select **Network > DHCP and DNS**. Review the information shown under Active DHCP Leases.

WR-IntelligentDevice

Status ▾System ▾Network ▾Logout

AUTO REFRESH ON

Server Settings

General Settings

Local server

/lan/

Local domain specification. Names matching this domain are never forwarded and are resolved from DHCP or hosts files only

Local domain

lan

Local domain suffix appended to DHCP names and hosts file entries

Log queries

☐

Write received DNS requests to syslog

Max. DHCP leases

unlimited

Maximum allowed number of active DHCP leases

Active DHCP Leases

Static Leases

Static leases are used to assign fixed IP addresses and symbolic hostnames to DHCP clients. They are also required for non-dynamic interface configurations where only hosts with a corresponding lease are served.
Use the *Add* Button to add a new lease entry. The *MAC-Address* identifies the host, the *IPv4-Address* specifies to the fixed address to use and the *Hostname* is assigned as symbolic name to the requesting host.

Add

Save & ApplySaveReset

12

Secure Repository

RPM Repository Server	67
Adding a Local Repository	67
Remote Repositories	68
Managing Repositories	71
Managing RPM Packages	73

RPM Repository Server

The RPM repository server maintains the customized packages for your Secure Remote management (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 LuCI interface to manage the repository, see [LuCI Interface Main Menus](#) on page 57.

Adding a Local Repository

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

Step 1 Build a platform project with the **wr-srm** layer (included by default) and boot your board in the standard way.

For more information see:

- [Building Platform Projects for Intel Bay Trail and Intel Haswell Boards](#) on page 95

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 <i>target_local_directory</i> is not exist	The RPM repository does not exist on the device.
Adding local repository <i>local_rpm_directory</i> error	The RPM repository exists but the command failed to add it.
Adding local repository <i>local_rpm_repo</i> 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.corei7_64.rpm
-rw-r--r-- 1 root root 536333 Nov  7 16:16 cups-libs-1.4.1-1_WR4.3.0.0.1.corei7_64.rpm
-rw-r--r-- 1 root root 15236 Nov  7 16:16 cups-lpd-1.4.1-1_WR4.3.0.0.1.corei7_64.rpm
-rw-r--r-- 1 root root 180338 Nov  7 16:16 dbus-glib-0.82-2_WR4.3.0.0.corei7_64.rpm
-rw-r--r-- 1 root root 5967 Nov  7 16:16 mentest-1.0-1_WR4.3.0.0.corei7_64.rpm
drwxr-xr-x 2 root root 4096 Apr 26 17:50 repodata
```

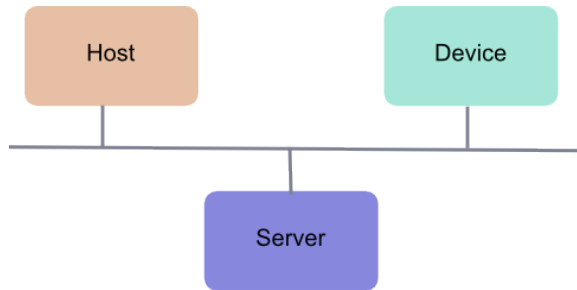
Remote Repositories

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.
- 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

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
```

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/corei7_64
$ sudo createrepo .
$ cd /var/www/html/rpm/intel-baytrail
$ sudo createrepo .
```

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.

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 40

[Signing Multiple RPMs](#) on page 40

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/corei7_64*, use the following command:

```
$ spm_repo --addremote myrepo http://192.0.2.0/rpm_repo/rpm/corei7_64
```

The following messages may occur:

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

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 <i>repository_name</i> error	The command failed to remove the repository.
Removing repository <i>repository_name</i> successfully	The RPM repository was successfully removed.

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
```

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 these examples, the package is **memtest-1.0-1_WR4.3.0.0.corei7_64.rpm**.

```
# spm_repo --installrpm memtest-1.0-1_WR4.3.0.0.corei7_64.rpm
```

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
```

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.corei7_64.rpm**.

```
# spm_repo --deleterpm memtest-1.0-1_WR4.3.0.0.corei7_64.rpm
```


13

Tamper-Proof File System

Application Integrity Measurement	75
Using the Tamper-Proof File System	76

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.

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

- `intel-quark`
- `intel-baytrail-64`
- `intel-haswell-64`

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-appraise** and boot the device. This includes the tamper-proof file system capability.

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

For more information see:

- [Building Platform Projects for Intel Bay Trail and Intel Haswell Boards](#) on page 95

The following configure command uses the **intel-baytrail-64** BSP as an example:

```
$ $WIND LINUX_CONFIGURE_CLI --enable-rootfs=idp --enable-addons=wr-idp \
--enable-kernel=idp --enable-board=intel-baytrail-64 --with-layer=wr-ima-appraise
```

Building this configuration creates two tar files:

wrlinux-image-idp-intel-baytrail-64.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-idp-intel-baytrail-64-dist.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 31.

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:imtools      --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_CLI** command.

```
$ $WIND_LINUX_CONFIGURE_CLI --enable-rootfs=idp --enable-addons=wr-idp \
--enable-kernel=idp --enable-board=intel-baytrail-64
```



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

14

Sign and Update RPM Packages, Kernel Images, GRUB Boot Loader, and initramfs Images

Generating and Installing a Signed RPM package with SST	79
Signing an RPM Package with GPG	80
Generating and Updating a Signed Kernel Image	83
Generating and Updating a Signed Boot Loader Image	84
Generating and Updating a Signed initramfs Image	85

Generating and Installing a Signed RPM package with SST

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

You only need to sign an RPM with SST if you include the **wr-ima-appraise** layer in your platform project. For more information about IMA, see [Using the Tamper-Proof File System](#) on page 76.

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

The following configure command uses the **intel-baytrail-64** BSP as an example:

```
$ cd projDir
$ $WIND LINUX_CONFIGURE_CLI --enable-board=intel-baytrail-64 \
--enable-addons=wr-idp --enable-kernel=idp --enable-rootfs=idp
```

For more information about how to configure IDP XT, see [Preparing to Build and Boot IDP](#) on page 91.

Step 2 Build your RPM package.

For example, to build an RPM package called **example-1.0**, do the following:

```
$ cd projDir
$ make example
```

The RPM package is created in the *projDir/build/package/deploy-rpms/arch* directory. For example:

```
$ ls build/example-1.0-r2/deploy-rpms/corei7_64/
example-1.0-r2.corei7_64.rpm
```

Step 3 Sign the RPM package with SST.

```
$ ./SST sign-rpm --verbose=no --mode=rpm --kernel-pkg=no --priv-key=./vendor-
private.pem \
./example-1.0-r2.corei7_64.rpm
```

The RPM package file is signed using the vendor's private key, **vendor-private.pem**.

For more information about signing RPMs, see [Signing a Single RPM](#) on page 40 and [Signing Multiple RPMs](#) on page 40.

Step 4 Copy the vendor certificate to the target.

Step 5 Install the vendor 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 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.corei7_64.rpm
Preparing... ##### [100%]
1:example ##### [100%]
```

The RPM package **example-1.0-r2.corei7_64.rpm** is successfully installed on the target.

Signing an RPM Package with GPG

You can use GPG (GNU Privacy Guard) to add a signature to an RPM or replace the existing signature of an RPM package.

You can use GPG (GNU Privacy Guard) to sign RPMs when you do not have access to the platform project to use the SST tool. Be aware that the kernel does not enforce GPG digital signatures.

Prerequisites

You must have RPM 5 installed on your Linux host. Otherwise, verification of the signed RPM on the target fails. Depending on your Linux host distribution, you may need to download, compile, and install the package on your host.

Step 1 On your host computer, generate the GPG key pairs.

When you generate the key, choose the key type, key size, a valid time, and specify your name and email address. When you choose a password, ensure that you save it for future use.

```
# gpg --gen-key
gpg (GnuPG) 2.0.26; Copyright (C) 2013 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Please select what kind of key you want:
  (1) RSA and RSA (default)
  (2) DSA and Elgamal
  (3) DSA (sign only)
  (4) RSA (sign only)
Your selection?1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) 2048
Requested keysize is 2048 bits
Please specify how long the key should be valid.
    0 = key does not expire
    <n> = key expires in n days
    <n>w = key expires in n weeks
    <n>m = key expires in n months
    <n>y = key expires in n years
Key is valid for? (0) 0
Key does not expire at all
Is this correct? (y/N) y

You need a user ID to identify your key; the software constructs the user ID
from the Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: example
Email address: example@example.com.com
Comment: ForExample
You selected this USER-ID:
    "example (ForExample) <example@example.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
You need a Passphrase to protect your secret key.
```

Step 2 In the window that appears, type a password, for example, 123456.

When key generation begins, the following output appears:

```
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
```

When key generation completes, the following output appears:

```
gpg: key 0EA5DF14 marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0 valid: 2 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 2u
pub 2048R/0EA5DF14 2016-02-10
    Key fingerprint = C2DE 7CC1 2E53 CBDC E31E A711 E5FA 5B21 0EA5 DF14
uid                               example (ForExample) <example@example.com>
sub 2048R/67D10EE4 2016-02-10
```

Step 3 Verify and export the public key.

```
# gpg --list-keys
sec 2048R/5A6A20D7 2015-11-12
```

```
uid          example (ForExample) <example@example.com>
ssb 2048R/8D61068E 2015-11-12
# gpg --export -a 5A6A20D7 > MYORG-GPG-KEY.public
```

Step 4 Verify and export the private key.

```
# gpg --list-secret-keys
sec 2048R/5A6A20D7 2015-11-12
uid          example (ForExample) <example@example.com>
ssb 2048R/8D61068E 2015-11-12
# gpg --export-secret-key -a 5A6A20D7 > MYORG-GPG-KEY.private
```

Step 5 Edit the RPM configuration file and update the required macros.



NOTE: If you want to sign the RPM on another host, you must import the private key as follows:

```
# gpg --import MYORG-GPG-KEY.private
```

In the `_signature` macro, specify the signature type and in the `_gpg_name` macro, specify the **Real Name** and **Comment** values you used in step 1 on page 81.

```
%_signature gpg
%_gpg_name example (ForExample)
```

Step 6 Sign the RPM package.

The following example signs the `tftp-hpa-server-5.2-r0.0.quark.rpm`. When prompted for the pass phrase, use the password you created when you generated the keys in step 2 on page 81.

```
# rpm --addsign tftp-hpa-server-5.2-r0.0.quark.rpm
Enter pass phrase:123456
Pass phrase is good.
tftp-hpa-server-5.2-r0.0.quark.rpm:
```

Step 7 Log in to the target and import the public key from the file you created in step 3 on page 81.

```
$ rpm --import MYORG-GPG-KEY.public
```

Step 8 Copy the RPM to the target.

Step 9 Verify the RPM on the target.

```
$ rpm -Kv tftp-hpa-server-5.2-r0.0.quark.rpm
tftp-hpa-server-5.2-r0.0.quark.rpm:
Header V4 RSA/SHA1 signature: OK, key ID
5a6a20d7
Header SHA1 digest: OK
(afc12411150683b3b3fa5407278236f5884a3f3d)
MD5 digest: OK
(a219faebc2cb40142236a8495d22445d)
```

If the RPM is successfully verified, the signature shows **OK** and the key ID is the public key.

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.

The following configure command uses the **intel-baytrail-64** BSP as an example:

```
$ $WIND_LINUX_CONFIGURE_CLI --enable-board=intel-baytrail-64 \  
--enable-addons=wr-idp --enable-kernel=idp --enable-rootfs=idp
```

For more information about how to configure IDP XT, see [Preparing to Build and Boot IDP](#) on page 91.

Step 2 Build the kernel image.

```
$ cd projDir  
$ make linux-windriver
```

This puts the kernel image in *projDir/build/linux-windriver/image/boot*

Step 3 Sign the kernel image with SST.

The following command uses the image for the **intel-baytrail-64** BSP as an example:

```
$ ./SST sign-kernel --verbose=no \  
--priv-key=./vendor-private.pem \  
--vendor-cert=./vendor-cert.pem \  
./bzImage-intel-baytrail-64.bin
```

vendor-private.pem is the vendor's private key and **vendor-cert.pem** is the vendor's certificate.

The command above updates the kernel image file **bzImage-intel-baytrail-64.bin**.

For more information, see [Signing Kernels](#) on page 38.

Step 4 If your boot method is UEFI and your boot device is a USB flash drive, format the USB flash drive into two partitions: VFAT and ext3.

For information about how to deploy a boot device, see [Preparing to Build and Boot IDP](#) on page 91.

Step 5 Connect your USB flash drive to the host machine.

Step 6 Mount the VFAT partition.

The following assumes that your USB flash drive is **/dev/sdc1** on your host

```
$ mount /dev/sdc1 ./vfat
```

Step 7 Replace the **bzImage** file in the VFAT partition with the updated **bzImage** file signed by SST.

```
$ cp bzImage ./vfat
```

Step 8 Unmount your boot device.

```
$ umount /dev/sdc1
```

Generating and Updating a Signed Boot Loader Image

A signed boot loader supports verified and secure boot, essential processes in the Secure Remote Management (SRM) feature in IDP XT.

The instructions in this section are for generating and updating a signed boot loader image for GIGABYTE GB-BXBT-3825, ADLINK MXE-5401, or Intel quark boards using a USB flash drive.

Prerequisites

You must have completed building your platform project. For more information, see [Building Platform Projects for Intel Bay Trail and Intel Haswell Boards](#) on page 95.

Step 1 Build the boot loader (**grub-efi**).

```
$ cd projDir
$ make grub-efi
```

This puts the **grub-efi** image file and GRUB configuration file in *projDir/build/grub-efi/image/boot/grub*.

Step 2 Sign the boot loader with SST.

```
$ ./SST sign-bootloader --verbose=no --grub-efi=yes \
--owner-cert=./owner-cert.pem \
--vendor-cert=./vendor-cert.pem \
--priv-key=./vendor-private.pem \
./grub.efi
```

The **owner-cert.pem** is the owner's certificate, **vendor-cert.pem** is the vendor's certificate, which is delegated by the owner, and **vendor-private.pem** is the vendor's private key.

The **grub.efi** file is updated.

Step 3 If you changed the boot loader configuration file to boot the target from a different storage media, sign the **grub.conf** configuration file.

```
$ ./SST sign-bootloader --verbose=no --grub-efi=no \
--vendor-cert=./vendorE-cert.pem \
--priv-key=./vendorE-private.pem \
./grub.conf
```

The **grub.conf.auth** file is generated.

Step 4 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 91.

Step 5 Connect your boot device to the host machine.

Step 6 Mount the VFAT partition.

The following command assumes that the VFAT partition on your USB flash drive is `/dev/sdc1` on your host machine.

```
$ mount /dev/sdc1 ./vfat
```

Step 7 Replace the boot loader and boot loader configuration files in the `./vfat/EFI/BOOT` directory with the updated `grub.efi` and `grub.conf.auth` files (or the `grub.conf` file if you did not need to sign the configuration file):

- If your board is a GIGABYTE GB-BXBT-3825 or ADLINK MXE-5401, do the following

```
$ cp grub.efi ./vfat/EFI/BOOT/BOOTX64.efi
$ cp grub.conf ./vfat/EFI/BOOT/BOOTX64.conf
$ cp grub.conf.auth ./vfat/EFI/BOOT/BOOTX64.conf.auth
```

- If your board is an Intel Quark, do the following

```
$ cp grub.efi ./vfat/EFI/BOOT/BOOTIA32.efi
$ cp grub.conf ./vfat/EFI/BOOT/BOOTIA32.conf
$ cp grub.conf.auth ./vfat/EFI/BOOT/BOOTIA32.conf.auth
```

Step 8 Unmount your boot device.

```
$ umount /dev/sdc1
```

Generating and Updating a Signed initramfs Image

Sign and update an initramfs image.

The Secure Remote Management (SRM) feature is enabled in IDP XT, which prevents the loading of a initramfs image that is not properly signed.

Prerequisites

You must have completed building your platform project. For more information, see [Building Platform Projects for Intel Bay Trail and Intel Haswell Boards](#) on page 95.

Step 1 Build the kernel image.

```
$ cd projDir
$ make wr-idp-initramfs
```

This creates the initramfs image `wr-idp-initramfs-intel-baytrail-64.cpio.gz` in `projDir/export/image`

Step 2 Sign the kernel image with SST.

The following command uses the image for the `intel-baytrail-64` BSP as an example:

```
$ ./SST sign-kernel --verbose=no \
--priv-key=./vendor-private.pem \
--vendor-cert=./vendor-cert.pem \
./wr-idp-initramfs-intel-baytrail-64.cpio.gz
```

`vendor-private.pem` is the vendor's private key and `vendor-cert.pem` is the vendor's certificate.

The **wr-idp-initramfs-intel-baytrail-64.cpio.gz.auth** signature file is generated.

Step 3 If your boot method is UEFI and your boot device is a USB flash drive, format the USB flash drive into two partitions: VFAT and ext3.

For information about how to deploy a boot device, see [Preparing to Build and Boot IDP](#) on page 91.

Step 4 Connect your USB flash drive to the host machine.

Step 5 Mount the VFAT partition.

The following assumes that your USB flash drive is **/dev/sdc1** on your host

```
$ mount /dev/sdc1 ./vfat
```

Step 6 Replace the initramfs file in the VFAT partition with the updated initramfs file signed by SST.

```
$ cp wr-idp-initramfs-intel-baytrail-64.cpio.gz ./vfat/idp-initramfs.img
$ cp wr-idp-initramfs-intel-baytrail-64.cpio.gz.auth ./vfat/idp-initramfs.img.auth
```

Step 7 Unmount your boot device.

```
$ umount /dev/sdc1
```

PART IV

Device Development Vendor Tasks

Introduction to Device Development Tasks.....	89
Building and Booting.....	91
Alternative Booting Methods.....	115
Configuring IDP Features.....	121
Installing Tools for Application Development and Control.....	129
Customizing LuCI.....	153
Updating WPAN Firmware for Intel Quark Boards.....	155

15

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 7.0 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.

16

Building and Booting

Preparing to Build and Boot IDP	91
About the wrenv.sh Script	92
About the deploytool Script	92
Building Platform Projects for Intel Quark Boards	93
Deploying Images to Intel Quark Boards	94
Building Platform Projects for Intel Bay Trail and Intel Haswell Boards	95
Configuring the BIOS for Intel Bay Trail and Intel Haswell Boards	97
Deploying Images to Intel Bay Trail and Intel Haswell Boards	98
Updating BIOS Images on GIGABYTE GB-BXBT-3825 Boards and other Intel Bay Trail Boards	99
Updating BIOS Images on ADLINK MXE-5401 Boards	99
Updating Flash Firmware for Intel Quark Boards	100
About the GRUB Boot Menu Information	105
Updating the Target System	108
Using the IA Recovery Image	109
IDP Preconfigured Profiles	111
Platform Boot Time Optimizations	112
Configuring the Target at Boot Time	112

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.

- The Git version control system installed on the host computer (required to install Wind River Linux 7.0).
- A Wind River IDP XT 3.1 installation with Wind River Linux 7.0 on a supported host.
- Any IDP XT 3.1 supported board:
 - Cross Hill
 - Clanton Hill
 - Galileo Gen 2
 - GIGABYTE GB-BXBT-3825
 - ADLINK MXE5401
- Ability to create boot medium with 8 GB capacity
- A display device compatible with the display port on the target. For Intel Quark boards, see your BSP documentation for information on the serial console.
- A connection between the development host and the board.

Step 2 Install Wind River Intelligent Device Platform XT 3.1, which includes Wind River Linux 7.0, on your development host.

Step 3 Connect the board to the host.

About the `wrenv.sh` Script

The **wrenv.sh** script sets all the Wind River Linux-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-7
```

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_CLI` in your **configure** line. The full path is similar to *installDir/wrlinux-7/wrlinux/configure*.

All the examples in this guide assume that you have executed **wrenv.sh**.

About the `deploytool` Script

The **deploytool** script enables you to deploy a kernel image and rootfs to a boot device or image file.

For a detailed list of options for the **deploytool** script, execute the script on your development host with the **-h** option:

```
$ cd projDir
$ ./deploytool -h
```

For usage examples and detailed information about error codes, see:

projDir/layers/wr-idp/wr-idp-devkit/recipes-devtools/deploy-tool/files/README.

Building Platform Projects for Intel Quark Boards

Configure and build a platform project configured for an Intel Quark board.

IDP XT supports the following Intel Quark boards:

- Clanton Hill
- Cross Hill
- Galileo Gen 2

There are several differences between Intel Quark boards and other Intel boards supported for previous versions of IDP XT:

- Intel Quark boards require that the system files in flash on the device match the system files on the boot media.
- Intel Quark boards do not have video output capability.
- Intel Quark boards can boot from either a USB flash drive or from an MMC card.

For more information, see your BSP documentation.

IDP XT provides a separate rootfs option (**--enable-rootfs=idp-dev**) for development that you can use to build a platform project without security capabilities enabled. The **wr-srm**, **wr-mcafee**, **wr-mcafee-essential**, and **wr-ima-appraise** layers are excluded, and although grsecurity is included in the kernel, the functionality is disabled. Do not use the images from this platform project in field deployments.

Step 1 Set the Wind River Linux environment variables on your host machine.

```
$ installDir/wrenv.sh -p wrlinux-7
```

The **wrenv.sh** script sets the **\$WIND_LINUX_CONFIGURE_CLI** environment variable to specify the location of the **configure** command.

Step 2 Create a platform project directory *projDir*.

```
$ mkdir projDir
$ cd projDir
```

Step 3 Configure the platform project.

Specify **intel-quark** as the board type.

The following configures a platform project with security features enabled:

```
$ $WIND_LINUX_CONFIGURE_CLI --enable-board=intel-quark --enable-addons=wr-idp \
--enable-kernel=idp --enable-rootfs=idp
```

The following configures a platform project with security features disabled:

```
$ $WIND_LINUX_CONFIGURE_CLI --enable-board=intel-quark --enable-addons=wr-idp \
--enable-kernel=idp --enable-rootfs=idp-dev --with-template=feature/engineering
```



NOTE: When you specify **--enable-rootfs=idp-dev**, you must also use **--with-template=feature/engineering** for Intel Quark boards. Otherwise, you cannot boot from the USB flash drive or MMC card when the flash firmware is programmed for secure boot.

Step 4 Build the project.

```
$ make
```

Building the project with the **--enable-rootfs=idp** option generates the following files in the *projDir/export/images* directory:

wr-idp-initramfs-intel-quark.cpio.gz

The gzipped cpio format initramfs for IDP XT.

bzImage-intel-quark.bin

The Linux kernel image for IDP XT.

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

A tar file containing the root file system image. The image is signed by default unless you use the **--without-layer=wr-srm** option in your configuration.

Building the project with the **--enable-rootfs=idp-dev** option generates the same initramfs and Linux kernel images. The following rootfs image is created:

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

A tar file containing the root file system image with the SRM, IMA, MEC, and grsecurity capabilities disabled by default. The image is not signed.

You are now ready to deploy your platform project to the device.

Deploying Images to Intel Quark Boards

Use the **deploytool** script to place the system files on the boot media and use the boot media 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 memory. The files in flash memory and on the USB flash drive must be from the same project to successfully boot the board.

Step 1 Deploy the file to the USB drive using **deploytool**.

If you built the platform project with the **idp** profile (**--enable-rootfs=idp**), you can use the following file:

projDir/export/intel-quark-idp-idp-dist.tar.bz2

If you built the platform project with the **idp-dev** profile (**--enable-rootfs=idp-dev**), use the following file:

projDir/export/intel-quark-idp-dev-idp-dist.tar.bz2



NOTE: Using the **-y** option on **deploytool** 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 **deploytool**. For information on creating partitions, see [Updating the Target System](#) on page 108.

For more information on the **deploytool** script, see [About the deploytool Script](#) on page 92.

For example, using a Cross Hill board and assuming your device is **/dev/sdb**:

```
$ cd projDir
$ sudo ./deploytool -f \
export/intel-quark-idp-idp-dist.tar.bz2 \
-d /dev/sdb -u -y
```

After deployment your USB device is formatted with a VFAT partition and an LVM partition formatted as an EXT3 file system.

You can set the rootfs partition size in bytes with the **-p** option. For example:

```
$ cd projDir
$ sudo ./deploytool -f \
export/intel-quark-idp-idp-dist.tar.bz2 \
-d /dev/sdb -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 Intel Quark Boards](#) on page 100

Step 4 Boot the target.

The boot behavior is as follows:

- When the Intel Quark board has a USB flash drive attached, but no MMC card, the board boots from USB and uses the USB flash drive as the root device.
- When the Intel 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 Intel 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.

Building Platform Projects for Intel Bay Trail and Intel Haswell Boards

Configure and build a platform project for Intel Bay Trail and Intel Haswell boards.

IDP XT supports the following boards:

- GIGABYTE GB-BXBT-3825 (Intel Bay Trail)
- ADLINK MXE-5401 (Intel Haswell)

IDP XT provides a separate rootfs option (**--enable-rootfs=idp-dev**) for development that you can use to build a platform project without security capabilities enabled. The **wr-srm**, **wr-mcafee**, **wr-mcafee-essential**, and **wr-ima-appraise** layers are excluded, and although grsecurity is

included in the kernel, the functionality is disabled. Do not use the images from this platform project in field deployments.

Step 1 Set the Wind River Linux environment variables on your host machine.

```
$ installDir/wrenv.sh -p wrlinux-7
```

Step 2 Create a platform project directory *projDir*.

```
$ mkdir projDir
$ cd projDir
```

Step 3 Configure the platform project.

Use the following **configure** command for the GIGABYTE GB-BXBT-3825 board to create a platform project with security features enabled:

```
$ $WIND_LINUX_CONFIGURE_CLI --enable-board=intel-baytrail-64 --enable-addons=wr-idp \
--enable-kernel=idp --enable-rootfs=idp
```

Use the following **configure** command for the GIGABYTE GB-BXBT-3825 board to create a platform project with security features disabled:

```
$ $WIND_LINUX_CONFIGURE_CLI --enable-board=intel-baytrail-64 --enable-addons=wr-idp \
--enable-kernel=idp --enable-rootfs=idp-dev
```

Use the following **configure** command for the ADLINK MXE-5401 board to create a platform project with security features enabled:

```
$ $WIND_LINUX_CONFIGURE_CLI --enable-board=intel-haswell-64 --enable-addons=wr-idp \
--enable-kernel=idp --enable-rootfs=idp
```

Use the following **configure** command for the ADLINK MXE-5401 board to create a platform project with security features disabled:

```
$ $WIND_LINUX_CONFIGURE_CLI --enable-board=intel-haswell-64 --enable-addons=wr-idp \
--enable-kernel=idp --enable-rootfs=idp-dev
```



NOTE: `$WIND_LINUX_CONFIGURE_CLI` is an environment variable set by `wrenv.sh` for the location of the **configure** command.



NOTE: There are additional configuration options available in Linux 7.0. For more information about the available Linux documents, see [Where to Find Information](#) on page 6.

Step 4 Build the project.

```
$ make
```

Building the project with the `--enable-rootfs=idp` option for the GIGABYTE GB-BXBT-3825 board generates the following in the *projDir/export/images* folder:

wr-idp-initramfs-intel-baytrail-64.cpio.gz

The gzipped cpio format initramfs for IDP XT.

bzImage-intel-baytrail-64.bin

The Linux kernel image for IDP XT.

wrlinux-image-idp-intel-baytrail-64.tar.bz2

A tar file containing the root file system image. The image is signed by default unless you use the **--without-layer=wr-srm** option in your configuration.

Building the project with the **--enable-rootfs=idp-dev** option generates the same initramfs and Linux kernel images. The following rootfs image is created:

wrlinux-image-idp-dev-intel-baytrail-64.tar.bz2

A tar file containing the root file system image with the SRM, IMA, MEC, and grsecurity capabilities disabled by default. The image is not signed.

Building the project with the **--enable-rootfs=idp** option for the ADLINK MXE540 board generates the following in the *projDir/export/images* folder:

wr-idp-initramfs-intel-haswell-64.cpio.gz

The gzipped cpio format initramfs for IDP XT.

bzImage-intel-haswell-64.bin

The Linux kernel image for IDP XT.

wrlinux-image-idp-intel-haswell-64.tar.bz2

A tar file containing the root file system image. The image is signed by default unless you use the **--without-layer=wr-srm** option in your configuration.

Building the project with the **--enable-rootfs=idp-dev** option generates the same initramfs and Linux kernel images. The following rootfs image is created:

wrlinux-image-idp-dev-intel-haswell-64.tar.bz2

A tar file containing the root file system image with the SRM, IMA, MEC, and grsecurity 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.

Configuring the BIOS for Intel Bay Trail and Intel Haswell Boards

Configure the default 64-bit BIOS for Intel Bay Trail boards (for example, GIGABYTE GB-BXBT-3825) and Intel Haswell boards (for example, ADLINK MXE-5401).

The default BIOS in Intel Bay Trail and Intel Haswell boards must be 64-bit, and the BIOS configuration among the boards might not match. IDP XT 3.x always uses UEFI to boot the board; IDP XT 3.x does not support 32-bit UEFI or legacy boot.



NOTE: The BIOS setup may vary among boards. The following instructions show how to configure the BIOS for the GIGABYTE GB-BXBT-3825 and ADLINK MXE-5401 boards.

Step 1 Power on and press **Del** to enter the BOIS setup menu.

Step 2 Ensure CSM is disabled.

- Intel Bay Trail boards: **Advanced > CSM Configuration > CSM Support > [Disabled]**
- Intel Haswell boards: **Boot > CSM parameters > Launch CSM > [Disabled]**

Step 3 Ensure that either the EHCI or XHCI USB controller is supported.



NOTE: The BIOS setup for the GIGABYTE GB-BXBT-3825 board does not have a menu to configure EHCI and XHCI.

Step 4 Ensure that Boot Option #1 points to the device you want to use.

To boot from a USB storage device in UEFI mode, do the following:

Boot > Boot Option #1 UEFI:<USB storage device name>

Deploying Images to Intel Bay Trail and Intel Haswell Boards

Use the **deploytool** script to place the system files on the boot media and use the boot media boot the device.

Step 1 Insert the USB flash drive into the host machine.

Step 2 Execute the **deploytool** script from your project directory.

This command deploys the image on your USB flash drive (**/dev/sdb**):

```
$ cd projDir
$ sudo ./deploytool -f \
export/images/fileName.tar.bz2 -d \
/dev/sdb -y -u
```

where *fileName.tar.bz2* is one of the following, depending on which BSP you used to build your project and which rootfs you specified:

BSP	rootfs	File name
intel-baytrail-64	idp	wrlinux-image-idp-intel-baytrail-64.tar.bz2
intel-baytrail-64	idp-dev	wrlinux-image-idp-dev-intel-baytrail-64.tar.bz2
intel-haswell-64	idp	wrlinux-image-idp-intel-haswell-64.tar.bz2
intel-haswell-64	idp-dev	wrlinux-image-idp-dev-intel-haswell-64.tar.bz2

You can set the rootfs partition size in bytes with the **-p** option. For example:

```
$ cd projDir
$ sudo ./deploytool -f \
export/images/fileName.tar.bz2 -d \
/dev/sdb -y -u -p "3G"
```

For more information on the **deploytool** script, see [About the deploytool Script](#) on page 92.

Step 3 Unplug the USB flash drive from your host machine.

Step 4 Plug the USB flash drive into the target and boot from it.



NOTE: The default user ID and password for Wind River targets are:

User ID: **root**

Password: **root**

Updating BIOS Images on GIGABYTE GB-BXBT-3825 Boards and other Intel Bay Trail Boards

Update a UEFI BIOS image on a GIGABYTE GB-BXBT-3825 and other Intel Bay Trail boards.



CAUTION: Ensure that the BIOS image you intend to use is for the corresponding GIGABYTE GB-BXBT-3825 board (see Step 1 on page 99). If you use the wrong BIOS image file for your board, you will damage the board beyond repair.



NOTE: The instructions of BIOS update vary among boards. The following instructions show how to update BIOS for the GIGABYTE GB-BXBT-3825 board.

To update a UEFI BIOS image on an Intel Bay Trail board, do the following:

- Step 1** Obtain the GIGABYTE GB-BXBT-3825 BIOS image file, for example **BAYADx64.F2a**, and copy it to your host.
- Step 2** Disconnect your GIGABYTE GB-BXBT-3825 board's power supply.
- Step 3** Boot to the UEFI shell.
- Step 4** Update the BIOS image with the following command:

```
fs1:\fpt -f fs1:\BAYADx64.F2a
fs1:\fpt -greset
```



NOTE: If the board is running with a 64-bit BIOS, use **fpt64** instead of **fpt**.

When you apply power to your target board, the BIOS image is updated.

Updating BIOS Images on ADLINK MXE-5401 Boards

Update a UEFI BIOS image on an ADLINK MXE-5401 board.



CAUTION: Ensure that the BIOS image you intend to use is for the corresponding ADLINK MXE-5401 board. If you use the wrong BIOS image file for your board, you will damage the board beyond repair.

The following instructions apply only to ADLINK MXE-5401 boards.

Prerequisites

You need a Windows machine to create a DOS system on a USB flash drive.

Obtain the BIOS update files and the **HP_USB_Disk_Storage_Format_Tool_v2.2.3** utility to create the bootable USB flash drive.

- Step 1** Disconnect the board from the power supply.
- Step 2** On your Windows host, create a FAT32 file system on the USB flash drive.
- Step 3** To create a bootable USB flash drive, on your Windows host, run **HP_USB_Disk_Storage_Format_Tool_v2.2.3**.



NOTE: This tool may be run in compatibility mode for Windows XP (Service Pack 3).

Step 4 Select **Create a DOS startup disk**.

Step 5 For the parameter **using system files located at**, select the path to **win98boot**.

Step 6 Copy the **PBIOS.BAT**, **AFUDOS.exe**, and **0303a.ROM** files and to the formatted FAT partition on the USB flash drive.

Step 7 Insert the USB flash drive into the board, reconnect the power supply, and power up the board.

Step 8 During the boot sequence, press **DEL** or **F2** to enter the BIOS setup.

Step 9 Boot into DOS from the USB flash drive.

Step 10 At the DOS prompt, type **PBIOS**.

Step 11 Wait for the BIOS update process to complete.

Step 12 Reboot the board.

Updating Flash Firmware for Intel 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: **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.

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:

projDir/export/images (symbolic link to *projDir/bitbake_build/tmp/deploy/images*)

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, Wind River recommends 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=xxxxxxxxxxx* 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 Connect the DediProg to the flash chip on your target board.

Step 4 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, the following error appears:

```
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** and then disable and re-enable the driver.

Updating Flash Firmware 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 includes 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** Press **DEL** or **F7** to open the boot manager menu.
- Step 5** Select **UEFI Internal Shell**.
- 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 for a Cross Hill board is **Flash-crosshill-8M-secured.cap**.

The CapsuleApp updates your SPI flash firmware, which takes about three minutes. If the update succeeds, the system boots normally.

Updating Flash Firmware 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 *projDir/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**.



CAUTION: 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.

Migrating Intel Quark Flash Firmware from IDP XT 2.0.x to IDP XT 3.1.x

When you migrate from IDP XT 2.0.4 to IDP XT 3.1.x, you must upgrade the flash firmware on Intel Quark-based boards.

If your board uses the Intel Quark SoC X1020 / X1021, contact your hardware vendor before following these instructions. The capsule images used for the firmware upgrade must be signed with the key owned by hardware vendor. The hardware vendor must provide the signed capsule images used for the firmware upgrade.

You can upgrade the firmware in one or two steps. The diagnostic tool specifies which process to use. The one-step process does not preserve the board serial number.

Prerequisites

You need the following:

- a target board running the latest IDP XT 2.0.x release
- the Linux capsule files for the firmware update

You can use the prebuilt capsule files provided with the binary release (recommended), or you can build the capsule files from your platform project. See the **README.txt** file provided in the *projDir/layers/wr-bsps/intel-quark/downloads/firmware_upgrade-tools-vversion.zip* file.

- boot media prepared with images for IDP XT 3.1.x

For information about how to create an image and the boot media for Intel Quark boards, see [Building Platform Projects for Intel Quark Boards](#) on page 93.

Step 1 Unzip the upgrade tools zip file and copy the firmware diagnostic tool from your host to your target board running IDP XT 2.0.x.

```
$ cd projDir
$ unzip /layers/wr-bsps/intel-quark/downloads/firmware_upgrade-tools-vversion.zip -d
$PWD
$ service sshd start
$ scp firmware_upgrade-tools-vversion/cln_fwtool_s root@targetIP:/root
```

Step 2 On the target board running IDP XT 2.0.x, verify the version of firmware running on the board.

```
# cd /root
# ./cln_fwtool_s diagnosis
```

Step 3 If the output of the diagnostic tool specifies that you must use the two-step process, on the target board running IDP XT 2.0.x, verify the board serial number.

```
# cat /sys/class/dmi/id/board_serial
# dd if=/dev/mem of=/tmp/pdata bs=64K skip=65521 count=2
# hexdump -C /tmp/pdata | head
```

You will use this information after the firmware update to confirm that the serial number is the same after the update.

Step 4 Insert the boot media you prepared with IDP XT 3.1.x into the board.
Do not reboot the board at this time. You will reboot it later as part of the capsule update process.

Step 5 Based on the output of the diagnostic tool in step 2 on page 103, use the appropriate capsule files and follow the procedure to update the flash firmware.

Diagnostic Tool Output	Upgrade Procedure
You need to run 1-step process of r1.2 firmware upgrade to preserve the contents of above asset	Follow the procedure in section Updating Flash Firmware Using Capsule Update in Linux on page 102 using the following capsule file: fw_upgrade_1step.cap
You need to run 2-step process of r1.2 firmware upgrade to preserve the contents of above asset	1. Follow the procedure in section Updating Flash Firmware Using Capsule Update in Linux on page 102 using the following capsule file: fw_upgrade_2step_1.cap 2. Follow the procedure in section Updating Flash Firmware Using Capsule Update in Linux on page 102 again using the following capsule file: fw_upgrade_2step_2.cap

During the boot process after the firmware update, if necessary, select the boot media prepared in step 4 on page 104 in the GRUB menu.

The flash capsule update takes place when you reboot the board. If you use the two-step method, the update is complete after the second capsule update. When your board first boots, it is now running IDP XT 3.1.x from your boot media.

Step 6 Verify the firmware version, MAC address, and board serial number (if applicable).

```
# cat /sys/class/dmi/id/bios_version
0x010200F4
# ifconfig eth0 | grep HWaddr
eth0      Link encap:Ethernet  HWaddr 00:08:a2:09:06:04
# cat /sys/class/dmi/id/board_serial
```

If you used the two-step process, the board serial number should match the value in step 3 on page 104.

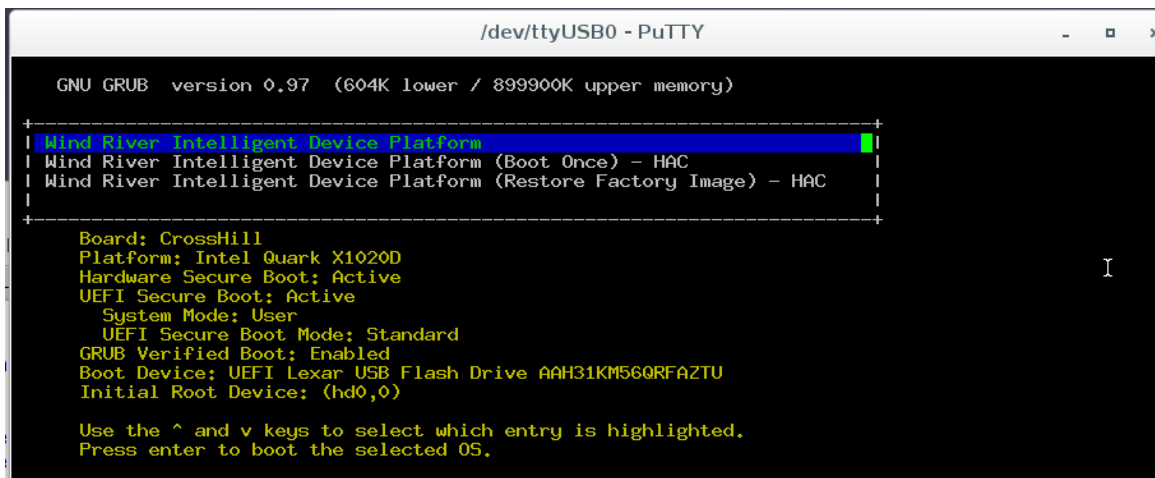
You have successfully migrated the firmware on your board.

About the GRUB Boot Menu Information

The GRUB boot menu displays security and platform statistics you can use to confirm that your board boots the correct software.

For example, if you changed the **grub.conf** file, you can use the information to confirm that the boot behavior matches your changes.

The following shows an example of the information displayed on the boot menu.



```

/dev/ttyUSB0 - PuTTY

GNU GRUB  version 0.97  (604K lower / 899900K upper memory)

+-----+
| Wind River Intelligent Device Platform |
| Wind River Intelligent Device Platform (Boot Once) - HAC |
| Wind River Intelligent Device Platform (Restore Factory Image) - HAC |
+-----+

Board: CrossHill
Platform: Intel Quark X1020D
Hardware Secure Boot: Active
UEFI Secure Boot: Active
System Mode: User
UEFI Secure Boot Mode: Standard
GRUB Verified Boot: Enabled
Boot Device: UEFI Lexar USB Flash Drive AAH31KM56QRFATU
Initial Root Device: (hd0,0)

Use the ^ and v keys to select which entry is highlighted.
Press enter to boot the selected OS.

```

The boot menu displays the following information:

Field	Description	Valid Values
Board	The board name retrieved from the SMBIOS (System Management BIOS) Baseboard (or Module) Information (Type 2), offset 5.	BIOS dependent
Platform	The codename or product name of the Intel processor on target.	The SKUs of Intel Bay Trail SoC series: • Intel Bay Trail-D • Intel Bay Trail-I • Intel Bay Trail-M • Intel Bay Trail-T The SKUs of Intel Quark SoC X1000 series: • Intel Quark X1000 • Intel Quark X1001 • Intel Quark X1010 • Intel Quark X1011

Field	Description	Valid Values
		- Intel Quark X1020 - Intel Quark X1020D - Intel Quark X1021 - Intel Quark X1021D The generic Intel processors: - Generic 32-bit x86 (vendor 0xX, device 0xY, platform 0xZ) - Generic 64-bit x86 (vendor 0xX, device 0xY, platform 0xZ) where X, Y, and Z indicate the PCI vendor ID, PCI device ID, and platform ID retrieved from the host bridge and MSR IA32_PLATFORM_ID, respectively.
UEFI Secure Boot	The status of UEFI secure boot during the UEFI boot phase.	Active UEFI Secure Boot is enabled. Inactive UEFI secure boot is disabled. Unsupported The UEFI firmware does not support UEFI secure boot.
System Mode	The system mode during the UEFI secure boot phase. This field does not exist if the value of the field UEFI Secure Boot is Unsupported .	Setup The UEFI firmware is not provisioned with a PK. User The UEFI firmware is provisioned with a PK.
UEFI Secure Boot Mode	The mode of UEFI Secure Boot. This field does not exist if the value of the field UEFI Secure Boot is Unsupported .	Custom Allows you manipulate the keys.

Field	Description	Valid Values
		Standard Disallows you from manipulating the keys.
GRUB Verified Boot	The status of GRUB verified boot during the GRUB boot phase.	Enabled GRUB verified boot is enabled, which will verify the signature files (.auth), including the configuration of GRUB, the kernel image, and the initramfs image. Disabled GRUB verified boot is disabled; therefore, there is no authentication check executed for the configuration of GRUB, the kernel image, and the initramfs image. Unsupported The GRUB boot loader does not support GRUB verified boot.
Boot Device	The description of current boot device.	Boot device dependent
Initial Root Device	The device from which the configuration of GRUB is initially loaded. If the root command in the GRUB boot menu is not explicitly specified, GRUB loads the kernel image and initramfs image from the initial device. If the root command in the GRUB boot menu is explicitly specified, the value of this field may be different from the initial device, and GRUB loads the kernel image and initramfs image from the device specified by the root command.	(hd X,Y), where X is the device drive number starting from zero and Y is the partition number on the drive starting from zero.

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) can be updated.

Prerequisites

You need the new tarball with the updated rootfs, kernel image, and boot loader. For information on generating the tarball, see [Preparing to Build and Boot IDP](#) on page 91.

Ensure that the volume group on your boot media has at least 2 GB of free space. For example, assuming the device on the host is `/dev/sdb` when you deploy the initial image, the following uses 30% of the volume group for the rootfs:

```
$ sudo ./deploytool -d /dev/sdb -f image.tar.bz2 \
-y -L 30%VG
```

The remaining 70% of the volume group must be at least 2 GB.



NOTE: By default, the LVM volume is used for the root file system.

For more information about deploying images, see the following:

- [Deploying Images to Intel Quark Boards](#) on page 94
- [Deploying Images to Intel Bay Trail and Intel Haswell Boards](#) on page 98

Step 1 Boot the target from removable media and log in.

Step 2 On the boot media, create a logical volume in the same volume group as the existing LVM partition on the media.

In this example, the existing volume group name is **VGWR6097**.

```
$ vgdisplay
--- Volume group ---
VG Name                VGWR6097
System ID
Format                 lvm2
Metadata Areas         1
Metadata Sequence No   11
VG Access               read/write
VG Status               resizable
MAX LV                 0
Cur LV                 2
Open LV                 2
Max PV                  0
Cur PV                 1
Act PV                  1
VG Size                 7.35 GiB
PE Size                 4.00 MiB
Total PE                1882
Alloc PE / Size         1358 / 5.30 GiB
Free PE / Size           524 / 2.05 GiB
VG UUID                 W9q51N-YQyQ-ii29-n933-5Rn1-b5y7-0onxd3

$ lvcreate --size 2G --name newlvm VGWR6097 -Z n
WARNING: "newlvm" not zeroed
Logical volume "newlvm" created
```

Step 3 Format the new partition and mount the partition in the file system.

```
$ mkfs.ext3 /dev/VGWR6097/newlvm
$ mkdir /mnt/newlvm
$ mount /dev/VGWR6097/newlvm /mnt/newlvm
```

Step 4 Extract the new tarball into the new volume.

The following example is for an Intel Bay Trail board. It assumes that you copied the image file from your development host to the **/root** directory on the target and that the new volume is mounted in the file system as **/mnt/newlvm**.

```
$ cd /root
$ tar -jxvf intel-baytrail-64-idp-idp-dist.tar.bz2 -C /mnt/newlvm
```

Step 5 To change the rootfs, update the **/boot/efi/rootfs** file to point to the new logical volume.

```
$ echo "LVM=/dev/VGWR6097-newlvm" > /boot/efi/rootfs
```

Step 6 To update the kernel, copy the **bzImage**, **idp-initramfs.img**, and the **.auth** files from the **/mnt/newlvm/boot** directory to the **/boot/efi** directory.

Step 7 To update the boot loader, copy the **.efi**, **.conf**, and **.auth** files from the **boot/efi/EFI/boot** directory on the new volume to the **/boot/efi/EFI/boot** directory.

Step 8 Ensure all changes are saved to the media.

```
$ sync
```

Step 9 For Intel Quark boards, update the firmware, if needed.

For more information, see [Updating Flash Firmware for Intel Quark Boards](#) on page 100.

Step 10 Reboot the target.

Related Links

[Signing Boot Loaders](#) on page 37

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

[Building Platform Projects for Intel Quark Boards](#) on page 93

Configure and build a platform project configured for an Intel Quark board.

[Building Platform Projects for Intel Bay Trail and Intel Haswell Boards](#) on page 95

Configure and build a platform project for Intel Bay Trail and Intel Haswell boards.

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.

Prerequisites

To use the recovery image, you must have the following minimum-size media:

- A USB Flash drive of at least 4 GB

- An MMC card or SSD of at least 2 GB

Creating a Recovery Image

Follow the instructions starting with step 1 on page 110.



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 copy them to a USB flash drive.
 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 110. Replace **export/usb.img** with the full path to **recovery.img**.

Step 1 Configure a platform project using **--with-template=feature/recovery**.

For example, for an Intel Bay Trail board, do the following:

```
$ $WIND_LINUX_CONFIGURE_CLI --enable-board=intel-baytrail-64 --enable-addons=wr-idp \
--enable-kernel=idp --enable-rootfs=idp \
--with-template=feature/recovery
```

Step 2 Build the rootfs.

```
$ make
```

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.

For example, for an Intel Bay Trail board, do the following:

```
$ sudo ./deploytool \
-f export/intel-baytrail-64-idp-idp-dist.tar.bz2 \
-d export/usb.img -r
```

Step 4 Copy the image to a USB flash drive.

```
$ sudo dd if=export/usb.img of=/dev/sdb bs=4M
$ sudo sync
```

Change **sdb** to match the device name of your USB flash drive on your host computer.

Step 5 Insert the USB flash 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 GIGABYTE GB-BXBT-3825 board:

```
# /sbin/deploytool -d /dev/sda --reset-media
```

This example copies the rootfs to the SD card:

```
# /sbin/deploytool -d /dev/mmcblk0 --reset-media
```

IDP Preconfigured Profiles

IDP XT 3.1 comes with two preconfigured profiles, **idp** and **idp-dev**. They add specific layers, features, and packages to your platform project.

Preconfigured Profile Customization

You can customize the preconfigured profiles by modifying the bitbake recipes:

idp profile

```
projDir/layers/wr-idp/wr-idp-base/recipes-base/images/wrlinux-image-idp.bb
```

idp-dev profile

```
projDir/layers/wr-idp/wr-idp-base/recipes-base/images/wrlinux-image-idp-dev.bb
```

Layers, Features, and Packages Included in Preconfigured Profiles

The following table shows which layers and features provide the packages for the preconfigured **idp** profile and indicates which layers and packages are included in the **idp-dev** profile.

Layer	Features and Packages	Included in idp-dev
wr-idp-devkit	feature/luci (including wifi , luci , and ntp)	Yes
	feature/firewall	Yes
	feature/grsec	No
	feature/pppoe	Yes
	feature/mqtt	Yes
wr-srm	srm (default)	No
wr-mcafee-essential	solidcores3 (package)	No
oe-core	dhcp (package) (including dhcpcv6)	Yes
	perl (package)	
	python (package)	

Layer	Features and Packages	Included in idp-dev
meta-networking (from Wind River Linux 7.0)	radvd (package)	Yes

Platform Boot Time Optimizations

There are build options available that can speed up target boot time.

By default, IDP XT uses **systemd** to run boot services in parallel to speed up boot time and utilize the multiprocessing capabilities of Linux. To improve boot time, you can disable services that are not required. For a list of services that are started automatically at boot time, see [IDP Services Reference](#) on page 203.

Disabling Security Features

IDP XT includes two layers with security features by default: **wr-srm** and **wr-mcafee-essential**. During development, you can use the **idp-dev** profile, which excludes all security features by default: **wr-srm**, **wr-mcafee-essential**, **wr-mcafee**, **wr-ima-appraise**, and **grsecurity**. Improvements in boot time vary for each board type.

For more information on **wr-srm**, **wr-mcafee-essential**, and **wr-mcafee**, see [IDP Features](#) on page 11.

Configuring the Target at Boot Time

You can use a script to configure the target system the first time the system boots.

If you want to configure your target system at run time with information that only needs to be specified once, you can create an executable shell script that runs only the first time the system boots. This mechanism is referred to as **runonce**. For example, you might want to use it to auto-provision the system.

If the **/etc/runonce/customer-runonce.sh** script exists, the system executes it the first time system boots. If the script returns zero (success), the system disables the **runonce** mechanism. If the script returns any non-zero value, the service runs again the next time the boots. This enables you to indicate success or failure to the system. You can also use the **reboot** command to try again instead of returning from the script.

If the script does not exist when the device first boots but is subsequently installed or created later, it will run the next time the system boots, even though it is not the first system boot.

Step 1 Create a file called **customer-runonce.sh** that contains the actions to perform the first time the device boots.

Step 2 To add the script to the root file system in the **/etc/runonce** directory, include the file in a package that can be installed as an RPM.

Ensure that your file is contained in an RPM, otherwise it will not be updated as part of a system-wide update.

For more information about creating packages and recipes, see the *Wind River Linux User's Guide, 7.0: Recipes*.

Example: Allowing SSH from the WAN

```
#!/bin/sh
uci add firewall rule
uci set firewall.@rule[-1].src=wan
uci set firewall.@rule[-1].target=ACCEPT
uci set firewall.@rule[-1].proto=tcp
uci set firewall.@rule[-1].dest_port=22
uci commit firewall
systemctl restart firewall
```

Example: Setting the Device Wireless SSID

```
#!/bin/sh
uci set wireless.@wifi-device[0].disabled=0
uci set wireless.@wifi-iface[0].ssid=customerNet
uci commit wireless
```


17

Alternative Booting Methods

[SRM and Alternative Booting Methods](#) 115

[Secure Booting](#) 115

[Performing a Verified Boot](#) 119

SRM and Alternative Booting Methods

Secure Remote Management (SRM) provides the infrastructure to boot to a trusted software stack and to securely manage devices remotely.

The IDP XT layer **wr-srm** provides SRM functionality. 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.

Secure Boot

Supported for Cross Hill, Clanton Hill, GIGABYTE GB-BXBT-3825, and ADLINK MXE-5401 boards.

Verified Boot

Supported for Cross Hill, Clanton Hill, GIGABYTE GB-BXBT-3825, and ADLINK MXE-5401 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 Intel Quark Boards](#) on page 93.

Use the following configure command:

```
$ $WIND_LINUX_CONFIGURE_CLI --enable-board=intel-quark --enable-addons=wr-idp \
--enable-kernel=idp --enable-rootfs=idp
```

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 Intel Quark Boards](#) on page 100

Step 5 Deploy the SRM signed rootfs on the USB flash drive.

Follow the instructions for deploying the image and rootfs and rebooting the board in the following sections:

- [Deploying Images to Intel Quark Boards](#) on page 94
- [Deploying Images to Intel Quark Boards Manually](#)

Performing a Secure Boot Using UEFI on Intel Baytrail Boards

A secure boot policy allows only the signed boot loader (**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 Intel Bay Trail and Intel Haswell Boards](#) on page 95. For example, for an Intel Bay Trail board, use the following command:

```
$ $WIND_LINUX_CONFIGURE --enable-board=intel-baytrail-64 --enable-addons=wr-idp \
--enable-kernel=idp --enable-rootfs=idp
```

Step 2 Sign the rootfs file with SST.

For more information, see [Signing the rootfs File](#) on page 39.

Step 3 Update the BIOS on the GIGABYTE GB-BXBT-3825 board or other Intel Bay Trail boards.

For more information, see [Updating BIOS Images on GIGABYTE GB-BXBT-3825 Boards and other Intel Bay Trail Boards](#) on page 99

Step 4 Deploy the signed rootfs on a USB flash drive.

For more information see the following:

- [Deploying Images to Intel Bay Trail and Intel Haswell Boards](#) on page 98
- [Deploying Images to Intel Bay Trail and Intel Haswell Boards Manually](#)

Step 5 Insert the USB flash drive into an Intel Bay Trail board.

Step 6 Configure the BIOS and reboot the board.



NOTE: The instructions may vary among boards. The following instructions are provided for Gigabyte GB-BXBT-3825 boards.

Select the following menu items:

- a) **Advanced > CSM Configuration > CSM Support > [Disabled] > F4 > Reboot**
- b) **Security > Secure Boot menu > Secure Boot > [Disabled]**
- c) **Security > Secure Boot menu > Secure Boot > Custom**
- d) **Security > Secure Boot menu > Key Management > Delete All Factory Default Keys**
- e) **F4 > Reboot**

Step 7 Enter the EFI shell console.

Press **F12** and select **UEFI: Built-in EFI shell**.

Step 8 Enter the USB flash drive VFAT partition.

Select **fs1: enter U-Disk VFAT partition**.

Step 9 Run the signed **grub.efi** file **BOOTX64.efi**.

```
# cd EFI\BOOT
# ./BOOTX64.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 **F12** and select **Enter Setup**.
- b) **Security > Secure Boot menu > Secure Boot > [Enable] > F4 > Reboot**

Step 11 Enter the USB flash drive VFAT partition.

Select **fs1: enter U-Disk VFAT partition**.

Step 12 Run the signed **grub.efi** file **BOOTX64.efi**.

The **BOOTX64.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 USB flash drive.
- b) Insert the USB flash drive into the Intel Bay Trail board.
- c) Press **F12** and select **UEFI: Built-in EFI shell**.
- d) Run the unsigned **grub.efi** file **UNSIGNED_BOOTX64.efi**.

The following message appears:

```
fs1:\EFI\BOOT> UNSIGNED_BOOTX64.efi
Error reported: Access Denied
```

This is the correct result when the secure boot policy is working correctly.

Performing a Secure Boot Using UEFI on Intel Haswell Boards

A secure boot policy allows only the signed boot loader (**grub.efi**) to run on the UEFI.

Context for the current task

Step 1 Configure and build a platform project.

Follow the configure and build steps in [Building Platform Projects for Intel Bay Trail and Intel Haswell Boards](#) on page 95. For example, for an Intel Haswell board, use the following command:

```
$ $WIND LINUX CONFIGURE --enable-board=intel-haswell-64 --enable-addons=wr-idp \
--enable-kernel=idp --enable-rootfs=idp
```

Step 2 Sign the rootfs file with SST.

For more information, see [Signing the rootfs File](#) on page 39.

Step 3 Update the BIOS on the ADLINK MXE-5401.

For more information, see [Updating BIOS Images on ADLINK MXE-5401 Boards](#) on page 99.

Step 4 Deploy the signed rootfs on a USB flash drive.

For more information see the following:

- [Deploying Images to Intel Bay Trail and Intel Haswell Boards](#) on page 98
- [Deploying Images to Intel Bay Trail and Intel Haswell Boards Manually](#)

Step 5 Insert the USB flash drive into an Intel Haswell board.

Step 6 Power on and press **Del** or **F2** to enter the BIOS setup menu.

Step 7 Configure the BIOS and reboot the board.



NOTE: The instructions may vary among boards. The following instructions are provided for ADLINK MXE-5401 boards.

Select the following menu items:

- Advanced > CSM Configuration > CSM Support > [Disabled] > F4 > Reboot**
- Security > Secure Boot menu > Secure Boot > [Disabled]**
- Security > Secure Boot menu > Secure Boot > Custom**
- Security > Secure Boot menu > Key Management > Delete All Secure Boot Variables**
- F4 > Reboot**

Step 8 When the grub boot menu appears, press **CTRL+ALT+DEL** to reboot the board.

Step 9 Configure the BIOS again and reboot the board.

- Press **DEL** or **F2** to enter the BIOS setup menu.
- Security > Secure Boot menu > Secure Boot Control > [Enable] > F4 > Reboot**

The board reboots. if you try to update the target with an unsigned boot loader, the target fails to boot.

Performing a Verified Boot

Performing a verified boot involves configuring and building a platform project, and rebooting the device.

Prerequisites

In addition to the standard IDP XT prerequisites, you need the following software 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. Grub-0.97 confirms that the ciphers of all the components it loads match the kernel image digest.

Step 1 Configure and build a platform project.

For details, see [Building Platform Projects for Intel Bay Trail and Intel Haswell Boards](#) on page 95.

The following configure command uses the **intel-baytrail-64** BSP as an example:

```
$ $WIND_LINUX_CONFIGURE_CLI --enable-board=intel-baytrail-64 --enable-addons=wr-idp \
--enable-rootfs=idp --enable-kernel=idp
```

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. 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          ##
##                                     ##
#####
```


18

Configuring IDP Features

Layers and Features	121
About Configuring Layers	123
Inspecting Layer Contents	125
About Configuring Default Features	125
About Configuring Non-Default Features	126
Non-Default Features Included in the idp Rootfs	126
About the Secure Remote Management Layer	127

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 (idp rootfs)	Cross Hill	Clanton Hill	GIGA BYTE GB-BXB T-382 5	ADLI NK MXE-5401
meta-java-dl	N/A	N/A	N/A	Yes	Yes	Yes	Yes
wr-digi-idigiconnector	default	Yes	No	Yes	Yes	Yes	Yes

Layers	Features	Default (layer included)	Default (idp rootfs)	Cross Hill	Clanton Hill	GIGA BYTE GB- BxB T-382 5	ADLI NK MXE- 5401
wr-wks-oneagent-oma-dm-ia	default	Yes	No	Yes	Yes	Yes	Yes
wr-wks-oneagent-tr069	default	Yes	No	Yes	Yes	Yes	Yes
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-ma	default	Yes	No	Yes	Yes	Yes	Yes
wr-mcafee	default	Yes	No	Yes	Yes	Yes	Yes
wr-mcafee-essential	default	Yes	Yes	Yes	Yes	Yes	Yes
wr-srm	default	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
	OpenSSL TPM Engine (No separate feature template)	Yes	Yes	Yes	No	No	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

Layers	Features	Default (layer included)	Default (idp rootfs)	Cross Hill	Clanton Hill	GIGA BYTE GB- BxB T-382 5	ADLI NK MXE- 5401
	firewall	No	Yes	Yes	Yes	Yes	Yes
	graphics_qt	No	No	No	No	Yes	Yes
	grsec	Yes	Yes	Yes	Yes	Yes	Yes
	ipsec_vpn	No	No	Yes	Yes	Yes	Yes
	l2tp	No	No	Yes	Yes	Yes	Yes
	luci	No	Yes	Yes	Yes	Yes	Yes
	mqtt	No	Yes	Yes	Yes	Yes	Yes
	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
	realtek	No	No	Yes	Yes	Yes	Yes
	recovery	No	No	Yes	Yes	Yes	Yes
	vlan	No	No	Yes	Yes	Yes	Yes
wr-ima-appraise	default	Yes	No	Yes	Yes	Yes	Yes

About Configuring Layers

To include any IDP XT-related layer, you must specify **--enable-addons=wr-idp** option in your configure line.

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.

For descriptions of all IDP XT layers and features, see [IDP Features](#) on page 11.

IDP XT supports two different types of rootfs, specified using the **--enable-rootfs** option. The layers that are included by default depend on the rootfs you choose. Specify one of the following types of rootfs:

idp
idp-dev

Wind River recommends using the **idp-dev** rootfs for development. It excludes all security layers and features.

By default, all platform projects include the **wr-idp-devkit** layer.

The **idp** rootfs includes the following IDP XT additional layers:

wr-srm
wr-mcafee-essential

The *projDir*/layers/wr-idp/wr-idp-base/templates/rootfs.cfg file specifies the layers that are automatically included in the **idp** rootfs.

The following example includes the three IDP XT default layers in a project for the **intel-baytrail-64** BSP:

```
$ $WIND_LINUX_CONFIGURE_CLI --enable-addons=wr-idp \  
--enable-board=intel-baytrail-64 \  
--enable-kernel=idp --enable-rootfs=idp
```

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 project for the **intel-baytrail-64** BSP:

```
$ $WIND_LINUX_CONFIGURE_CLI --enable-addons=wr-idp \  
--enable-board=intel-baytrail-64 \  
--enable-kernel=idp --enable-rootfs=idp --without-layer=wr-srm
```

To confirm which layers are included in your project, view 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-ima-appraise** layer to a project for the **intel-baytrail-64** BSP:

```
$ $WIND_LINUX_CONFIGURE_CLI --enable-addons=wr-idp \  
--enable-board=intel-baytrail-64 --enable-kernel=idp \  
--enable-rootfs=idp --with-layer=wr-ima-appraise
```



NOTE: You do not need to include the default IDP XT layers again using the **--with-layer=** option. For example:

```
$ $WIND_LINUX_CONFIGURE_CLI --enable-addons=wr-idp \  
--enable-board=intel-baytrail-64 --enable-kernel=idp \  
--enable-rootfs=idp --with-layer=wr-srm
```

This configure line attempts to include the **wr-srm** layer twice: once by specifying **--enable-rootfs=idp** and again by specifying **--with-layer=wr-srm**. This does not cause problems with the build, but is unnecessary.

Inspecting Layer Contents

You cannot inspect IDP XT layers by listing the installation directory contents because layers are distributed as compressed **git** trees. Build a platform project to view the layer contents.

You can view all the **README** files for all the layers included in your project in the *projDir/READMEs* directory after running the **configure** command.

The source code for some layers, such as the **wr-idp-devkit** and **wr-srm** layers, is not available when you list the directory. For example:

View the layers in a platform project.

a) Create a platform project.

The following configure command uses the **intel-baytrail-64** BSP as an example:

```
$ $WIND_LINUX_CONFIGURE_CLI --enable-addons=wr-idp \
--enable-board=intel-baytrail-64 \
--enable-kernel=idp --enable-rootfs=idp
```

b) List the project layers directory.

```
$ ls -l projDir/layers/wr-idp/wr-srm
total 64
drwxr-xr-x  2 user users 4096 Jan  4 14:11 conf
drwxr-xr-x  2 user users 4096 Jan  4 14:11 downloads
drwxr-xr-x  3 user users 4096 Jan  4 14:11 files
-rw-r--r--  1 user users 8903 Jan  4 14:11 README
drwxr-xr-x  3 user users 4096 Jan  4 14:11 recipes-base
drwxr-xr-x 11 user users 4096 Jan  4 14:11 recipes-core
drwxr-xr-x  5 user users 4096 Jan  4 14:11 recipes-devtools
drwxr-xr-x  4 user users 4096 Jan  4 14:11 recipes-extended
drwxr-xr-x  3 user users 4096 Jan  4 14:11 recipes-initramfs
drwxr-xr-x  3 user users 4096 Jan  4 14:11 recipes-kernel
drwxr-xr-x  5 user users 4096 Jan  4 14:11 recipes-luci
drwxr-xr-x  3 user users 4096 Jan  4 14:11 recipes-samples
drwxr-xr-x  4 user users 4096 Jan  4 14:11 recipes-uefi
drwxr-xr-x  4 user users 4096 Jan  4 14:11 templates
```

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 features using the **--with-template** option.

The following configure command uses the **intel-baytrail-64** BSP as an example:

```
$ $WIND_LINUX_CONFIGURE_CLI --enable-board=intel-baytrail-64 --enable-kernel=idp \
--enable-rootfs=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 121

However, certain non-default features from certain layers are included automatically; 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*.

Non-Default Features Included in the idp Rootfs

Some non-default features are included in your project automatically whenever you specify **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
luci	wr-idp-devkit
mqtt	wr-idp-devkit
pppoe	wr-idp-devkit

All other non-default features can be included using the **--with-template** option.

The following configure command uses the **intel-baytrail-64** BSP as an example:

```
$ $WIND_LINUX_CONFIGURE_CLI --enable-board=intel-baytrail-64 --enable-addons=wr-idp \
--enable-kernel=idp --enable-rootfs=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 121.

About the Secure Remote Management Layer

Secure Remote Management (SRM) is included in IDP XT layer 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 or unless you specify the **--enable-rootfs=idp-dev** option.

SRM is supported for the following BSPs:

- intel-quark** for Cross Hill and Clanton Hill
- intel-baytrail-64** for GIGABYTE GB-BXBT-3825
- intel-haswell-64** for ADLINK MXE540

For more information on including the SRM layer or excluding it from a platform project, see [About Configuring Layers](#) on page 123.

19

Installing Tools for Application Development and Control

Including Bluetooth in a Platform Project	129
Enabling 3G WWAN	130
Enabling IMA Appraise	131
Configuring OpenJDK	132
OSGi Development Workflow	136
Installing Sqlite3	141
Installing MQTT and Lua	142
Configuring Encrypted Storage	142
Installing OneAgent TR-069	149
Installing OMA-DM	150
Configuring PaX in the Kernel	151
Installing Wind River OPC	151

Including Bluetooth in a Platform Project

Enabling Bluetooth support requires configuring your platform project with Bluetooth support.

Step 1 Run your desired configure line.

The following configure command uses the **intel-baytrail-64** BSP as an example:

```
$ $WIND_LINUX_CONFIGURE_CLI --enable-board=intel-baytrail-64 \  
--enable-kernel=idp --enable-addons=wr-idp --enable-rootfs=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.

Do not access carrier networks using unauthorized devices unless expressly authorized. You cannot use a SIM card in an unauthorized device unless you have authorization from the carrier or the applicable regulatory authority.

Prerequisites

Before you proceed, ensure that you have configured and built your platform project. For more information, see:

[Building Platform Projects for Intel Bay Trail and Intel Haswell Boards](#) on page 95

[Building Platform Projects for Intel Quark Boards](#) on page 93

Step 1 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 2 Deploy the kernel image and rootfs to a USB Flash drive and boot the target.

For more information, see:

[Deploying Images to Intel Bay Trail and Intel Haswell Boards](#) on page 98

[Deploying Images to Intel Bay Trail and Intel Haswell Boards Manually](#)

[Deploying Images to Intel Quark Boards](#) on page 94

[Deploying Images to Intel Quark Boards Manually](#)

3G starts automatically on boot.

Step 3 Confirm that **/etc/config/network** contains the correct information.

Alternatively, you can use the LuCI interface to confirm this information. Start LuCI and select **Network > Interfaces**.

This example shows sample values for the Telit HE910 and an AT&T SIM card.

```
# 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 'us'
option apn 'att'

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 'lbc7'
option ProdID '0021'
option SerialNumber '357164040778055'
option Rev '12.00.003'

config device 'sim_card'
option name 'sim_card'
option present 'Yes'
option IMSI '3104101123456789'
option operator 'AT&T'
option operator_code '3104101'
```

Step 4 Confirm that the pppoe module is present on the target.

```
# lsmod | grep "pppoe"
```

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.

Configure and build a platform project for your board.

In your `$WIND_LINUX_CONFIGURE_CLI` 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_CLI` command when you use `--without-layer=wr-srm`.

The following configure command uses the **intel-baytrail-64** BSP as an example:

```
$ $WIND_LINUX_CONFIGURE_CLI --enable-board=intel-baytrail-64 --enable-kernel=idp \
--enable-rootfs=idp --enable-addons=wr-idp --with-layer=wr-ima-appraise
```

For more information, see:

- [Building Platform Projects for Intel Bay Trail and Intel Haswell Boards](#) on page 95

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 Intel Bay Trail and Intel Haswell Boards](#) on page 95

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,recipes-core,templates}
$ mkdir -p templates/default
$ mkdir -p recipes-core/helloworld/files
```

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 += "myjavayer"
BBFILE_PATTERN_myjavayer := "^${LAYERDIR}/"
BBFILE_PRIORITY_myjavayer = "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/default** directory.

For this example, **template.conf** contains the following code:

```
MULTILIB_USE_LIB32 += "helloworld"
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=7491d70e1949a6b5e036c44aaa7296d4"
FILESEXTRAPATHS_prepend := "${THISDIR}/files:"

SRC_URI += "file://HelloWorld.tar.gz"
S = "${WORKDIR}/HelloWorld"
DEPENDS += "ant-native"

PACKAGES = "${PN}"

FILES_${PN} += "/usr/share/java"
RDEPENDS_${PN}_append_x86 += "openjdk-bin"
RDEPENDS_${PN}_append_multilib-lib32 += "lib32-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] = "a7af3b8d16ce382152f73cd40a4ce878"
SRC_URI[sha256sum] = "5d6484e7f40cf66fd7158265f3fbeb373ba9ab63070d7a2397715fa614a93e11"
```



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, 7.0: Recipes*.

e) Verify the directory structure.

```
$ tree
.
|-- conf
|   |-- layer.conf
|-- recipes-core
|   |-- helloworld
|       |-- files
|           |-- HelloWorld.tar.gz
|           |-- helloworld.bb
|-- templates
|   |-- default
|   |-- template.conf
```

Step 6 Copy the tar file into the **downloads** directory of the custom layer directory.

```
$ cp HelloWorld.tar.gz \
myJavaLayer/recipes-core/helloworld/files
```

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 helloworld
```

The built package is located in the following directory:

projDir/build/helloworld/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**

Step 10 Execute the HelloWorld program on the IDP XT target.

```
# java -jar /usr/share/java/HelloWorld.jar  
Hello, World
```

Related Links

[Building Platform Projects for Intel Bay Trail and Intel Haswell Boards](#) on page 95
Configure and build a platform project for Intel Bay Trail and Intel Haswell boards.

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.

The following configure command uses the **intel-baytrail-64** BSP as an example:

```
$ $WIND_LINUX_CONFIGURE_CLI --enable-board=intel-baytrail-64 --enable-kernel=idp \  
--enable-rootfs=idp --enable-addons=wr-idp --with-template=feature/openjdk-bin
```

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.

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.

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-baytrail-64, intel-haswell-64	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 GIGABYTE GB-BXBT-3825 and ADLINK MXE5401 boards:

```
$ unzip ProSyst_mBS_SH_SDK_7.5_Board_Extension_Atom.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 step 2 on page 137.

Postrequisites

Once the SDKs are installed, see [Enabling Eclipse for ProSyst Smart Home Development](#) on page 138, to complete the process of setting up your development environment.

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 version 4.2 (Juno) or higher 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 140

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 140.

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 139), 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 GIGABYTE and ADLINK boards: IDP_XT_2.0.2/Atom For Intel 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 140.

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 140), 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 Intel Bay Trail and Intel Haswell Boards](#) on page 95

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 transfer 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` or

`http://targetIpAddr:8080/system/console`

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 Intel Quark Boards](#) on page 93
- [Building Platform Projects for Intel Bay Trail and Intel Haswell Boards](#) on page 95

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 broker 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 Intel Bay Trail and Intel Haswell Boards](#) on page 95

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:

lvm2-2.02.97
cryptsetup-1.6.6
trousers-dev-0.3.13

- Familiarity with how TPM works. For more information, see *Wind River Intelligent Device Platform XT Security Guide*.

Enabling Encrypted Storage

Enabling encrypted storage requires building a platform project, creating a sealed key, and setting up a **dm-crypt** partition.

Step 1 Configure and build a platform project.

For more information, see:

- [Building Platform Projects for Intel Quark Boards](#) on page 93
- [Building Platform Projects for Intel Bay Trail and Intel Haswell Boards](#) on page 95

The following configure command uses the **intel-baytrail-64** BSP as an example:

```
$ $WIND_LINUX_CONFIGURE_CLI --enable-rootfs=idp --enable-addons=wr-idp \
--enable-kernel=idp --enable-board=intel-baytrail-64
```

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 flash drive.
3. Booted the device from the USB flash 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) Save and exit the BIOS settings; press **F4** and then to access the BIOS settings again, press **DEL** or **F7**.

- e) In the menu, enable TPM.

**Advanced > Trusted Computing > TPM Configuration > Security Device Support
[Enabled] TPM State [Enabled]**

- f) Confirm the TPM status.

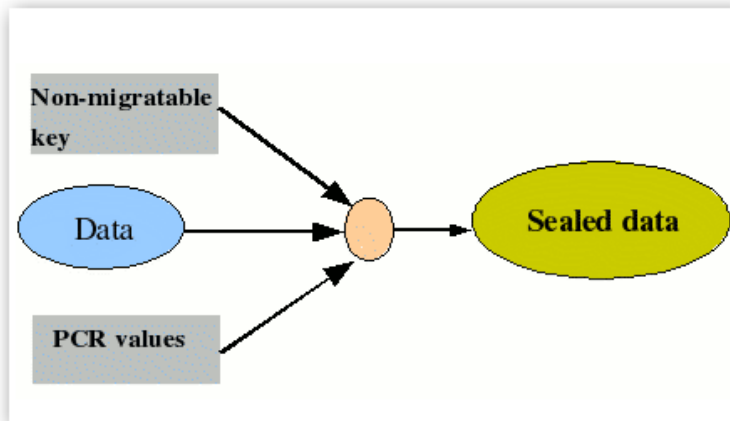
```
Current TPM Status Information
TPM Enabled Status: [Enabled]
TPM Active Status: [Activated]
```

- g) Save and exit the BIOS settings; press **F4**.

Step 3 Log in to 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** daemon 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
```

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 **HI** value before the **SI** 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 -l /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/mysecret_dir  
# mount /dev/mapper/secret-loop /home/mysecret_dir
```

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/mysecret_dir/secret_text_file.txt << EOF  
this is a secret text  
EOF
```

Step 2 Read the contents of the file.

```
# cat /home/mysecret_dir/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 -H1 unlimited
# ulimit -S1 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-loop --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 -H1 unlimited
# ulimit -S1 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/mysecret_dir
```

The device mounts successfully.

Step 8 Confirm the mount by reading `secret_text_file.txt`.

```
# cat /home/mysecret/secret_text_file.txt
this is a secret text
```

You can access and read `secret_text_file.txt` because the keys match.

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**.



NOTE: The `mkfs.ext3` utility used to format the USB flash drive is included in the rootfs.

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 USB flash drive.

```
# fdisk -l
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 -l /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/mysecretmdir
```

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/mysecretmdir/secret_text_file.txt << EOF
this is a secret text
EOF
```

Step 2 Read the contents of the file.

```
# cat /home/mysecretmdir/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 14.04. 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 -H1 unlimited
# ulimit -S1 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/
sdcl 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.

Installing OneAgent TR-069

OneAgent TR-069 Agent enables you to remotely manage your device using software that complies with the TR-069 standard.

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

Add the following layer in your **configure** command:

```
--with-layer=wr-wks-oneagent-tr069
```

Postrequisites

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

Related Links

[OneAgent TR-069 Agent](#) on page 181

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

Installing OMA-DM

The OMA DM client and OMA DM server enable you to remotely manage your device using software that complies with the OMA Device Management standard.

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

Include the following layer in your **configure** command:

--with-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.

- 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 PaX in the Kernel

Modify the kernel configuration to enable the grsecurity PaX by making a temporary modification on the **make** command line.

Step 1 To include Pax in the kernel, modify the command line as follows:

```
$ make -C build linux-windriver.menuconfig && make \  
-C build linux-windriver.rebuild
```

The Linux kernel configuration menu appears.

Step 2 From the **Grsecurity** menu, use the arrow keys to select the options you want.

Step 3 When you finish, select **Exit** at each menu level you entered..

The kernel is rebuilt with the options you selected enabled.

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.

Include the following template in your **configure** command:

Template: **feature/opc**

Related Links

[Building Platform Projects for Intel Quark Boards](#) on page 93

Configure and build a platform project configured for an Intel Quark board.

[Building Platform Projects for Intel Bay Trail and Intel Haswell Boards](#) on page 95

Configure and build a platform project for Intel Bay Trail and Intel Haswell boards.

20

Customizing LuCI

About Customizing LuCI 153

About Customizing LuCI

The device owner's specification may require that you customize LuCI on the device during development.

You can use the Wind River LuCI customizations that are included with IDP XT as a guide for your own customizations. For additional information, see the following:

- <https://github.com/openwrt/luci>
- <http://wiki.openwrt.org/doc/howto/luci.essentials>
- <http://wiki.openwrt.org/doc/techref/luci>
- <https://github.com/openwrt/luci/issues>

21

Updating WPAN Firmware for Intel Quark Boards

Cross Hill boards provide a utility for updating WPAN firmware.

Run the firmware update command on the device.

```
# q58_programmer_x86_v0.9.9
```

PART V

Application Development Vendor Tasks

Application Development.....	159
Exegin ZigBee Stack.....	161
Wind River OpenJDK.....	163
OSGi Development with the MBS Smart Home SDK.....	165
Sqlite3 Database.....	169
MQTT and Lua.....	175
Encrypted Storage.....	179
OneAgent TR-069 Agent.....	181
Works Systems OneAgent OMA Agent.....	183
The grsecurity Tool.....	187
Authentication, Authorization, and Auditing.....	193

22

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 22.

Wind River OpenJDK

Provides Java-based virtual machine (VM) platform development. See [Installing OpenJDK](#) on page 132.

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 165.

SQLite3

Provides a lightweight database for embedded applications. See [SQLite3 Command Reference](#) on page 169 and [SQLite3 Data Element Reference](#) on page 170.

MQTT

Consists of the open source broker Mosquitto and a client that includes utilities for publishing and subscribing to MQTT topics. See [About MQTT and Lua](#) on page 175.

Lua

Provides an open source, powerful, light-weight programming language designed for extending applications. It is also frequently used as a general-purpose, standalone language. See [About MQTT and Lua](#) on page 175.

Encrypted storage

Provides secure storage for application data on Intel Architecture devices. See [Encrypted Storage](#) on page 19.

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 181.

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 26.

grsecurity RBAC

Provides full learning mode for generating security policy rules. See [grsecurity and Related Tools](#) on page 187.

Authentication, Authorization, and Auditing

Supports AAA (Authentication, Authorization, and Auditing) services, such as Microsoft Active Directory, RADIUS, LDAP and TACACS+.

The layer descriptions in [Layers and Features](#) on page 121 provide the layer name and installation location for each development option listed in this section.

23

Exegin ZigBee Stack

[About the ZigBee Stack](#) 161

[Setting Up a ZigBee Network](#) 161

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 3.1 installation on top of a Wind River Linux 7.0 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 3.1 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.

24

Wind River OpenJDK

[About OpenJDK](#) 163

[Basic OpenJDK Command Reference](#) 163

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)

Basic OpenJDK Command Reference

The virtual machine supplied with OpenJDK is Zero Virtual Machine.

Commands

- To execute programs:

```
# java name-of-class-with-main-method arguments
```



NOTE: You must compile your Java source file (**.java**) with the Java compiler **javac** before you can execute on the IDP XT target.

- To execute JAR programs:

```
# java -jar name-of-jar-file.jar
```

- To check the version of the Java environment:

```
# java -version
```


OSGi Development with the MBS Smart Home SDK

[OSGi Development with the mBS Smart Home SDK](#) 165

[Developing with OSGi](#) 166

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-baytrail-64** and **intel-haswell-64** BSPs.

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.

Prerequisites

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 136.

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, the following message appears:

```
[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` or

`http://targetIpAddr:8080/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

26

Sqlite3 Database

[Using Sqlite3](#) 169

[Sqlite3 Command Reference](#) 169

[Sqlite3 Data Element Reference](#) 170

[Sqlite3 Examples](#) 171

Using Sqlite3

Sqlite3 is a lightweight database used primarily for embedded systems.

Prerequisites

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>
```

Sqlite3 Command Reference

Use the `sqlite3` command to enter the Sqlite3 command terminal.

<code>.database</code>	Check database file information
<code>.schema</code>	Show the <code>create</code> statements
<code>.schema table_name</code>	Show specified table <code>create</code> statements

.dump <i>table_name</i>	Output table content as sql statements
.help	Output help information
.quit or .exit	Exit Sqlite terminal

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.

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;
```


27

MQTT and Lua

About MQTT and Lua	175
The Lua Language	175
Starting the MQTT Broker	177
About Publishing and Subscribing to Messages	177

About MQTT and Lua

MQTT consists of the open source broker Mosquitto and a client that includes utilities for publishing and subscribing to MQTT topics. The client uses the Lua language.

The MQTT broker is Mosquitto, an open source implementation of a broker 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.

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 package is located at:

`projDir/layers/wr-idp/wr-idp-devkit/downloads/lua-version.tar.gz`

where the default *version* is 5.1.5.

The Lua source code package contains the reference documentation. The reference documentation is the official definition of the Lua language. After you uncompress the source package, the documentation is located in the following file:

`lua_sources_path/doc/contents.html`



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.

Examples: Using the Lua Program Examples

The Lua source code package contains the Lua application program examples.

Step 1 On your host computer, uncompress the Lua source code package.

```
$ tar -zxvf projDir/layers/wr-idp/wr-idp-devkit/downloads/luar-version.tar.gz \
-C lua sources path
```

Step 2 Transfer the **.lua** files from your host to your IDP XT target.

```
$ cd lua_sources_path/lua-version/test
$ scp *.lua root@IP-Address-of-IDP-Target:/root/examples
```

Step 3 On your target, execute the Lua example files.

The following examples show the commands and the system output for the various example files.

```
# cd /root/examples
# lua hello.lua
Hello world, from Lua 5.1!
# luac hello.lua
# lua luac.out
Hello world, from Lua 5.1!
# lua echo.lua WindRiver IDP 2.1
0      echo.lua
1      WindRiver
2      IDP
3      2.1
# 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
# lua life.lua
-----
--00-----0-----
-00-----0-0-----
--0-----0-----
-----
-----
-----
-----
-----
-----
--0-----
--0-0-----
--0-0-----
--0-----
-----
-----00-----
-----00-----
-----
Life - generation 2000
```

Starting the MQTT Broker

Start the MQTT broker using the **mosquitto** service. You can customize the Mosquitto configuration by modifying the **/etc/mosquitto/conf.d** file.

Step 1 Check to see if the mosquitto service is already running.

```
$ ps -ef | grep mosquitto
```

Step 2 If mosquito service is not running, start or restart it.

```
$ systemctl {start|restart} mosquitto
```

Step 3 (Optional) Modify the configuration of the MQTT broker.

- a) If you need to update any configuration items, update **/etc/mosquitto/conf.d**.
- b) Restart the service.

```
# systemctl restart mosquitto
```

About Publishing and Subscribing to Messages

MQTT allows you to publish messages on the broker 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
```

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 broker 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**.

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.

28

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 encryption 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.

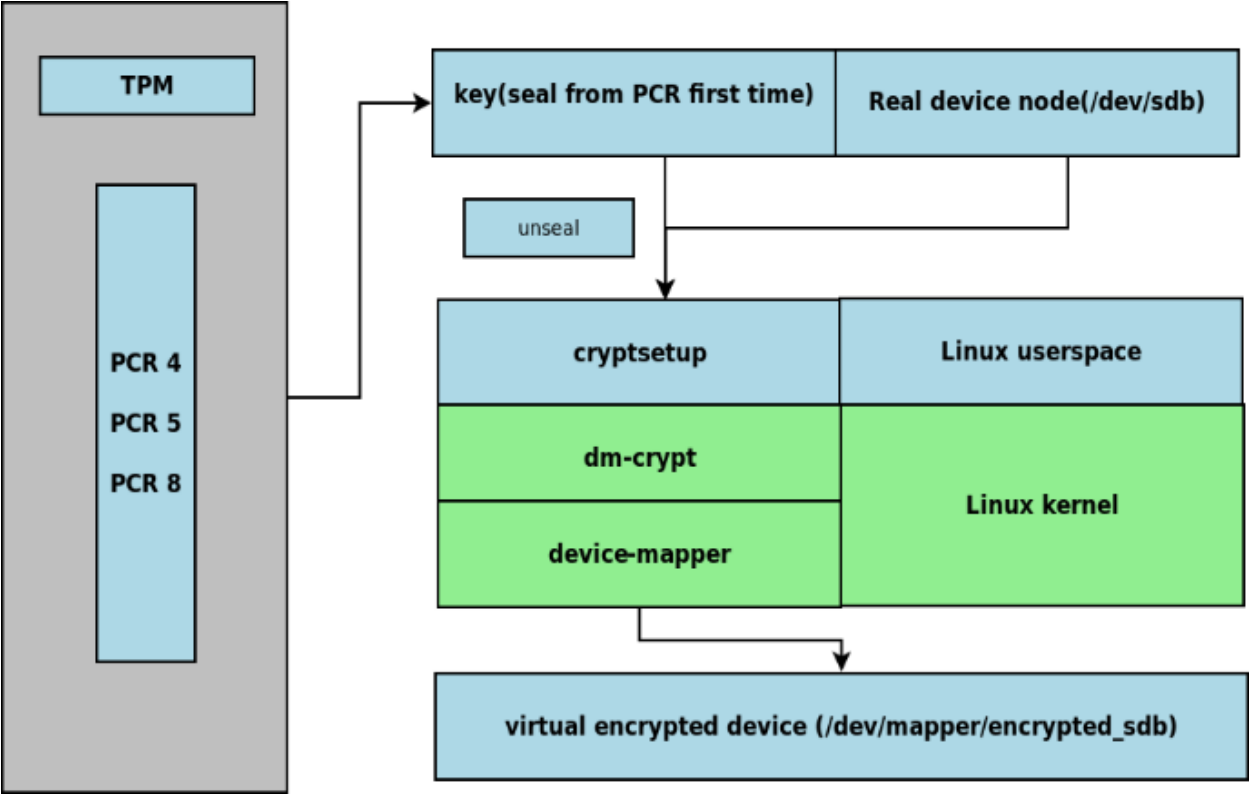
Encryption Process

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.



29

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-idp/wr-wks-oneagent-tr069/templates/default* directory:
 - **README**
 - *OneAgent TR 5.5 Integration Guide*

Works Systems OneAgent OMA Agent

[About the OMA-DM Agent](#) 183

[About MO Wrappers](#) 184

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/*
```

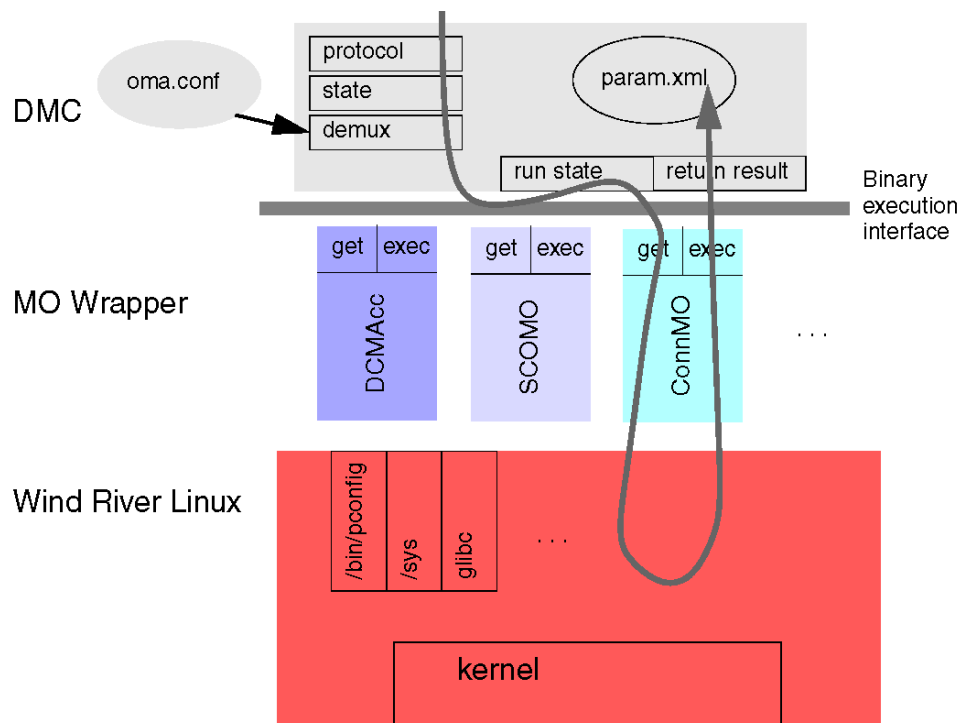
Most of the configuration files for the OMA-DM client are placed in **/etc/oma/**.

You can view additional OMA-DM debug messages in the **/var/log/messages** 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`

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.

31

The grsecurity Tool

grsecurity and Related Tools	187
grsecurity RBAC Command Reference	188
paxctl Reference	188
Generating a Security Policy for the Package	190
The grsecurity sysctl Interface	190
Troubleshooting grsecurity	191

grsecurity and Related Tools

IDP XT uses grsecurity to provide an example security policy and tools to customize the policy.

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. An example policy that you can use as a starting point is located in the `/etc/grsec/policy.example` file.

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 190. 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 190.

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/>

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	Disable 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

paxctl Reference

The user-space utility for controlling the PaX flags of executable files is **paxctl**.

The following 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
-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 `paxctl` 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
```

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 <code>/etc/grsec/policy</code> and restart grsecurity RBAC system in one of the following ways: Method 1: <pre># echo "windriver" gradm -D # gradm -E</pre> Method 2: <pre># gradm -a admin # gradm -R</pre>
If you have experience with the grsecurity RBAC policy:	Start full-learning mode, which can generate a reference policy file for you. <pre># 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</pre>
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

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/grsec_lock
1
```

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. You have the following options:

- 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 188.
- You can disable other flags using the **paxctl** tool depending on your error message.

Using Breakpoints

PaX may prevent GDB from setting software breakpoints, depending on how the kernel is configured. PaX disallows writes in executable memory for security reasons; therefore, the debugger cannot modify the code to add the breakpoint. This includes the breakpoint at **main()** where debugging starts. There are two workarounds with different effects and constraints to solve this.

Option 1: Use software breakpoints and remove the RANDEXEC and MPROTECT flags

The advantage of software breakpoints is that there is no limit on the number of breakpoints you can set.

To allow software breakpoints, use the **-x** (set by default) and **-m** flags with the **paxctl** command to disable the RANDEXEC and MPROTECT features for the binary to debug.

```
# /sbin/paxctl -m binary
```

You should now be able to use GDB to add software breakpoints on the binary. If not, use the **-s** and **-p** flags to disable the SEGMEEXEC and PAGEEXEC features.

```
# /sbin/paxctl -sp binary
```

Option 2: Use hardware breakpoints

Hardware breakpoints do not require any changes to PaX behavior, but the number of breakpoints is usually limited (for example, some processors allow only four breakpoints,

including address watchpoints), and also require the program to be running before adding breakpoints.

To use hardware breakpoints, in GDB, use the **hbreak** command instead of the **break** command.

To restore the system to its normal state after debugging, use the **-z** flag to reset all the options. If you want to keep trampoline emulation disabled, use the **-e** flag.

```
# paxctl -ze binary
```

32

Authentication, Authorization, and Auditing

About Authentication, Authorization, and Auditing Examples	193
Example: Client Application	193
Example: Authentication with Microsoft Active Directory	195
Example: Authentication with an LDAP Server	196
Example: Authorization with a TACACS+ Server	197
Example: Authentication with a RADIUS Server	198
Example: Using LDAP as a Name Service	199

About Authentication, Authorization, and Auditing Examples

IDP XT supports AAA (Authentication, Authorization, and Auditing) services, such as Microsoft Active Directory, RADIUS, LDAP and TACACS+.

The examples show a sample user application running on the target communicating with different server types on the host computer or another server in the network to validate users.

Example: Client Application

The client application **check_user** is an example program to demonstrate the use of AAA (Authentication, Authorization, and Auditing) functions.

This sample program uses PAM (Pluggable Authentication Module) APIs. The application authenticates and authorizes the username and password provided. If the username and password successfully authenticates, the application returns **Authenticated**. Otherwise, it returns **Not Authenticated**.

Step 1 On the host computer, write the application.

```
#include <security/pam_appl.h>
#include <security/pam_misc.h>
#include <stdio.h>
static struct pam_conv conv = {
    misc_conv,
    NULL
};
int main(int argc, char *argv[])
{
    pam_handle_t *pamh = NULL;
    int retval;
    const char *user = "nobody";
    if (argc == 2) {
        user = argv[1];
    }
    if (argc > 2) {
        fprintf(stderr, "Usage: check_user [username]\n");
        exit(1);
    }
    retval = pam_start("check_user", user, &conv, &pamh);
    if (retval == PAM_SUCCESS)
        retval = pam_authenticate(pamh, 0); /* is user really user? */
    if (retval == PAM_SUCCESS)
        retval = pam_acct_mgmt(pamh, 0); /* permitted access? */
    /* This is where we have been authorized or not. */
    if (retval == PAM_SUCCESS) {
        fprintf(stdout, "Authenticated\n");
    } else {
        fprintf(stdout, "Not Authenticated\n");
    }
    if (pam_end(pamh, retval) != PAM_SUCCESS) { /* close Linux-PAM */
        pamh = NULL;
        fprintf(stderr, "check_user: failed to release authenticator\n");
        exit(1);
    }
    return ( retval == PAM_SUCCESS ? 0 : 1 ); /* indicate success */
}
```

Step 2 Compile the program. `${PRJ_DIR}/host-cross/usr/bin/x86_64-wrs-linux/x86_64-wrs-linux-gcc \`
`--sysroot=${PRJ_DIR}/bitbake_build/tmp/sysroots/intel-${board}-64 \ check_user.c \ -o`
`check_user \ -lpam -lpam_misc`

```
$ ${PRJ_DIR}/host-cross/usr/bin/x86_64-wrs-linux/x86_64-wrs-linux-gcc \
--sysroot=${PRJ_DIR}/bitbake_build/tmp/sysroots/intel-${board}-64 \
check_user.c \
-o check_user \
-lpam -lpam_misc
```

Step 3 Build the `check_user`, add it to rootfs image and deploy the image to the target.

For more information, see [Building Platform Projects for Intel Bay Trail and Intel Haswell Boards](#) on page 95.

Postrequisites

Follow the steps in the following sections to configure different server types and modify the `/etc/pam.d/check_user` to specify the configuration for each type of server.

- [Example: Authentication with Microsoft Active Directory](#) on page 195
- [Example: Authentication with an LDAP Server](#) on page 196
- [Example: Authentication with a RADIUS Server](#) on page 198

- [Example: Authorization with a TACACS+ Server](#) on page 197
- [Example: Using LDAP as a Name Service](#) on page 199

Example: Authentication with Microsoft Active Directory

This example shows how to configure the application on the target to authenticate users with a Microsoft Active Directory server.

Prerequisites

You need a Microsoft Active Directory server to authenticate against.

Step 1 On your host computer, add the **libpam-krb5** package to the rootfs and build the image.

```
$ make libpam-krb5.addpkg
$ make fs
```

Step 2 Deploy the image and boot the target.

For more information, see [Building Platform Projects for Intel Bay Trail and Intel Haswell Boards](#) on page 95.

Step 3 On the target, add the following to the **/etc/krb5.conf** configuration file to refer to the domain controller as the Kerberos KDC.

Add the following:

```
[libdefaults]
    default_realm = EXAMPLE.DOMAIN.COM

[realms]
    EXAMPLE.DOMAIN.COM = {
        kdc = 128.224.192.11:88
        admin_server = 128.224.192.11
    }

[domain_realm]
    .example.domain.com = EXAMPLE.DOMAIN.COM
    example.domain.com = EXAMPLE.DOMAIN.COM
```

Step 4 On the target, modify the **/etc/pam.d/check_user** file to configure the PAM policy.

Add the following:

```
auth    required pam_krb5.so
account required pam_krb5.so
```

Step 5 Run **check_user** on the target.

```
# check_user john
Password: johnldap
Authenticated
```

Example: Authentication with an LDAP Server

This example shows how to authenticate a user using an LDAP (Lightweight Directory Access Protocol) server.

This example runs an `openldap` server on Ubuntu host. For more information, see <https://help.ubuntu.com/lts/serverguide/openldap-server.html>.

Step 1 On your Ubuntu host, edit the `/etc/hosts` file to configure the DNS to resolve the LDAP server.
Add the following:

```
127.0.1.1      hostname.example.com  hostname
```

Step 2 On your host computer, install the packages required to run the LDAP server.

```
$ sudo apt-get install ldap-utils slapd
```

Step 3 On your host computer, create the `add_content.ldif` file to configure the LDAP directory structure.

Add the following:

```
dn: ou=People,dc=example,dc=com
objectClass: organizationalUnit
ou: People

dn: ou=Groups,dc=example,dc=com
objectClass: organizationalUnit
ou: Groups

dn: cn=miners,ou=Groups,dc=example,dc=com
objectClass: posixGroup
cn: miners
gidNumber: 5000

dn: uid=john,ou=People,dc=example,dc=com
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: john
sn: Doe
givenName: John
cn: John Doe
displayName: John Doe
uidNumber: 10000
gidNumber: 5000
userPassword: johnldap
gecos: John Doe
loginShell: /bin/bash
homeDirectory: /home/john
```

Step 4 Add the directory structure to the LDAP server.

```
$ sudo ldapadd -x -D cn=admin,dc=example,dc=com -W -f add_content.ldif
Enter LDAP Password: *****
adding new entry "ou=People,dc=example,dc=com"
adding new entry "ou=Groups,dc=example,dc=com"
adding new entry "cn=miners,ou=Groups,dc=example,dc=com"
adding new entry "uid=john,ou=People,dc=example,dc=com"
```

Step 5 Add the **libpam-ldap** package to the rootfs and build the image.

```
$ make libpam-ldap.addpkg
$ make fs
```

Step 6 Deploy the image and boot the target.

For more information, see [Building Platform Projects for Intel Bay Trail and Intel Haswell Boards](#) on page 95.

Step 7 On the target, modify the **/etc/ldap.conf** openldap configuration file to use the LDAP server on your host computer.

Add the following:

```
host 192.52.160.222
base dc=example,dc=com
ldap_version 3
pam_password md5
```

Step 8 On the target, modify the **/etc/pam.d/check_user** file to configure the PAM policy.

Add the following:

```
auth    required pam_ldap.so
account required pam_ldap.so
```

Step 9 Run **check_user** on the target.

```
# check_user john
Password:johnldap
Authenticated
```

Example: Authorization with a TACACS+ Server

This example shows how to authenticate using a TACACS+ (Terminal Access Controller Access-Control System Plus) server.

This example runs a TACACS+ server on your Ubuntu host.

Step 1 On your host computer, install the required packages to run TACACS+.

```
$ sudo apt-get install tacacs+
```

Step 2 On your host computer, edit the **/etc/tacacs+/tac_plus.conf** file to configure TACACS+.

Add the following:

```
key = testing123

user = john {
    global = cleartext john
    service = raccess {}
}
```

Step 3 Restart the **tacacs_plus** service.

```
$ sudo /etc/init.d/tacacs_plus restart
```

Step 4 Add the **libpam-tacplus** package to the rootfs and build the image.

```
$ make libpam-tacplus.addpkg
$ make fs
```

Step 5 Deploy the image and boot the target.

For more information, see [Building Platform Projects for Intel Bay Trail and Intel Haswell Boards](#) on page 95.

Step 6 On the target, modify the **/etc/pam.d/check_user** file to configure the PAM policy.

Add the following:

```
auth        required pam_tacplus.so server=128.224.158.123 secret=testing123
prompt=password:
account     required pam_tacplus.so service=raccess protocol=ip
```

Step 7 Run **check_user** on the target.

```
# check_user john
Password:john
Authenticated
```

Example: Authentication with a RADIUS Server

This example shows how to authenticate using a RADIUS (Remote Authentication Dial-In User Service) server.

This example runs a freeradius server on your Ubuntu host.

Step 1 On your host computer, install the required packages to run RADIUS.

```
$ sudo apt-get install freeradius freeradius-utils
```

Step 2 On your host computer, edit the **/etc/freeradius/users** file to configure RADIUS.

Add the following:

```
john        Cleartext-Password := "john"
            Reply-Message = "Hello, %{User-Name}"
```

Step 3 Restart the **freeradius** service.

```
$ sudo service freeradius restart
```

Step 4 Add the **pam-radius-auth** package to the rootfs and build the image.

```
$ make pam-radius-auth.addpkg
$ make fs
```

Step 5 Deploy the image and boot the target.

For more information, see [Building Platform Projects for Intel Bay Trail and Intel Haswell Boards](#) on page 95.

Step 6 On the target, modify the **/etc/pam_radius_auth.conf** configuration file to point to your RADIUS server.

Add the following:

```
# server[:port]  shared_secret  timeout (s)
freeradius      testing123      3
```

Step 7 Copy the `/etc/pam_radius_auth.conf` file to `/etc/raddb/server`.

Step 8 Modify the `/etc/pam.d/check_user` file to configure the PAM policy.

Add the following:

```
auth      sufficient  pam_radius_auth.so
account   sufficient  pam_radius_auth.so
```

Step 9 Run `check_user` on the target.

```
# check_user john
Password:john
Authenticated
```

Example: Using LDAP as a Name Service

The example shows how to use LDAP (Lightweight Directory Access Protocol) for identity and authentication.

This example uses the **nss-pam-ldapd** package to provide a Name Server Switch (NSS) to enable your LDAP server to retrieve identity information and do authentication.

Prerequisites

You need an LDAP server configured and running. For an example of how to set up a server on your Ubuntu host, see [Example: Authentication with an LDAP Server](#) on page 196.

Step 1 Add the **nss-pam-ldapd** package to the rootfs and build the image.

```
$ make nss-pam-ldapd.addpkg
$ make fs
```

Step 2 Deploy the image and boot the target.

For more information, see [Building Platform Projects for Intel Bay Trail and Intel Haswell Boards](#) on page 95.

Step 3 On the target, modify the `/etc/nslcd.conf` file to configure LDAP and Active Directory (AD) authentication.

Add the following:

```
....
ldap_version 3

uri ldap://192.52.160.222
base dc=corp,dc=example,dc=domain,dc=com
binddn xouyang@example.domain.com
bindpw mypassword
#filter passwd (&(objectClass=*) (! (objectClass=computer)) (uidNumber=*)
(unixHomeDirectory=*))
#map      passwd homeDirectory  "/home/$uid"
#map      passwd homeDirectory  unixHomeDirectory
```



NOTE:

- You need the **binddn** and **bindpw** lines if the Active Directory settings do not allow anonymous LDAP binding.
 - The **filter** and **map** lines are optional. The lines could be based on the result of running **ldapsearch**.
 - See the comments in the **/etc/nslcd.conf** file for more options.
-

Step 4 On the target, restart the **nslcd** service.

```
# systemctl restart nslcd
# systemctl status -l nslcd
```

Step 5 Edit the **/etc/nsswitch.conf** file and add LDAP.

Add the following:

```
passwd:      compat ldap
group:       compat ldap
shadow:      compat ldap
```

Step 6 Test user names with LDAP and AD.

```
# id yourusername
uid=18087(yourusername) gid=100(users) groups=100(users)
```

References

IDP Services Reference..... 203

IDP Packages Not Included in Any Feature..... 205

Packages Required for SST..... 207

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
arptables	Yes	Starts arptables , which applies ARP firewall rules from a configuration file.
dcom	Yes	Provides DCOM service used by OPC servers.
dnsmasq	Yes	Starts dnsmasq , which implements DNS and DHCP functions for the gateway device.
encrypt-storage	Yes	Starts encrypt-storage , which sets up encrypted storage automatically at boot time.
fcgiwrap	No	Starts FastCGI, which supports CGI scripts (CGI support for nginx)
firewall	Yes	Starts firewall for the system with pre-configured rules.
firstboot	No	Runs first boot tasks for the system.
gradm	No	Enables the GRSecurity RBAC system.
ima-digsig	No	Starts the IMA signature flush service daemon.
krb5-admin-server	No	Starts the MIT Kerberos KDC administrative daemon.
krb5-kdc	No	Starts MIT Kerberos KDC.

Service Name	Started By Default	Description
mosquitto	No	Starts an mqtt broker. 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 daemon that manage the priority of upstream interfaces according to a user configuration and determines the active upstream interface in the system according to priority and connection status.
netifd	Yes	Starts the Network Interface Daemon (netif) in the system that deals with various network interface configurations.
nginx	Yes	Starts the nginx web server.
oma_dmc	No	Starts OMA-DM agent if the OMA-DM agent is included in image.
openl2tp	Yes	Starts the openl2tp daemon which acts as an L2TP server.
runonce	Yes	Starts the runonce , service, which enables you to perform initial device configuration once.
scsrvc	Yes	Starts the McAfee Solidifier service.
tcsd	No	Starts the tcsd daemon, which enables applications to use the Trousers APIs to communicate with a TPM chip.
tr-agent	No	Starts the CWMP(TR-069) client.
ubus	Yes	Starts ubusd , which is an RPC DAEMON server that enable interprocess communication.

34

IDP Packages Not Included in Any Feature

Most IDP XT packages are included in a layer. However, some packages are not included with a layer and are not part of any feature template; they must be included by building the package.

To include a package into your rootfs explicitly, add **--with-package=pkgName** to your **configure** command when you build your platform project.

The following is a list of additional Wind River Linux packages provided by IDP XT but not part of any IDP XT layer or feature.

Package	Description
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.
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.

Package	Description
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.
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.
schroedinger	schroedinger 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.

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.
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.
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.
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.

Package	Description
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.

