Reader and Gateway Embedded Developers Guide

Version 5.12.3

http://www.impinj.com

Contents

1 Introduction **5**

1.1 Purpose . . . 5
1.2 Scope . . . 5
1.3 References . . . 5
1.4 Acronyms . . . 5

2 Platform Architecture **8**

2.1 Speedway Reader Enclosure . . . 8
2.2 xPortal Gateway Enclosure . . . 9
2.3 xArray/xSpan Gateway Enclosure . . . 9
2.4 Speedway Power Supply . . . 10
2.5 Speedway GPIO . . . 10
2.5.1 Using the GPIO Serial Port . . . 12
2.6 Speedway Reader LEDs . . . 12
2.7 xArray Gateway LEDs . . . 14
2.8 xSpan Gateway LED . . . 15
2.9 Speedway Console Port . . . 16
2.10 Platform System Overview . . . 19
2.11 Flash Organization . . . 21
2.11.1 System Operating Partition (SOP) . . . 22
2.11.2 System Persistent Partition (SPP) . . . 22
2.11.3 Custom Application Partition (CAP) . . . 23
2.11.4 Upgrade File System . . . 23
2.12 RAM Allocation . . . 24
2.12.1 Linux OS . . . 24
2.12.2 Reader Applications . . . 24
2.12.3 RAM File System . . . 24

2.12.4 Custom Applications . . . 25
2.12.5 RAM Allocation by Product . . . 25
2.13 Linux File System . . . 25
2.14 USB HID Keyboard Emulation . . . 26
2.14.1 Requires On-Reader Application . . . 27
2.14.2 USB Cabling . . . 27
2.14.3 USB Presence . . . 28
2.14.4 Keyboard Emulation Program (hidkey) . . . 28
2.14.5 Disconnected Operation . . . 30
2.14.6 Example USB HID Application . . . 30
2.14.7 Supported Characters . . . 32
2.14.8 Performance . . . 32
2.15 USB Flash Drive AutoRun . . . 32
2.15.1 USB Flash Drive Preparation . . . 33
2.15.2 HardwareEvents Stanza in Reader.conf . . . 33
2.15.3 Example Storage Event Handler . . . 33

3 Platform Boot Process 35
3.1 Bootloader . . . 36
3.1.1 Default Restore . . . 36
3.1.2 Image Selection . . . 37
3.2 Operating System . . . 37
3.2.1 Mounter . . . 38
3.2.2 Run-Time Applications . . . 41

4 Software Development Process 43
4.1 Tools . . . 43
4.2 Linux syslog . . . 43
4.3 CAP Upgrade Creation . . . 44
4.3.1 Using cap_gen . . . 46

4.4 Accessing the Linux Shell 47
4.4.1 Executing the Custom Application 48
4.4.2 Automatically Starting a Custom Application 49
4.4.3 Custom Application Library Dependencies 50
4.5 Firmware Upgrade Procedure 50
4.5.1 Image Versioning Scheme 50
4.5.2 RShell Upgrade Methods 51
4.5.3 OSShell Upgrade Methods 52
4.6 Reader Configuration 53
4.6.1 Configuration File Format 53
4.6.2 Hardware Events 54
4.6.3 Partner Features 55
4.6.4 Software Features 55
4.6.5 Configuration File Example 55

5 Document Revision History **57**

6 Notices **58**

1 Introduction

1.1 Purpose

This document describes the Impinj Speedway platform and high-level architecture. It is intended for embedded software developers designing custom application software to run on the Impinj Speedway Reader, xPortal, xArray, and xSpan Gateway products.

1.2 Scope

This document describes at a high level how to use the basic functionality of the Impinj Speedway Reader. The Impinj xPortal, xArray and xSpan Gateway products embed Impinj Speedway Readers, and hence inherit all of the Reader's capabilities with regard to on-reader application development. This document also provides detailed information about the platform architecture provided by Impinj to custom application embedded software developers.

Information in this document pertains to the Octane 5.12.3 firmware.

1.3 References

Table 1.1 References

Document	**Version**
EPCglobal: Low Level Reader Protocol (LLRP)	1.0.1
Speedway Installation and Operations Guide	5.12.3
RShell Reference Manual	5.12.3
Firmware Upgrade Reference Manual API	5.12.3
Octane LLRP	5.12.3

1.4 Acronyms

API: Application Programming Interface

CAP: Custom Application Partition

CDR: Configuration Default Restore

CLI: Command Line Interface

CPU: Central Processing Unit

DC: Direct Current

DE9: 9-pin D-sub E-shell connector

DE15: 15-pin D-sub E-shell connector

DHCP: Dynamic Host Configuration Protocol

FCR: Factory Configuration Restore

FDR: Factory Default Restore

FPGA: Field Programmable Gate Array

FS: File System

FTP: File Transfer Protocol

GPIO: General Purpose Input/Output

IP: Internet Protocol

LED: Light Emitting Diode

LLRP: Low Level Reader Protocol

MAC: Media Access Control

MIPS: Million Instructions per Second

MTD: Memory Technology Device

NC: Not Connected

NFS: Network File System

NTP: Network Time Protocol

NVP: Name/Value Pair

PoE: Power over Ethernet

RAM: Random Access Memory

RF: Radio Frequency

RFID: Radio Frequency Identification

RP-TNC: Reverse Polarity-Threaded Neill-Concelman connector

SDK: Software Development Kit

SDRAM: Synchronous Dynamic Random Access Memory

SOP: System Operating Partition

SPP: System Persistent Partition

SSH: Secure Shell

UHF: Ultra-High Frequency

URI: Universal Resource Identifier (RFC 3986)

USB: Universal Serial Bus

LLA: Link-Local Addressing

mDNS: Multicast Domain Name System

Note DE9 and DE15 connectors are often incorrectly referenced as DB9 and DB15.

2 Platform Architecture

The Speedway R120, R220 and R420 Readers are fixed UHF Gen2 RFID tag Readers that provide network connectivity between tag data and enterprise system software. The Impinj xPortal, xArray and xSpan Gateways integrate a reader with antenna arrays for portal and wide-area monitoring applications respectively.

This Embedded Developer's Guide provides instructions on how to configure the Speedway platform application program interface, called the Custom Application Partition (CAP). It assumes the embedded software developer is familiar with appropriate networking facilities, the EPCglobal Gen2 specification, and general principles of RFID system management.

2.1 Speedway Reader Enclosure

Refer to the figures below for pictures of the Speedway Reader enclosure with major ports, connectors, and status indicators clearly labeled. Figure 2.1 depicts the hardware (network, power, Ethernet, etc.) ports, and Figure 2.2 depicts the antenna ports and status LEDs.

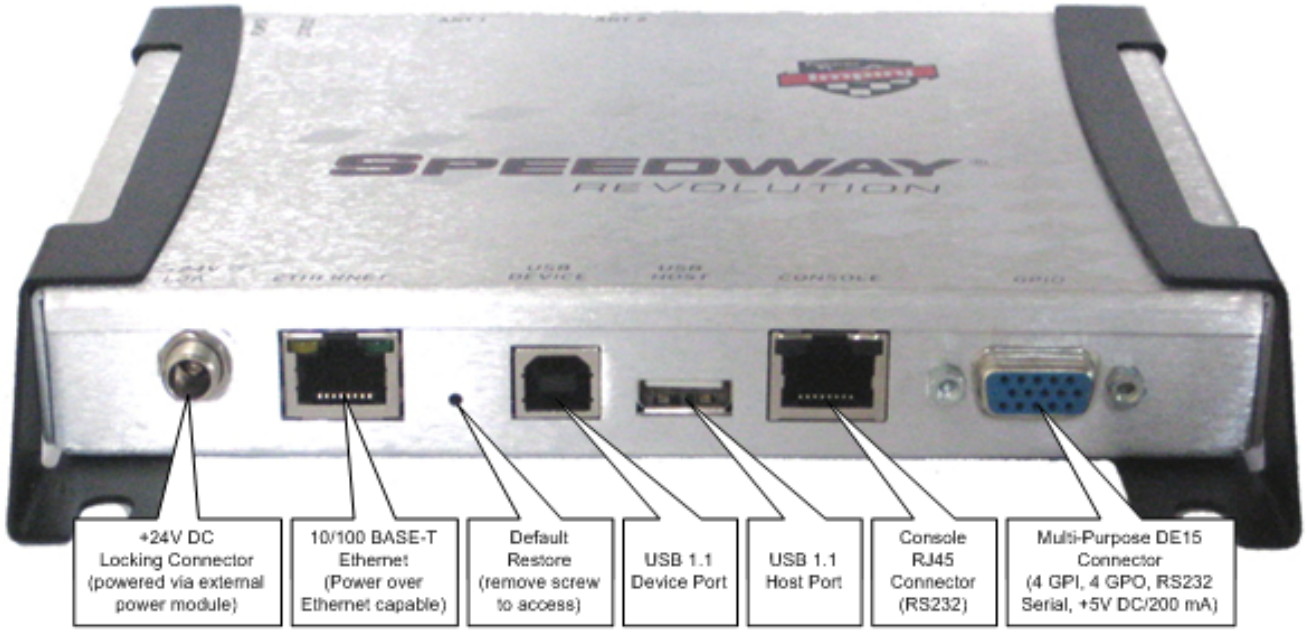

Figure 2.1 Speedway Hardware Connections

The Speedway Reader is equipped with the following hardware connectors:

- Locking 24V DC power supply connector (labeled +24V DC 1.0A)

- RJ45 Power-over-Ethernet capable Ethernet jack (labeled ETHERNET)
- USB 1.1 Type B port (labeled USB DEVICE)
- USB 1.1 Type A port (labeled USB HOST)
- RJ45 RS232 Serial Console connector (labeled CONSOLE)
- Female DE15 connector with user I/O capability. Secondary RS232 serial, four optoisolated inputs, four optoisolated outputs. See the Speedway GPIO section for pinouts and I/O details.

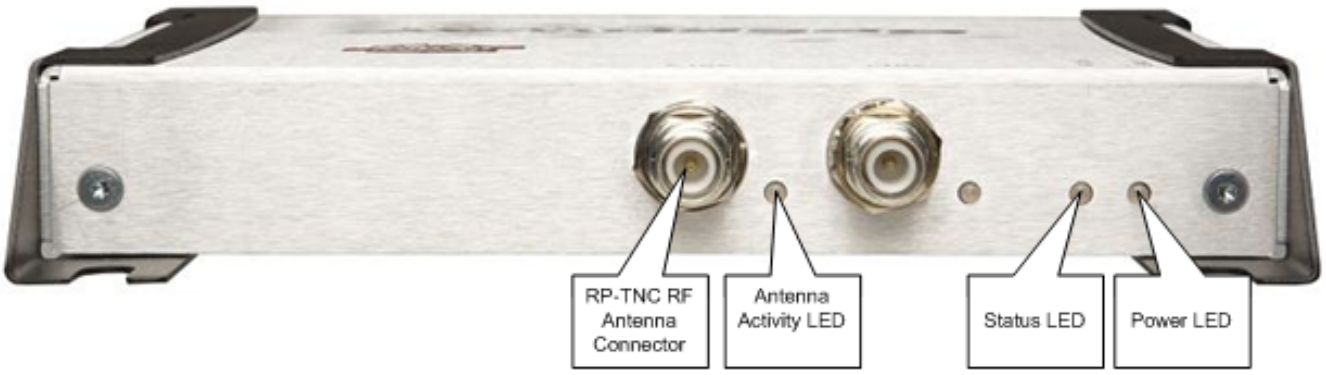

Figure 2.2 Speedway Antenna Connections

The Speedway Reader is equipped with the following antenna connectors and LEDs:

- Two (R120 and R220, pictured) or four (R420) female RP-TNC antenna connectors (labeled ANT1-ANTn).
- Two (R120 and R220, pictured) or four (R420) antenna activity LEDs. For more information, see the Speedway Reader LEDs section.
- One status and one power LED. For more information, see Speedway Reader LEDs.

2.2 xPortal Gateway Enclosure

The antennas are connected to the internal antennas and therefore aren't accessible.

2.3 xArray/xSpan Gateway Enclosure

Figure 2.3 shows the xArray/xSpan Gateway enclosures connections.

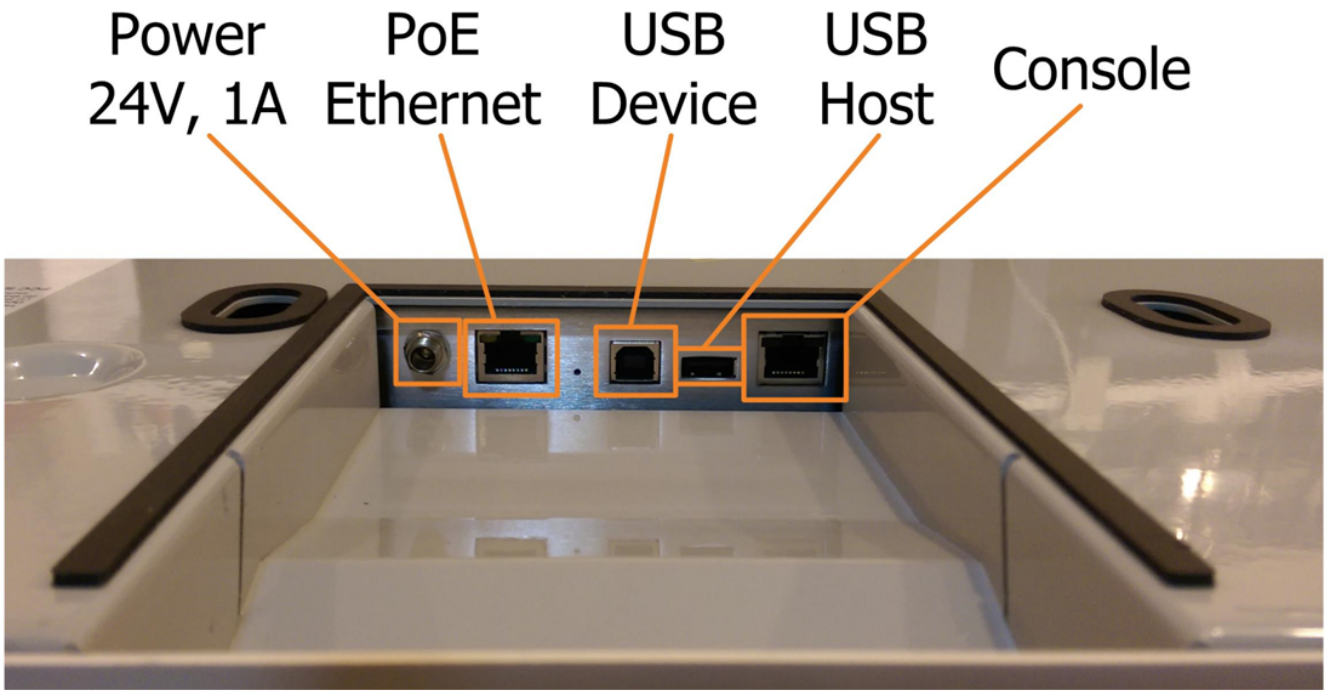

Figure 2.3 xArray/xSpan Hardware Connections

The antennas are connected to the internal antennas and therefore aren't accessible. In addition, the embedded reader's female DE15 I/O connector is used internally and therefore isn't available for on-reader applications.

2.4 Speedway Power Supply

Speedway Readers can be powered by either an external 24V power supply, or via Power over Ethernet (PoE). Only one power supply is active at a time, and the default power supply is via the 24V external supply when connected. To use PoE, the 24V external supply must be disconnected, and the Ethernet port connected to a cable that is attached to either a PoE injector or a PoE-enabled network switch. A change in active power supply, such as pulling the 24V external supply while a PoE-enabled Ethernet cable is attached, will result in a Reader reset.

2.5 Speedway GPIO

Speedway Readers provide the user with a multipurpose I/O port that contains a RS232 serial port, four optoisolated inputs, four optoisolated outputs, and a +5V supply. These features are accessed through a DE15 connector mounted on the back of the Reader.

The four optoisolated inputs have a range of 0-30V. The Reader will treat an input of 0-0.8V as logic 0, and an input of 3-30V as logic 1. The Reader has a per-input de-bounce interval that is configurable via LLRP. For more information, see the *Octane LLRP* document. This value dictates the minimum pulse width of an input. Impinj recommends that external devices guarantee a minimum pulse width of at least 100 milliseconds.

Four optoisolated outputs are also provided. An external power supply must be connected between V+ and V- for the GPIO outputs to function. The maximum voltage for this supply is 30V. When the user configures a selected GPIO output via LLRP to output logic 0, an isolated FET switch in the Reader effectively shorts that output to V-, with a current sink capability of up to 200mA. When the user configures a selected GPIO output to logic 1, the selected output is pulled to V+ through a 10K resistor. If GPIO isolation is not required, the Reader provides a +5V supply and a ground pin on the DE15 that can be connected to V+ and V-.

See Figure 2.4 and Table 2.1 for details regarding the pin-out of the DE15.

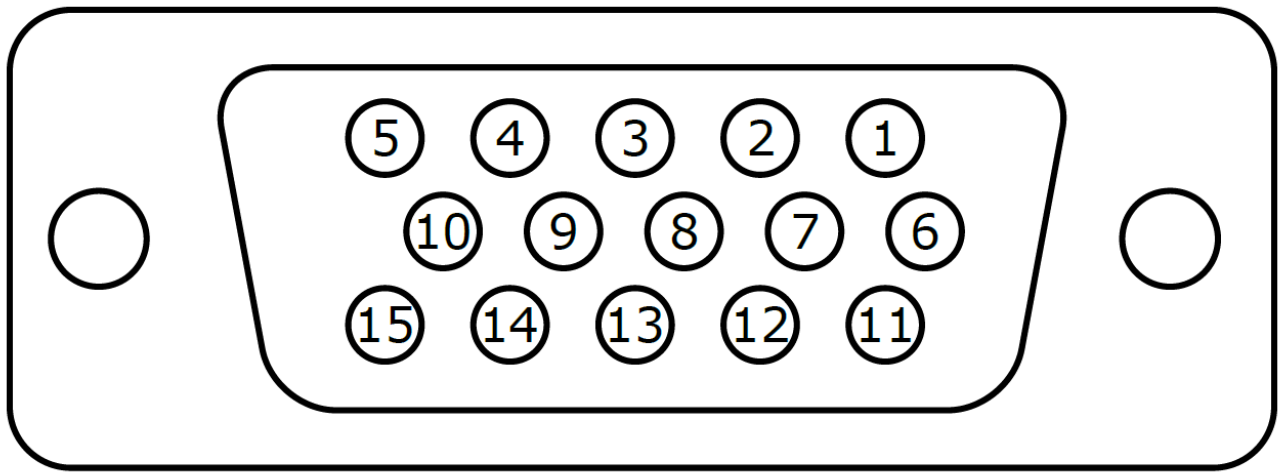

Figure 2.4 DE15 Female Connector

Table 2.1 provides the signal name and a description for each DE15 Connector pin.

Table 2.1 DE15 Connector Pin-Out

Pin	Signal Name	Description
1	USER_5V	+5V supply
2	RS232_RXD	RS232 serial receive
3	RS232_TXD	RS232 serial transmit
4	DB_DEFAULT_RST	Factory default restore (currently unused)
5	VPLUS	Positive supply for optoisolated GPO

Pin	Signal Name	Description
6	VMINUS	Negative supply for optoisolated GPO
7	GND	Ground
8	USEROUT_0	General purpose output 0 (LLRP 1)
9	USEROUT_1	General purpose output 1 (LLRP 2)
10	USEROUT_2	General purpose output 2 (LLRP 3)
11	USEROUT_3	General purpose output 3 (LLRP 4)
12	USERIN_0	General purpose input 0 (LLRP 1)
13	USERIN_1	General purpose input 1 (LLRP 2)
14	USERIN_2	General purpose input 2 (LLRP 3)
15	USERIN_3	General purpose input 3 (LLRP 4)

When the Speedway Reader is used with Speedway Reader Antenna Hub accessories, the GPIO port is used to interface to the GPIO Adaptor Kit. In this configuration, the GPIOs and the serial port are no longer available for embedded applications to use.

2.5.1 Using the GPIO Serial Port

The Speedway Revolution reader has two serial ports: The first is labeled 'Console' and is accessible through an RJ-45 jack on the rear panel. This serial port is primarily used for the RShell interface. It's referred to as /dev/ttyS0. The second serial port (/dev/ttyS1) is unused and can be accessed through the DB-15 GPIO port. Your cable should be wired in a null modem configuration, with the RX and TX wires swapped.

For in-depth examples, see this resource for Linux serial programming: http://tldp.org/HOWTO/Serial-Programming-HOWTO/

2.6 Speedway Reader LEDs

The Speedway Reader has several LEDs to indicate Reader operational status. The three primary LED categories are power, Reader status, and antenna status. Each LED has its own blink patterns to convey status to the user. Table 2.2 documents the defined patterns for the Power LED. Table 2.3 documents the defined patterns for the Reader Status LED. Table 2.4 documents the defined patters for the Antenna Status LEDs.

Table 2.2 Power LED Patterns

LED State	Reader State
Solid RED (after power-on or reset)	Power applied, attempting to start boot code
OFF	Default Restore button pressed
One short RED blink	Configuration Default Restore detected See Default Restore.
Two short RED blinks	Factory Default Restore detected See Default Restore.
Blinking RED (4 Hz)	Unable to boot (see console for details)
Solid GREEN	Done booting, starting application image
Blinking ORANGE (1Hz)	USB flash drive upgrade in progress
Blinking RED (2 Hz)	USB flash drive upgrade failure

Table 2.3 Reader Status LED Patterns

LED State	Reader State
OFF	Application image booting, RFID not available
Alternating RED and GREEN	Application image booting, RFID not available, File system operation in progress (after upgrade)
Solid GREEN	Application image booted, RFID available, No LLRP connection
Two short GREEN blinks	Active LLRP connection
One short GREEN blink	No LLRP connection, LLRP ROSpec configured and enabled
Blinking ORANGE	Inventory active, blinking rate increases with an increased number of tags in the Reader FOV

Table 2.4 Antenna Status LED Patterns

LED State	Reader State
OFF	Antenna inactive
Solid GREEN	Antenna actively transmitting

The Speedway Reader's antenna, power and status LEDs are not visible in the xPortal, xArray

and xSpan Gateway products.

2.7 xArray Gateway LEDs

The xArray Gateway has four LEDs to indicate operational status, their position is shown in Figure 2.5. The Power LED is ON (GREEN) when power is applied.

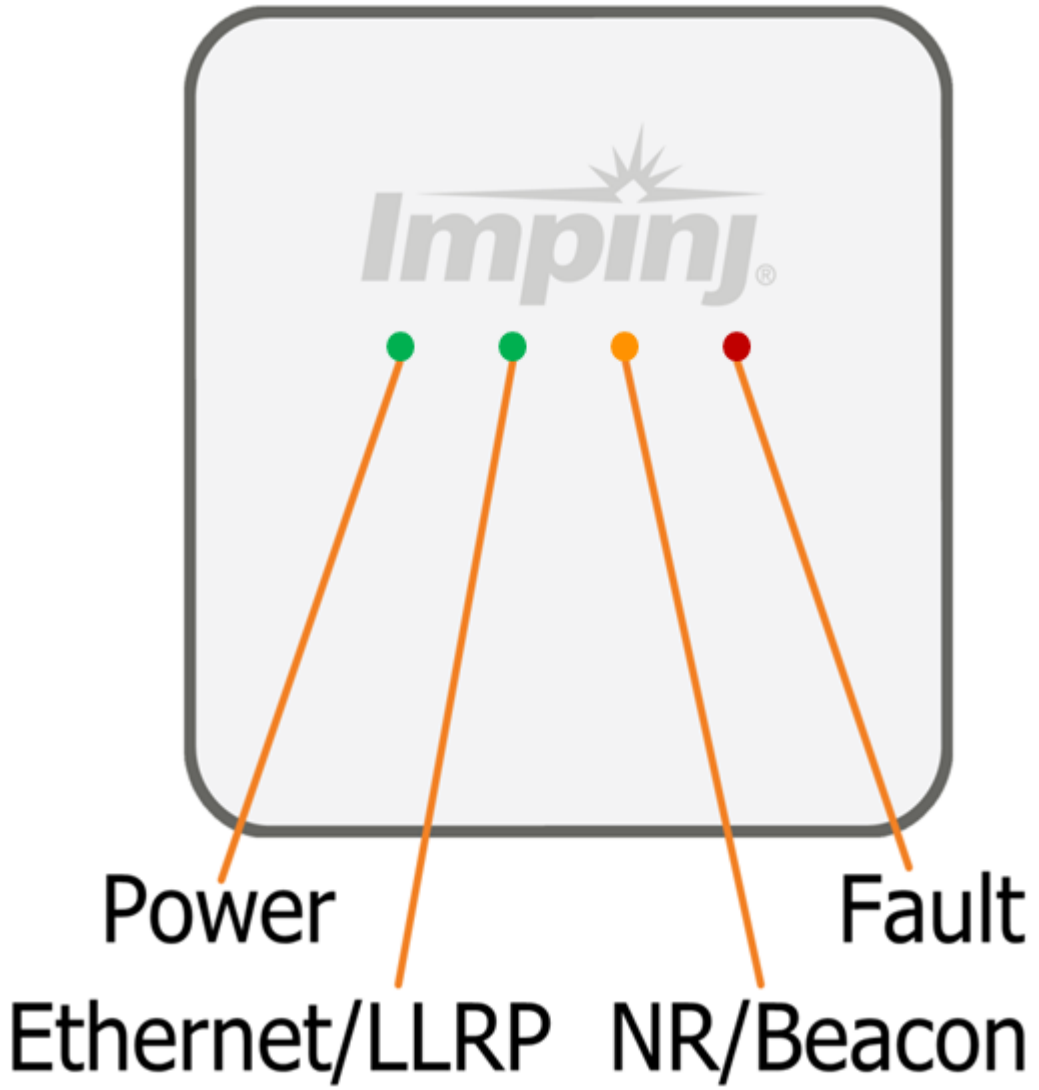

Figure 2.5 xArray Status LEDs

Each LED has its own blink patterns to convey status to the user. Table 2.5 documents the defined patterns for the xArray Ethernet/LLRP LED. Table 2.6 documents the defined patterns for the xArray NR/Beacon LED. Table 2.7 documents the defined patters for the xArray Fault LED.

Table 2.5 xArray Ethernet/LLRP LED

LED State	xArray State
OFF	During startup
Blinking (GREEN)	Ready for an LLRP host connection
Solid (GREEN)	LLRP host connected

Table 2.6 xArray NR/Beacon LED

LED State	xArray State
Blink (ORANGE)	Briefly during startup
OFF	(Default)
ON (ORANGE)	When turned ON by LLRP command

Table 2.7 xArray Fault LED

LED State	xArray State
Blink (RED)	Briefly during startup
OFF	Normal (no faults)
ON (RED)	Fault condition occurred

2.8 xSpan Gateway LED

The xSpan Gateway has one tri-color LED to indicate operational status. It's position is shown in Figure 2.6. The LED is Solid RED when power is applied.

Figure 2.6 xSpan Status LED

Table 2.8 documents the defined patterns for the xSpan LED.

Table 2.8 xSpan LED

LED State	xSpan State
OFF	No Power
Solid RED	Power
Solid YELLOW	Ready for an LLRP host connection
Solid BLUE	LLRP host connected
Blinking RED	Fault condition occurred
Blinking BLUE	Beacon turned on by LLRP command

2.9 Speedway Console Port

The Speedway console port uses RS232 serial over a RJ45 cable. Because most computers do not have a RJ45 COM port (DE9 COM ports are typical), a RJ45-to-DE9 converter cable will likely be required to connect to the Reader's console port. The Speedway RJ45 port uses a Cisco-compatible pin out and these cables are readily available from many third-party vendors. However, in the event that such a cable is unavailable and a cable must be built by hand, Table 2.9 documents the pin mapping from a female DE9 connector to a male RJ45 connector. Note that the table below is complete in terms of the pin-to-pin mapping, but only RJ45 pins 3 (RD), 4 (GND), and 6 (TD) are required for proper operation. The remaining pins are not utilized by the Reader and are left unconnected.

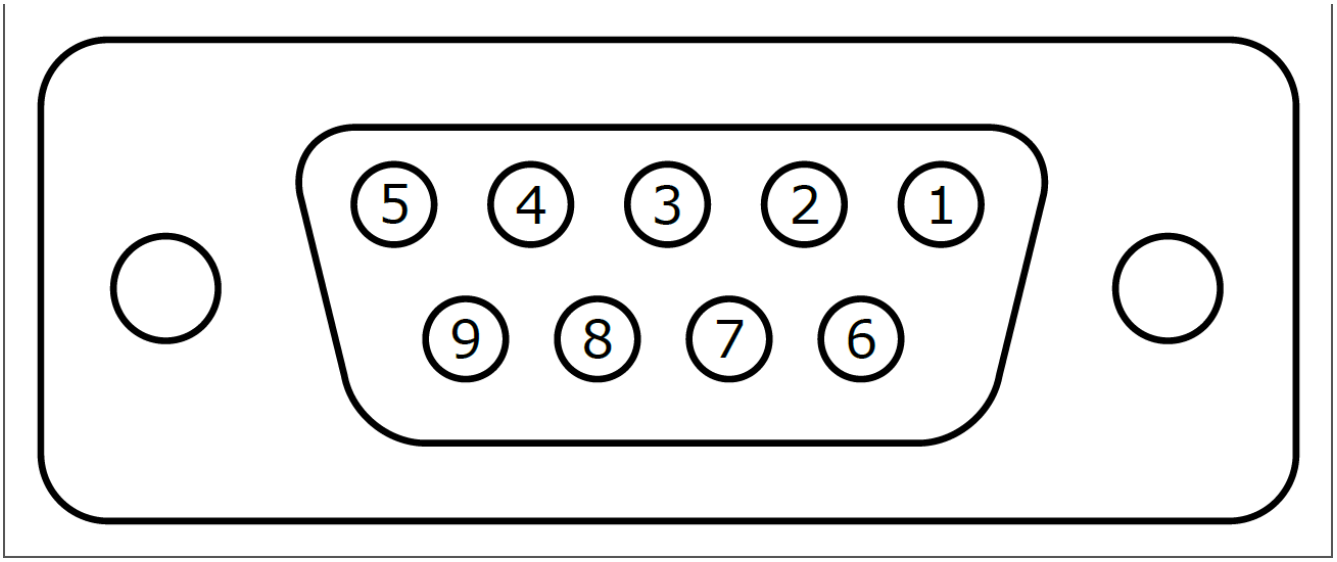

Figure 2.7 DE9 Female Connector

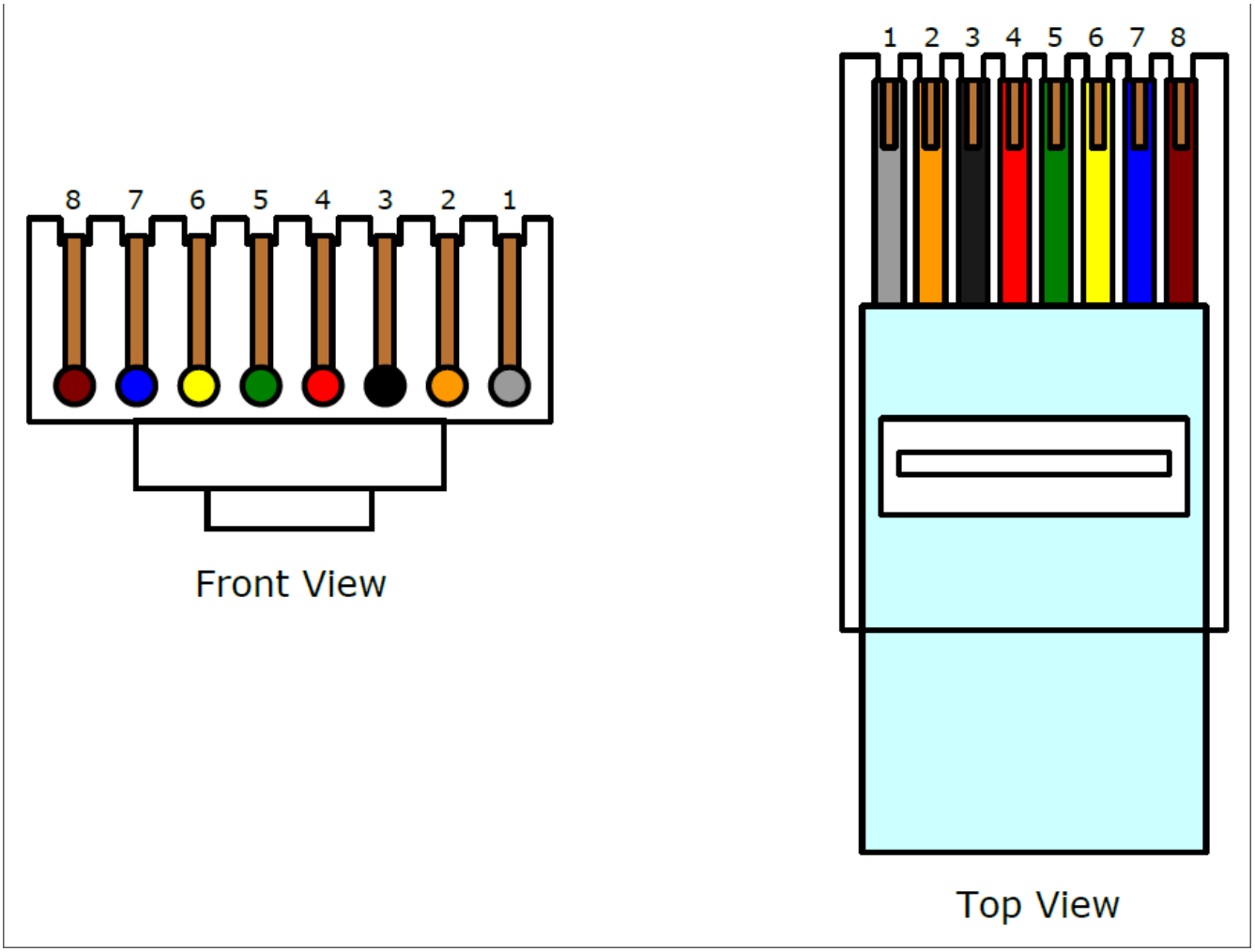

Figure 2.8 RJ45 Male Connector

Table 2.9 documents the pin mapping from a female DE9 connector to a male RJ45 connector.

Table 2.9 RS232 DE9-to-RJ45 Pin Mapping

RS232 Signal	DE9 Pin	RJ45 Pin	Color (typ.)
Data Carrier Detect	1	NC	N/A
Receive Data	2	3	Black
Transmit Data	3	6	Yellow
Data Terminal Ready	4	7	Blue
Signal Ground	5	4	Red
Data Set Ready	6	2	Orange

RS232 Signal	DE9 Pin	RJ45 Pin	Color (typ.)
Request to Send	7	8	Brown
Clear to Send	8	1	Gray
Ring Indicator	9	NC	N/A

2.10 Platform System Overview

The Speedway Reader is a single processor system, with the control platform based on an Atmel AT91SAM series controller. For a hardware block diagram, see Figure 2.9. Air-protocol and time-critical functions are implemented within an FPGA. This architecture arrangement leaves anywhere from 50% to 90% of the control CPU MIPS (depending on the inventory load) available for custom application software. The control microprocessor also has NAND flash memory and SDRAM available for firmware and data storage. See Flash Organization and RAM Allocation sections for flash and SDRAM allocation schemes, respectively, to determine how much is available for custom application development.

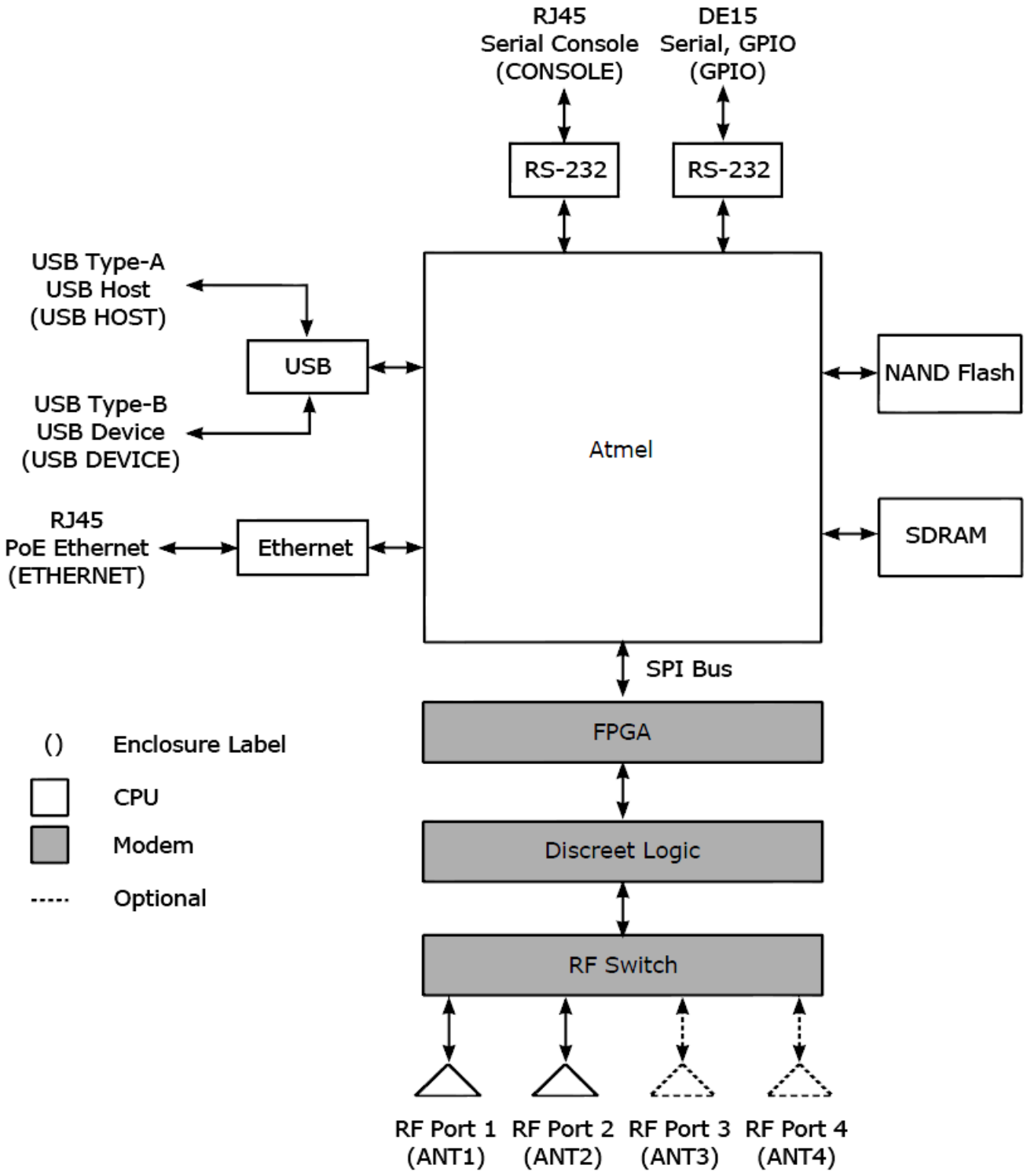

Figure 2.9 Speedway Hardware Block Design

The Speedway Reader's CPU hardware subsystem was updated in later revisions of the hardware. The PCBA version can be used to determine which components are present.

Table 2.10 CPU Hardware Update Summary

H/W Component	PCBA v4.xx (and lower)	PCBA v5.xx (and higher)
Micro-controller, clk	AT91SAM9260, 200Mhz	AT91SAM9G20, 400Mhz
NAND Flash	128 Mbytes	256 Mbytes
SDRAM	64 Mbytes	128 Mbytes (xArray/xSpan) 64 Mbytes (others)

For the purposes of this document, SDRAM and RAM are synonymous.

2.11 Flash Organization

The Speedway flash device is logically partitioned as shown in Figure 2.10. The dual image architecture supports a minimal downtime during firmware upgrades and provides for maximum robustness for upgrade failure scenarios (power outage, etc.). The firmware upgrade agent downloads the secondary image into the upgrade file system in the background while the primary image runs. The availability of the second image supports a firmware fallback if the latest upgrade does not work properly or if the primary image is corrupted. This dual image partitioning also supports a return to factory default firmware.

Figure 2.10 Speedway Flash Organization

Within each firmware image are individual JFFS2 formatted partitions that are used to logically organize the system software. Each image is partitioned as shown in Figure 2.11. Nominal sizes are given, but the actual sizes reported by the Reader include an allowance for bad flash blocks. The details regarding each partition are in the sections that follow. The Linux OS may report the file systems being larger than specified here, this is to account for redundancy in the NAND flash memory.

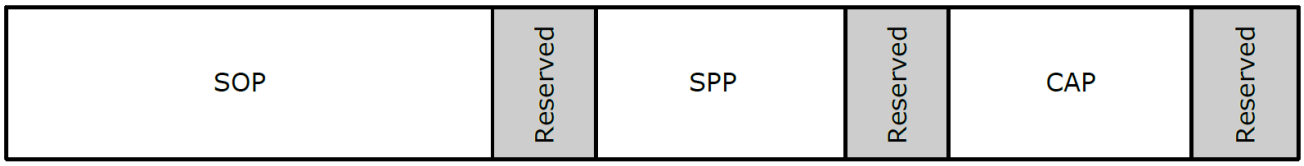

Figure 2.11 Speedway Image Organization

The Speedway Reader's NAND Flash Partition sizes were expanded in later revisions of the hardware. The PCBA version can be used to determine which components are present.

Table 2.11 Speedway Image Partition Sizes

Image Partition Name	PCBA v4.xx (and lower)	PCBA v5.xx (and higher)
Octane Firmware Partition (SOP)	16 Mbytes	28 Mbytes
System Persistent Partition (SPP)	8 Mbytes	16 Mbytes
Custom App Partition (CAP)	8 Mbytes	32 Mbytes

2.11.1 System Operating Partition (SOP)

This is the primary system partition of the Speedway Reader. Embedded developers can't modify the contents of this partition. This partition is mounted read-only at / (root). The primary contents of the SOP are:

- Linux 2.6.39.4 kernel
- FPGA firmware
- RFID management software
- Reader management (RShell - for more information, see *RShell Reference Manual*)
- Logging management software
- Firmware upgrade control
- System watchdog software
- Factory default data

2.11.2 System Persistent Partition (SPP)

This is the persistent partition of the Speedway Reader. Embedded developers can't modify the contents of this partition. Files in this partition are automatically generated and maintained by

the software running on the Reader. Manual manipulation of these files will result in undefined Reader behavior. This partition is mounted read-write at /mnt/spp. The primary contents of the SPP are:

- Reader configuration (e.g. network settings, LLRP configuration, log settings)
- Reader logs (both application logs and error logs)
- Reader crash information (used internally by Impinj for debugging)

2.11.3 Custom Application Partition (CAP)

This is the custom application partition of the Reader. Embedded developers are free to make use of this partition as required by their application (subject to the constraints within). The use of this partition is the primary focus of this document. This partition might or might not be present on a Reader, depending on whether a CAP partition was included in the upgrade file. If a CAP is present on the Reader, this partition is mounted read-write at /cust. If a CAP is not present on the Reader, the /cust directory will be mounted read-only and will be empty. The primary contents of the CAP are typically:

- Custom application software
- Extra libraries or tools required by the custom application
- Configuration files used by the custom application
- Custom application logs

Note that, while this partition is mounted read-write, custom application developers should limit the use of this file system for dynamic data storage to avoid excessive wear of the flash memory. Wherever frequent modifications are required within a file system, we recommend that custom application developers use the RAM file system. For more information see the RAM File System section.

2.11.4 Upgrade File System

The upgrade file system is not included in a firmware image and thus is not considered a partition like those discussed in previous sections. Instead, the upgrade file system is used by the active image to perform firmware upgrades, regardless of which image is active. During an upgrade, the firmware upgrade application downloads the upgrade image to this file system. In this way the image file doesn't have to be stored in RAM, which increases the RAM available for Speedway and custom applications. However, this means that the upgrade file system must always be empty or the upgrade process will fail. Therefore, embedded developers are prohibited from using this space,

and anything placed here could be deleted at any time. This file system is mounted read-write at /mnt/ufs.

Table 2.12 shows Upgrade File System sizes. Nominal sizes are given; the actual sizes reported by the Reader include an allowance for bad flash blocks.

Table 2.12 Speedway Upgrade File System Size

Image Partition Name	PCBA v4.xx (and lower)	PCBA v5.xx (and higher)
Upgrade File System (UFS)	32 Mbytes	63 Mbytes

2.12 RAM Allocation

The Speedway RAM is used for several purposes during run-time. While the Linux kernel manages the allocation of the available RAM, the total RAM available is limited. Custom application developers should be aware of the restrictions imposed by a finite amount of RAM and zero swap space. The RAM on the Reader is shared between the Linux OS, Reader applications, custom applications, and the RAM file system.

2.12.1 Linux OS

It is impossible to accurately measure the total RAM required by the Linux kernel and its associated drivers. However, certain kernel file systems and drivers have well-known requirements. As of Octane 4.6, the Linux kernel requires approximately 10 Mbytes of RAM.

2.12.2 Reader Applications

The Reader applications stored in the SOP, as described in the System Operating Partition section, control every aspect of the Speedway Reader. Each of these applications is critical to the Reader operation, whether it is RFID operations, firmware upgrades, system management, or log control, and each application requires RAM.

2.12.3 RAM File System

The RAM file system is available as temporary storage for dynamic data. This is the recommended file system to use for data that is volatile, requires fast access, or doesn't need to persist across Reader resets. Embedded software developers should use this file system whenever possible in place

of the CAP (discussed in the Custom Application Partition section) to avoid flash wear. This file system is mounted read-write at /tmp and consumes 1 Mbyte of RAM.

A larger RAM file system can be mounted as follows:

```
/bin/mkdir -p /var/bigram
/bin/mount tmpfs /var/bigram -t tmpfs -o size=8M
```

2.12.4 Custom Applications

In theory, custom applications can use the remaining RAM at their discretion. However, it is imperative that embedded developers understand the dynamic memory requirements of a real-time system. While the used memory presented in this section is accurate, system activity can cause fluctuations in the required memory (such as during a firmware upgrade). To ensure proper system operation, Impinj recommends that custom applications consume no more than 16 Mbytes of RAM. Failure to do so may result in abnormal system behavior and possible system reset as the available memory approaches zero.

2.12.5 RAM Allocation by Product

Table 2.13 Octane RAM Allocation (starting with Octane 5.2)

Product	Used by Octane	Available
Speedway Reader	~32 Mbytes	16 Mbytes
xPortal Gateway	~32 Mbytes	16 Mbytes
xArray Gateway	~48 Mbytes	64 Mbytes
xSpan Gateway	~48 Mbytes	64 Mbytes

Note that Speedway Readers that are embedded in xArray/xSpan Gateways have 128 Mbytes of SDRAM.

2.13 Linux File System

During the boot process, the mounter application determines which of the two images is active and mounts the appropriate file systems on the flash MTD devices, as described in the Mounter section. Therefore, custom applications don't need to concern themselves with the dual image design. The

standard Linux file system is always guaranteed to map to the correct image. Figure 2.12 depicts the Speedway file system and how each directory is mapped into the associated memory devices.

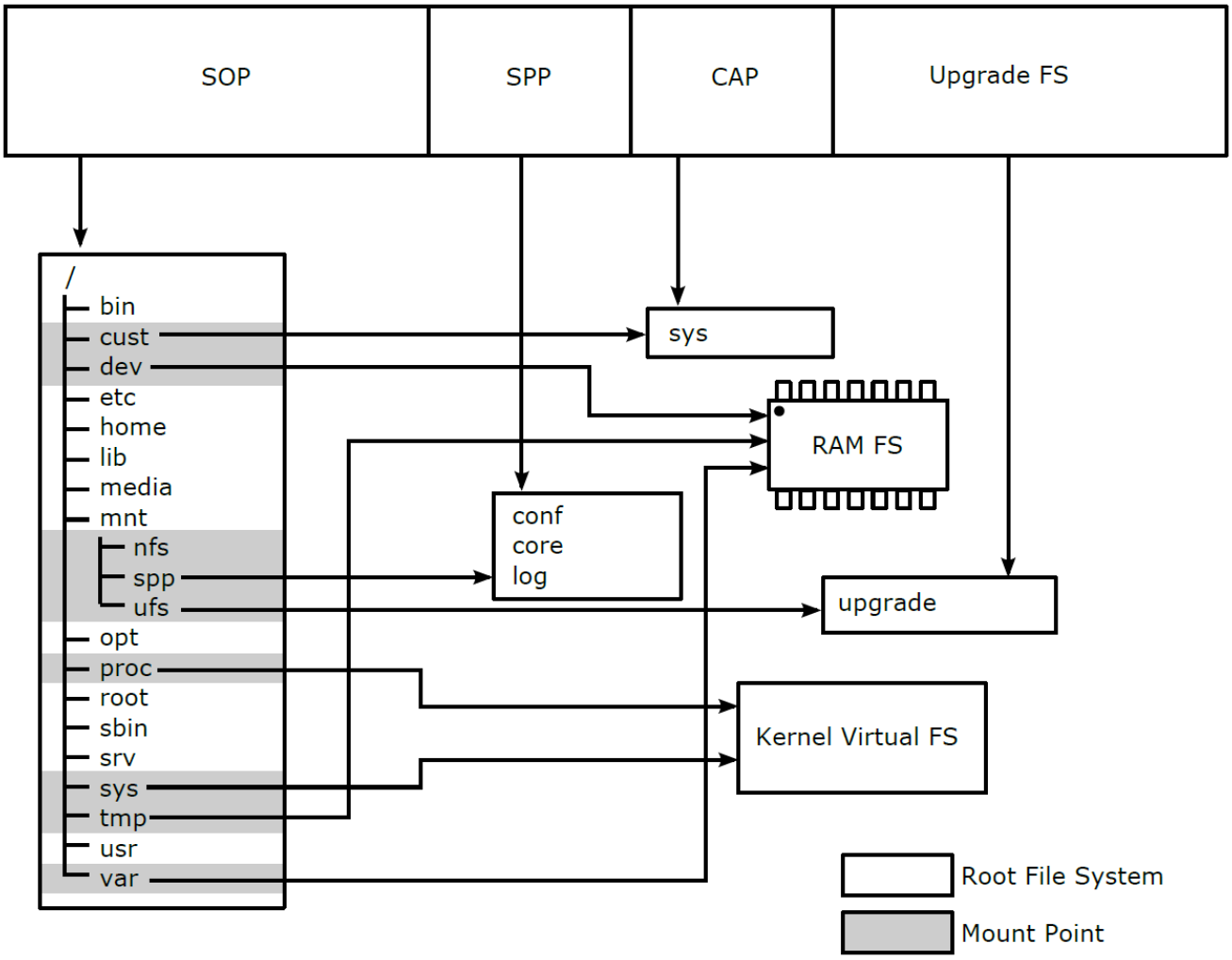

Figure 2.12 Speedway File System Layout

2.14 USB HID Keyboard Emulation

It is possible for your on-Reader application to send data through USB to a PC or other computer. This section describes how to use the hidkey program which provides an emulation of a USB HID (Human Interface Device) keyboard. To the PC, the Speedway RFID Reader appears to be a keyboard with somebody typing on it.

2.14.1 Requires On-Reader Application

To take advantage of the USB device interface, an on-Reader application is necessary, which must be cross-compiled for the target. The USB functionality is only accessible to Linux OS, so OSShell access is required to develop such an application. An example of an on-Reader application is shown in Figure 2.13.

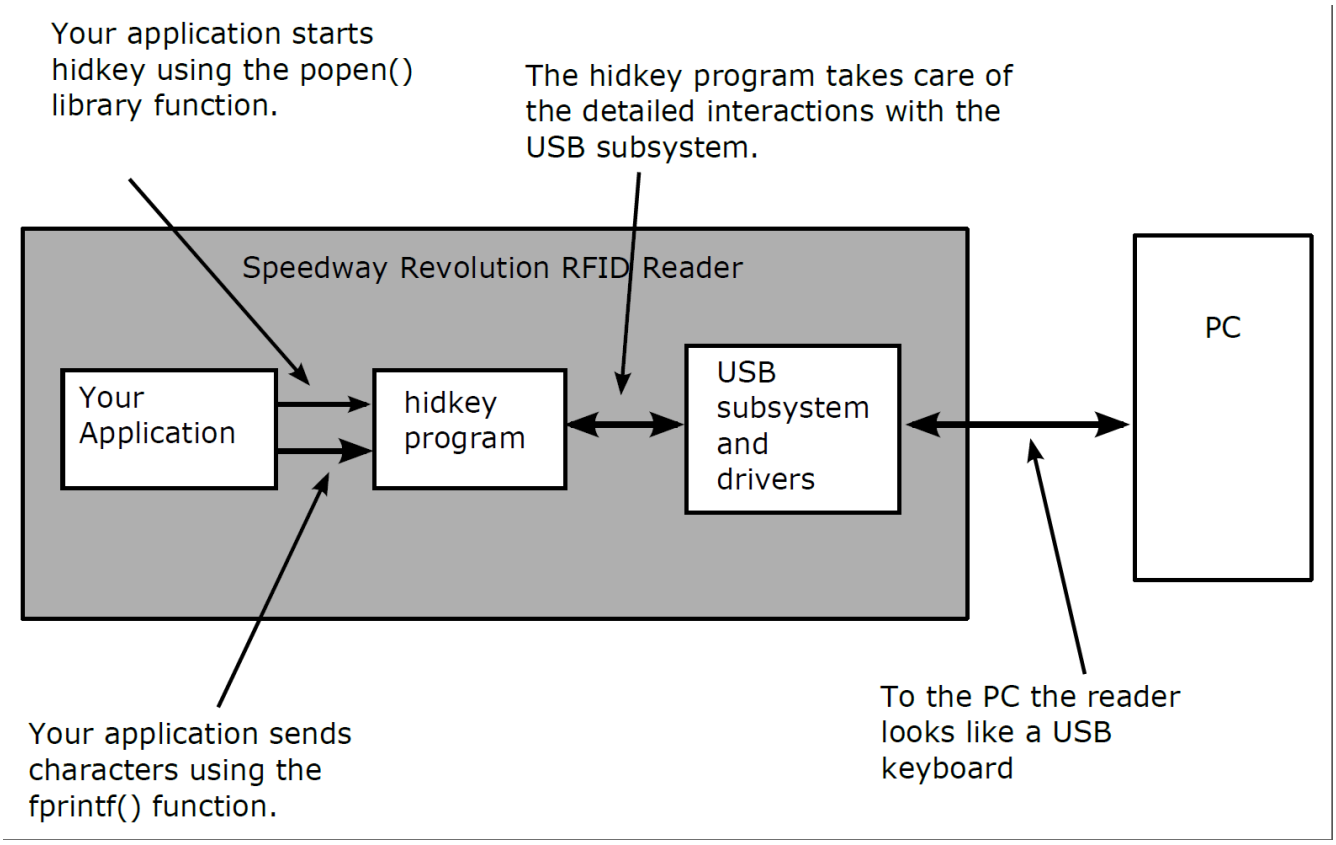

Figure 2.13 Speedway On-Reader Application

2.14.2 USB Cabling

Of course, an **A-B** USB cable is necessary to connect the Reader (device) to the host (e.g. a PC). Connection to the host will almost certainly be via one or more USB hubs. The host and any hubs should be USB 2.0-compliant.

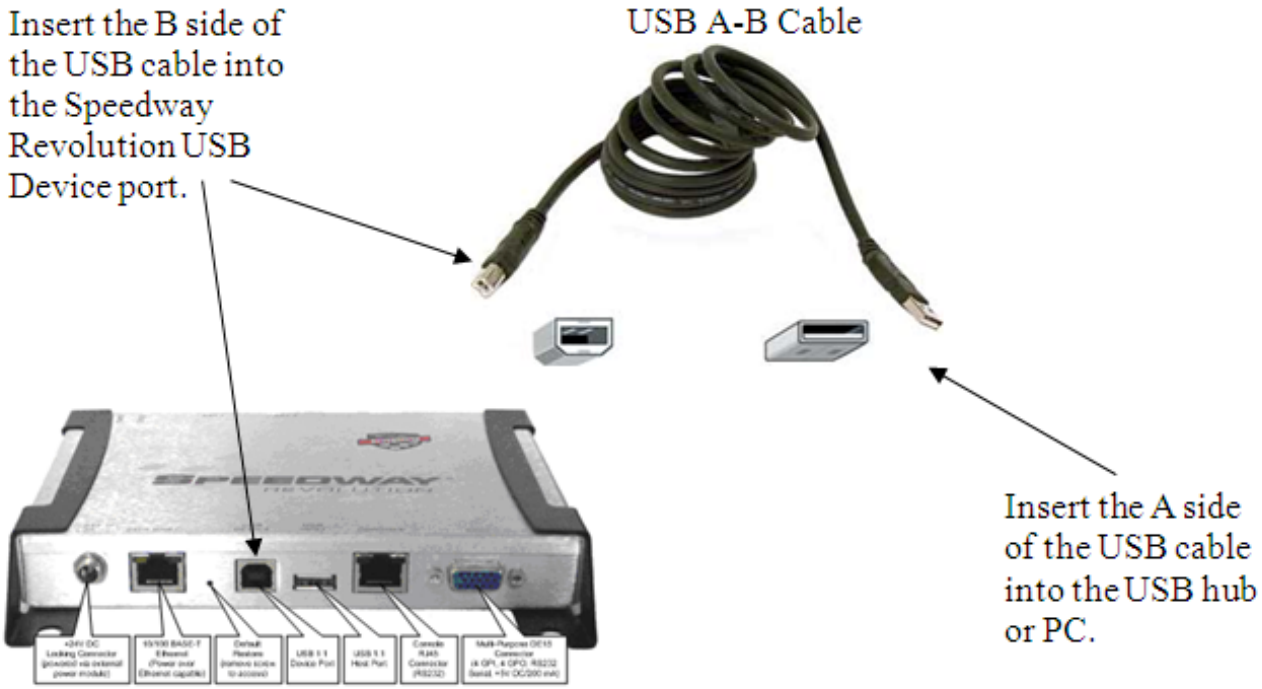

Figure 2.14 USB Cabling

2.14.3 USB Presence

Based on the behavior of other USB devices, when the Reader is initially attached to the host, you would expect the host to identify the Reader. This is not the case. An on-Reader application must start the USB HID Keyboard Emulation utility before any USB activity can start. The Reader can automatically start the application each time it boots, in which case the Reader will respond as a USB device, either when attached or at power-up while still attached.

2.14.4 Keyboard Emulation Program (hidkey)

The USB HID Keyboard Emulation utility is an executable program named **hidkey**, which is included in the Speedway's read-only System OS Partition (SOP) at the following path:

/opt/ys/usb/hidkey

When **help** is passed as a command line option, it shows a summary of its use and command line options.

```
root@SpeedwayR-00-00-00:~# /opt/ys/usb/hidkey help

USB HID Keyboard Emulation Driver
Version: Comp=hidkey;Ver= 5.12.3;…

Usage: /opt/ys/usb/hidkey [Options]

Options:
  ReportInterval=<N> (default is 10ms)
  KeyDelay=<K>,<D>[:<K>,<D>[…]] (upto 10 pairs)
  SerialNum=Device|Random (default is Device)
  SysLog=Emerg|Alert|Crit|Error|Warning|Notice|Info|Debug
  DbgLvl=Emerg|Alert|Crit|Error|Warning|Notice|Info|Debug|Debug1|Debug2

Notes:
  Intervals and <D>elays are in mS, <K>eys are ASCII codes
  Default SysLog is Error, default DbgLvl is Emerg

root@SpeedwayR-00-00-00:~
```

Because piping commands is well supported by the Bash shell, we can start with a shell-based example, as described in this procedure.

1. In order to proceed, you need to start an application on the Windows PC host that accepts keyboard input, such as an editor (e.g. Notepad) or Spreadsheet (e.g. Excel).
2. Now the Reader can be connected to the PC. Nothing will happen until the **hidkey** executable is started. Then the Reader will behave like a USB device and begin to enumerate.

   From OSShell, type the following:

   # echo "RFID That Just Works!" | /opt/ys/usb/hidkey
3. Before enumeration completes, keyboard input must be directed to the Windows application.
4. If this is the first time the driver is used, pop-ups that identify the device will flash up from the Windows system tray's USB icon.
5. After a short pause (approx. 20 seconds), the echoed string should appear in the application on the host. This pause is inserted by **hidkey** to allow enumeration to complete.

2.14.5 Disconnected Operation

If the Reader becomes disconnected, the **hidkey** program will continue to consume and discard new data. Because there is no pre-defined delimiter that groups reports, a partial report might be sent as a consequence of a disconnection or re-connection. It is expected that applications can be started and re-started while there is no report activity.

When you start the **hidkey** program, if a USB connection is not present, **hidkey** will discard all input data until a connection is established and enumeration completes.

2.14.6 Example USB HID Application

Here is an example of how the **hidkey** program can be used.

```
/*****************************************************************************
Application Example
*****************************************************************************
Outline:
An example application that opens the USB HID Keyboard Emulation program
(hidkey) in a non-blocking (by default) mode.

*****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>
#define ARRAYSIZE(a) sizeof(a)/sizeof(a[0])
#define name "MyApp"
int main(void)
{
   int cntr;
   FILE * pipe_fp;
   int ret;
   int pipe_fd;
   int c;
   /* RFID report like test data */
   char *strings[] = {
      "2009-09-03T00:12:39.705135\tEPC96\t76636575773F6B493FD7A981\n",
```

```
    "2009-09-04T00:12:40.124143\tEPC96\t76636575773F6B493FD7C74B\n",
    "2009-09-06T00:12:40.635294\tEPC96\t76636575773F6B4934B472A9\n"
};
/* Open the application with a pipe to its stdin */
fprintf(stderr, "%s: Opening pipe\n", name);
if (NULL == (pipe_fp = popen("/opt/ys/usb/hidkey", "w" )))
{
   perror("popen");
   exit(1);
}
/* Get the file descriptor (from the file pointer) */
if (-1 == (pipe_fd = fileno(pipe_fp)))
{
   perror("fileno");
   exit(1);
}
fprintf(stderr, "%s: Got file descriptor (%d)\n", name, pipe_fd);
/* Limit the pipe's buffer size to a (minimal) fixed size */
/* - this stops additional buffering being allocated */
setbuf(pipe_fp, NULL);
/* Wait for enumeration to complete, otherwise driver discards */
sleep(20);
/* Use fputs() */
for (cntr=0; cntr<ARRAYSIZE(strings); cntr++)
{
   ret = fputs(strings[cntr], pipe_fp);
   if (EOF == ret)
   {
      /* Did the write fail because the device is backed-up ? */
      if (errno != EWOULDBLOCK)
      {
         perror("fputs");
         exit(1);
      }
   }
}
/* Flush the pipe (won't necessarily empty it) */
fflush(pipe_fp);
```

```
    /* Close stream opened by popen. */
    /* - waits for and returns termination status */
    fprintf(stderr, "%s: Closing pipe ...", name);
    if (-1 == (ret = pclose(pipe_fp)))
    {
       perror("pclose");
    }
    fprintf(stderr, "%s: Closed. Exiting.\n", name);
    return 0;
}
```

Figure 2.15 Example USB HID Application

2.14.7 Supported Characters

During normal operation, the hidkey program converts ASCII characters into keyboard scan codes. When an ASCII character doesn't obviously map to a keyboard scan code, it is mapped to a substitute character, a "__", rather than raising an error or warning.

2.14.8 Performance

During testing, it became clear that sending characters at the fastest rates (smallest polling intervals) can overflow input buffering on some slower host applications. Specifically, MS Excel would garble characters when this situation occurred. No such issues were observed when the host application was a text editor. This is likely a function of the host's performance, the host's CPU load and the host application, so it is difficult to provide any guidance.

2.15 USB Flash Drive AutoRun

Speedway allows a firmware upgrade from a USB flash drive. When a USB flash drive is inserted into the Reader's USB slot, a program, informally referred to as **AutoRun**, is automatically invoked that checks for the presence of firmware upgrade files. The AutoRun program also checks to see whether an application-specific program is configured, so that the application can check for files on the USB flash drive.

2.15.1 USB Flash Drive Preparation

The USB flash drive program **AutoRun** works with VFAT USB flash drives. MAC and Linux (ext2 or ext3) formatted drives are not supported. Some very old flash drives support an older version of FAT with 8-character file names (with 3 character extensions); these drives are not supported. Use Windows XP or Windows Vista to manage files and directories on the flash drive, because this ensures compatibility with VFAT.

On the flash drive, create a directory \impinj\revolution\upgrade\images. Copy the upgrade file to that directory. The upgrade file must end with a .upg extension.

Note: prior to Octane 5.10, the upg file could not contain any other period (‘.’) symbols other than the period before the upg extension.

Remember that, on Windows, the path name separator is a backslash (\), while on the Reader the path name separator is a forward slash.

2.15.2 HardwareEvents Stanza in Reader.conf

The following stanza and parameter have been added to the /cust/sys/reader.conf file on the CAP to allow a CAP developer to indicate which command line to execute on a Mass Storage event.

```
[HardwareEvents]
OnMassStorageEvent=/cust/bin/mass_storage_event
```

This callback is executed with two parameters, the action and the path to the mounted directory. The action is either **add** or **remove**. The mounted directory is the path where the root of the USB flash drive appears. For example:

```
/cust/bin/mass\_storage\_event add /mnt/usbfs/usbsda1
```

2.15.3 Example Storage Event Handler

Here is an example of how the hidkey program can be used.

```
#!/bin/bash
if [ $# -ne 2 ]
then
  logger Invalid argument count
```

```
    exit 1
fi

ACTION="$1"
PREFIX="$2"

if [ "$ACTION" == "add" ]
then
    if [ -f $PREFIX/company/product/config_file ]
    then
        cp $PREFIX/company/product/config_file /cust/config/config_file
        logger /cust/config/config_file updated
    fi
elif [ "$ACTION" == "remove" ]
then
    : # The USB flash drive was removed, nothing to do
else
    logger Invalid action
fi
```

Figure 2.16 Storage Event Handler

3 Platform Boot Process

The Speedway boot process is a multi-stage procedure that transitions the Atmel from its internal IROM ultimately to the Linux OS. Figure 3.1 depicts this process in detail. The initial stages of this procedure are outside the scope of this document. However, the Bootloader and the Linux OS are relevant to embedded developers and will be covered in the sections that follow.

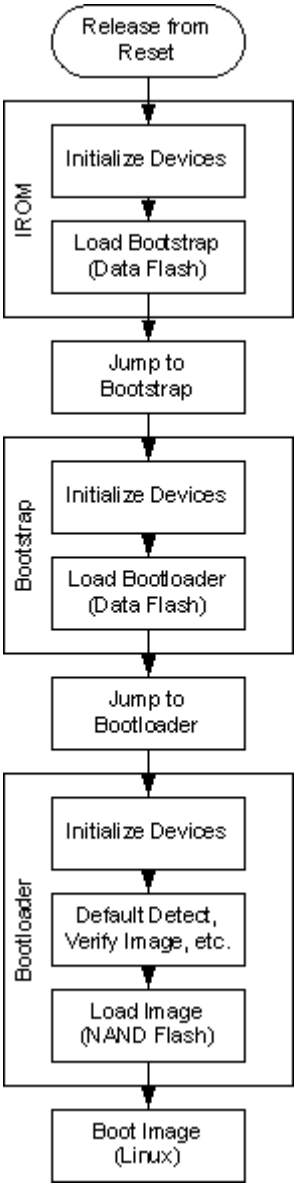

Figure 3.1 Speedway Boot Procedure

3.1 Bootloader

The Speedway Bootloader, U-Boot (Universal Bootloader) is open source software. U-Boot performs processor and hardware initialization, detects factory default restore, and performs firmware image validation, invoking the newest valid image. U-Boot is not intended for direct control by a user in the field, and Impinj does not provide direct support for U-Boot.

3.1.1 Default Restore

There are two types of default restore, Factory Default Restore (FDR), and Configuration Default Restore (CDR). The purpose of these default restore options is to allow the end user to reset the Reader to a known state. The difference between the two restores is summarized in Table 3.1.

Table 3.1 Factory Configuration and Default Restore

	Manufacturing Data	**SPP**	**CAP**
Configuration Default Restore	Unaffected	Restored to defaults	Notified
Factory Default Restore	Unaffected	Restored to defaults	Removed

Because there is only one default restore button, as shown in Figure 2.1, the two different restore types are invoked by pressing the default restore button when the power on for different durations. As an aid, the Power LED is used to provide feedback for the duration of the button press. Table 3.2 provides the different durations required to achieve the two different default restore options. For more information about the Power LED status patterns, see Speedway Reader LEDs.

Table 3.2 Default Restore Button and LED Behavior

	Default Restore Button	**Power LED**
Configuration Default Restore	Press for 3 seconds	OFF, 3 seconds, blink RED
Factory Default Restore	Press for 10 seconds	OFF, 3 seconds, blink RED, 7 seconds, blink RED twice

3.1.2 Image Selection

The bootloader determines which of the two application images are valid, and which of the two is the latest. The selected image is automatically booted. If neither image is valid, the bootloader displays an error message on the console, and the Power LED blinks red indefinitely (see Table 2.2 for more information about the Power LED status). The watchdog remains enabled in this scenario, and the bootloader simply waits for the watchdog reset. See Figure 3.2 for a depiction of this process.

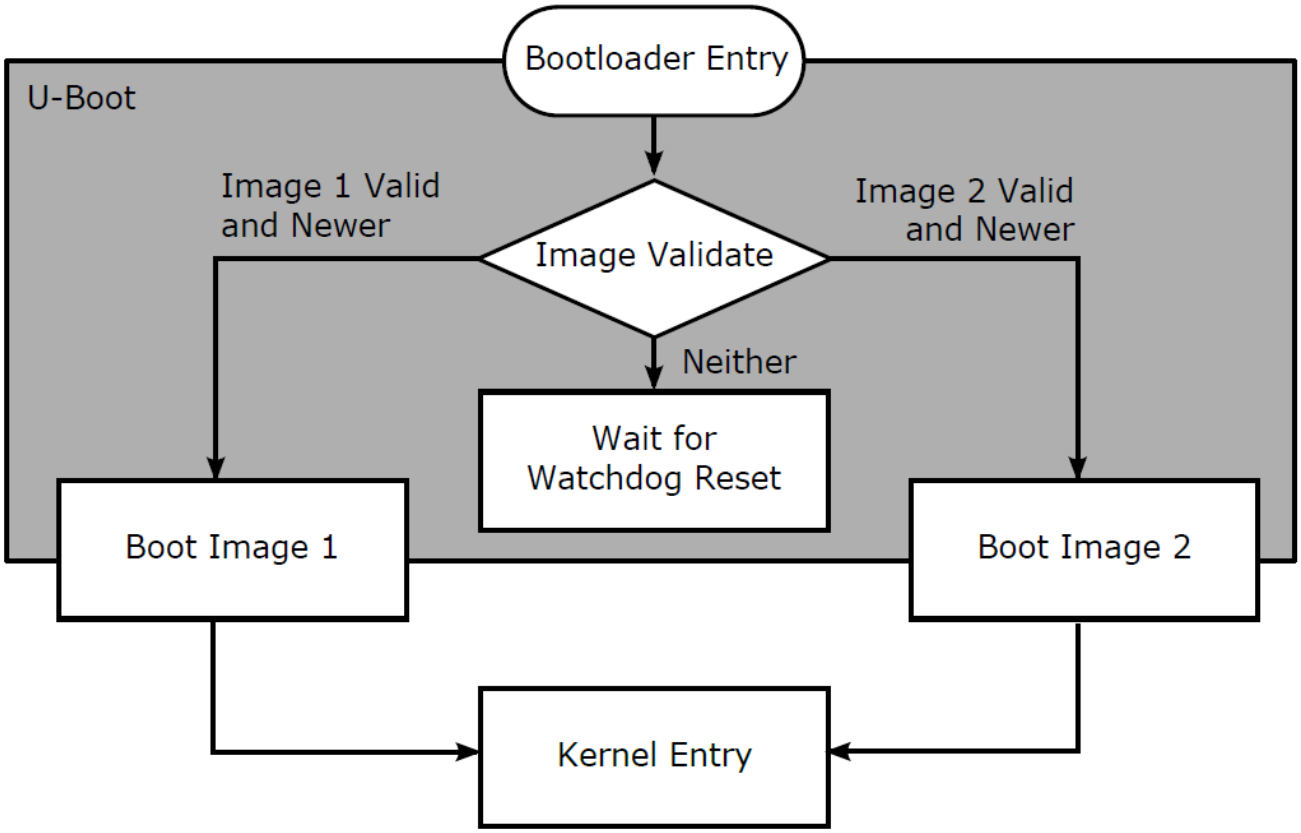

Figure 3.2 Image Selection and Boot

3.2 Operating System

After the bootloader has determined the appropriate image to boot, the root file system from the SOP of that image is mounted, and the Linux operating system is started. One of the first applications to start is **mounter**, which handles the upgrade scenarios.

After **mounter** has completed, the run-time Reader applications are started, including any custom applications that have been loaded.

3.2.1 Mounter

Mounter is the application that is responsible for completing upgrades after the new image has been booted. Mounter has three primary responsibilities:

- Perform CDR and FDR operations
- Copy partitions not included in the upgrade image from the old image to the new image.
- Notify any CAP applications of CDR and upgrade operations

Configuration Default Restore and Factory Default Restore

If the bootloader detects that either a configuration default restore or a factory default restore has been requested, it conveys this information via image metadata that is outside the scope of this document. After **mounter** recognizes that either a CDR or FDR has been requested, it is responsible for taking the appropriate actions.

For an FDR, **mounter** will erase the SPP and the CAP partitions, which results in a Reader with a default configuration and no custom applications.

For a CDR, **mounter** will erase the SPP, and notify the CAP of the CDR operation via the custom application's startup script. For more information about this process, see **CAP Notification** later in this section.

Partition Copy

Upgrade image files can contain different combinations of the CAP and SOP partitions. Based on the content of the upgrade image file, certain partitions might need to be copied from the previously active image to the new image in order for the upgrade to complete. Table 3.3 documents the allowable image file partition combinations, and lists the actions that the mounter application will take in each scenario.

Note that SOP images are maintained and released by Impinj. Embedded developers will receive these partitions from Impinj and can then choose to include them with their CAP partition if desired.

The SPP partition cannot be upgraded and thus is never included in an upgrade image.

Table 3.3 Mounter Operations Based on Upgrade Image Content

CAP	SOP	Mounter Operation
X		Copy SOP/SPP
	X	Copy CAP/SPP
X	X	Copy SPP

CAP Notification

During a firmware upgrade or a configuration default restore, the **mounter** application notifies any custom applications of the operation so that the application can take appropriate action. This notification is performed by invoking an executable (the upgrade application) within the CAP partition called:

```
/cust/cust_app_upgrade
```

This application is optional and is only required if the custom application wants to receive notifications of upgrade or default restore events.

On the first boot after a configuration default restore has been performed, the upgrade application is called as:

```
/cust/cust_app_upgrade cdr
```

This notifies the custom application that the Reader has been restored to its default configuration. The custom application should then likewise restore any configuration as deemed necessary.

On the first boot after a firmware upgrade (assuming a CAP exists on the secondary image), the upgrade application is called as:

```
/cust/cust_app_upgrade upg <cust_dir> <old_cust_dir>
```

This notifies the custom application on the primary image that an upgrade has been performed and that a previous version of the CAP image exists. The argument <cust_dir> represents the new CAP root directory (in this case, typically /cust), and <old_cust_dir> represents the previous CAP root directory.

The upgrade application can then temporarily access the previous image in read-only mode, and can copy any configuration or persistent files to the new partition prior to starting up the custom application.

In addition to the command line arguments, several environment variables are made available to the upgrade application so that it can infer context. Table 3.4 documents the environment variables that are exported to the upgrade application. Note that, in instances where a partition does not exist, such as when there is no secondary SOP or CAP installed on the Reader, the environment variables are not defined.

Table 3.4 Upgrade Application Environment Variables

Variable Name	Description	Example
primary_sop_vsn	Current SOP version	5.2.0.240
primary_cap_vsn	Current CAP version	1.0.2.0
secondary_sop_vsn	Previous SOP version	4.12.0.240
secondary_cap_vsn	Previous CAP version	1.0.1.0

Because the upgrade application is run very early during Reader initialization, there are several restrictions on this application and what it is permitted to do. Failure to follow these restrictions could result in a failure of the Reader to initialize. These restrictions include:

- When the upgrade application is invoked, the Reader has not completed its boot sequence yet. This implies that no RFID applications are available, nor is there any network connectivity. Therefore, the application should not start the actual custom application, nor should it attempt to access the network.
- When the upgrade application is invoked after an upgrade, <old_cust_dir> is mounted temporarily and is read-only. This means that all access to this mount point must occur within the upgrade application. The mount point is no longer available after the upgrade application terminates.
- Because the upgrade application is called prior to the run-time Reader applications, it must ensure prompt execution so that Reader initialization can proceed. Lengthy execution times for the upgrade application (several minutes) could result in a Reader reset. If the situation persists, a fallback to the secondary image might be required.

Figure 3.3 shows an example of an upgrade application in the form of a Bash shell script.

```
#!/bin/sh
event=$1
echo “Customer application upgrade script enters”
echo “Primary SOP version is $primary_sop_vsn”
echo “Primary CAP version is $primary_cap_vsn”
test ! -z $secondary_sop_vsn &&
  echo “Secondary SOP version is $secondary_sop_vsn”
test ! -z $secondary_cap_vsn &&
  echo “Secondary CAP version is $secondary_cap_vsn”
if [ $event = “cdr” ] ; then
```

```
    echo "Reader has restored its default configuration"

    # Copy default to current configuration
    if [ -f /cust/config/default_config ] ; then
      cp /cust/config/default_config /cust/config/my_config
    fi

  elif [ $event = "upg" ] ; then

    echo "Reader has been upgraded"

    cust_dir=$2
    old_cust_dir=$3

    # Copy previous to current configuration
    if [ -f $old_cust_dir/config/my_config ] ; then
      cp $old_cust_dir/config/my_config $cust_dir/config/my_config
    fi

  fi
```

Figure 3.3 Example Customer Upgrade Application

3.2.2 Run-Time Applications

After the appropriate image partitions have been mounted and any upgrade operations have completed, the run-time Reader applications are started. This includes applications related to network management, logging, upgrade, watchdog, and of course, RFID. Applications are started using standard Linux boot procedures and therefore should not assume a boot order.

As the run-time applications are being started, any custom applications that are present on the Reader are also started. Custom applications are started like any other run-time application and therefore must adhere to the same restrictions. These restrictions include:

- No boot order may be assumed. Custom applications must be robust enough to handle scenarios in which required resources are not yet available. Figure 3.4 shows how an application's start script can wait for network availability before calling the application.
- Applications are launched as daemons, and should behave as such. Custom monitoring of application health can be implemented.

For more information about starting custom applications, see the Executing the Custom Application and Automatically Starting a Custom Application sections.

```
#!/bin/sh

# Wait until the network is both connected and we have a DNS server
while true; do
   netconf | grep -q "connectionStatus='Connected"' && dnsconf | grep -q Server &&
break
   sleep 1
done

# Network is now up!
# Now start custom application …
```

Figure 3.4 Delaying Custom Application for Network

4 Software Development Process

This section provides information about embedded developer resources and procedures from the Speedway SDK. This information enables a developer to write an example application, generate the appropriate binaries, install the example application on the Reader, and execute the application.

4.1 Tools

The Speedway Reader filesystem is based on the 'Jessie' release of the Debian distribution. The Linux kernel version is 2.6.39.4. See https://www.debian.org/releases/jessie for more information.

An embedded software developer can develop a custom application by using the Octane Embedded Development Tools (ETK) available from the Impinj Support Portal. This package includes the Impinj LLRP libraries for C and C++, a sample application, a cross-compiled toolchain, and other utilities.

Note that the Embedded Developers Kit (EDK) that is based on the Linux Virtual Machine (VM) is no longer supported, and therefore is not recommended for developing new applications.

4.2 Linux syslog

An on-reader application can log information to the Linux application syslog using the BusyBux *logger* command line tool. The application syslog content is stored in the /mnt/spp/log/syslog-app.log file.

```
Usage: logger [OPTIONS] [MESSAGE]
  -s         Log to stderr as well as the system log
  -t TAG     Log using the specified tag (defaults to user name)
  -p PRIO    Priority (numeric or facility.level pair)
```

Example usage:

 # logger -s "Test Output to SysLog 1"

 # logger -s -t TEST1 "Test Output to SysLog 2"

 # logger -s -t TEST1 -p 7 "Test Output to SysLog 3"

 # logger -s -t TEST1 -p user.notice "Test Output to SysLog 4"

Resulting content in the syslog-app.log file:

```
Jul 18 15:59:18 (none) root: Test Output to SysLog 1
Jul 18 15:59:19 (none) TEST1: Test Output to SysLog 2
Jul 18 15:59:19 (none) TEST1: Test Output to SysLog 3
Jul 18 15:59:20 (none) TEST1: Test Output to SysLog 4
```

Note: The syslog priority levels are Emerg (0), Alert (1), Crit (2), Error (3), Warning (4), Notice (5), Info (6), and Debug (7). For more information on syslog levels, refer to the *Impinj RShell Reference Manual*.

4.3 CAP Upgrade Creation

An on-reader application is installed onto the reader via an upgrade file using the same upgrade mechanism as the Octane firmware. Impinj provides a utility to convert a directory structure along with its contents into an upgrade (.upg) file. This tool is called *cap_gen* and is included in the ETK. Most of *cap_gen*'s arguments are passed in a text description file. An explanation of each option is included in the ETK's example *cap_description.in* file. Note: *cap_gen* depends on *fstool*, this is also included in the ETK.

Here is an example of *cap_gen* description file:

```
# Upgrade Description File
#
# This file contains the settings used by the upgrade generation tool
# to produce an Impinj firmware upgrade file.
[Description]

# Version is a 4 part number in decimal with each part limited to
# 0-255. It is the version of the upgrade file to be generated.
Version = 1.2.3.4

# Valid Reader Hardware is a 3 part number in decimal representing
# the reader model and major/minor revisions on which the upgrade may be
# loaded. Each field may be replaced by the wildcard '*' or '255',
# which matches any.
#   Format = aaa.bbb.ccc
#     aaa - Model number
#     bbb - Major revision
#     ccc - Minor revision
Valid Reader Hardware = 225-*-*
Valid Reader Hardware = 240-*-*
```

```
Valid Reader Hardware = 250-*-*
Valid Reader Hardware = 260-*-*
Valid Reader Hardware = 270-*-*
Valid Reader Hardware = 280-*-*
Valid Reader Hardware = 290-*-*

# File System Layout is an value used by the reader to determine how
# the upgrade partition should be loaded to flash. Currently the only
# supported layout version is 10.
File System Layout = 10

# Input Directory is the top-level directory of the filesystem to
# create. The files under this directory will be available on the
# reader under /cust after the upgrade is loaded.
Input Directory = ./cap

# Encrypt the upgrade filesystem content. Note this will be
# decrypted when the upgrade file is processed by the reader.
# The default is 'False' (don't encrypt)
#Encrypt = True

# Use the expanded NAND Flash partition sizes available on
# all Speedway based readers and gateways with PCBAs equal or
# greater than v5.00.
# The default is 'False' (build for all PCBAs).
#ExpandedNAND = True
```

Several of these options are described in further detail below.

The *Version* field should be version number of the on-reader application. This is the version number that RShell will report for the CAP. It is also used by the auto upgrade mechanism to determine if the CAP needs to be upgraded, details of which are covered in the *Firmware Upgrade Reference Manual*.

The *Valid Reader Hardware* field(s) list which versions of the Reader the upgrade is suitable for. The version corresponds to the PCBA number and revision printed on the readers identity label, it is also reported by RShell ('show system platform'). An asterisk (or '255') in any position is a wildcard that matches anything. If the Reader's PCBA doesn't match, it will reject the upgrade file.

The *Input Directory* tells *cap_gen* the 'root' of the file structure it should include in the CAP upgrade file.

The content of a CAP firmware upgrade file cen be encrypted by setting the *Encrypt* field to 'True'. This applies a symmetric block cypher to the data (directories and files) contained in the CAP upgrade file. Provided the Reader is running Octane 5.4 (or higher) it will automatically decrypt the content when processing the upgrade file. Impinj recommends that an OSShell password is included in the CAP to further protect the content of the encrypted CAP upgrade file.

Later model Readers, those with PCBA's greater than v5.00, have more available on-reader Flash memory. The *ExpandedNAND* field should be set to 'True' to take advantage of the additional Flash memory.

4.3.1 Using cap_gen

The *cap_gen* utility has the following options:

```
Usage: cap_gen [-v] [<-d file -o file> | <-i file> | -V]
  -v                Output verbose information
  -d file           The upgrade description file to use for settings
  -o file           The output file to write
  -i file           The image file to inspect and display metadata
  -V                Output version information
```

cap_gen would typically be invoked as follows:

To generate an upgrade file:

```
$ cap_gen -d cap_description.in -o my_cap.upg
```

To view details of an existing upgrade file use:

```
$ cap_gen -i my_cap.upg
Partition START
  ValidHardware[00]      : 225-255.255
  ValidHardware[01]      : 240-255.255
  ValidHardware[02]      : 250-255.255
  ValidHardware[03]      : 260-255.255
  ValidHardware[04]      : 270-255.255
  ValidHardware[05]      : 280-255.255
  ValidHardware[06]      : 290-255.255
  FormatVersion          : 3
  Flags                  : 0x0000
  FirmwareVersion        : 1.2.3.4
  FileSystemLayout       : 10
  Partition              : CAP
```

4.4 Accessing the Linux Shell

The Speedway Reader has its own CLI named RFID-Shell, or RShell. End users can access this interface by using the RS-232 console port, or by connecting to the Reader over the network via SSH if it is configured. For how to configure network services, see the Software Features section.

Using this interface, you can configure the Reader and display system information. However, for an embedded developer, this interface is insufficient and access to the underlying Linux shell is required. Because of the critical nature of this interface and the ability to adversely affect Reader behavior if it is misused, access to the Linux shell (named OSShell) is protected from the end user.

For an embedded developer, access to this interface is made possible via a custom application configuration file that is built into the CAP upgrade image. For details about creating the CAP partition, please refer to the instructions provided with the Embedded Development Tools from the Impinj Support Portal.

To enable access to OSShell via RShell, You must create the following file within the CAP file system, for example:

```
/cust/sys/reader.conf
```

This file does much more than just enable access to OSShell. More specific information is provided in the Reader Configuration section.

For OSShell access, the following lines (or *stanza*) must appear within the file.

```
[rshell]
password=developer
```

Figure 4.1 Enabling Access to OSShell

After you build an upgrade image with the CAP partition containing this file, and the upgrade image is loaded onto the Reader, you can enable access to OSShell by using the following RShell command:

```
> osshell developer
```

The password **developer** in this example must match the password that exists in the Reader configuration file loaded on to the Reader. Invalid passwords will be rejected as if the command was invalid for security purposes. The OSShell feature is intended for embedded developers only. If the password is left blank (that is, **password=**), then password checking is disabled.

Note: Impinj highly recommends that the OSShell feature be disabled in deployed CAP image files.

File Transfer Protocol (FTP)

There is an FTP server on the Reader that can be used to transfer files to the Reader. As a security measure, this service is disabled by default. To use the FTP server on the Reader, the service must be enabled using the Reader configuration file. For information about how to enable FTP, see the Configuration File Example section.

Note: Similar to the OSShell password, Impinj does not recommend deploying Readers with this feature enabled.

4.4.1 Executing the Custom Application

Now that the custom application has been loaded on to the Reader, it can be tested. Connect to the Reader via SSH and attach a serial cable to the DE15 connector on the Reader. The procedure is illustrated here by using **CustApp**, a hypothetical example that accepts some input, prints a message, and exits.

To test the custom application CustApp

1. Log on to the Reader as the root user.
2. Access OSShell:

   > osshell developer
3. Navigate to the directory where the custom application binary was stored. If this was a development build and the file was transferred via NFS or FTP, go to the directory where the binary was transferred. If this is a deployed image, navigate to the CAP.

   # cd /cust
4. Run the custom application from the command line. Note that when the application runs in the foreground, the command prompt will hang up.

   # ./CustApp
5. Now enter several characters in the terminal program. If everything is working correctly, those characters should now appear in both the terminal window and the SSH window where the application is running. When you type a customized character (**A** in the example code), the result should be that the customized string appears in the SSH window. Figure 4.2 is a sample output from the SSH window.

6. Press **ESC** to quit the program.

```
root@SpeedwayR-00-00-00:/cust# ./CustApp
abcdefghijklmnopqrstuvwxyz0123456789
Hello Friends!
The End!
root@SpeedwayR-00-00-00:/cust#
```

Figure 4.2 Sample Example Application Output

4.4.2 Automatically Starting a Custom Application

The custom application in this example is one that is intended to be run manually, and is for demonstration purposes only. A typical custom application would be started along with the other run-time Reader applications, as described in the Run-Time Applications section.

To configure a custom application to be started when the Reader is reset, the Linux boot process invokes the following application at system startup:

```
/cust/start
```

This application is provided by the embedded developer and must have executable permissions. This application is started only once in the background, and its responsibility is to launch the custom application(s) as required. Of course, this may itself be the custom application, or it could be a shell script that launches, and perhaps monitors, other applications.

Figure 4.3 is an example of this type of script. This example tests for the presence of a custom application, and, if it is executable, starts it. If the application ends, it logs a message and re-starts the application, after a delay to prevent spinning.

```
#!/bin/sh
export LD_LIBRARY_PATH=/cust/lib
(( count = 1 ))
while true ; do
  if [ -f /cust/app/rfid_control ] ; then
    if [ -x /cust/app/rfid_control ] ; then
      /cust/app/rfid_control
```

```
        /usr/bin/logger -p user.notice \
        "Restarting custom application, count $count."
        (( count = count + 1 ))
      fi
      sleep 10
    else
      exit 0
    fi
  done
```

Figure 4.3 Example Customer Start Application

4.4.3 Custom Application Library Dependencies

The Speedway firmware contains a minimal set of libraries that are required for the applications which it supports. It is conceivable that a custom application could require additional libraries to operate that are not included in the SOP. Embedded applications that require additional library support can include these libraries in their CAP image, and can configure Linux via the LD_LIBRARY_PATH environment variable to scan the CAP partition when it searches for dynamic library dependencies.

A typical example of this might be if a custom application is written in C++ and requires the dynamic library libstdc++, which is not included in the SOP. To handle this scenario, developers should include this library in their CAP file system. Then the custom application start script should set the LD_LIBRARY_PATH environment variable to point to the appropriate directory where the shared objects are located, as is shown in the sample start script in Figure 4.3.

4.5 Firmware Upgrade Procedure

Speedway provides three methods for managing the firmware image: upgrade to a new image, fallback to a previous valid image, and default restore. Upgrades can be accomplished without disturbing the current Reader operation, and do not take effect until the Reader is rebooted, which could be automatic. This section describes the upgrade process.

4.5.1 Image Versioning Scheme

Each flash partition shown in Figure 2.10 has a version number associated with it. The length of the version number is 32 bits and the Reader internally maintains it as an unsigned integer. The

version number is represented as a string consisting of four fields separated by "**.**" as shown in the example below.

Example: *ddd.ddd.ddd.ddd*

Each field is a decimal number ranging from 0 to 255. The left-most field maps to the most significant byte of the internal **uint32_t** version number.

For the purpose of upgrades, when two version numbers are compared, a larger integer number is considered a higher version and is therefore a newer image. There is no other meaning to the version string. The custom application can use any versioning scheme as long as the version is 32 bits long, is represented in the format shown above, and has a larger number to indicate a newer version.

4.5.2 RShell Upgrade Methods

Speedway provides two methods to upgrade the firmware: **manual** and **automatic**. Use manual mode to perform a one-time upgrade of an individual Reader. The manual upgrade is started by entering commands at the RShell interface. For the purposes of embedded software development, the manual method is the most useful.

You can use the automatic method to upgrade a large network of Readers automatically by managing an upgrade configuration file called a **metafile**. The automatic method is outside the scope of this document. If you want to use the automatic method, see the *Firmware Upgrade Reference Manual* for more information.

Table 4.1 documents the various manual upgrade commands that are available via the RShell interface. This table is only intended to be a summary. For complete documentation of all available image management commands, see the *RShell Reference Manual*.

Table 4.1 RShell Upgrade Command Summary

RShell Command	Description	Auto-Reset
config image upgrade	Invoke an immediate upgrade. The image at the provided URI is downloaded to the Reader and installed over the secondary image.	No, manual reset required
config image default	Perform a configuration default restore. See Table 3.1 for complete information.	Yes
config image fallback	Fallback to the secondary image.	Yes

RShell Command	Description	Auto-Reset
config image removecap	Completely remove the CAP partition. The CAP is not mounted after the reset.	Yes
show image summary	Query the status of an upgrade and display both the primary and secondary image version information (SOP, SPP, and CAP)	N/A

4.5.3 OSShell Upgrade Methods

In addition to the commands available to end users via RShell, there are commands available to embedded developers only via OSShell. These commands are intended to directly manipulate the CAP partition, without the need to perform an upgrade. All of these commands, if given valid data, will return a status of Command-Being-Processed and will result in an automatic Reader reset when completed. Table 4.2 provides a list of OSShell upgrade commands.

Table 4.2 OSShell Upgrade Command Summary

OSShell Command	Description	Auto-Reset
uaconf formatcap=<ver>	Format the CAP partition. The <ver> field must be a valid 4-digit dotted version number (see the Configuration File Format section). The CAP will be mounted after the reset with the new version but will be empty.	Yes
uaconf removecap=true	Completely remove the CAP partition. The CAP will not be mounted after the reset.	Yes

It should be noted that these commands manipulate the secondary image to achieve their purpose. Therefore, when a CAP is formatted or removed, the actions occur on the secondary images. After the Reader is reset, **mounter** performs the appropriate partition copy procedures, as described in the Mounter section. This is significant, because performing these operations means that the fallback image is no longer available after they are completed.

4.6 Reader Configuration

Speedway is designed to allow custom applications to control many of the features and services available to the Reader. This customization is achieved by using the /cust/sys/reader.conf configuration file that we first visited when discussing OSShell access in the section Accessing the Linux Shell. This section documents the other capabilities of this file.

4.6.1 Configuration File Format

The Reader configuration file /cust/sys/reader.conf is logically organized into *stanzas*. A stanza is defined as a section of the file that starts with a stanza name enclosed in brackets, and ends with either the end of the file, or the next stanza name. The opening bracket ([) of the stanza name must appear as the first character in the line, followed immediately by the stanza name, then immediately closed with a terminating bracket (]). Characters beyond the closing bracket of a stanza name are ignored.

Inside a stanza is a set of name value pairs (NVPs) in the form:

```
name=value
```

When parsing the NVPs included in a stanza, whitespace is ignored and therefore lines can be indented for clarity. The name portion of the NVP is case-sensitive, so it must appear exactly as documented to work correctly.

Table 4.3 lists the available stanzas and gives a brief summary of what each configures. As customizable services are added to the Speedway firmware, additional stanzas may be added in the future.

Table 4.3 Configuration Stanzas

Stanza Name	Description
[rshell]	Configuration of the OSShell access password.
[HardwareEvents]	Configuration of callback notifications to CAP applications based on certain hardware events.
[PartnerFeatures]	Configuration of specific Reader features only available to select Impinj partners.
[SoftwareFeatures]	Configuration of network services and other software features.

4.6.2 Hardware Events

Speedway Readers have both a USB device and USB host port. When the Reader detects that a device was inserted into one of the USB ports, the Octane firmware can notify custom applications of the event. Each notification type has different characteristics, which are described in Table 4.4 and the sections that follow.

Table 4.4 Hardware Event Notification Types

Hardware Event	Port	NVP Name	NVP Value
Mass Storage Device	Host	OnMassStorageEvent	Callback Script

Mass Storage Device

This event is delivered to the custom application via a callback script. The script provided in the configuration file must be present in the file system with executable permissions. When the insertion or removal of a mass storage device is detected by the USB host port, the script is invoked, with specific command line arguments and environment variables that convey what event has occurred. The callback script can then take whatever action is required, such as transfer configuration files, transfer log files, etc. Table 4.5 lists the arguments and environment variables made available to the callback script.

Table 4.5 Mass Storage Event Variables

Variable Name	Description
Command Line Argument 1	Event Type: <add | remove>
Command Line Argument 2	File System Path
Environment ACTION	Event Type: <add | remove>
Environment MDEV	Device Name: e.g. <sda1>

Note that when a mass storage device insertion has been detected, the callback script does not need to mount the device. The Speedway firmware will already have mounted the device by the time the callback script is invoked and will pass that path via the second command line argument (e.g. /mnt/usbfs/usbsda1/).

4.6.3 Partner Features

Some Speedway features were developed for specific Impinj partners, and must be explicitly enabled before they will operate. This section of the Reader configuration file is used to enable these features. The details of how this is performed are outside the scope of this document. For information about specific features and how they are enabled, please contact your Impinj Sales Representative.

4.6.4 Software Features

Speedway firmware supports a number of commonly used network services. However, by default, only a subset of those services is enabled for security purposes. Embedded developers can customize those services by using the Reader configuration file.

Table 4.6 lists the available network services on the Reader, and whether they can be customized. The NVP name is the case-sensitive line that must appear within the SoftwareFeatures stanza of the Reader configuration file in order to customize the service.

Table 4.6 Customizable Network Services

Service	Default	NVP Name
mDNS	On	StartmDNS
NTP	On	StartNTP
SNMP	On	DefaultStartSnmp
FTP	Off	StartFTP
HTTP	On	StartHTTP
FTP Upgrade	On	AllowUpgradeFTP
HTTP Upgrade	On	AllowUpgradeHTTP
Web Upgrade	On	AllowUpgradeWeb

4.6.5 Configuration File Example

Figure 4.4 depicts a sample Reader configuration file. This is provided as an illustrative example for embedded developers to better understand the file format and its contents.

```
#
# This is a sample reader configuration file.
```

```
# Note that lines beginning with a # are ignored.
#
[rshell]
  # This is the OSShell password. It should be something more secure
  # than this example, and should be disabled after it is deployed.
  password=developer
[HardwareEvents]
  # Invoke special script on a mass-storage device
  OnMassStorageEvent=/cust/bin/sdAction.sh
[PartnerFeatures]
  # Enable an example reader feature here
[SoftwareFeatures]
  # Turn on FTP for development
  StartFTP=yes
  # Turn off NTP and rely on the RTC for time
  StartNTP=no
```

Figure 4.4 Example Reader Configuration File

5 Document Revision History

Date	Revision	Comments
04/08/2009	4.0	Original Release
08/27/2009	4.2	Updated for Octane Release 4.2
03/31/2010	4.4	Updated for Octane Release 4.4
06/02/2010	4.4.1	Moved description of tools to EDK web pages
12/14/2010	4.6.1	Updates for release of 4.6 and EDK 1.0
04/25/2011	4.8.0	Updates for Octane 4.8 release
02/13/2012	4.10.0	Added comment on unused serial pins on RJ45. Updated for Octane 4.10 release 2.1.4 The remaining pins are not utilized by the Reader and are left unconnected.
12/16/2014	5.2.0	Added information about xPortal and xArray Gateway products Added information about Speedway H/W changes (PCBA v5.xx an up) Refer to the Embedded Development Tools from Portal instead of EDK Updated versions for Octane 5.2 release. Edited, updated cross references, and made figure and table names consistent.
06/02/2015 06/16/2015	5.4.0	Updates for Octane 5.4.0 release - Added cap_gen information
11/18/2015	5.6.0	Updates for Octane 5.6.0 release - Debian version upgraded to Jessie in Octane firmware
12/21/2015	5.6.2	Updates for Octane 5.6.2 release
10/10/2016	5.8.0	Updates for Octane 5.8.0 release - Added information about xSpan Gateway product.
5/16/2017	5.12.0	Updates for Octane 5.12.0 release - Added information about R120 Gateway product. - Deprecated TFTP support.

6 Notices

Impinj gives no representation or warranty, express or implied, for accuracy or reliability of information in this document. Impinj reserves the right to change its products and services and this information at any time without notice.

EXCEPT AS PROVIDED IN IMPINJ'S TERMS AND CONDITIONS OF SALE (OR AS OTHERWISE AGREED IN A VALID WRITTEN INDIVIDUAL AGREEMENT WITH IMPINJ), IMPINJ ASSUMES NO LIABILITY WHATSOEVER AND IMPINJ DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATED TO SALE AND/OR USE OF IMPINJ PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT.

NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY PATENT, COPYRIGHT, MASK WORK RIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT IS GRANTED BY THIS DOCUMENT.

Impinj assumes no liability for applications assistance or customer product design. Customers should provide adequate design and operating safeguards to minimize risks.

Impinj products are not designed, warranted or authorized for use in any product or application where a malfunction may reasonably be expected to cause personal injury or death or property or environmental damage ("hazardous uses") or for use in automotive environments. Customers must indemnify Impinj against any damages arising out of the use of Impinj products in any hazardous or automotive uses.

Impinj, GrandPrix ™, Indy ®, Monza ®, Octane ™, QT ®, Speedway ®, STP ™, True3D ™, xArray ®, and xSpan ® are trademarks or registered trademarks of Impinj, Inc. All other product or service names are trademarks of their respective companies.

These products may be covered by one or more U.S. patents. See `http://www.impinj.com/patents` for details.

For more information, contact support@impinj.com