Speedway® Reader Application Series

Embedded Developer's Guide

REV 4.8.0 25-April-2011 Proprietary and Confidential www.impinj.com

For more information, contact rfid_info@impinj.com

Table of Contents

Table of Figures

Table of Tables

1 Introduction

1.1 Purpose

This document describes the Speedway Revolution platform and high-level architecture. It is intended for embedded software developers designing custom application software to run on the Speedway Revolution reader.

1.2 Scope

This document describes at a high-level how to use the basic functionality of the Speedway Revolution reader. It also provides detailed information about the platform architecture provided by Impinj to custom application embedded software developers.

This document concentrates on an embedded developer's view of the Speedway Revolution Reader platform. The Revolution Embedded Developers Kit (EDK) contains further documentation (as web pages) detailing the use of several Impinj supplied tools necessary to build and deploy embedded applications.

1.3 References

Table 1-1 References

Document	Version
EPCglobal: Low Level Reader Protocol (LLRP)	1.0.1
Speedway Revolution User's Guide	4.8
Speedway Revolution RShell Reference Manual	4.8
Speedway Revolution Firmware Upgrade Reference Manual API	4.8
Octane LLRP	4.8

1.4 Acronyms

API	Application Programming Interface
CAP	Custom Application Partition
CLI	Command Line Interface
CPU	Central Processing Unit
DC	Direct Current
DE9	9-pin D-sub E-shell connector[1]
DE15	15-pin D-sub E-shell connector[1]
DHCP	Dynamic Host Configuration Protocol
FCR	Factory Configuration Restore
FDR	Factory Default Restore

[1] *DE9 and DE15 connectors are often incorrectly referenced as DB9 and DB15.*

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

2 Platform Architecture

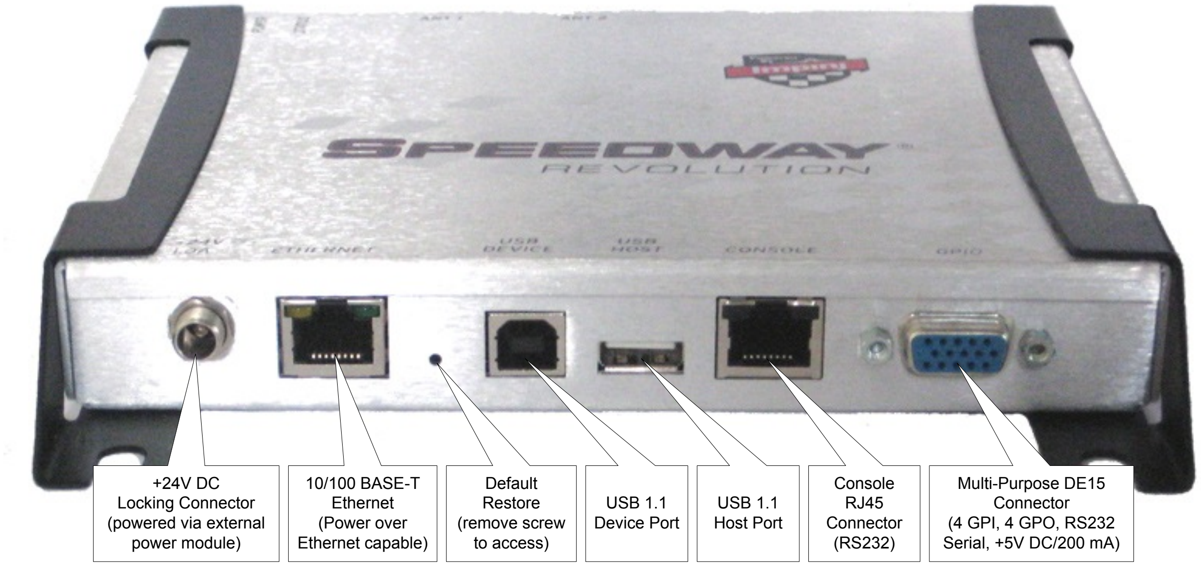
The Speedway Revolution R220 and R420 readers are fixed UHF Gen2 RFID tag readers that provide network connectivity between tag data and enterprise system software.

This Embedded Developer's Guide provides instructions on how to configure the Speedway Revolution reader 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 Reader Enclosure

Refer to the figures below for pictures of the Speedway Revolution reader enclosure with major ports, connectors, and status indicators clearly labeled. Figure 2-1 depicts the hardware (network, power, Ethernet, etc.) ports, and Figure 2-3 depicts the antenna ports and status LEDs.

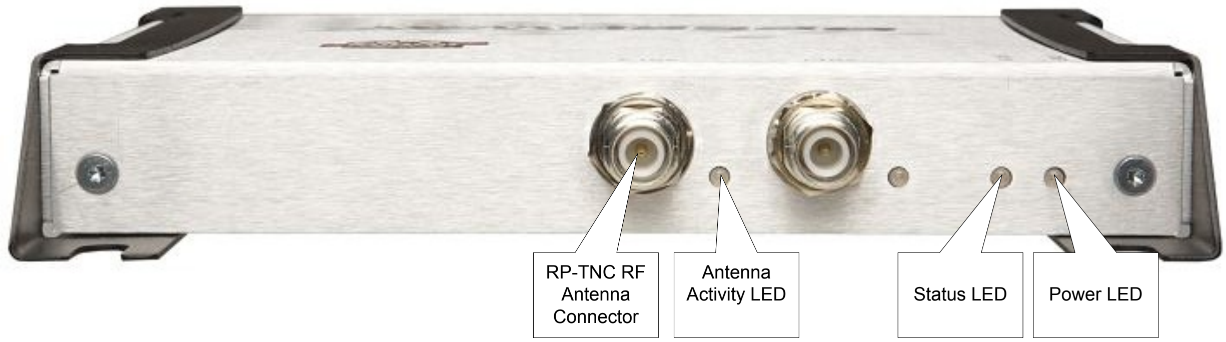
Figure 2-1 Speedway Revolution Hardware Connections


+24V DC Locking Connector (powered via external power module)
10/100 BASE-T Ethernet (Power over Ethernet capable)
Default Restore (remove screw to access)
USB 1.1 Device Port
USB 1.1 Host Port
Console RJ45 Connector (RS232)
Multi-Purpose DE15 Connector (4 GPI, 4 GPO, RS232 Serial, +5V DC/200 mA)

The Speedway Revolution 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 section 2.1.2 for the details and pin out. (labeled GPIO)

Figure 2-2 Speedway Revolution Antenna Connections



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

- Two (R220, pictured) or four (R420) female RP-TNC antenna connectors (labeled ANT1-ANTn)
- Two (R220, pictured) or four (R420) antenna activity LEDs (see section 2.1.3)
- One status and one power LED (see section 2.1.3)

2.1.1 Speedway Revolution Power Supply

Speedway Revolution 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 Power over Ethernet, the 24V external supply must be disconnected, and the Ethernet port connected to a cable 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.

2.1.2 Speedway Revolution GPIO

Speedway Revolution 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 debounce interval that is configurable via LLRP (see Octane LLRP documentation for more information). 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 within 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 which can be connected to V+ and V-.

See Figure 2-3 and Table 2-1 for details regarding the pin out of the DE15.

Figure 2-3 DE15 Female Connector

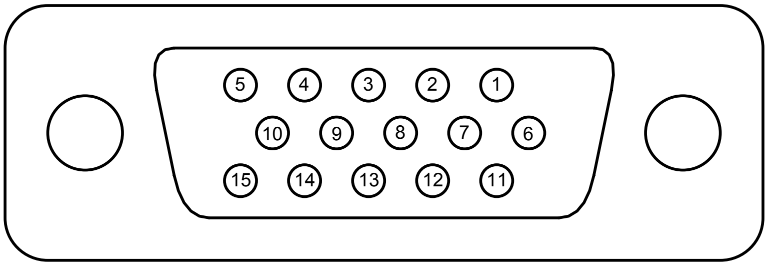


Table 2-1 DE15 Connector Pin-Out

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
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)

2.1.3 Speedway Revolution LEDs

The Speedway Revolution 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. Lastly, 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 section 3.1.1)
Two short RED blinks	Factory Default Restore detected (see section 3.1.1)
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
Two short GREEN blinks	LLRP connection active
One short GREEN blink	LLRP active, but no LLRP connection
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

2.1.4 Speedway Revolution Console Port

The Speedway Revolution console port uses RS232 serial over a RJ45 cable. As 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 Revolution RJ45 port uses a Cisco-compatible pin out and thus 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 2-5 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.

Figure 2-4 DE9 Female Connector

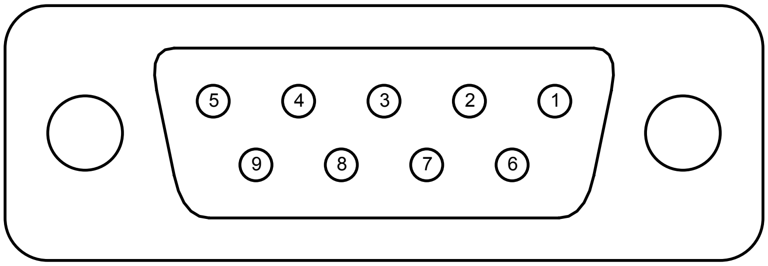


Figure 2-5 RJ45 Male Connector

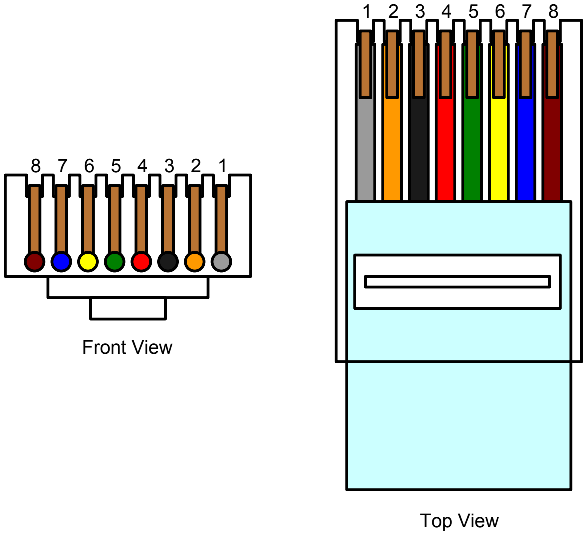


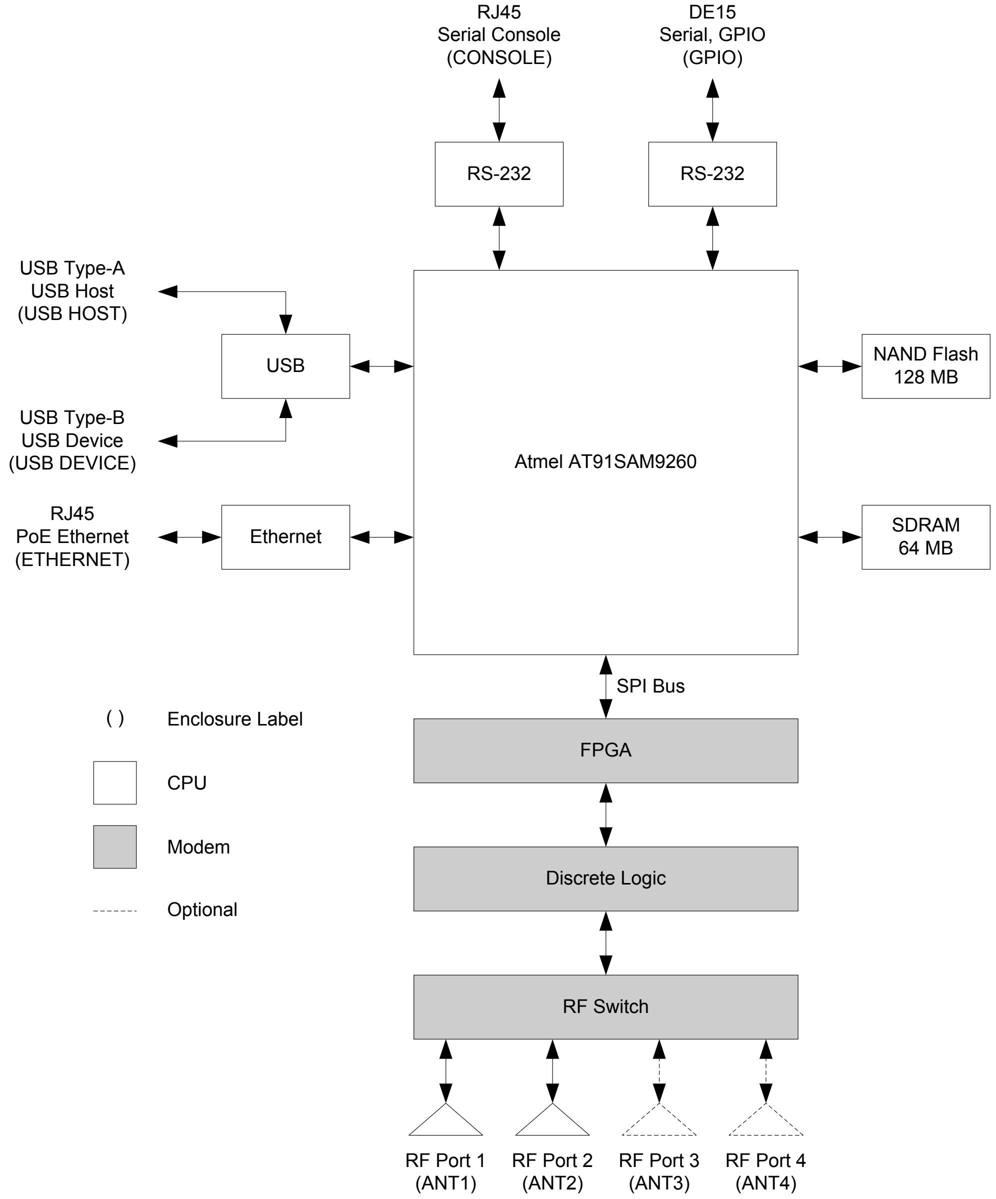
Table 2-5 RS232 DE9-to-RJ45 Pin Mapping

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
Request to Send	7	8	Brown
Clear to Send	8	1	Gray
Ring Indicator	9	NC	N/A

2.2 Platform Overview

The Speedway Revolution Reader is a single processor system (see Figure 2-6 for a hardware block diagram) with the control platform based on an Atmel AT91SAM9260 running at 200 MHz. 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 128 Mbytes of flash memory and 64 Mbytes of SDRAM available for firmware and data storage. See sections 2.3 and 2.4 for flash and SDRAM allocation schemes, respectively, to determine how much is available for custom application development.

Figure 2-6 Speedway Revolution Hardware Block Design



2.3 Flash Organization

The Speedway Revolution flash device is logically partitioned as shown in Figure 2-7. 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 2-7 Speedway Revolution Flash Organization

Reserved	Image 1	Image 2	Upgrade FS	Reserved
4 MB	39.5 MB	39.5 MB	34.5 MB	10.5 MB

Within each firmware image are individual JFFS2 formatted partitions used to logically organize the system software. Each image is partitioned as shown in Figure 2-8. 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 2-8 Speedway Revolution Image Organization

SOP	Reserved	SPP	Reserved	CAP	Reserved
16 MB	2.5 MB	8 MB	2.5 MB	8 MB	2.5 MB

2.3.1 System Operating Partition (SOP)

This is the primary system partition of the Speedway Revolution reader. Embedded developers may not modify the contents of this partition. This partition is mounted read-only at / (root) and consumes 16 Mbytes of flash memory. The primary contents of the SOP are:

- Linux 2.6.18 kernel
- FPGA firmware
- RFID management software
- Reader management (RShell, see Speedway Revolution RShell documentation)
- Logging management software
- Firmware upgrade control
- System watchdog software
- Factory default data

2.3.2 System Persistent Partition (SPP)

This is the persistent partition of the Speedway Revolution reader. Embedded developers may not 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` and consumes 8 Mbytes of flash memory. 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)

2.3.3 Custom Application Partition (CAP)

This is the custom application partition of the Speedway Revolution 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 may or may 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` and consumes 8 Mbytes of flash memory. 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. Where frequent modifications are required within a file system, it is recommended that custom application developers use the RAM file system (section 2.4.3).

2.3.4 Upgrade File System

The upgrade file system is not included within a firmware image and thus is not considered a partition like the previous sections have discussed. Instead, the upgrade file system is used by the active image (regardless of which is active) to perform firmware upgrades. During an upgrade, the firmware upgrade application downloads the upgrade image to this file system. This avoids requiring the image file to be stored in RAM, thus increasing the RAM available for Speedway and custom applications. However, this means that this file system must always be empty or the upgrade process will fail. Thus embedded developers are prohibited from using this space, and anything placed here may be deleted at any time. This file system is mounted read-write at `/mnt/ufs` and consumes 34.5 Mbytes of flash memory.

2.4 RAM Allocation

The Speedway Revolution SDRAM is used for several purposes during runtime. While the Linux kernel manages the allocation of the available RAM, the total RAM available is limited and 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.

2.4.1 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.

2.4.2 Reader Applications

The reader applications stored in the SOP (section 2.3.1) control every aspect of the Speedway Revolution reader. Be it RFID operations, firmware upgrades, system management, or log control, each of these applications is critical to the reader operation and each application requires RAM. As of Octane 4.4, reader applications require approximately 32 Mbytes of RAM.

2.4.3 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 need not persist across reader resets. Embedded software developers should use this file system whenever possible in place of the CAP (section 2.3.3) to avoid flash wear. This file system is mounted read-write at `/tmp` and consumes 1 Mbyte of RAM.

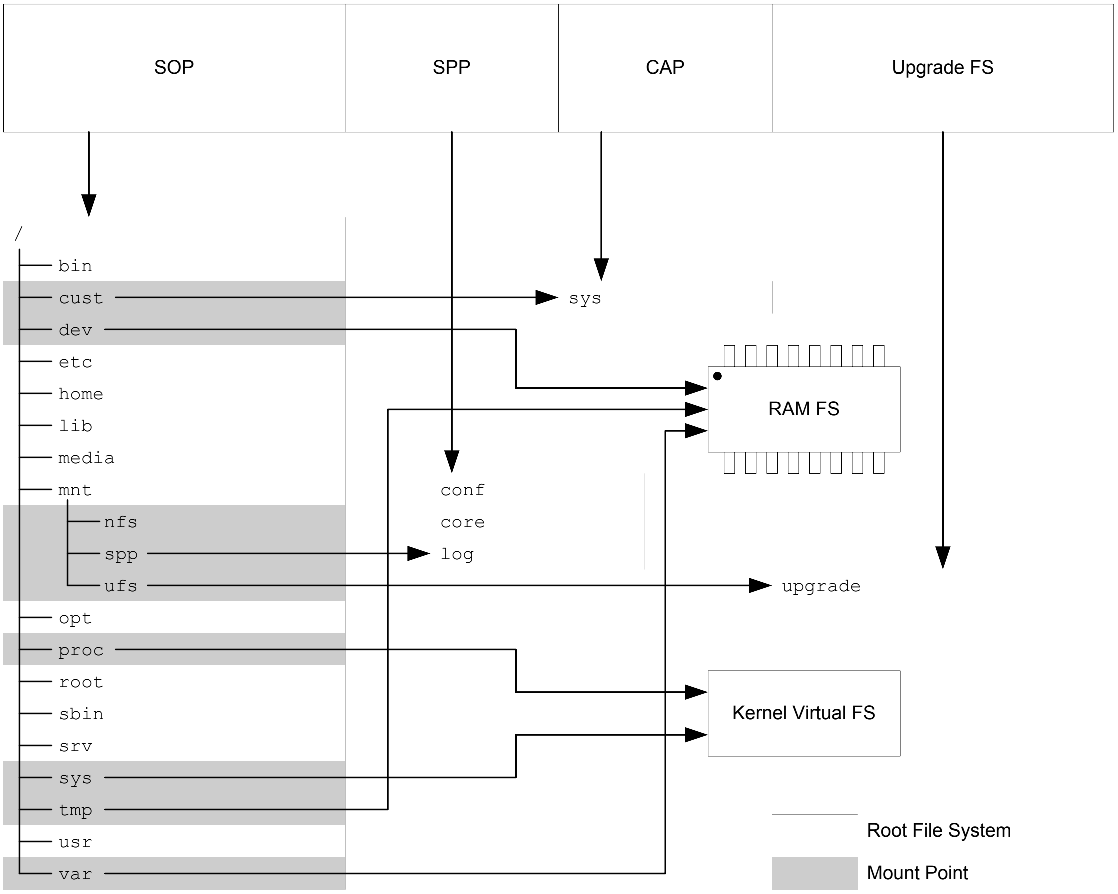
2.4.4 Custom Applications

Custom applications in theory may 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 may 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 8 Mbytes of RAM. Failure to do so may result in abnormal system behavior and possible system reset as the available memory approaches zero.

2.5 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 (see section 3.2.1). Thus custom applications need not concern themselves with the dual image design. The standard Linux file system is always guaranteed to map to the correct image. Figure 2-9 depicts the Speedway Revolution file system and how each directory is mapped into the associated memory devices.

Figure 2-9 Speedway Revolution File System Layout



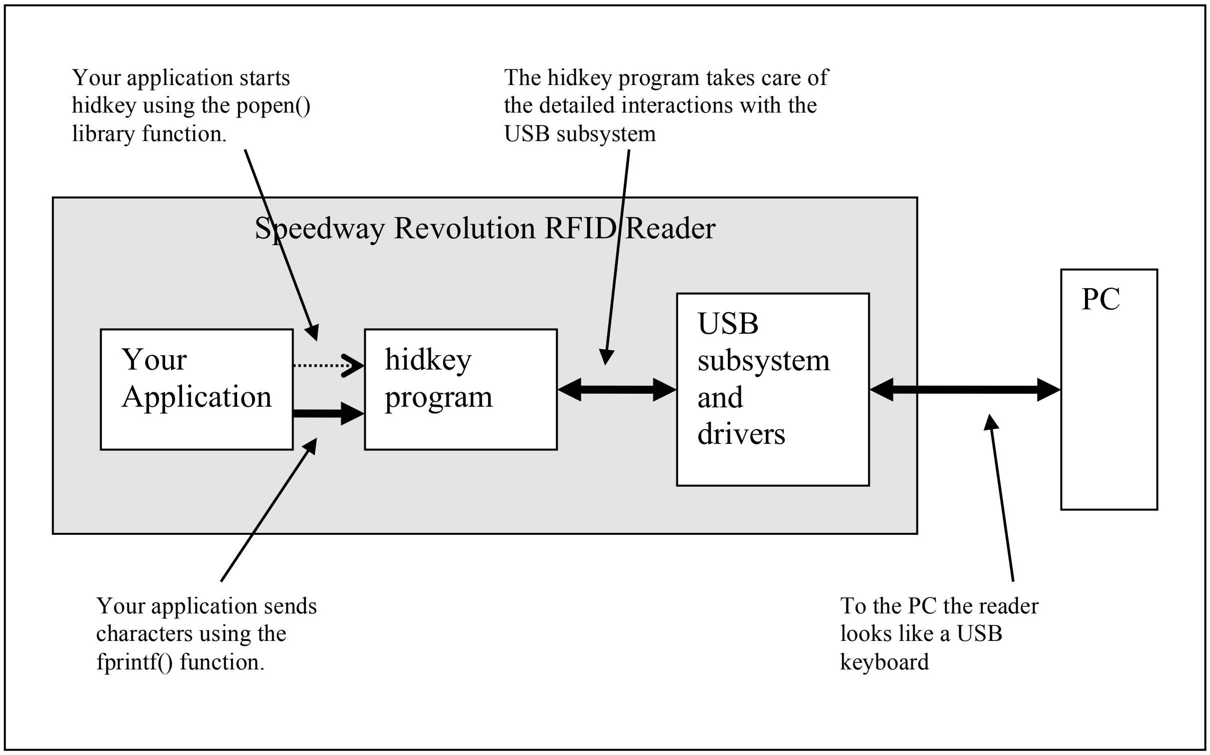
2.6 USB HID Keyboard Emulation

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 Revolution RFID Reader appears to be a keyboard with somebody typing on it.

Octane 4.4.0 or later is required for this feature.

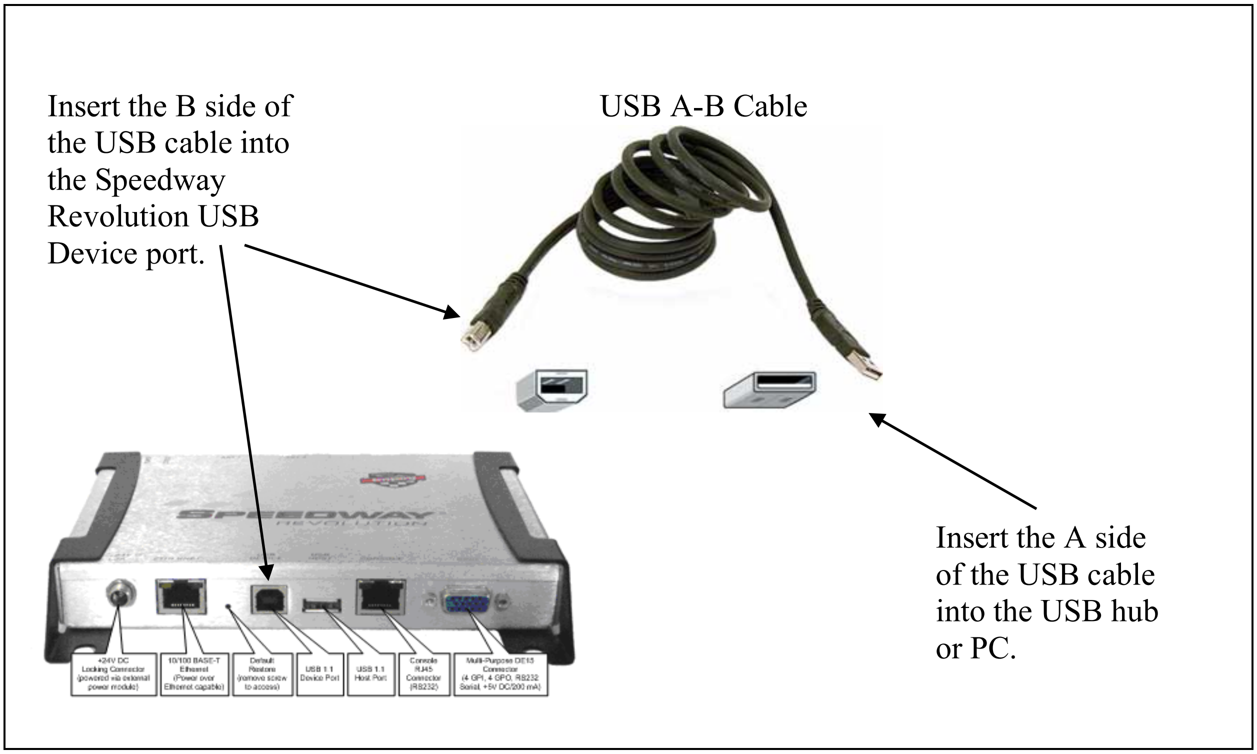
2.6.1 Requires On-Reader Application

To take advantage of the USB device interface, an on-reader application is necessary; this must be cross-compiled for the target. The Impinj SpeedwayR EDK would be an ideal environment to create such an application. The USB functionality is only accessible Linux OS, so OSShell access is required to develop such an application.


Your application starts hidkey using the popen() library function.
The hidkey program takes care of the detailed interactions with the USB subsystem
Speedway Revolution RFID Reader
Your Application
hidkey program
USB subsystem and drivers
PC
Your application sends characters using the fprintf() function.
To the PC the reader looks like a USB keyboard

2.6.2 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.



Insert the B side of the USB cable into the Speedway Revolution USB Device port.

Insert the A side of the USB cable into the USB hub or PC.

USB A-B Cable
SPEEDWAY
REVOLUTION
+24V DC Locking Connector (powered via external power module)
10/100 BASE-T Ethernet (Power over Ethernet capable)
Default Restore (remove screw to access)
USB 1.1 Device Port
USB 1.1 Host Port
Console RJ45 Connector (RS232)
Multi-Purpose DE15 Connector (4 GPI, 4 GPO, RS232 Serial, +5V DC/200 mA)

2.6.3 USB Presence

Based on the behavior of other USB devices, when initially attaching the reader to the host, one 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 will begin. The reader can automatically start the application each time it boots, in which case the reader will respond as a USB device when either attached or at power-up (while attached).

2.6.4 Keyboard Emulation Program (hidkey)

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

```
/opt/ys/usb/hidkey
```

‘help’ passed as a command line option 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: 000.000.010400.009_126_835_2102

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. In order to proceed we'll need to start an application on the Windows PC host that accepts keyboard input, an editor (e.g. Notepad) or Spreadsheet (e.g. Excel) will work fine.

Next, the reader can be connected to the PC. Nothing will happen yet, only when the hidkey exe is started will the reader behave like a USB device and begin to enumerate.

From OSShell, type the following:

```
# echo "RFID That Just Works!" | /opt/ys/usb/hidkey
```

Before enumeration completes keyboard input must be directed to the Windows application.

If this is the first time the driver is used pop-ups identifying the device will flash up from the Windows system tray's USB icon.

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.

2.6.5 Disconnected Operation

Should the reader become disconnected the hidkey program will continue to consume and discard new data. As there is no pre-defined delimiter grouping reports, a partial report may 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 starting the hidkey program, if a USB connection is not present, hidkey will discard all input data until a connection is established and enumeration completes.

2.6.6 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 addional 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;
}
```

2.6.7 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 rather than raise an error or warning it is mapped to a substitute character, a '_'.

2.6.8 Performance

During testing it became clear that the 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 to be a function of the host's performance, the host's CPU load and the host application, so it is difficult to provide any guidance.

2.7 USB Flash Drive AutoRun

Speedway Revolution, as of Octane 4.4.0, allows 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 whether an application specific program is configured so that the application may check for files on the USB Flash Drive.

2.7.1 USB Flash Drive Preparation

USB Flash drive 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, as this ensures compatibility with VFAT.

On the flash drive create a directory X:\YourCompany\YourProduct. For example, Impinj uses \impinj\revolution to contain firmware upgrade files. It's best to use only alphanumeric characters and no spaces for file and directory names.

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

2.7.2 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 what 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
```

2.7.3 Example Storage Event Handler

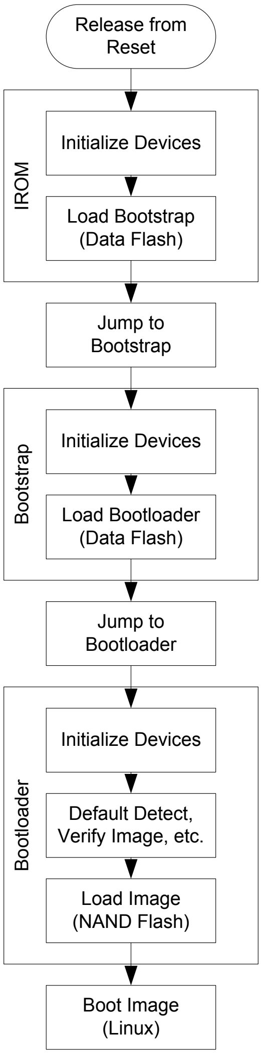
Here is an example of how the hidkey program can be used.

```
#!/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
```

3 Platform Boot Process

The Speedway Revolution 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 Revolution Boot Procedure



3.1 Bootloader

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

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

As there is only one default restore button (see Figure 2-1), the two different restores are invoked by pressing the default restore button at power on for different durations. As an aid, the Power LED is used to provide feedback for the duration of the button press (see Table 2-2). Table 3-2 provides the different durations required to achieve the two different default restore options.

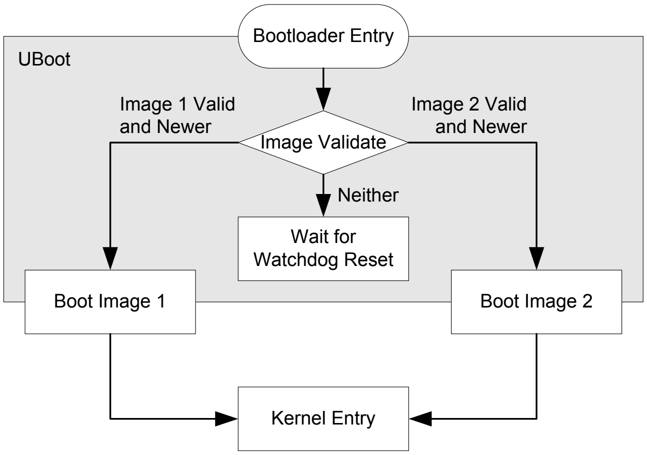
Table 3-2 Default Restore Button and LED Behavior

	Default Restore Button Press	Power LED
Configuration Default Restore	3 seconds	OFF, 3 seconds, blink RED
Factory Default Restore	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. Should neither image be valid, the bootloader will display an error message on the console and blink the Power LED red indefinitely (see Table 2-2). The watchdog remains enabled in this scenario and the bootloader simply waits for the watchdog reset. Figure 3-2 documents this process.

Figure 3-2 Image Selection and Boot



3.2 Operating System

Once 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 that start is mounter, which handles the upgrade scenarios. Once mounter has completed, the run-time reader applications are spawned, including any custom applications that have been loaded.

3.2.1 Mounter

Mounter is the application 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

3.2.1.1 Configuration and Factory Default Restore

If the bootloader detects that either a configuration or factory default restore has been requested, it conveys this information via image metadata that is outside the scope of this document. Once mounter recognizes that either a CDR or FDR has been requested, it is responsible for taking the appropriate actions. For a FDR, mounter will erase the SPP *and* the CAP partitions, resulting 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 on this process, see section 3.2.1.3.

3.2.1.2 Partition Copy

Upgrade image files may contain different combinations of the CAP and SOP partitions. Based on the content of the upgrade image file, certain partitions may 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 may 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

3.2.1.3 CAP Notification

During a firmware upgrade or a configuration default restore, the mounter application notifies any custom applications of the operation so that the app may 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 only required if the custom application wishes 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 existed. 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 (read-only) access the previous image and may copy any configuration or persistent files to the new partition prior to the custom application startup.

In addition to the command line arguments, several environment variables are made available to the upgrade application so that it may infer context. Table 3-4 documents the environment variables 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 will not be defined.

Table 3-4 Upgrade Application Environment Variables

Variable Name	Description	Example
`primary_sop_vsn`	Current SOP version	4.2.0.240
`primary_cap_vsn`	Current CAP version	1.0.2.0
`secondary_sop_vsn`	Previous SOP version	4.0.1.240
`secondary_cap_vsn`	Previous CAP version	1.0.1.0

As 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 may result in a failure of the reader to initialize.

- 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. As such, the application should not spawn 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. Thus all accesses must occur within the upgrade

application. This mount point is no longer available once the upgrade application terminates.

- As the upgrade application is called prior to the run-time reader applications, it must ensure prompt execution so that reader initialization may proceed. Lengthy upgrade application execution times (several minutes) may result in a reader reset and if the situation persists, a fallback to the secondary image.

An example upgrade application (in the form of a Bash shell script) appears in Figure 3-3.

Figure 3-3 Example Customer Upgrade Application

```
#!/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
```

3.2.2 Run-Time Applications

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

As the run-time applications are being spawned, any custom applications that are present on the reader are also started. Custom applications are started like any other run-time app and thus must adhere to the same restrictions. Specifically:

- No boot order may be assumed. Custom applications must be robust enough to handle scenarios whereby required resources are not yet available. Figure 3-4 shows how an application’s start script can wait for network availability before calling the app.

- Applications are launched as daemons, and should behave as such. Custom monitoring of application health may be implemented.

For more information about the spawning of custom applications, see section 4.3.3.

Figure 3-4 Delaying Custom Application for Network

```
#!/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 ...
```

4 Software Development Process

This section provides information about the Speedway Revolution SDK embedded developer resources and procedures. 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 Revolution reader uses MontaVista® Linux Professional Edition 5.0 (Pro) as its operating system. MontaVista® Pro 5.0 is an enhanced version of the open-source Linux 2.6.18 kernel that has advanced capabilities optimized for embedded applications. See http://www.mvista.com for more information. Any of the versions of Linux supported by MontaVista® Pro 5.0 may be used to run the scripts and utility programs described below. These include: Red Hat® Enterprise Linux 3.0 or 4.0, Solaris® 8.0 or 9.0, SUSE® Linux 10.1 Workstation, and SUSE® Linux Enterprise Server 9 service pack 2.

Note: MontaVista is a registered trademark of MontaVista Software, Inc., Red Hat is a registered trademark of Red Hat, Inc., Solaris is a registered trademark of Sun Microsystems, Inc., SUSE is a registered trademark of Novell Inc.

This document assumes users familiarity with the MontaVista® platform development kit (PDK) and with GDB, the GNU Project debugger. For more information on GDB, see http://www.gnu.org. An embedded software developer may develop a custom application using the Octane EDK and/or Impinj supplied LLRP libraries in C or C++ using any tools based on gcc version 4.2.0 cross-compiled for the Atmel AT91SAM9260 configured in little-endian mode.

4.2 Distribution

The components necessary to develop on-reader custom applications are distributed as RPMs. Each component is contained within a single RPM file, the list of components is likely to grow, here is a list of the current RPMs and their purpose:

- **revolution_ltk** – Speedway Revolution LLRP Toolkit (libraries and examples).
- **revolution_tools** – EDK tools, scripts and custom application development documentation.
- **revolution_examples** – Examples, specifically demonstrating the use of the EDK.
- **revolution_logging** – Configuration of the syslog utility.
- **revolution_web** – Mostly documentation infrastructure.
- **revolution_compiler** – Re-packaged Montavista cross-development tools (e.g. compiler, linker etc).
- **revolution_octaneSDK** – easy to use libraries allowing quick application development without the need to learn LLRP

As both an accelerator to getting started and as a reference development platform a Virtual Machine (VM) is also provided and maintained by Impinj. The Impinj Speedway Revolution EDK VM comes with all the necessary RPMs pre-installed. Updates to the content of each package will be accrued and released periodically.

The YUM package manager is used to manage updates to the packages, all of which are available from a publicly accessible YUM server.

4.3 Accessing the Linux Shell

The Speedway Revolution reader has its own CLI termed RFID-Shell, or RShell. End users may access this interface by using the RS-232 console port, or by connecting to the reader over the network via SSH or Telnet (if configured, see section 4.5.4).. Using this interface, the reader may be configured and system information displayed. 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 misused, access to the Linux shell (termed 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 in to the CAP upgrade image. For details about creating the CAP partition, please refer to the Speedway Revolution EDK, “Getting Started” link. To enable access to OSShell via RShell, the following file must be created within the CAP file system.

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

This file does much more than just enable access to OSShell, and specific details can be found in section 4.5. But for OSShell access, the following lines (or ‘stanza’) must appear within this file.

Figure 4-1 Enabling Access to OSShell

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

Once an upgrade image built with the CAP partition containing this file is loaded on to the reader, access to OSShell is enabled 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 (i.e. ‘password=’), then password checking is disabled.

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

4.3.1.1 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 on how to enable FTP, see section 4.5.4.

Note that as with the OSShell password, Impinj does not recommend deploying readers with this feature enabled.

4.3.2 Executing the Custom Application

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

To test the custom application:

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 as the application was run in the foreground, the command prompt will hang up.

```
# ./CustApp
```

5: Now enter several characters in the terminal program. If all is working correctly, those characters should now appear in both the terminal window and the SSH window where the application is running. Typing a customized character ('A' in the example code), should result in the customized string appearing in the SSH window. Figure 4-2 is a sample output from the SSH window.

6: Hit 'ESC' to quit the program.

Figure 4-2 Sample Example Application Output

```
root@SpeedwayR-00-00-BF:/cust#
root@SpeedwayR-00-00-BF:/cust# ./CustApp
abcdefghijklmnopqrstuvwxyz0123456789

Hello Friends!

The End!
root@SpeedwayR-00-00-BF:/cust#
```

4.3.3 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 (see section 3.2.2).

To configure a custom application to be spawned 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 spawned only once in the background and it is its responsibility 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 one such script. This example tests for the presence of a custom application, and if it is executable, spawns it. If the application ends, it logs a message and respawns the application after a delay to prevent spinning.

Figure 4-3 Example Customer Start Application

```
#!/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
```

4.3.4 Custom Application Library Dependencies

The Speedway Revolution firmware contains a minimal set of libraries required for the applications which it supports. It is conceivable that a custom application may require additional libraries to operate that are not included in the SOP. As part of the Speedway Revolution SDK, all of the supported MontaVista® Pro 5.0 libraries are included. Embedded applications that require additional library support may include these libraries in their CAP image and configure Linux via the `LD_LIBRARY_PATH` environment variable to scan the CAP partition when searching 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++.so` that is not included in the SOP. To handle this scenario, developers should include this library in their CAP file system, and 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 done in the sample start script in Figure 4-3).

4.4 Firmware Upgrade Procedure

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

4.4.1 Image Versioning Scheme

Each flash partition shown in Figure 2-8 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 a dot '.': ddd.ddd.ddd.ddd, where each field is a decimal number ranging from 0–255. The left-most field maps to the most significant byte of the internal uint32_t version number. For the purposes of upgrades, a larger integer number is considered a higher version. There is no other meaning to the version string. The custom application may use any versioning scheme as long as the version is 32 bits long, represented in the format shown above, and has a larger number to indicate a newer version.

4.4.2 RShell Upgrade Methods

Speedway Revolution provides two methods to upgrade the firmware: manual and automatic. The manual method is where upgrades are performed on a single reader by entering commands at the RShell interface. The automatic method is where a large network of readers may be upgraded automatically by managing an upgrade configuration file called a metafile. For the purposes of embedded software development, the manual method is the most useful. The automatic method is outside the scope of this document. If the automatic method is desirable, reference the Speedway Revolution Firmware Upgrade API documentation.

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, reference the 'Speedway Revolution 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
`config image removecap`	Completely remove the CAP partition. The CAP will not be 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

4.4.3 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 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 section 4.4.1). 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. Thus, when a CAP is formatted or removed, the actions occur on the secondary images and after the reader reset, mounter performs the appropriate partition copy procedures (see section 3.2.1.2). This is significant because performing these operations means the fallback image is no longer available once completed.

4.5 Reader Configuration

Speedway Revolution is designed to allow custom applications to control many of the features and services available by the reader. This customization is achieved using the `/cust/sys/reader.conf` configuration file that we first visited when discussing OSShell access (see section 4.3). This section documents the other capabilities of this file.

4.5.1 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 thus lines may be indented for clarity. The name portion of the NVP is case-sensitive and thus 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 Revolution 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.

Stanza Name	Description
`[PartnerFeatures]`	Configuration of specific reader features only available to select Impinj partners.
`[SoftwareFeatures]`	Configuration of network services and other software features.

4.5.2 Hardware Events

Speedway Revolution readers have both a USB device and USB host port. Upon detection of device insertion into one of the USB ports, the Octane firmware has mechanisms whereby custom applications can be notified of the event. Each notification type has different characteristics, the details of which are found 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

4.5.2.1 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. On detection of a mass storage device insertion or removal from the USB host port, the script will be invoked with specific command line arguments and environment variables to convey what event has occurred. The callback script can then take whatever action is required (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 Revolution 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/`).

4.5.3 Partner Features

Some Speedway Revolution 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 regarding specific features and how they are enabled, please contact your Impinj Sales Representative.

4.5.4 Software Features

Speedway Revolution 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 may customize those services using the reader configuration file.

Table 4-6 lists the available network services on the reader, and whether they may be customized. The NVP name is the case-sensitive line that must appear within the `SoftwareFeatures` stanza of the reader configuration file to customize the service.

Table 4-6 Customizable Network Services

Service	Default	NVP Name
DNS-SD (HTTP)	On	StartDnsSDHttp
DNS-SD (LLRP)	On	StartDnsSDLlrp
FTP	Off	StartFTP
LLA	On	StartLLA
mDNS	On	StartmDNS
NTP	On	StartNTP
SNMP	On	DefaultStartSnmp
Telnet	On	StartTelnet
FTP Upgrade	On	AllowUpgradeFTP
HTTP Upgrade	On	AllowUpgradeHTTP
Web Upgrade	On	AllowUpgradeWeb
TFTP Upgrade	On	AllowUpgradeTFTP
HTTP	On	StartHTTP

4.5.5 Example Configuration File

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.

Figure 4-4 Example Reader Configuration File

```
#
# 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 disabled once deployed.
    password=developer

[HardwareEvents]
    # Invoke special script on a mass-storage device
    OnMassStorageEvent=/cust/bin/sdAction.sh

[PartnerFeatures]
    # Enable some example reader feature here

[SoftwareFeatures]
    # Turn off telnet for security purposes
    StartTelnet=no
    # Turn on FTP for development (TODO: turn off)
    StartFTP=yes
    # Turn off NTP and rely on the RTC for time
    StartNTP=no
    # Do not advertise any reader services
    StartDnsSDHttp=no
    StartDnsSDLlrp=no
```

5 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

Notices:

The information contained in this document is confidential and proprietary to Impinj, Inc. This document is conditionally issued, and neither receipt nor possession hereof confers or transfers any right in, or license to, use the subject matter of any drawings, design, or technical information contained herein, nor any right to reproduce or disclose any part of the contents hereof, without the prior written consent of Impinj and the authorized recipient hereof.

Impinj reserves the right to change its products and services at any time without notice.

Impinj assumes no responsibility for customer product design or for infringement of patents and/or the rights of third parties, which may result from assistance provided by Impinj. No representation of warranty is given and no liability is assumed by Impinj with respect to accuracy or use of such information.

These products are not designed for use in life support appliances, devices, or systems where malfunction can reasonably be expected to result in personal injury.

The "Powered by Impinj" shield is your assurance of RFID integrity