Raccoon System

Embedded Software Integration

SRS

(Software Requirements Specification)

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Rev** | **Changes** | **Author** | **Date** |
| Rev1 | Initial version | Shlomit Morad | 20.7.2014 |
| Rev2 | Adding parameters to be hidden | Shlomit Morad | 6.8.2014 |
| Rev3 | Refining the logging requirements | Shlomit Morad | 6.10.2014 |

Table of Content

[1 Introduction 2](#_Toc402106185)

[1.1 Tag Concepts 2](#_Toc402106186)

[1.2 Raccoon System Architecture 2](#_Toc402106187)

[1.3 Raccoon Software Architecture 3](#_Toc402106188)

[1.4 Acronyms & Abbreviations 3](#_Toc402106189)

[2 Printer Software Integration 4](#_Toc402106190)

[2.1 System Init 4](#_Toc402106191)

[2.2 Cartridge Detection 4](#_Toc402106192)

[2.3 Cartridge Insertion 5](#_Toc402106193)

[2.4 Consumption Update 5](#_Toc402106194)

[2.5 Remount 6](#_Toc402106195)

[2.6 Logging 6](#_Toc402106196)

[3 Raccoon Constants 7](#_Toc402106197)

# Introduction

The Raccoon system introduces a mechanism that identifies Stratasys materials and alerts upon usage of non-genuine material and/or usage the material inappropriately. It is composed of 2 parts: the burning station located in the material factory & burns Identification Tags (IDTs) on new cartridges, and the printer which authenticates the IDTs, reads the materials data and updates the tags with the remaining material.

## Tag Concepts

The tag attached to the cartridge is a highly secured module (chip) which is commonly used when cloning must be prevented. The chip provides FIPS compliant services such as: random key-pair generation, cryptographic algorithms, digital signature and random number generation.

The data burned on the IDT consists of material data required for identification & usage control, manufacturing details required for troubleshooting if the cartridge is found faulty, and other data items required for cartridge authentication:

* Version number (of material info)
* Material information:
  + Material name
  + Material batch number
  + Production date & time
  + Material expiration date
  + Full cartridge weight
* The IDT chip Serial Number
* The PubKC (cartridge public key)

## Raccoon System Architecture

On the printer, the Raccoon hardware includes up to 4 Front Connection Boards (FCBs), one per cabinet. Each FCB controls 8 cartridges. The first FCB is connected to the printer computer; the next FCBs are linked to each other in a chain using I2C.

Raccoon HW

Printer Computer

IDT

Interface board

Housing

Cartridge

Housing

Cartridge

I2C

Ethernet

Housing

Cartridge

I2C x8

I2C

I2C x8

## Raccoon Software Architecture

The chip vendor (Inside Secure or Atmel) provides an API library (IDT-API). This library forces the basic architecture: a Raccoon server (IDT-Srv) that uses the IDT-API and provides the printer software with all Raccoon services, and a communication layer which is called by the IDT-API for delivering messages to & from the IDT via the IDT-HW.

This communication layer delivers commands to a specific IDT (using its I2C address). Since the printer has multiple cartridges with the same I2C address, the Raccoon hardware shall provide a mechanism for interfacing each cartridge individually.

In order to communicate with an IDT, the hardware shall be able to detect it when the cartridge is inserted (and thus identifies when a cartridge is removed). This function shall replace the In-Place micro-switches that detect the cartridge insertion today.

The IDT-HW might change: a new hardware might replace the FCB, the FCB might use different components, etc. In order to increase the software maintainability, the communication layer shall be divided into 3 layers:

(1) IDT-Comm converts the IDT-API into a generic hardware interface

(2) IDT-HW-Inf converts the generic hardware interface into specific hardware interface

(3) IDT-HW Driver

The software architecture (including its internal interfaces) is illustrated in the figure below:

|  |  |
| --- | --- |
| EM-SW (Printer) | |
| IDT-Srv | |
| Multi-channel control | IDT-API |
| IDT-Comm |
| Single IDT commands |
| RS-HW-Inf | |
| FCB Driver | |
| FCBs & IDTs | |

## Acronyms & Abbreviations

|  |  |  |
| --- | --- | --- |
| EM-SW | - | Embedded (printer) SW |
| IDT | - | ID Tag |
| IDT-Srv | - | IDT services requires by the application |
| RS-EMS | - | Raccoon System – Embedded Software – All Raccoon software elements that serve the printer (production + testing tools) |

# Printer Software Integration

The RS-EMS provides the printer software (EM-SW) with all services required for controlling the cartridges via their IDTs. It is the EM-SW responsibility to use these services as necessary.

This section specifies the states and conditions in which these services shall be called, the printer data required by these services and the processing required on data obtained by these services.

The IDT-Srv provides the following services:

* Set a log callback function.
* IDT-HW Init: initializes the IDT-HW and returns how many cartridges are installed.
* Remove Cartridge: indicates that the cartridge was removed.
* Get In-Place Status: returns a 32 bit status (bit per cartridge) of all cartridges.
* Authenticate Cartridge: performs the complete authentication procedure for a single IDT while obtaining the cartridge data.
* Update Consumption: performs the consumption update process per cartridge.

## System Init

When the EM-SW starts, it shall first set the log callback function. Then it shall call the HW Init function that initializes the hardware (FCBs).

When the software is started, it shall detect and authenticate all the inserted cartridges. Since this function is used very frequently when the system is up & running, the EM-SW shall initialize the cartridge detection mechanism (see the next section) in a way that upon startup, all the inserted cartridges will be detected and authenticated right away.

## Cartridge Detection

In order to detect a cartridge insertion/removal, the EM-SW must check the In-Place status held by the FCBs, very frequently. Every 500 msec, it shall call the Get In-Place Status function and compare the returned 32 bit status with the status of the previous call (initial value is 0). For each bit whose value was changed, the software shall behave as follows:

* Calculate the absolute cartridge ordinal number (bit number in the status)
* If the bit value has changed from 0 to 1 – a cartridge was inserted: the software shall execute the cartridge insertion procedure for that cartridge (see the next section). If succeeded, it shall turn on the cartridge in-place indication on the “Actuators & Sensors” screen.
* If the bit value has changed from 1 to 0 – a cartridge was removed: the software shall call the remove function. This function informs the IDT-Srv that there is no cartridge in that location, and subsequent consumption updates shall be considered as invalid. Then, it shall turn off the cartridge in-place indication on the “Actuators & Sensors” screen.

**Note**: when the software compares the current & previous statuses, it **must** handle **all** differences before reading the next status. Since multiple insertions at the same time is a very rare situation (after system startup), there is **no** need to implement **concurrency** for accelerating this scenario.

## Cartridge Insertion

When the cartridge detection mechanism detects a newly inserted cartridge it shall first call the Authenticate Cartridge function. This function verifies that the cartridge is genuine and it returns the material data and the cartridge weight (how much material it contains).

When the authentication succeeds, it returns the material data as a binary stream which starts with a structure version number (1 byte). The data shall be converted into the appropriate structure and shall be transferred to the cartridge enabling process, as exists today. The cartridge weight shall be transferred to the consumption update mechanism (see the next section).

If it fails or the cartridge weight is 0, the software shall disable this cartridge, as it does today.

## Consumption Update

The consumption update mechanism runs constantly during the printing process. It calculates the consumption of each material and updates the remaining material on the appropriate IDTs.

The following consumption update procedure shall be executed per cartridge, and all the counters it defines shall be allocated and managed per cartridge. The formulas use constant names; their values and location are specified in the next section:

* After (or while) converting the slice bitmaps into the firing data, the software shall calculate how many drops will be consumed by this cartridge in printing this pass. It shall convert the # of drops into weight, using the following formula, and add it to an internal counter:

Cartridge consumption = Ndrops (# of drops) X Wdrop (the drop weight in milligrams)

Wdrop is an integer constant – but its value is different per print mode.

* The firing data is calculated per Travel (a full head movement along the X axis) or per Pass (2 travels, back and forth). When the software updates the consumption counter, it shall also update a travel/pass counter - number of Travels/Passes since last consumption update.
* Before the purge process starts, the software shall accumulate the estimated consumption to the internal consumption counter.

Purge consumption = Wpurge (in milligrams)

Wpurge is an integer constant.

* After updating the internal consumption counter and the slice counter, the software shall:
  + Compare the consumption counter with the consumption threshold ‘Wthreshold’.
  + Compare the travel/pass counter with the travel/Pass threshold ‘Tthreshold’ / ‘Pthreshold’.
  + If one of the counters exceeds its threshold, the software shall:
    - Call the consumption update function to that cartridge.
    - Reset the consumption counter and the slice counter.

The update consumption function returns the updated weight. If this value is 0, the software shall disable this cartridge, as it does today. From this point, the cartridge is considered as removed / not-exist.

## Remount

From time to time (or under special conditions), the EM-SW asks to refresh the cartridges status. This refresh procedure shall recheck which cartridges are inserted, authenticate them and read the burned resin info.

Since the Raccoon cartridge detection is based on frequent comparison between the current & the previous in-place status, the remount request shall only initialize the previous in-place status to 0. The next in-place check (within the coming 500 msec) will automatically identify the cartridges, authenticate them and refresh their resin info.

## Python Engine

Since Python scripts can easily change significant properties, including some that affect the resin consumption, the ability to execute Python scripts (via a keyboard shortcut) shall be hidden from the user. It will be enabled for Stratasys people after inserting a special dongle with ID=1001.

## Logging

### Application Level Logging

The printer software shall write a log message for 6 events:

* System initialization: the message shall specify the number of cartridges returned by the InitHW function.
* An in-place status was read: the message shall specify the returned status.
* A cartridge was inserted and successfully authenticated: the message shall contain the cartridge #, all tag details (+ the material full name) and its consumption counter.
* A cartridge was removed: the message shall specify the cartridge # & the material name.
* An error was occurred: the message shall contain the cartridge #, the material name, the failed operation and the error code.
* Consumption update: the message shall contain the cartridge #, the consumption to be decremented and the new value.

### Raccoon Domain Tags

In order to differentiate between messages written in very low frequency (InitHW & cartridge authentication) and messages written in very high frequency (in-place & consumption update), the EM-SW shall use 2 Raccoon-specific tags: Raccoon Control and Raccoon In-Action.

### RS\_EMS Log Messages

The RS-EMS has no logging system of its own. It writes its log messages into a log function supplied by the EM-SW (during initialization). The log message contains plain text.

The EM-SW shall add the appropriate domain tag to indicate Raccoon messages (for filtering purposes).

# Raccoon Constants

The formulas specified above use several constants. These constants might vary between printer models, and they might require further adjustments. Thus, they shall be defined as constants to be easily modified.

**Pay attention: the Wdrop unit is nanograms (10 \*\*-9) and the Wpurge unit is Grams, while the consumption units are milligrams (10 \*\* -3)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Printer** | **Mode** | **Wdrop** |  | **Printer** | **Wpurge per head** |
| Alaris (just in case) | HQ | 35 ng | Alaris | 2.5 grams |
| HS | 70 ng | XL Phase 2 | 2.5 grams |
| XL Phase 2 | HQ | 90 ng | Triplex | 2.5 grams |
| HS | 90 ng | Keshet | 2.5 grams |
| Triplex | DM | 78 ng |  | |
| HQ | 78 ng |
| HS | 78 ng |
| Keshet | High-Mix | 78 ng | Wthreshold | 5 gr = 5000 milligrams |
| HQ | 78 ng | Tthreshold | 10 travels |
| HS | 78 ng | Pthreshold | 5 passes |

Holding these constants in a configuration file is risky, since a hacker can change them in a way that bypasses the entire Raccoon protection. It is recommended to store them in an encrypted file. If the use of encrypted file is not feasible, these constants shall be embedded in the code.