**Raccoon**

**(Embedded side)**

**Software Design Review**

****

Written by: Slava Chuhovich

August 2014

Table of Contents

[1. Introduction 3](#_Toc395648425)

[1.1 Background 3](#_Toc395648426)

[1.2 Major design goals 3](#_Toc395648427)

[1.3 Abbreviations and acronyms 3](#_Toc395648428)

[2. Architecture 4](#_Toc395648429)

[2.1 Overview 4](#_Toc395648430)

[2.2 Operation sequences 4](#_Toc395648431)

[2.3 Special requirements 5](#_Toc395648432)

[2.4 User interface 5](#_Toc395648433)

[3. Software implementation 6](#_Toc395648434)

[3.1 Implementation 6](#_Toc395648435)

[3.2 Consumption calculation 7](#_Toc395648436)

[3.3 Container module explanation (prior Raccoon implementation) 7](#_Toc395648437)

[3.4 Logging 8](#_Toc395648438)

[3.5 Parameters 8](#_Toc395648439)

[4. Development 9](#_Toc395648440)

[4.1 Limitations & Risks 9](#_Toc395648441)

[4.2 Unclear issues 9](#_Toc395648442)

[4.3 Development stages 9](#_Toc395648443)

# Introduction

## Background

The Raccoon system introduces a mechanism that identifies Stratasys materials and alerts upon usage of non-genuine material and/or inappropriate material usage. Material identification and authentication is performed using ID Tags (IDT), which is built into each material cartridge.

IDT 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. Data burnt into IDT consists of material details required for identification & usage control, manufacturing details required for troubleshooting if the cartridge is found faulty, and other data items required for cartridge authentication.

The Raccoon system comes to replace currently available RFID mechanism and provide expanded and more secure functionality.

## Major design goals

1. Remove RFID software layer, which is currently used for material identification and authentication.
2. Implement cartridge authentication with a new API provided by Giga.
3. Remove cartridge “in-place” implementation with micro switches.
4. Implement “in-place” functionality using Giga’s API.
5. Implement material consumption update, which is calculated from actual printed data.
6. Implement additional parameters, UI changes and design modification that may be required.

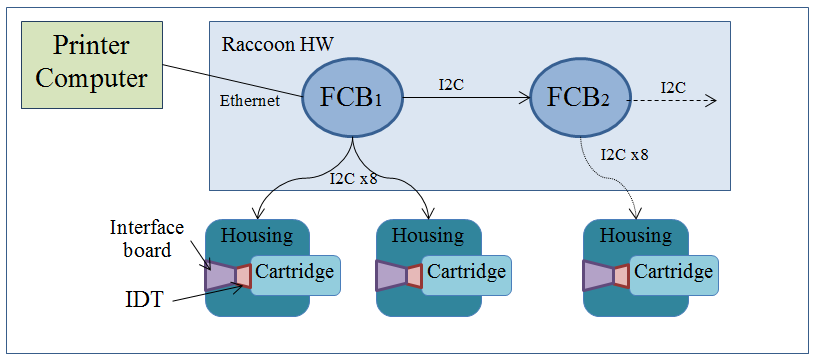
## Abbreviations and acronyms

|  |  |
| --- | --- |
| **Abbreviation** | **Description** |
| EM-SW | Embedded (printer control) software |
| IDT | Identification tag |
| RS-EMS | Raccoon system elements for printer service |
| API | Application Programing Interface |
| FCB | Front Connection Board |
| PM | Parameters Manager |

# Architecture

## Overview

Raccoon hardware includes up to 4 electronic Front Connection Boards (FCBs), one per cabinet. Each FCB controls 8 material cartridges. First FCB is connected to the printer’s embedded computer using Ethernet communication and linked to following FCBs in a chain using I²C.



When material cartridge is inserted into the machine, IDT of this cartridge performs mechanical connection with the Interface Board, which is located in the cartridge’s housing. The FCB, which is responsible for this Interface Board senses the connection and authentication process may commence.

## Operation sequences

**System Power-up**

1. Load Giga’s DLL.
2. Initialize software API.
3. Initialize hardware and start communication.
4. Start cartridges detection and authentication module (thread).

**Detection**

1. Call API method, which returns the ‘in-place’ status of each cartridge.
2. If cartridge was removed, call API method for cartridge removal.
3. If cartridge was inserted, start authentication sequence.

**Authentication**

1. Call API method, which authenticates the cartridge.
2. If failed, block the cartridge with existing mechanism and logic.
3. If succeeded, read the cartridge data and start existing cartridge enabling state machine.

**Consumption update (during print only)**

1. Each printed slice, calculate the material weight, which is required from specific cartridge, by analyzing the printed data.
2. Before the start of each purge sequence, add the purge weight (constant value) to the material consumption. Distribution between cartridges – TBD.
3. If consumption calculation of specific cartridge exceeds 5 Gr or it is more than 5 slices from the last update, call API method to update the consumption of this cartridge.
4. If a remaining weight of a specific cartridge after consumption update equals 0, mark this cartridge as removed.

## Special requirements

1. Raccoon system constants may vary between machine types and operation modes.
2. System constants must be protected under encrypted file – TBD.
3. Raccoon predefined parameters must be hidden from Parameters Manager. Exact list – TBD.
4. Opening python console must be disabled in Release version.

## User interface

No special requirements.

# Software implementation

## Implementation

1. Load IDT-SRV DLL with EM-SW application startup.
2. Implement hardware initialization by calling InitHW method at system power-up.
3. Implement Giga API initialization by calling InitApi method at system power-up.
4. Redesign CContainer class to remove all RF related logic and state machines.
5. Create new CRaccoon module as system thread, which executes every 500 mSec. The module will be created as a singleton object.
6. Use GetInPlaceStatus API call in CRaccoon thread to read the insertion status of all cartridges and store the value as a class member. Each bit in a 32 bit returned value indicates the insertion status of each cartridge. Store current and previous values.
7. Implement observer design pattern in CRaccoon class to notify the required modules, such as CContainer or CQSingleContainer, if an “in-place” status has been changed for each cartridge.
8. Implement CRaccoon::Authenticate method (thread safe), which uses AuthenticateCartridge API call. This method will be called from each CQSingleContainer object when authentication is required. The received data will be stored in each CQSingleContainer object as a class member.
9. Implement CRaccoon::RemoveCartridge method, which will use RemoveCartridge API call to mark specific cartridge as “removed”. It will be called from CContainer object.
10. Implement CRaccoon::UpdateConsumption method, which will use UpdateConsumption API call. It will be called from CContainer object for each CQSingleContainer object.
11. Implement CLayerProcess::CalculateMaterialConsumption method, which will be called from CLayerProcess::GenerateOnePassData and will return the material consumption in grams for each material type in printed bitmap. If reached required criteria (See “Consumption calculation” section), it will update the material consumption of a specific cartridge through CContainer module. If a returned weight value by consumption updating method equals 0, this cartridge will be blocked (marked as “removed” ?) by CContainer module.
12. Implement consumption calculation and call consumption updating methods through CContainer module in CMachineSequencer::PurgeSequence method. (See “Consumption calculation” section).
13. Integrate CRaccoon module with existing cartridge enabling logic.
14. Implement secure file for consumption calculation constants – TBD.
15. Implement Raccoon system related parameters in PM and mark them as hidden.
16. Implement Raccoon system related logging entries.

## Consumption calculation

1. Consumption update will be done using CRaccoon::UpdateConsumption method, which will be called from CContainer objet for each CQSingleContainer.
2. Previous material weight and previous slice number at consumption updating time will be stored in each CQSingleContainer object as class members.
3. If consumed material exceeded 5 grams (stored constant value) or passed 5 slices (stored constant value) since the last update for the specific cartridge, call consumption updating methods.
4. Printed data consumption calculations will be done by the following formula:

Consumption = *num\_of\_drops* \* *single\_drop\_weight* (mGr)

1. Purge consumption calculations will be done by the following formula:

Purge consumption *= purge\_weight* (mGr)

## Container module explanation (prior Raccoon implementation)

Container is a module in embedded application, which is responsible for material (model and support) and waste cartridges operation and logic. Its **current** implementation and the main tasks of the module are:

1. “In place” (micro-switches) functionality for all tanks.
2. Load cell weights for all tanks.
3. RFID state machine and decision making for each tank.
4. “Hot swap” mechanism.
5. Communication with OCB regarding all tanks data (weights, in-places, RF …)
6. Tanks operation modes and activation for each print-block chamber.
7. Parameters updates regarding tanks operation and relations (PumpTankRelation, ChamberTankRelation …)
8. User notification (pop-ups) RFID messages.

Container module consists of a several main classes:

1. CQSingleContainer – Functionality and data for a single cartridge of any type.
2. CContainerBase – Base class for the module.
3. CContainer – Inherits from CContainerBase. Functionality when working with hardware.
4. CContainerDummy – Inherits from CContainerBase. Functionality for Emulation.

CContainer object holds a list of CQSingleContainer objects, which represents each cartridge in the system. All functionality and data collection, such as: weights and in-places receiving from the OCB, “hot-swap” between the cartridges in each pair, RFID state machine for each cartridge…. Is performed by iterating (looping) through the list of Single Containers and performing required logic.

CContainer object runs as a separate thread and sampled each 200 msec.

CContainer

CQSingleContainer

CQSingleContainer

CQSingleContainer

…

CQSingleContainer

CQSingleContainer

## Logging

Following scenarios will be written to application log file:

1. Cartridge insertion and removal.
2. Cartridge authentication results and received material values.
3. Consumption update and its values.
4. API calls return statuses (during development).
5. Any additional messages and data – TBD.

## Parameters

1. Needed parameters for the Raccoon system will be added to PM, but will be marked as hidden. Exact parameters list – TBD.
2. System constants will be added to additional secured file. File definition, security type, loading time, structure … - TBD.

# Development

## Limitations & Risks

1. Prior to starting Raccoon integration into EM-SW, Giga’s software API must be available and functioning.

## Unclear issues

1. How purge weight consumption is distributed between the cartridges? Divided equally by number of active cartridges?
2. When reaching 0 at consumption update “Remove” API call should be used? If so, the cartridge is still physically located in the housing, but not at the UI, which may confuse the user.
3. Securing constants in the encrypted file – it’s not clear what security and signature to use.
4. Suggesting disabling python console in Release version entirely.
5. List of exact parameters, which must be hidden in parameters manager.
6. How material cartridge parameters are received from AuthenticateCartridge API method? Why not fill and return a structure?

## Development stages

1. All development will be done in Triplex 58.1 release branch.
2. At initial stage Giga’s API will be integrated into EM-SW and logic implemented.
3. At next stage more advanced development and testing will be done at development machine using actual hardware, FCB boards and IDTs.
4. At final stage the testing will be done on actual Triplex machine using full hardware.