Raccoon System - EMbedded Software

(RS-EMS)

Interfaces Specification

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Rev** | **Changes** | **Author** | **Date** |
| Rev1 | Initial version | Shlomit Morad | 17.8.2014 |
| Rev2 |  |  |  |

Table of Content

[1 Introduction 3](#_Toc396377772)

[1.1 Acronyms & Abbreviations 3](#_Toc396377773)

[1.2 IDD - IDentification Data 3](#_Toc396377774)

[1.3 Software Architecture 3](#_Toc396377775)

[1.4 Deliverables 4](#_Toc396377776)

[2 IDT-Srv Services 5](#_Toc396377777)

[2.1 IDT-API Init 5](#_Toc396377778)

[2.2 IDT-HW Init 5](#_Toc396377779)

[2.3 Set PubKS 5](#_Toc396377780)

[2.4 Get In-Place Status 5](#_Toc396377781)

[2.5 Authenticate Cartridge 5](#_Toc396377782)

[2.6 Remove Cartridge 6](#_Toc396377783)

[2.7 Update Consumption 6](#_Toc396377784)

[3 IDT-HW-Inf 7](#_Toc396377785)

[3.1 Init 7](#_Toc396377786)

[3.2 Get In-Place Status 7](#_Toc396377787)

[3.3 Select Channel 7](#_Toc396377788)

[3.4 Send/Receive Data 8](#_Toc396377789)

[4 Error Codes 9](#_Toc396377790)

# Introduction

The Raccoon project introduces the mechanism that identifies Stratasys materials & alerting upon using non-genuine material and/or using the material inappropriately.

The system is composed of 2 parts located in different physical locations: a burning station which is located in the material factory and burns tags on new cartridges, and the printer software which authenticates the tag, reads the material data and updates the tag with the remaining material.

The software that implements the Raccoon functions on the printer is the RS-EMS.

## Acronyms & Abbreviations

|  |  |  |
| --- | --- | --- |
| IDD | - | Tag ID (identification) Data |
| IDT | - | ID (identification) Tag |
| IDT-HW | - | General term for the IDT interfacing HW (FCB is the specific implementation for PolyJey printers) |
| IDT-HW-Inf | - | Raccoon hardware interface library |
| IDT-Srv | - | The main library that provides IDT services to the application |
| PrvKC - PubKC | - | Cartridge Private & Public Keys |
| RS-EMS | - | Raccoon System – Embedded Software – All Raccoon software elements that serve the printer (production + testing tools) |

## IDD - IDentification Data

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

Since each system may use different structure and since the structure may change in the future, the first byte of the IDD is a version number (an ordinal number, starting with 1). The IDD is composed of the following items:

* Version Number
* Material information
* The tag chip Serial Number
* The Cartridge public key – in full key format

The following structure defines the IDD (.h):

## Software Architecture

The interface library provided by the chip vendor dictates the software architecture: It provides services to be used by the IDT-Srv and it uses a communication API for delivering data to/from the IDT via the IDT-HW.

Since the hardware used for interfacing the IDTs may change from system to system, a hardware interface layer has been added: IDT-HW-Inf. This layer converts a generic interface into specific hardware interface, and should be changed for any new/modified hardware.

If a printer has multiple cartridges, the IDT-HW shall manage all cartridges and their IDTs. The software that controls this hardware must be able to identify IDTs or switch between IDTs. The multi-channel functions & parameters were added to the generic hardware interface. When using RS-EMS on a printer with a single IDT, these multi-channel items in the IDT-HW-Inf should be replaced with empty functions & constant parameters.

The figure below shows the complete RS-EMS architecture:

|  |  |
| --- | --- |
| Printer SW | |
| IDT-Srv | |
| Multi-channel control | IDT-API |
| Comm. Layer |
| Single IDT commands |
| RS-HW-Inf | |
| IDT (proprietary) HW | |
| IDTs | |

The **Green** components are the generic services to be used by the application.

The **Purple** component shall be modified when the IDT-HW is changing.

## Deliverables

The main library of the RS-EMS is IDT-Srv.dll. It uses RSCommon.dll, log4cplus.dll, TinyXml.dll, & VaultIC\_API\_1XX.dll (the chip API provided by the chip vendor + the general comm. layer).

The program that uses the RS-EMS should include .h files

The IDT-HW-Inf layer is provided as a dll and its source code.

# IDT-Srv Services

The IDT-Srv provides the printer application with the main IDT services: authenticate & update consumption. The context in which these services are used is under the responsibility of the printer software. Some of these services shall be used while burning the IDTs.

The IDT-Srv services are provided via the interface class IMaterialMonitor.

All functions return 0 as operation succeeded. The error codes are attached in Section 4:

## IDT-API Init

This function loads & initializes the IDT-API library.

int **InitApi** ();

## IDT-HW Init

This function initializes the IDT-HW. It shall returns how many cartridges are installed.

int **InitHW** (unsigned char \*pucCartridgesCount) = 0;

|  |  |
| --- | --- |
| pucCartridgesCount | Number of installed cartridges |

## Set PubKS

This function initializes the Public Key of Stratasys.

int **SetPubKS** (const unsigned char \*aucPubKS);

|  |  |
| --- | --- |
| aucPubKS | Stratasys public key |

## Get In-Place Status

This function provides a status of all connected cartridges.

int **GetInPlaceStatus** (int \*pStatus) = 0;

|  |  |
| --- | --- |
| pStatus | A 32-bit status - bit per cartridge - 1 means cartridge is in-place |

## Authenticate Cartridge

This function performs the entire authentication procedure for a single cartridge.

int **AuthenticateCartridge** (unsigned char ucCartridgeNum, const unsigned char \*aucPubKS, unsigned char \*aucIdd, unsigned short \*usIddLength, unsigned int \*uiCurrentVolume);

Version is missing

|  |  |
| --- | --- |
| ucCartridgeNum | The cartridge number [0..31] |
| aucPubKS | PubKS, if the function SetPubKS was skipped. When value is NULL, the preset key is used |
| aucIdd | The tag identification data (output parameter). This is a byte stream of the IDD, starting with 1 byte with the structure version number. |
| usIddLength | The tag identification data length (output parameter) |
| uiCurrentVolume | The current material weight (output parameter) |

The tag chip keeps the data as a byte stream. When the tag is burned, the material information is serialized; when the printer authenticates the IDT, it gets the byte steam and it is its responsibility to deserialize it into the correct structure.

## Remove Cartridge

This function indicates that there is no authenticated cartridge (it resets all cartridge information kept by the library).

int **RemoveCartridge** (unsigned char ucCartridgeNum);

|  |  |
| --- | --- |
| ucCartridgeNum | The cartridge number [0..31] |

## Update Consumption

This function performs the consumption update process per cartridge

int **UpdateConsumption** (unsigned char ucCartridgeNum, unsigned int uiComsumption, unsigned int \*uiNewVolume)

|  |  |
| --- | --- |
| ucCartridgeNum | The cartridge number [0..31] |
| uiComsumption | The material consumption to be decremented |
| uiNewVolume | Cartridge new weight |

# IDT-HW-Inf

The IDT-HW-Inf defines a generic API for communicating with the IDTs.

The DLL provided by the RS-EMS implements this API for communication via FCB (used in the PolyJet printers). When the IDTs are communicating via a different hardware, this API should be implemented accordingly. If the system uses a single IDT, the multi-channel commands should be replaced with empty functions & multi-channel parameters should be replaced with constants.

The sections below describe the interface of this layer. It describes the parameters as used for the FCB, and how it should be modified when only one IDT is used.

## Library Init

The function loads the FCB interface library.

int **InitIdtLibrary** ();

## Hardware Init

The hardware initialization function initializes IDT-HW. When FCBs are used, it initializes all installed FCBs and all the attached IDTs.

int **IdtInit** (int \*cartridgesCount);

|  |  |
| --- | --- |
| cartridgesCount | Number of cartridges [8, 16, 24, 32] - number of FCBs x 8 |

## Get In-Place Status

The get in-place status function returns a bitmap of all inserted cartridges. The FCBs maintain a register that holds the presence status (in-place) of the cartridges under its control. This function reads all statuses from all installed FCBs and merges them into a single 32bit status.

int **IdtGetInPlaceStatus** (int \*status);

## Select Channel

The FCB design allows access to a single IDT at a time. Before any send/receive operation (or a sequence of such operations) with a specific IDT, the IDT-Srv must select the cartridge channel.

int **IdtSelectChannel** (int cartridgeNum);

|  |  |
| --- | --- |
| cartridgeNum | The cartridge number [0..31] |

If a single IDT is used, this function should do emptied – it should return 0 constantly.

## Send/Receive Data

The functions SendData and ReceiveData shall receive the data buffer and its length. They shall delegate the operation to the FCB driver by calling the corresponding FCB functions.

int **IdtSendData** (int bufferSize, char \*buffer)

int **IdtReceiveData** (int bufferSize, char \*buffer)

|  |  |
| --- | --- |
| bufferSize | The buffer size |
| Buffer | The buffer to be sent of a buffer for the received data |

# Error Codes

The following list of error codes is not final. An ENUM definition will replace this table:

|  |  |  |
| --- | --- | --- |
| **Method** | **Error Code** | **Description** |
| InitApi | e200 | module or method not found |
| e110 | module or method not found |
| InitHw | e205 | module or method not found |
| GetInPlaceStatus | e206 | module or method not found |
| AuthenticateCartridge | e110 | module not initialized |
| e207 | switching failed |
| 6581 | memory error |
| 6981 | transaction not allowed |
| 6a82 | file not found |
| 6700 | input data expected |
| 6985 | persmission denied |
| e300 | invalid certificate signature |
| e301 | invalid token signature |
| f000 | internal exception caught |
| UpdateConsumption | e110 | module not initialized |
| e207 | switching failed |
| e400 | cartridge not authenticated |
| e401 | invalid volume signature |
| e402 | cannot consume |
| 6a80 | counter underflow |