# Developers Guide for RaccoonSafenet

This guide explains how to build and use RaccoonSafenet project

SVN:

<http://srv-ire-svn/svn/ControlSW/Trunk/Raccoon/Raccoon-Safenet>

Path on disk:

C:\svn\Raccoon\Raccoon-Safenet

Inside Secure delivery: not checked with real tags.

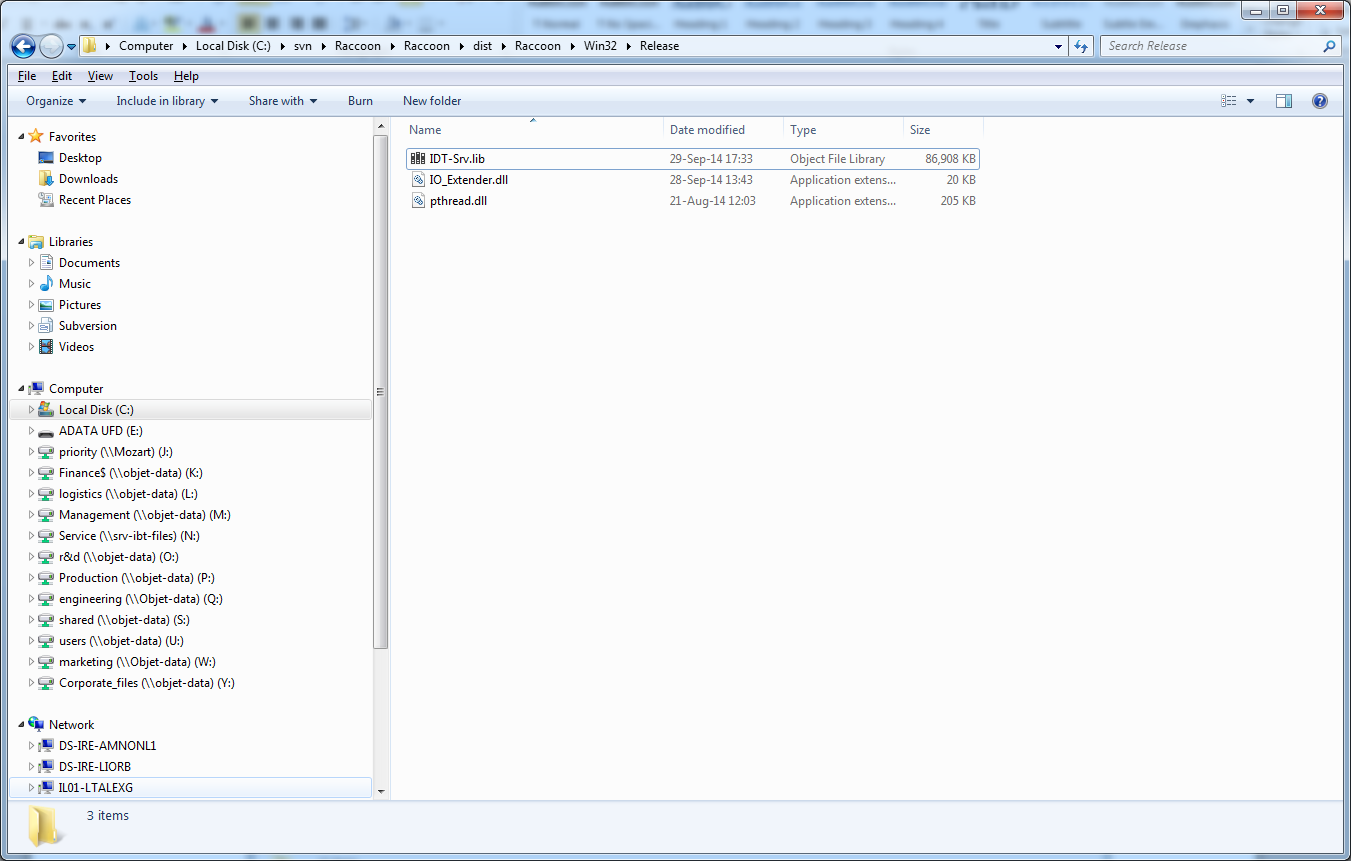
## Building the library for windows

Raccoon software is delivered in three different packages:

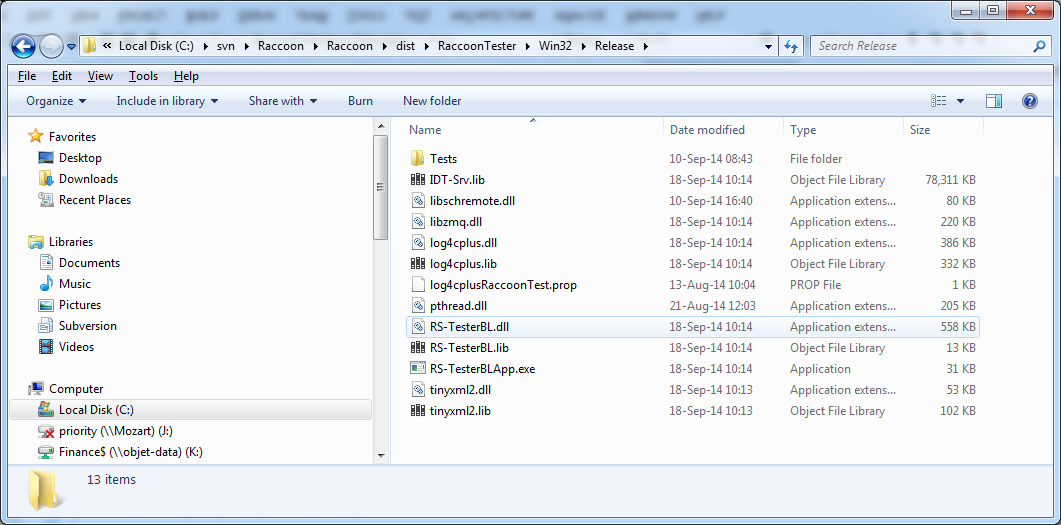
* Raccoon,
* RaccoonTester,
* RacconTesterSimulator.

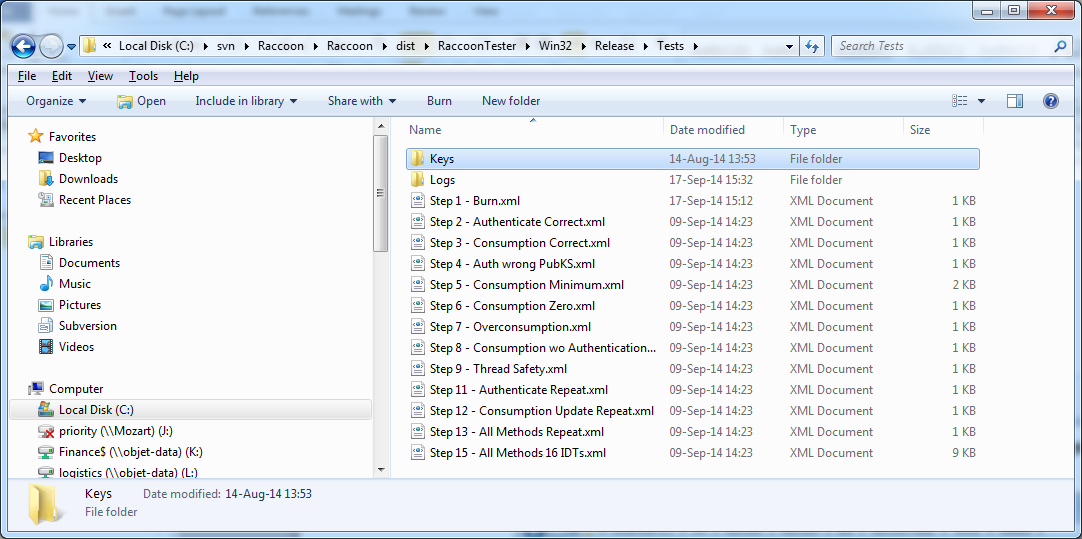
Each one is built and deployed by its own visual studio solution. All solutions supports debug and release configurations and Win32 platform.

* **Raccoon.sln**: build and deploys the operational library. Package destination folder is dist\Raccoon\Win32\Release:

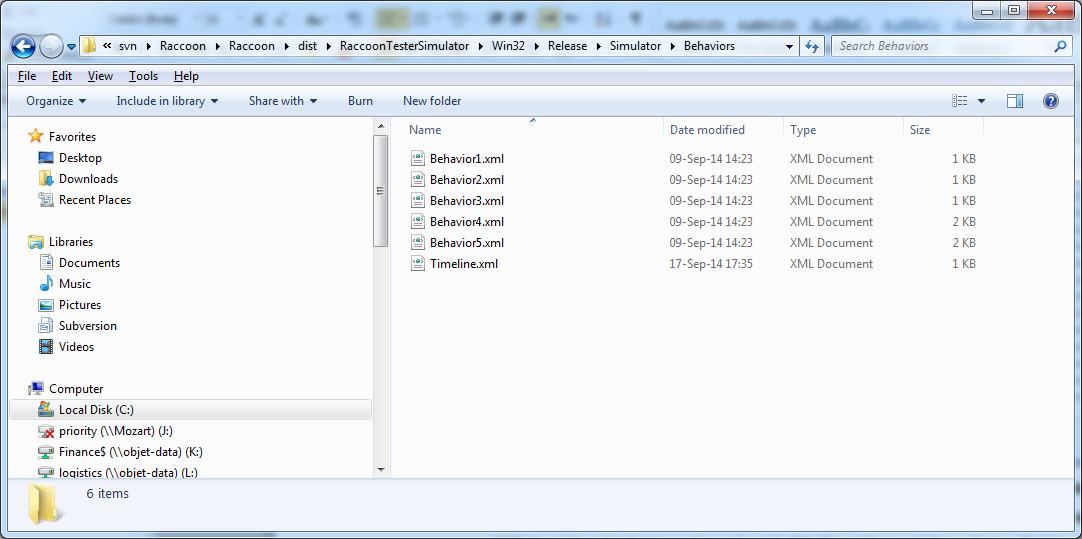


* **RaccoonTester.sln**: build and deploys the tester library. Package destination folder is dist\RaccoonTester\Win32\Release:

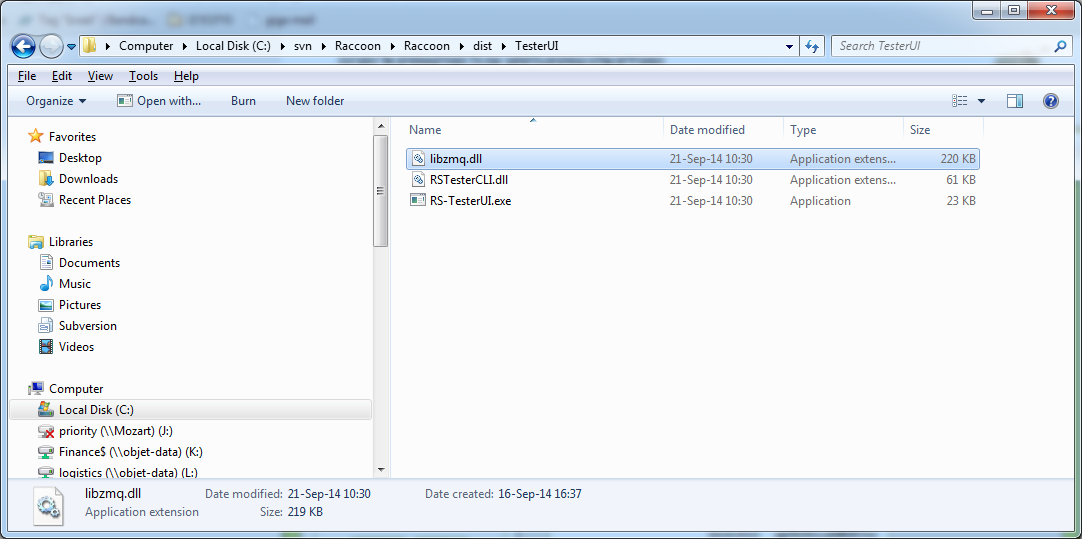




* **RacconTesterSimulator**: build and deploys the tester library. Package destination folder is dist\RaccoonTesterSimulator\Win32\Release. It contains the same files as **RacoonTester**, except IDT-Srv.lib which is replaced with RS-Sim.lib, and a Simulator\Behaviors folder



TesterUI: tester UI is built on both RacoonTester and RaccoonTesterSimulator solutions; destination folder is dist\TesterUI:



## Building the library for Linux

Precondition: install PCSC software, such as PCSClite (yum install pcsc-lite).

As in windows build, each Raccoon software package is built and deployed by its own eclipse workspace. log4cplus, zeroMQ and VaultIC100API libraries have their own makefile; zeroMQ and log4cplus shall be configured before build (sh ./configure). After building those libraries, the workspace can be built on Eclipse.

Build target is the same as windows build.

## Using Raccoon Package

### General

* Add IDT-Lib and IDT-Lib\RSCommon to include directories.
* Add IDT-Srv.lib to linker input files.

### Creating IMaterial Monitor instance:

#include "IDT-Srv/MaterialMonitor.h"

// Create an IMaterialMonitor instance:

RSCommon::IMaterialMonitor\* materialMonitor = RSCommon::CreateMaterialMonitor();

### Setting log function:

#include "Utils.h" // SetLogMessageFunction() is defined here

// set log callback function:

SetLogMessageFunction(&LogMessage);

// Example for log function:

void LogMessage(const char\* message){

       cout << message << endl;

}

### Initializing material monitor:

unsigned char ucCartridgesCount;

int returnValue = materialMonitor->InitHW(&ucCartridgesCount);

### Getting in-place status:

int status;

int returnValue = materialMonitor->GetInPlaceStatus(&status);

### Authenticate cartridge and decode its material info:

#include "IdentificationData.h"

unsigned char iddBuf[32];

unsigned short iddBufLength;

**Previouse interface:**

// If SSYS public key is already set, pucPubKS argument can be replaced with null:

int result = materialMonitor->AuthenticateCartridge(ucCartridgeNum, pucPubKS, iddBuf, &iddBufLength, &m\_volumes[ucCartridgeNum]);

**New / changed interface:**

* Was removed input parameter pucPubKS – not relevant, because pucPubKS burned on Dongle;
* Was added input parameter isActive, that means active tag or not;

int result = materialMonitor->AuthenticateCartridge(ucCartridgeNum, isActive, iddBuf, &iddBufLength, &m\_volumes[ucCartridgeNum]);

// Decode material info from buffer:

MaterialInformation materialInfo;

unsigned short materialInfoLength = (unsigned short)materialInfo.Decode(iddBuf);

// Copy tag serial number (not a part of MaterialInfo class) from buffer:

unsigned char serialNo[SERIAL\_NO\_SIZE];

memcpy(serialNo, iddBuf + materialInfoLength, SERIAL\_NO\_SIZE);

### Update material consumption:

unsigned int volume;

int returnValue = materialMonitor->UpdateConsumption(ucCartridgeNum, uiConsumption, &volume);

## Information burnt on tag

|  |  |  |
| --- | --- | --- |
| **Region** | **File** | **Content** |
| Static File System | /cert.cer | A binary file containing serialized IDC, i.e. serialized identification data followed by its signature by dummy SSYS private key. |
| Key Ring | Device public key | Public key of the key pair generated by tag. |
| Device private key | Private key of the key pair generated by tag. |
| Host public key | Dummy SSYS public key. |
| ECDSA domain parameters | Defined by K-283 curve. |
| Device ID | Device serial number followed by 6 dots, e.g. if serial number is 1A2B3C4D5F, then device ID is “1A2B3C4D5F……” |
| Host ID | The text “StratasysPrinter” |

The file system of the tag has two regions: the static file system and the key ring.

The static file system is empty except for the certificate file; its content is specified in the content column.

The key ring contains the keys and the domain parameters for signing data and two ID files.

The ID files are two text files; each one is 16 characters long. They are used in the internal authentication process. They don’t need to be confidential.