***Detailed Design Specifications***

***for***

**RS-BSS**

Document ID:

Document Revision:

|  |  |  |  |
| --- | --- | --- | --- |
|  | ***Function*** | ***Name*** | ***Elec. Signature & Date*** |
| **Prepared by:** | Software Engineer | Itay Sofer |  |
| **Reviewed by:** | Architect or HW/SW Engineer |  |  |
| **Approved by:** | Project Architect / Project Manager |  |  |

Table of Contents

[1.0 Identification 3](#_Toc352596061)

[2.0 Introduction 3](#_Toc352596062)

[2.1. Purpose 3](#_Toc352596063)

[2.2. Scope 3](#_Toc352596064)

[2.3. Definitions, Acronyms, and Abbreviations 3](#_Toc352596065)

[2.4. References 3](#_Toc352596066)

[3.0 Design Considerations 4](#_Toc352596067)

[3.1. Major Design Constraints/Limitations 4](#_Toc352596068)

[3.2. Major Design Decisions 4](#_Toc352596069)

[3.3. Tools and Technologies Used 4](#_Toc352596070)

[3.4. Safety Mitigation 4](#_Toc352596071)

[4.0 Module Design 4](#_Toc352596072)

[4.1. System Architecture 4](#_Toc352596073)

[4.2. High Level Design 4](#_Toc352596074)

[4.3. Detailed Component Design 4](#_Toc352596075)

[5.0 Requirements Cross Reference 4](#_Toc352596076)

[6.0 Packaging 4](#_Toc352596077)

[7.0 Revision History 4](#_Toc352596078)

[Appendix 1 4](#_Toc352596079)

# Identification

This document describes the top-level design of RS-BSS. RS-BSS (Raccoon system – burning station software) is the main software of Raccoon burning station, which enables the user to:

* Burn confidential information on a secured tag placed on SSYS cartridge.
* Validate the information burned on existing cartridge.
* Generate and view burning reports.

RS-BSS is combined of a BSP (burning station proxy) module deployed on a central burning server, and a BSA (burning station application) module deployed on the burning station itself. BSA exposes RS-BSS functionalities, and responsible to request the confidential burning data from the BSP.

# Introduction

## Purpose

This document describes in details the top-level design of RS-BSS.

## Scope

The document describes the detailed design of RS-BSS. It describes the internal components of the features (involving models, view models, views, tools, etc.) and the logic flow.

## Definitions, Acronyms, and Abbreviations

**Batch** is a material production unit: a 1 m³ tank to be used for bottling different cartridge types/sizes.

**Work-Order** is a record that holds bottling instructions for a certain material and a certain cartridge type/size, per month.

**Lot** is a sub-work-order (as defined in manual work-plan): bottling instructions for a certain material from a certain batch and a certain cartridge type/size.

|  |  |
| --- | --- |
| BSA | Burning Station Application (on the BrS) |
| BSp | Burning Station Proxy (on the CBS) |
| BSS | Burning Station Software: BSA + BSP + relevant data |
| CBS | Central Burning Server (computer) |
| FCB | Front Communication Board (with the IDTs) |
| IDD | ID Data – the tag data to be signed |
| IDT | ID Tag |
| IDT-Srv | IDT services library to be used by the application |
| PrvKC - PubKC | Cartridge Private Key & Public Key |
| PrvKS - PubKS | Stratasys Private Key & Public Key |
| VLL | VaultIC List Loader: VLL-A + VLL-P |
| VLL-A | VaultIC List Loader – Application (on a user’s PC) |
| VLL-P | VaultIC List Loader – Proxy (on the CBS) |

## References

RS-BSS SRS

RS-EMS DDS

# Design Considerations

## Major Design Constraints/Limitations

It shall be combined of two main modules: BSP (burning station proxy) module deployed on a central burning server, and a BSA (burning station application) module deployed on the burning station itself.

BSP shall be accessible by authorized users only.

BSA and BSP shall be executed over Win32 machines running Windows 7 and up.

BSA and BSP shall be integrated with peripherals under development such as traffic light and buzzer and an HSM client.

BSA and BSP shall be testable.

## Tools and Technologies Used

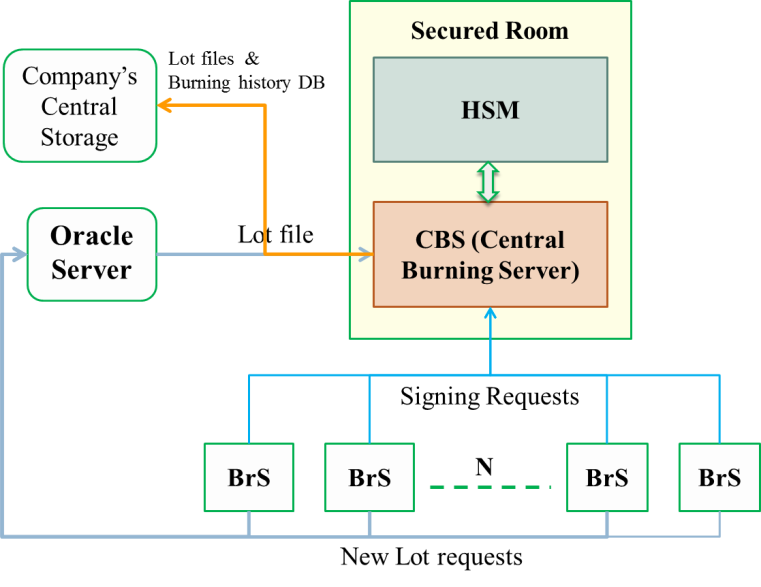
Software is designed according to MVVM pattern; it will be developed in .Net framework 4.5. Galasoft MVVM is used for basic MVVM components.

# Module Design

## System Architecture

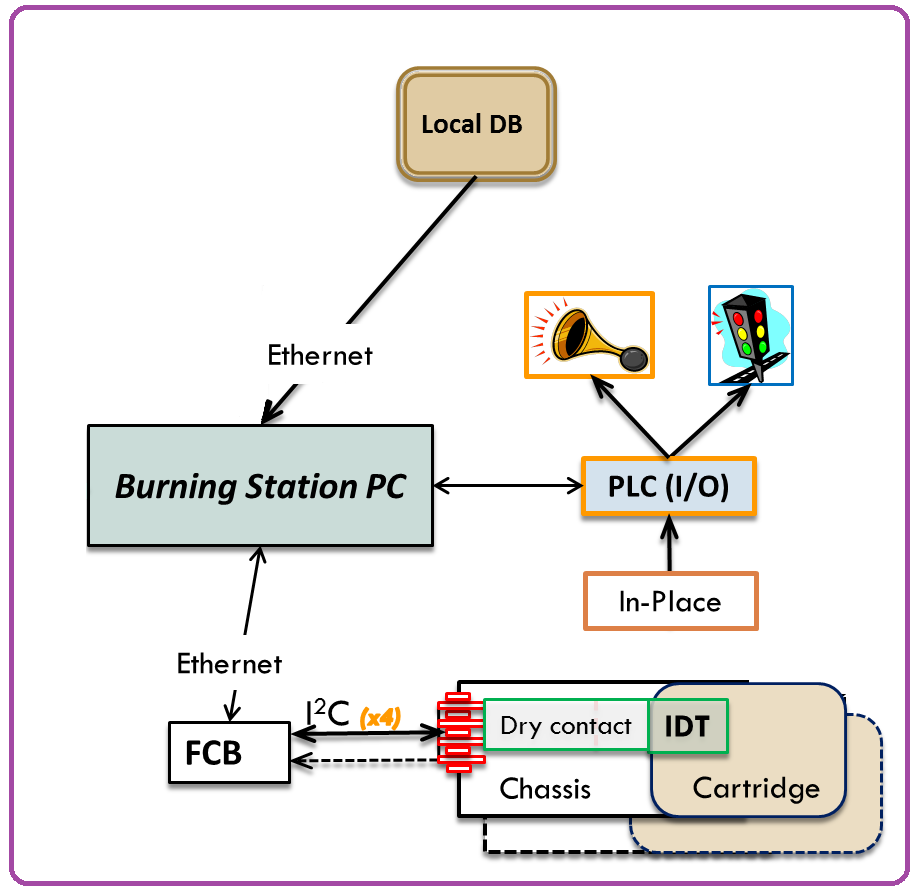
RS-BSS is being designed from scratch (hardware & software). Yet, it must keep the external interfaces and the work procedures as exist today in the RFID-based station:

* The Lot is defined by a work-order number & a batch number accepted as barcodes. These numbers are converted into material details and bottling instructions by Oracle Application.
* The station contains up to 4 chassis for cartridges to be burned, a buzzer and 4 traffic lights that signal process failure/completion (for operator that controls multiple stations).
* The insertion of a cartridge is identified by a special in-place indicator.
* The material information is sent to a local Data-Base (DB) for manufacturing control & statistics.



The CBS executes the BSP, which is deployed as a windows service executed automatically on startup. The Oracle application is also executed by CBS. BSP uses an HSM client to sign IDD by SSYS private key.

Each Brs executes a BSA process. Communication with BSP is established, maintained and secured using WCF platform.



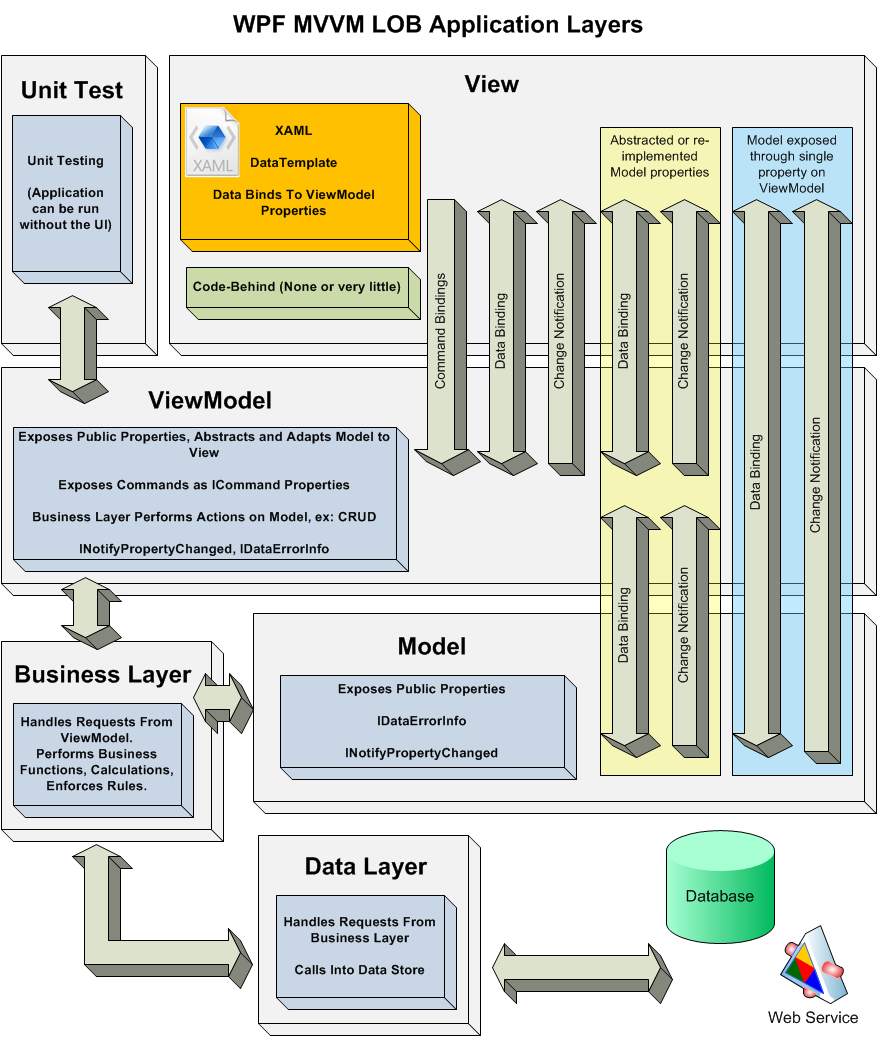
BSA communicates PLC using ModBus protocol using RS232 connection.

The IDT is based on a security device (chip) that communicates via I2C Bus over a dry contact to a Front Control Board (FCB). As any bus, the I2C Bus is also based on node addresses. Since we cannot initiate each IDT with a unique address, each IDT has a dedicated line to the FCB. The FCB uses a MUX element that switches between IDTs, according to the software request.

## High Level Design

### Overview

RS-BSA design follows the general MVVM line of business, as illustrated below:



BSA layers are packaged into the following class libraries:

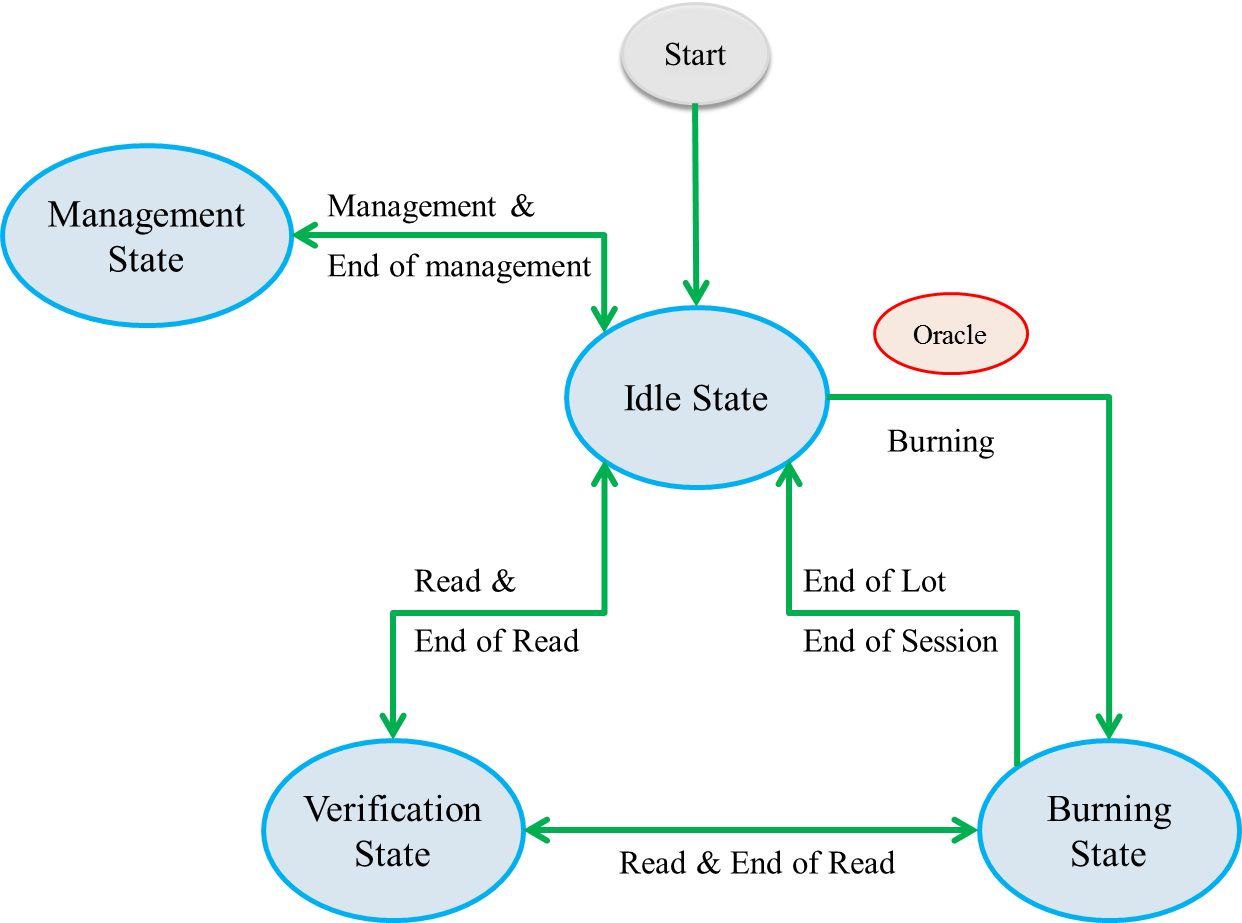
* **BSS.App** – packages the application, the main view and the main view-model.
* **IDT-Srv-Burner** – packages the part of the model regarding to Idt-Srv that has to be implemented in native code.
* **BSS.Contracts** – packages the data that has to be shared between BSA and BSP.
* **BSS.MVVM** – packages view, view-model and the rest of the model.
* **BSS.UnitTests** – packages the unit-tests layer.

BSP modules are packaged into the following class libraries:

* **BSPHost** – hosts the WCF services.
* **BSP** – contains the business logic of the burning server.
* **BSS.Contracts** – packages the data that has to be shared between BSA and BSP.
* **HSMClientWrapper** – packages the part of the business logic regarding to HSM client that has to be implemented in native code.

### Basic flow

RS-BSA contains 5 system states: Idle, burning, verifying, report and management.



#### Idle state

BSA does no action in Idle state. It is the initial system state, and used in transition between other states.

#### Burning state

Transition to burning from idle state is allowed when two conditions are satisfied:

* A new Lot is read by Oracle application in the CBS or current Lot is not closed yet.
* No cartridge is placed in chassis.

User may transit back to idle state by ending Lot and waiting for a new one to be read, or by ending the burning session and later resume it as if burning was not interrupted. Any other state termination (e.g. closing the application, shutdown, etc.) is equivalent to session ending.

The actual burning is executed automatically when cartridge is inserted. The yellow light is turned on at the insertion moment. After burning ends, the green or the red light are turned on according to success or failure of burning. Light is turned of after cartridge is removed from chassis, as illustrated below:



#### Verification State

The purpose of the IDT verification function is to authenticate tag, read the IDD burned on it and validate the IDD. BSA can transit to verification state from Idle or from burning state when all chassis are empty. After verification is done, user may transit back to the origin state.

The actual verification is performed in the same manner as burning does, as illustrated below:



#### Report state

In report state user may view burning history of Lot or tags, and export it to excel. This state is accessible from any state, but the user may only trasit back to the origin state.

#### Management state

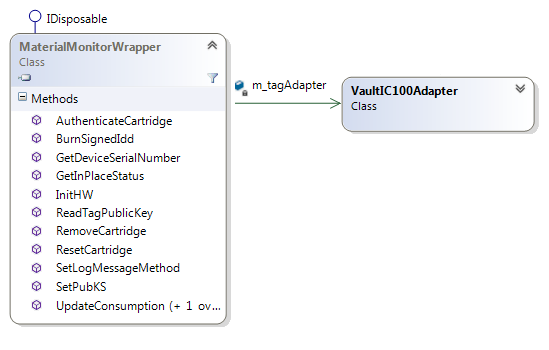
In manage state user may view system configuration parameters and modify some of them. This state is accessible only from idle state.

### Model

The model layer lies within three class libraries: IDT-Srv-Burner, BSS.Contracts and BSS.MVVM. IDT-Srv-Burner is a native module for the IDT burning and verification. BSS.Contracts provides service and data contracts to be shared between BSA and BSP. The BSA business logic as well as entities that are not needed to be shared are contained in BSS.MVVM module.

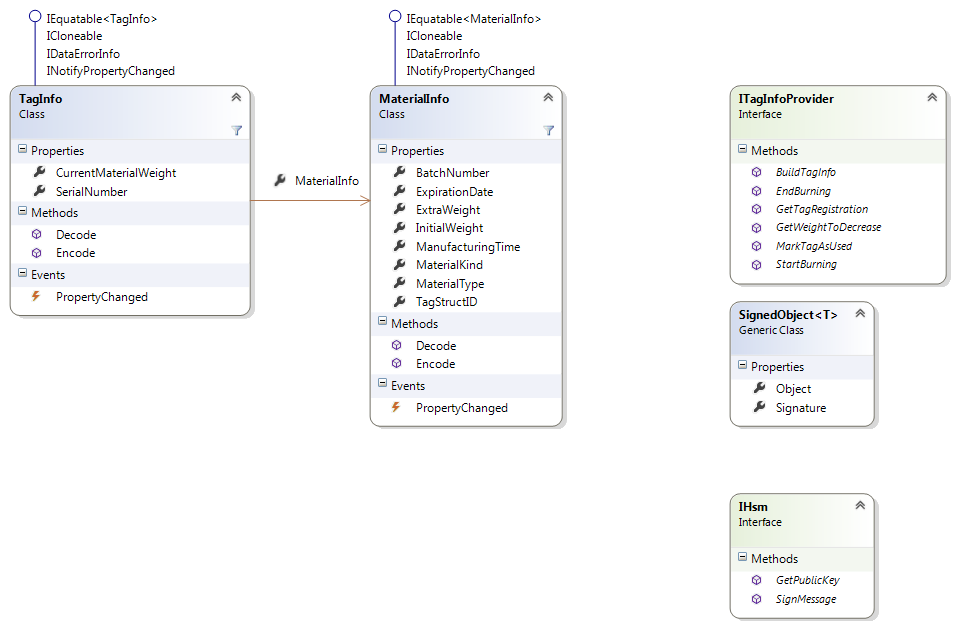
#### IDT-Srv-Burner

Existing class libraries of RPS-SW package already contains the VaultIC100 API and the communication components, as well as the IDT authentication and get in-place status functionalities. However, the burning and correcting functionalities are missing. To avoid massive code duplication, IDT-Srv is not used as a building block, but as a base for source codes. The exiting VaultIC100API, RS-Common and the IDT-HW-Inf are taken as they are; IDT-Srv library is renamed to IDT-Srv-Burner, and MaterialMonitor and VaultIC100Adapter are extended to include the burning and correcting functionalities, but no existing method is modified. The only exception is defining the ENABLE\_USB preprocessor definition to be able to work with evaluation kit.



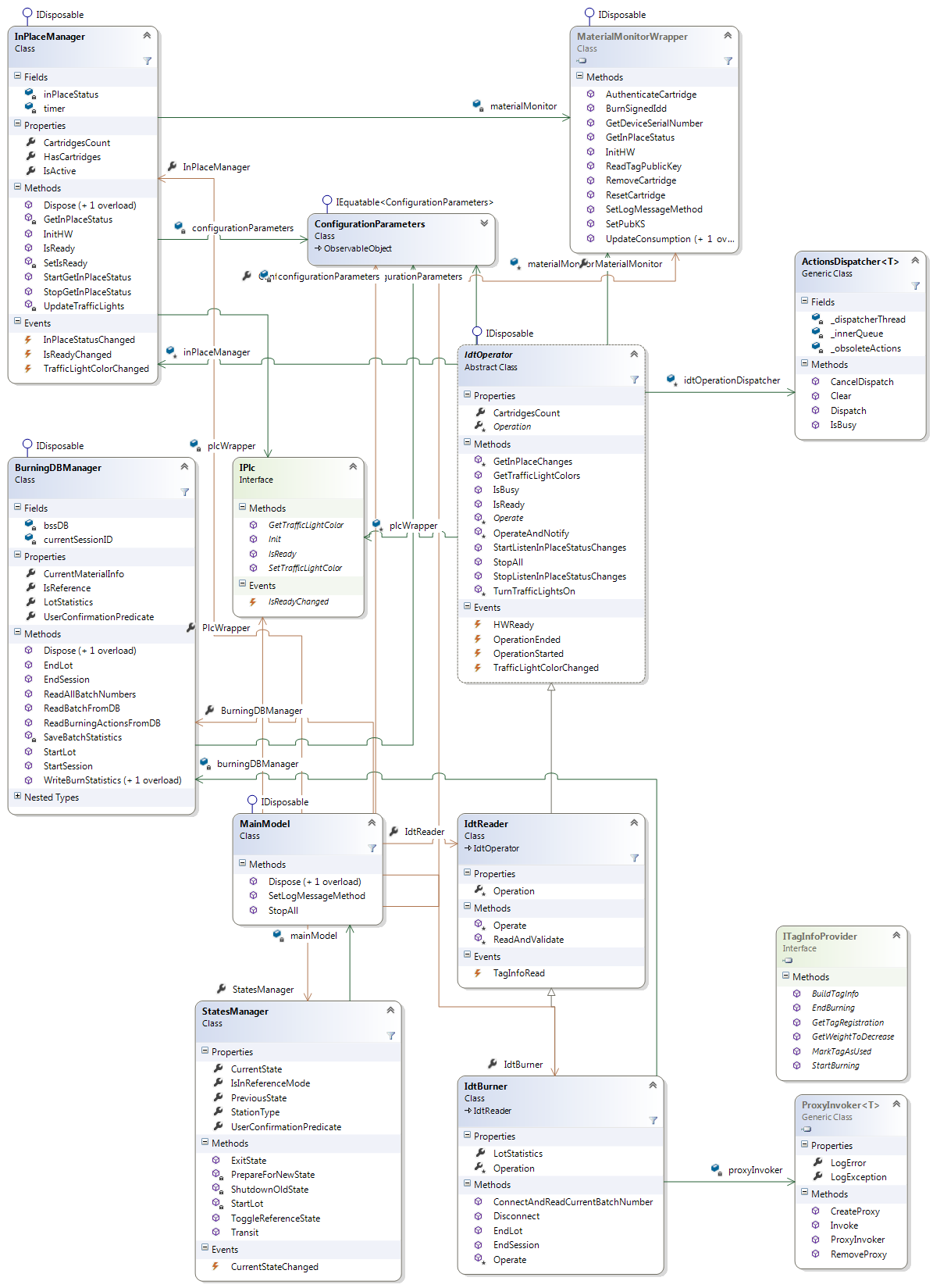
#### BSS.Contracts

BSS.Contracts defines service and data contracts to be shared between BSA and BSP. Each data contract is responsible for its serialization and error validation.



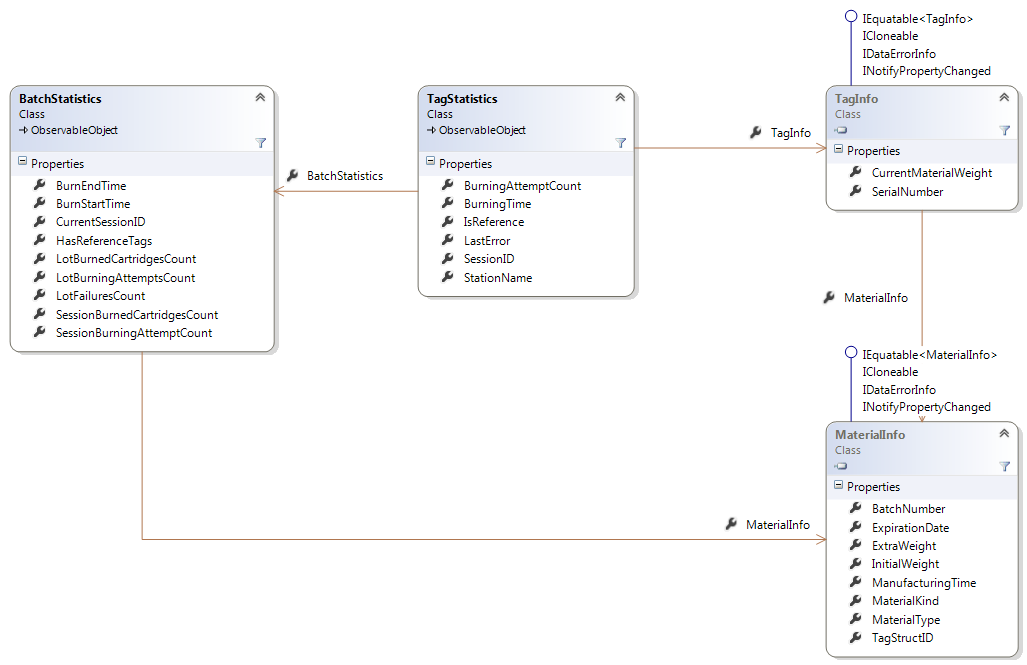
#### Business Logic

BSS.MVVM.BusinessLogic namespace groups the classes holding the business logic of the burning station application. It holds the StatesManager class to hold the system state and transit between states, The InPlaceManager class to setect insertion and removal of cartridges, IdtBurner to burn tags, IdtReader to verify tags, BurningDBManager to read and write burning records to DB, ConfigurationParameters class to load and save configuration parameters, PLC classes to control the PLC devices and other helping classes such as ActionsDispatcher. See 4.3.4 for details.



#### BSS.MVVM.Entities

The BSS.MVVM.Entities namespace contains entities batch and tag statistics to be read and written to burning DB.



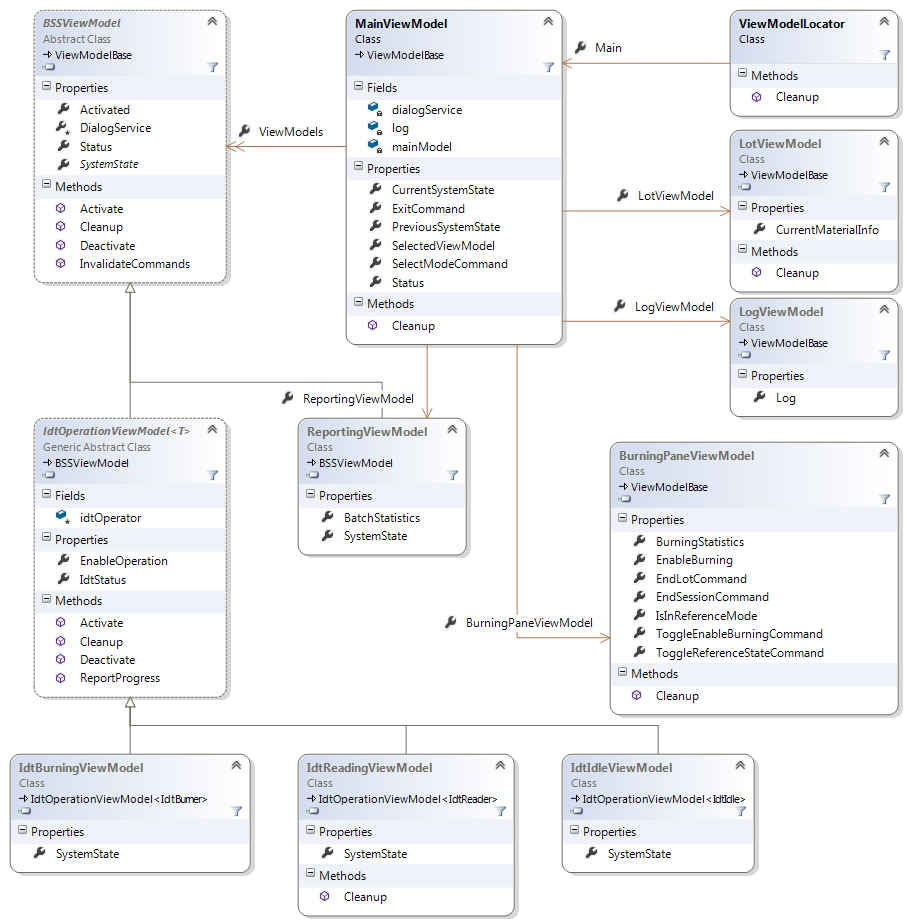
### View and View-Model

View and view-model layers are packaged in BSS.MVVM.View and BSS.MVVM.ViewModel namespaces, except for application view and view-model, which is packaged in BSS.App class library.

Classes in view layer are derived from UI design, with no code behind.

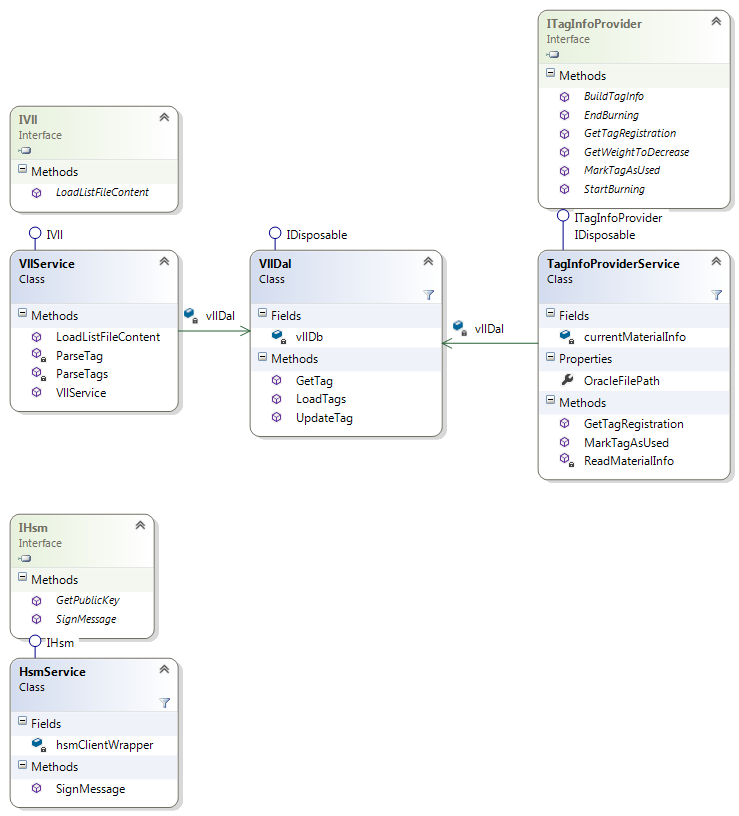
All view-models inherit an abstract BSSViewModel, which holds common properties and methods. Both BSSViewModel and MainViewModel inherit ViewModelBase class of MVVM-lite toolkit.

As a thumb rule, each functionality of a view model is covered by a public method and a command. For instance, when user clicks the "EndLot" button on burning modes, EndLotCommand in the view model is executed, which calls the EndLot() method. This method, as well as all command methods, is declared public to support unit testing.



### BSP

The BSP is a windows service which hosts three WCF services: the TagInfoProviderService, HsmService and VllService. The burning station application is a client of the former two services, while VLL application is a client of the latter.



### Threading Model

Communicatio with CBS, as well as any interaction with peripheral devices, such as FCB or PLC, may be slow or even blocking; hence, methods as a rule of thumb, are called asynchronously.

#### Interaction with PLC

Calls to PLC returns immediately when communication is ok, but when PLC is not connected, call is blocking; to avoid blocking the UI, any call to PLC will be invoked asynchronously following task-based async pattern (TAP). If PLC does not respond after timeout is reached, operation is cancelled.

#### Communication with CBS

If CBS is not connected, proxy operations may block for a period of timed configured by WCF service behavior; any call via proxy is invoked asynchronously following TAP.

#### Interaction with FCB

IDT-Srv-Burner is a thread-safe module, because VaultIC API is single threaded. It synchronizes all operations on IDT (burn, authenticate, get in-place status) by owning a mutex as the original IDT-Srv does. Those operations are time comsuming, and calling them synchronously from UI thread may nake UI unresponsive. In addition, burn and validate operations are composed of several calls to IDT-Srv-Burner, but they shall be atomic. To satisfy these, burn and validate operations are invoked by ActionDispatcher class, which runs operation asynchronously, and executes simultaneous requests one by one. See detailed component design for details.

In rare cases, FCB initialization method may be blocking; as done in PLC, initialization is called asynchronously.

### Key Sequences

#### Initialization

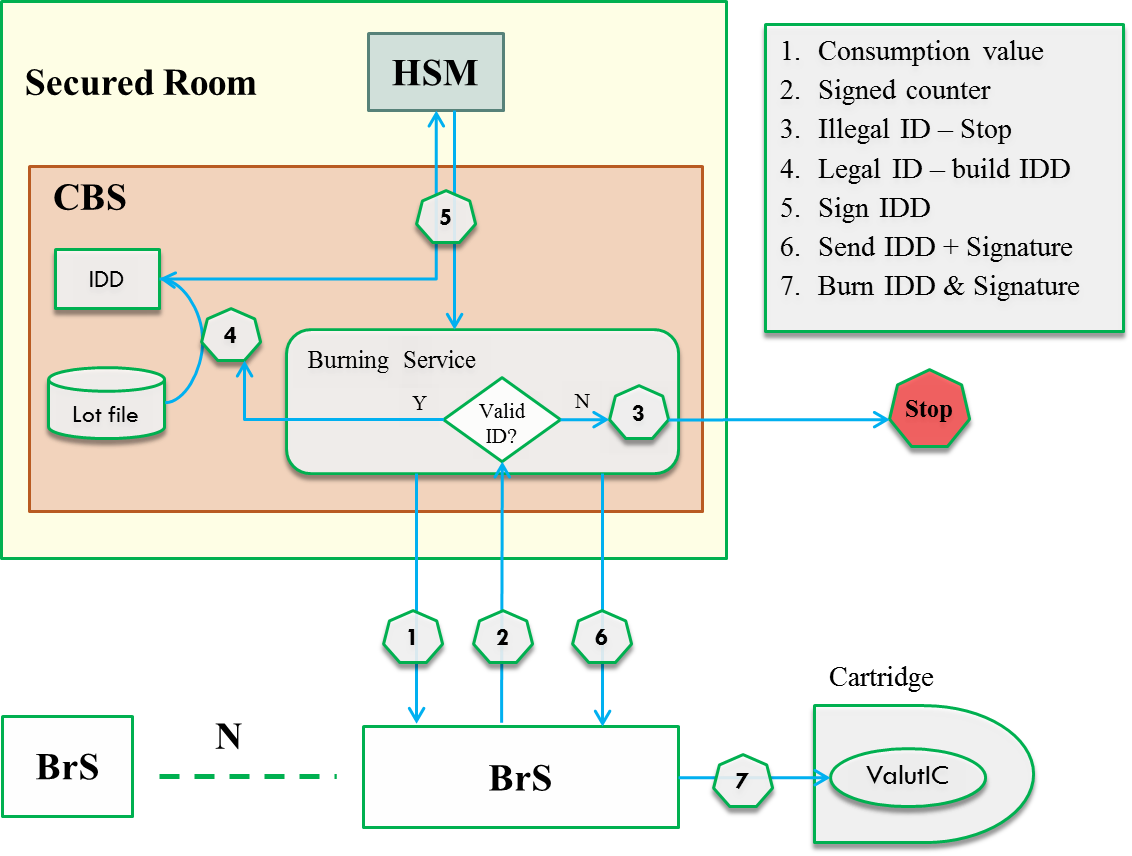
BSS.App view-models are initialized using service locator pattern in order to inject BL classes into view-models at initialization. Using this pattern decouples view from business logic layer. ViewModelLocator class role is to be the service locator, and it is created as application static resource.

ViewMoelLocator provides the Main property of type MainViewModel, which is bound to main view data context; consequently, ViewModelLocator static constructor is called on startup. On its static constructor, ViewModelLocator takes advantage on SimpleIoc class of MVVM light package to initialize and register LotReader and MaterialInfoBurner. By registering the view-models, SimpleIoc injects those classes into MainViewModel constructor.



#### Burning IDT

* User activates burning window.
* Cartridge inserted.
* InPlaceManager identifies insertion, turns on yellow light and raises event. If any failure occurs on the following stages, burning is terminated, and both red light and buzzer are turned on.
* IdtBurner requests the serial number of the inserted tag from material monitor.
* IdtBurner requests from TagInfoProviderService to check if tag is found in VLL and its status.
* TagInfoProviderService passes the request to VLL DB using VllDal, and sends the response to IdtBurner.
* IdtBurner requests counter update value from TagInfoProviderService.
* IdtBurner updates tag counter using material monitor.
* IdtBurner requests the signed IDD from tag preparation service.
* IdtBurner burns the signed IDD on tag using material monitor.
* IdtBurner verifies the burn using authenticate function.
* IdtBurner writes burning statistics into database.
* IdtBurner turns green light on success or red light on failure.
* Container removed.
* User deactivates burning window.



### Error Handling

To be consistent with MaterialMonitor existing error policy, its new public methods are exception-safe, returns 0 for success and error codes for failures. On the other hand, BSS.BusinessLogic component throws exceptions on failures, and view-model components catches them and displays their messages. For detailed list of exception types thrown, see detailed component design section.

### Security

Most BSS functions need no protection: process management, hardware interfaces, cartridges access, reports, etc. However, the BSS shall be protected from unauthorized modifications of the IDD (mainly the cartridge weight) and the tag consumption counter.

In order to protect the sensitive data, it will be located on CBS and it will be modified by authorized users only.

#### BSP Security

Securing BSP takes advantage on WCF security. The default security settings for TCP binding are:

* Message protection level: messages are encrypted and signed.
* Transfer security: transport.
* Authentication: windows authentication.

The default protection level and transfer security are satisfactory. To ensure that only burning station could access the BSP, they will log in to windows with a specified user name, and only that user name will be able to access the HSM and TagInfoProvider services; access of other user throws a security exception.

#### HSM Security

HSM client could be accessed from a specified address and port, which is taken by BSP.

#### DB Security

The control reports are the final “gatekeeper”: they intend to reveal mismatches & exceptions due to unauthorized data modifications or fake signing requests that succeeded penetrating the security mechanisms. This is achieved by adding signature column to each table. When a record is read, its signature is verified by HSM client.

## Detailed Component Design

### MainModel

This class functions as the container of all components. It is responsible for model initializing and cleanup and used by main view model to set references between view-model components to their corresponding BL compon

### StatesManager

This class is responsible for holding the system state and transit between states. StatesManager provides the following members:

* CurrentState: BssState – gets the current state.
* Transit(newState: BssState): void – transits into new state.
* Exit(): void – returns to the previous state.
* CurrentStateChanged – occurs when current state is chaged.

Transit() and Exit() methods may throw InvalidStateTransitionException if transition is not allowed.

### ConfigurationParameters

This class provides system configuration properties and methods for loading and saving them. The provided properties are:

* BurningDBConnectionString – gets the connection string to burning DB.
* ChassisCount – gets the number of chassis.
* InPlaceStatusInterval – gets or sets the interval between calls to get in-place status.
* StationName – gets the station name.
* StationType – gets the station type (standalone or inline).
* UsePlcSimulator – gets a value indicating whether use PLC simulator.

### InPlaceManager

This class is responsible for getting in-place status and notifying about status changes.

InPlaceManager provides the following public members:

* InitHW(): bool – initializes the FCB hardware.
* StartGetInPlaceStatus(): void – starts periodic call to get cartridges in-place status. The default period
* StopGetInPlaceStatus(): void – stops the periodic call to get cartridges in-place status.
* InPlaceStatus: int – gets the current in-place status.
* IsActive: bool – gets a value indicating if periodic call to get in-place status is active.
* InPlaceStatusChaged – occurs when in-place status changed.
* IsReadyChanged – occurs when FCB becomes ready or unavailable.

### IdtOperator

This class holds common functionality for burning and verifying operations on IDT.

IdtOperator provides the following public members:

* StartListenInPlaceStatusChanges(): void – starts listening to changes of in-place status.
* StopListenInPlaceStatusChanges(): void – stops listening to changes of in-place status.
* Operate(cartridgeNumber: byte): void - when overriden in a derived class, operates on a cartridge specified by its number.
* IsBusy(): bool – returns a value indicating whether IDT is currently being operated.
* OperationStarted – occurs when operation on IDT starts.
* OperationEnded – occurs when operation on IDT ends.
* TrafficLightColorChanged: occurs after a change in traffic light color. Event arguments contains cartridge number and the new color.

### IdtReader

This class is Responsible for reading material info from IDT when cartridge is inserted.

IdtReader provides the following members:

* ReadAndValidate(cartridgeNumber: byte): TagInfo - Reads tag information from a specified cartridge and validates it.
* TagInfoRead – occurs when tag info is read from cartridge.

### IdtBurner

This class is responsible for burning material info into IDT when cartridge is inserted.

IdtBurner provides the following methods:

* Burn(cartridgeNumber: byte): bool – burns current material info into cartridge specified by its number. Burning algorithm follows the steps describes in (4.2.7.2). The burning steps are dependent on tag info provider service running on BSP, see (4.3.12) for details.

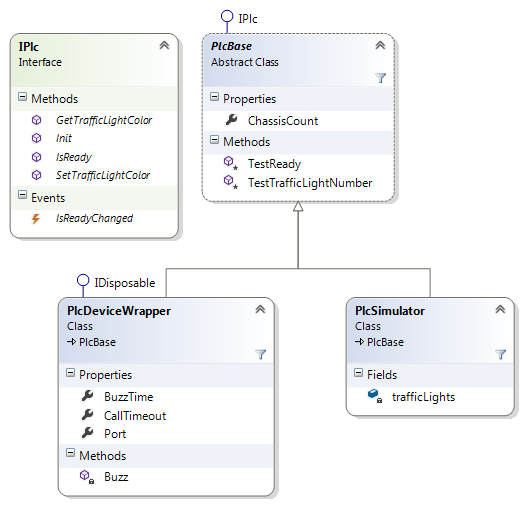
### PLC

The IPlc interface defines the PLC functionality:

* Init(): bool – initializes connection to PLC.
* IsReady(): bool - determines whether connection to PLC is ready.
* GetTrafficLightColor(trafficLightNumber: byte): color - gets the color of the traffic light specified by its number.
* SetTrafficLightColor(trafficLightNumber: byte, color: color): void – sets the color of the traffic light specified by its number.
* IsReadyChanged: occurs when PLC becomes ready or unavailable.

PlcBase is an abstract class implementing IPlc, implementing input validation.

PlcDeviceWrapper communicates with the PLC device using ModBus protocol, and PlcSimulator simulates the PLC device. Both classes implement PlcBase.



### RaccoonCLI

This class is a C++/CLI class wrapping native material monitor of Idt-Srv. The original interface of the material monitor was extended to implement burning functionality, as described in (4.2.7.2). The following methods were added to material monitor, and hence to RaccoonCLI:

* BurnSignedIdd(cartridgeNum: byte, idd: char\*, signature: char\*) : int – write IDD and signature into tag file system.
* GetDeviceSerialNumber(cartridgeNum: byte, serialNumber: char\*) : int – gets the tag serial number.

All methods return 0 for success and non-zero values for errors; see (4.2.7) for error codes returned by MaterialMonitor.

### BurningDbManager

This class is responsible for reading and writing burning statistics to DB. It provides the following methods:

* StartLot(): void – initializes burning statistics of current Lot.
* StartSession():void – initializes burning statistics of current burning session.
* EndLot():void – closes statistics of current Lot.
* EndSession():void – closes statistics of current burning session.
* ReadBatchFromDB(batchNumber: string): Batchstatistics – reads batch statistics from DB by batch number.
* ReadAllBatchNumbers(): string[] – reads numbers of all batches from DB.
* WriteBurnStatistics(tagInfo: TagInfo): void – writes burned tag information into DB.

### ActionsDispatcher

This class stores actions in a queue and dispatches them sequentially. Actions are executed on background by a dispatcher thread, which starts when queue becomes non-empty and stops when queue is empty. ActionDispatcher is used by IdtBurner and IdtReader to handle simultaneous cartridge insertion sequentially.

ActionsDispatcher provides the following methods:

* Dispatch(action: Action): void – inserts action into queue. When action becomes the first item in queue, it is executed and popped out.
* CancelDispatch(action: Action): void – recall execution of an action if not executed yet; when action becomes the the first item in queue, it is popped out immediately.
* Clear(): void – clears all queue items.
* IsBusy(): bool – determines whether dispatcher queue is not empty.

### TagInfoProviderService

This class implements the ITagInfoProvider service contract, including the following operations:

* StartBurning(): string – Reads Lot file generated by Oracle client and returns the batch number.
* GetTagRegistration(serialNumber: byte[]): TagRegistration – gets the tag registration (unknown, new ,read or used) from VLL.
* GetWeightToDecrease(serialNumber: byte[]): uint – gets the weight to decrease from the counter initial value to reflect the actual container weight.
* BuildTagInfo(serialNumber: byte[], pubKC: byte[]): TagInfo – builds the tag information and sign it by HSM.
* MarkAsUsed(serialNumber: byte[]): void – mark tag as used in VLL.
* EndBurning(): void – ends Lot burning. This is a termintating operation.

TagInfoProviderService instance is created per session, meaning that a new instance is created on StartBurning() and disposed after EndBurning() is called.

### HsmService

This class implements the IHsm service contract, including the following operations:

* GetPublicKey(): byte[] – gets SSYS public key.
* SignMessage(message: byte[]): byte[] – Signs a message using SSYS private key.

HsmService passes the request to HSM client.

HsmService instance is created per call, meaning that a new instance is created on each method call and disposed when method ends.

### VllDal

### This class functions as the data access layer for the Vaultic list. It provides the following members:

* LoadTags(content: string): void – parse tags from textual content and loads them into DB.
* GetTag(serialNumber: byte[]): Tag – gets tag information from DB.
* UpdateTag(serialNumber: byte[], tagRegistration: TagRegistration): void – updates tag registration in DB.

# Requirements Cross Reference

|  |  |
| --- | --- |
| **DRS** | **DDS** |
|  |  |
|  |  |

# Packaging

RS-BSA package contains the following files:

* BSS.App.exe
* BSS.App.exe.config
* BSS.MVVM.dll
* BSS.Contracts.dll
* WcfInfras.dll
* RaccoonCLI.dll

# Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Revision** | **Date** | **Status** | **Changed by:** | **Reason/ Description of change.  (Insert Technical Review record)** |
| 1.0 | 24-Oct-2017 |  | Itay Sofer | Creating |
|  |  |  |
|  |  |  |
|  |  |  |

# Appendix 1

*< Contains any additional pertinent information too lengthy or that does not have a logical spot anywhere else in the organization of the document. Examples of this are detailed explanations of a complicated algorithm or justifications of important design decisions.*

*Appendix should also be used to describe other design alternatives that were considered for this module, please specify them and explain the process and rationales for rejecting them (as well as the reasons for accepting the alternative that is finally chosen).>*

**End of document**