Software Design Specification

for

WLENS DATA PIPELINE

Table of Contents

Table of Contents ii

Revision History ii

1. Introduction 3

1.1 Purpose 3

1.2 References 3

2. Overall Description 3

2.1 Perspective 3

2.2 Functions 3

2.3 Users 3

2.4 Operating Environment 3

2.5 Design and Implementation Constraints 4

3. Design Description 4

3.1 High Level Block Diagram 4

3.2 Test Function Flows 6

4. Database Table 13

4.1 Log Rule DB: 13

4.2 Test Function LUT DB 14

4.3 Test Function DB 14

4.4 Test Case Status DB 15

5. Authorization & Authentication 15

6. Time Estimate 15

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Cicy,Saranya | 07/17/2017 | Initial draft | 1.0 |

# Introduction

## Purpose

The purpose of this document is to capture the high level software design specifications for the WLENS Data Pipeline of engineering data and project status.

## References

### SDS Document for W-LENS Application Version 2

# Overall Description

## Perspective

WLENS Web application is an initiative by the WISE team to solve the challenges in the world of data collection, data formatting, data processing & analytics and data distribution. And WLENS Data Pipeline is the back-end design to serve WLENS Web application with raw data collection and database formatting.

## Functions

Below are the functions targeted by the WLENS data pipeline:

* Upload Raw Data to S3
* Process Raw Data to Database

## Users

Users of WLENS Web Application will be internal Amazon employees. However, user management will allow for restricted access to functions based on credentials.

* Typically, internal WTG engineers will frequently access Product Status to track updates on the product development. Users from the management community and external teams will also occasionally access the Product Status.
* Members of the WSP team will frequently access the Test Management function and publish information that will be made available in the Product Status page.

## Operating Environment

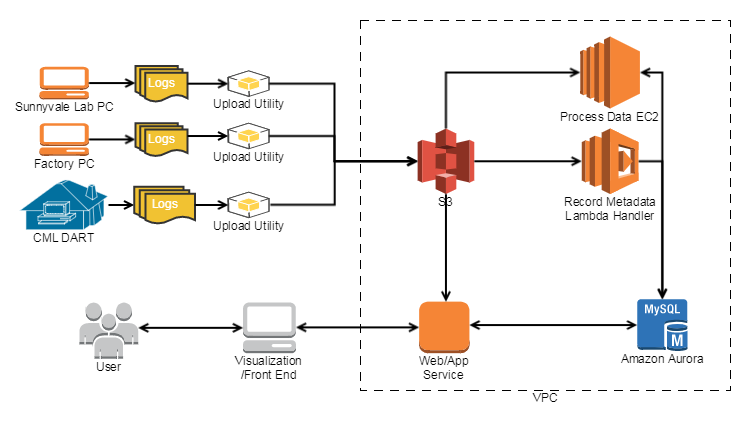
Upload utility will be integrated into any test tool that is connected to the network. It includes lab automation tools and factory automation tools developed by WISE team, and IQfact+ tools, CMWrun tools and Antenna test tools provided by vendors. And these tools are based on Windows system or Linux system.

## Design and Implementation Constraints

Since S3/Lambda/MySQL is using AWS resources, it will comply with the corporate regulatory policies defined for AWS.

# Design Description

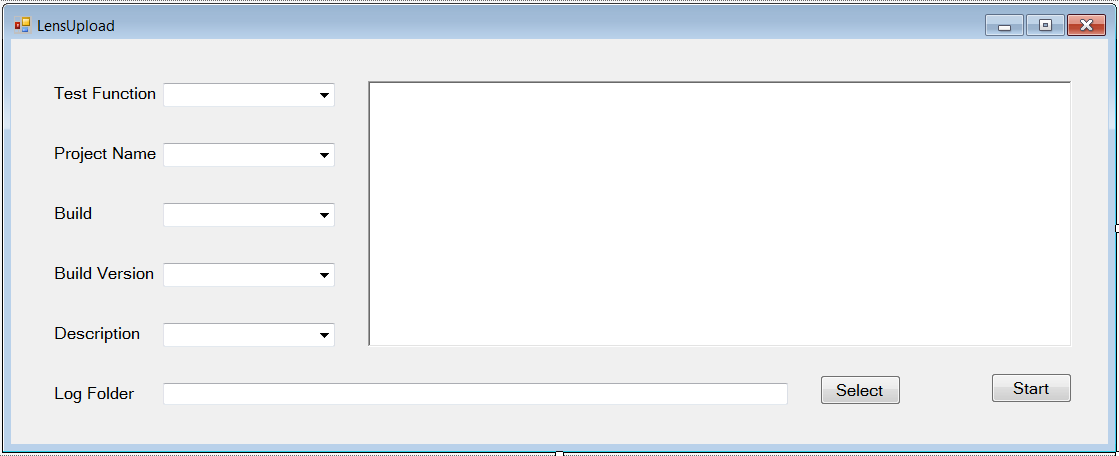
## High Level Block Diagram



### Upload Utility - Upload Raw Data to S3

* Define raw data with different naming rules and formats based on each test function characteristic
* Non-real-time upload: move raw data folder to a network available PC, then make use of GUI to upload all logs to S3.

**Upload Utility GUI** (**Description** is optional, other options are mandatory)

****

* Real-time upload: integrity utility into test tool, it uploads raw data to S3 automatically once test done when network is available.

**Upload Utility Public API**

// Summary:

// Check all sensitive parameters could be found in database

// Check log format and name meet the Log Rule DB

// Upload log to S3 if it pass above checks, otherwise upload fail: return false and provide ErrorMessage

public bool UploadLog(string TestFunction, string ProjectName, string Build, string BuildVersion, string Log, ref string ErrorMessage, string Description = null)

// Summary:

// Get Test Function list from database

public string[] GetTestFunction()

// Summary:

// Get Project Name list from database

public string[] GetProjectName()

// Summary:

// Get Build list from database

public string[] GetBuild()

// Summary:

// Get Build Version list from database

public string[] GetBuildVersion()

// Summary:

// Get Description list from database

public string[] GetDescription()

### Lambda – Record Metadata to Database

* Lambda function is triggered by S3 raw data and records the metadata into database

### EC2 – Process Metadata to Database

* EC2 querys database for unprocessed metadata, fetches it from S3, processes it into database

### Database

Test function includes:

* Log Rule DB: Upload Utilty follows it to check log format and name
* Metadata Status DB: EC2 querys it to get unprocessed metadata
* Test Function LUT DB: EC2 looks it up to locate Test Case ID
* Test Function DB: EC2 inserts test-function-data into it
* Test Case Status DB: EC2 updates test case status once it’s done

## Test Function Flows

Bascially, we could separate the test flow into 2 kind of cases:

* One is integrate Upload Utility into tools developed by WISE: LAB RVR, CML RVR, Congestion, LAB Co-existence and CML Co-existence.
* Another one is integrate Upload Utility into tools provided by vendor: Non Signaling, Signaling and Antenna.

Below, list all formates and rules defined for each test function, will take LAB RVR & Non Signaling as examples for above 2-case with detail flows.

### LAB RVR

*Log format:*

**SN\_Timestamp.zip** including **SN\_Timestamp.csv inside**

*S3 structure:*

**ProjectName/Build/BuildVersion/LABRVR\_SN\_Timestamp\_Description.zip**

* Description list: Buttercup TV, Direct TV, HDMI off, P2P remote, P2P SAP/WFD, WFD & BT remote & A2DP

*Lambda:*

Save S3 log path to Metadata Status DB

*EC2:*

Fetch unprocessed log by **LABRVR** prefix& **.zip** suffix log.

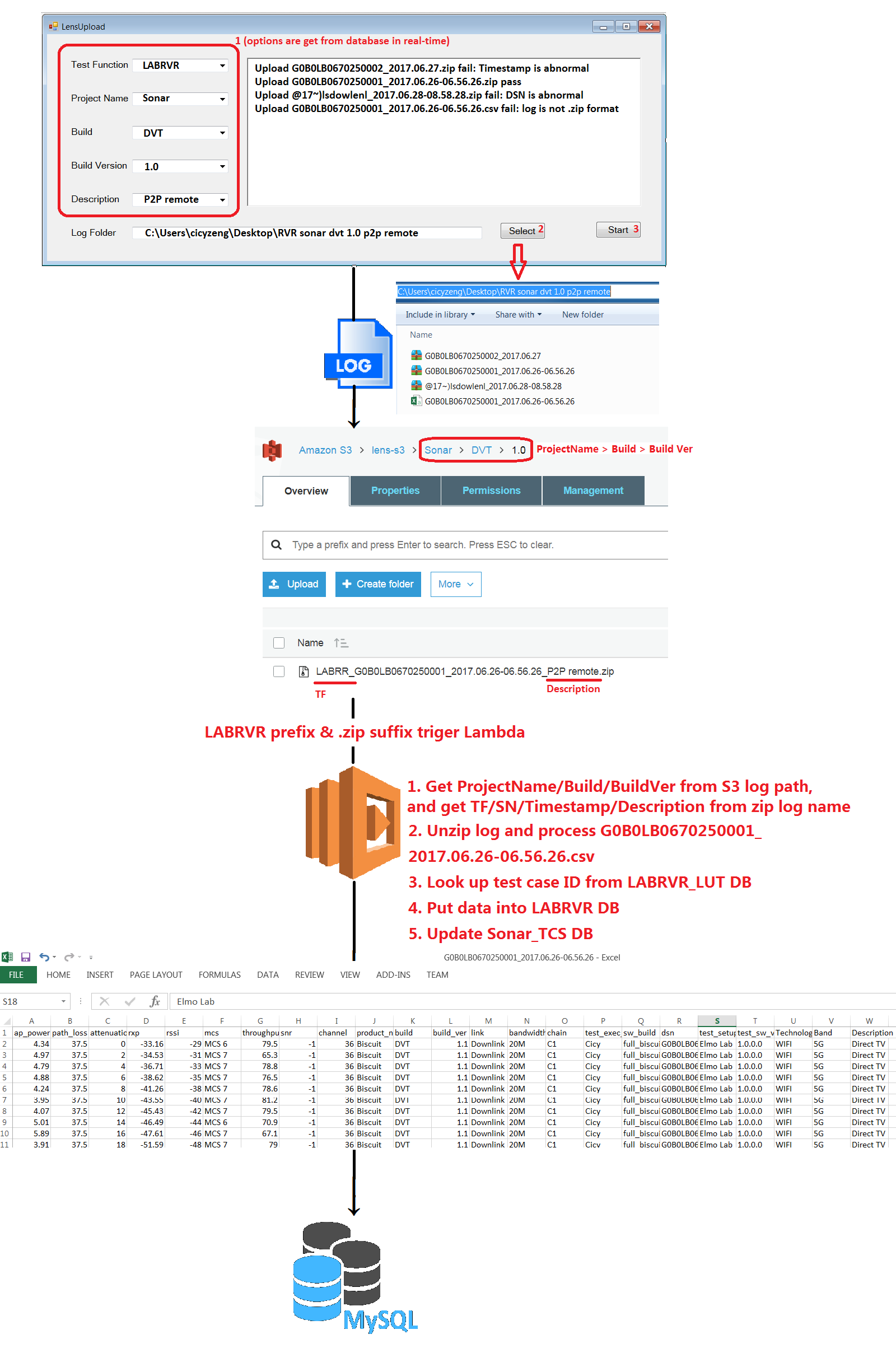
Get **ProjectName/Build/BuildVersion** from S3 log path.

Get **SN/Timestamp/Description** from zip log name.

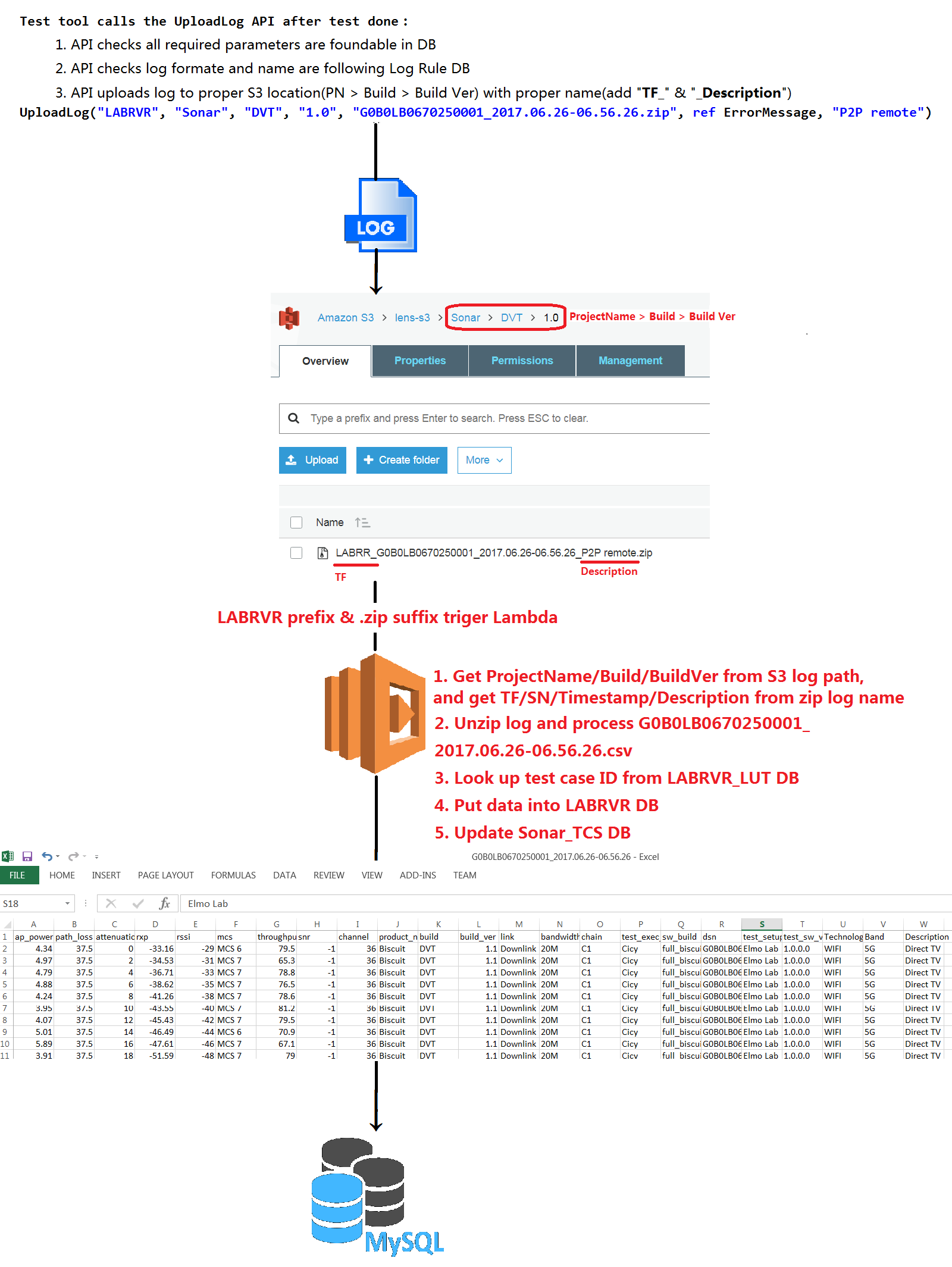
Unzip log.

Process **SN\_Timestamp.csv** to database: for any new headings in csv file, there would be a new column created on database automatically and order does not matter.

1. *Non-real-time upload example:*



1. *Real-time upload example:*

**

### CML RVR

*Log format:*

**SN\_Timestamp.csv**

*S3 structure:*

**ProjectName/Build/BuildVersion/CMLRVR\_SN\_Timestamp\_Description.csv**

*Lambda:*

Save S3 log path to Metadata Status DB

*EC2:*

Fetch unprocessed log by **CMLRVR** prefix& **.csv** suffix log.

Get **ProjectName/Build/BuildVersion** from S3 log path.

Get **SN/Timestamp/Description** from csv log name.

Process **ProjectName/Build/BuildVersion/CMLRVR\_SN\_Timestamp\_Description.csv** to database: for any new headings in csv file, there would be a new column created on database automatically and order does not matter.

### Non Signaling

*Log format:*

**Config\_SN\_****Temprature\_TestItem\_Timestamp\_Note.txt**

* TestItem list:

**Wi-Fi**

Tx\_Multi\_Verification - wtx0, wtx1, wtx2, wtx3, wtxmimo

Tx\_Timing\_Verification - wtiming0, wtiming1, wtiming2, wtiming3

Rx\_SensitivityVerification - wrx0, wrx1, wrx2, wrx3, wrxmimo

Rx\_MaxInputVerification - wmaxsens0, wmaxsens1, wmaxsens2, wmaxsens3

Rx\_ACR - wacr0, wacr1, wacr2, wacr3

Rx\_nonACR - wnonacr0, wnonacr1, wnonacr2, wnonacr3

**BT/BLE**

Tx\_BDR/EDR/LE - btx0, btx1, btx2, btx3

Rx\_Sensitivity\_BDR/EDR/LE - brx0, brx1, brx2, brx3

Rx\_MaxInput\_BDR/EDR/LE - bmaxsens0, bmaxsens1, bmaxsens2, bmaxsens3

**ZigBee**

Tx\_Multi\_Verification - ztx0, ztx1, ztx2, ztx3

Rx\_Sensitivity - zrx0, zrx1, zrx2, zrx3

Rx\_MaxInput - zmaxsens0, zmaxsens1, zmaxsens2, zmaxsens3

*S3 structure:*

**ProjectName/Build/BuildVersion/****NONSIG\_Config\_SN\_Temprature\_TestItem\_Timestamp\_Note\_Description.txt**

*Lambda:*

Save S3 log path to Metadata Status DB

*EC2:*

Fetch unprocessed log by **NONSIG** prefix& **.txt** suffix log.

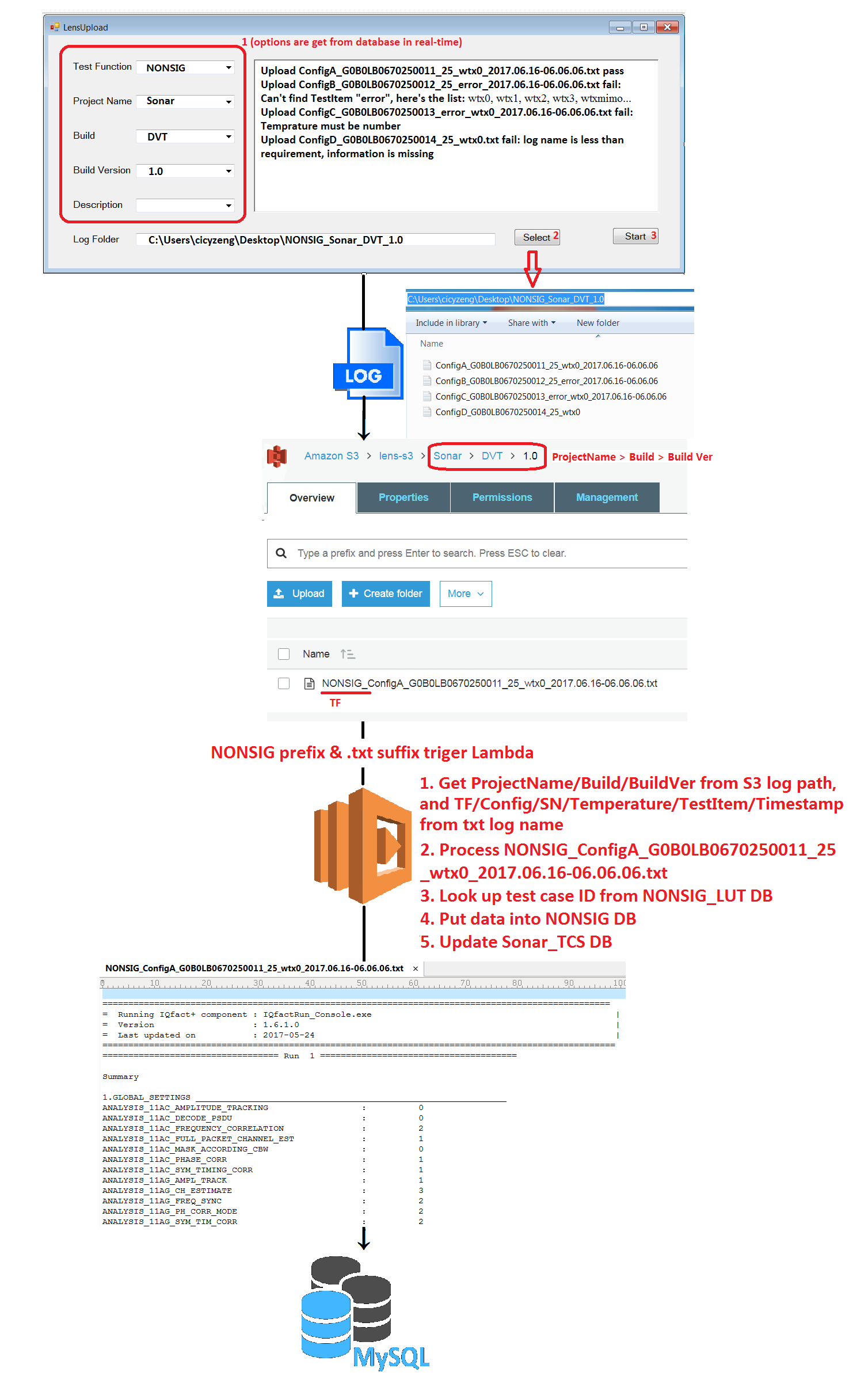
Get **ProjectName/Build/BuildVersion** from S3 log path.

Get **Config/SN/Temprature/TestItem/Timestamp/Note/Description** from txt log name.

Process **ProjectName/Build/BuildVersion/NONSIG\_Config\_SN\_Temprature\_TestItem\_**

**Timestamp\_Note\_Description.txt** to database: for any new test items in txt file, there would be a new column created on database automatically and order does not matter.

1. *Non-real-time upload example:*

**

1. *Real-time upload example:*

**

### Signaling

*Log format:*

**Config\_SN\_TestItem\_Timestamp.txt**

*S3 structure:*

**ProjectName/Build/BuildVer/SIG\_Config\_SN\_TestItem\_Timestamp\_Description.txt**

*Lambda:*

Save S3 log path to Metadata Status DB

*EC2:*

Fetch unprocessed log by **SIG** prefix& **.txt** suffix log.

Get **ProjectName/Build/BuildVersion** from S3 log path.

Get **Config/SN/TestItem/Timestamp/Description** from txt log name.

Process **ProjectName/Build/BuildVer/SIG\_Config\_SN\_TestItem\_Timestamp\_**

**Description.txt** to database: for any new test items in txt file, there would be a new column created on database automatically and order does not matter.

### Congestion

*Log format:*

**SN\_Timestamp.zip** including **.csv inside**

*S3 structure:*

**ProjectName/Build/BuildVersion/CONG\_SN\_Timestamp\_Description.zip**

*Lambda:*

Save S3 log path to Metadata Status DB

*EC2:*

Fetch unprocessed log by **CONG** prefix& **.zip** suffix log.

Get **ProjectName/Build/BuildVersion** from S3 log path.

Get **SN/Timestamp/Description** from zip log name.

Unzip log.

Process **SN\_Timestamp.csv** to database: for any new headings in csv file, there would be a new column created on database automatically and order does not matter.

### Antenna

*Log format:*

**Test\_Chamber\_WirelessStandard\_Channel\_AntennaChain\_DSN\_Timestamp.csv**

*S3 structure:*

**ProjectName/Build/BuildVersion/ANT\_Test\_****Chamber\_WirelessStandard\_Channel\_AntennaChain\_DSN\_Timestamp\_Description.csv**

*Lambda:*

Save S3 log path to Metadata Status DB

*EC2:*

Fetch unprocessed log by **ANT** prefix& **.csv** suffix log.

Get **ProjectName/Build/BuildVersion** from S3 log path.

Get **Test/Chamber/WirelessStandard/Channel/AntennaChain/DSN/Timestamp**

**/Description** from csv log name.

Process **ProjectName/Build/BuildVersion/ANT\_Test\_Chamber\_WirelessStandard\_**

**Channel\_AntennaChain\_DSN\_Timestamp\_Description.csv** to database: for any new headings in csv file, there would be a new column created on database automatically and order does not matter.

### LAB Co-existence

*Log format:*

**SN\_Timestamp.zip** including **.csv inside**

*S3 structure:*

**ProjectName/Build/BuildVersion/LABCOEX\_SN\_Timestamp\_Description.zip**

*Lambda:*

Save S3 log path to Metadata Status DB

*EC2:*

Fetch unprocessed log by **LABCOEX** prefix& **.zip** suffix log.

Get **ProjectName/Build/BuildVersion** from S3 log path.

Get **SN/Timestamp/Description** from zip log name.

Unzip log.

Process **SN\_Timestamp.csv** to database: for any new headings in csv file, there would be a new column created on database automatically and order does not matter.

### CML Co-existence

*Log format:*

**SN\_Timestamp.csv**

*S3 structure:*

**ProjectName/Build/BuildVersion/CMLCOEX\_SN\_Timestamp\_Description.csv**

*Lambda:*

Save S3 log path to Metadata Status DB

*EC2:*

Fetch unprocessed log by **CMLCOEX** prefix& **.csv** suffix log.

Get **ProjectName/Build/BuildVersion** from S3 log path.

Get **SN/Timestamp/Description** from csv log name.

Process **ProjectName/Build/BuildVersion/CMLCOEX\_SN\_Timestamp\_Description.csv** to database: for any new headings in csv file, there would be a new column created on database automatically and order does not matter.

# Database Table

## Log Rule DB:

In order to minimum data pipeline mistake, UploadLog() API checks log format and name following the Log Rule DB.

Log Naming Rule Basic Principles:

1. No space in log name
2. No “@” in log name, becase it’s used for identify the check item
3. “\_” is used for splitting each rule itme

Log Rule DB:



Rules: Test function columns set log format, log item number and rule for each item, “@” tail columns set the list for required log item.

1. LABRVR: SN\_Timestamp.zip

* Log item number is 2
* SN is a string contains A to Z, a to z, 0 to 9 only
* Timestamp format is %Y.%m.%d-%H.%M.%S, eg: 2017.06.16-06.06.06
* Log format is .zip

1. CMLRVR: SN\_Timestamp.csv
2. NONSIG: Config\_SN\_Temprature\_**TestItem**\_Timestamp\_Note.txt

* Log item number is 6
* “NoRule” for Config
* SN is a string contains A to Z, a to z, 0 to 9 only
* Temprature is a string that contains number 0 to 9 only and a negative sign is optional
* **TestItem has “@” tail, it should be foundable in NONSIG\_TestItem@ list**
* Timestamp format is %Y.%m.%d-%H.%M.%S, eg: 2017.06.16-06.06.06
* “NoRule” for Note
* Log format is .txt

1. SIG: Config\_SN\_TestItem\_Timestamp.txt
2. CONG: SN\_Timestamp.zip
3. ANT: Test\_Chamber\_WirelessStandard\_Channel\_AntennaChain\_DSN\_Timestamp.csv
4. LABCOEX: SN\_Timestamp.zip
5. CMLCOEX: SN\_Timestamp.csv

## Metadata Status DB

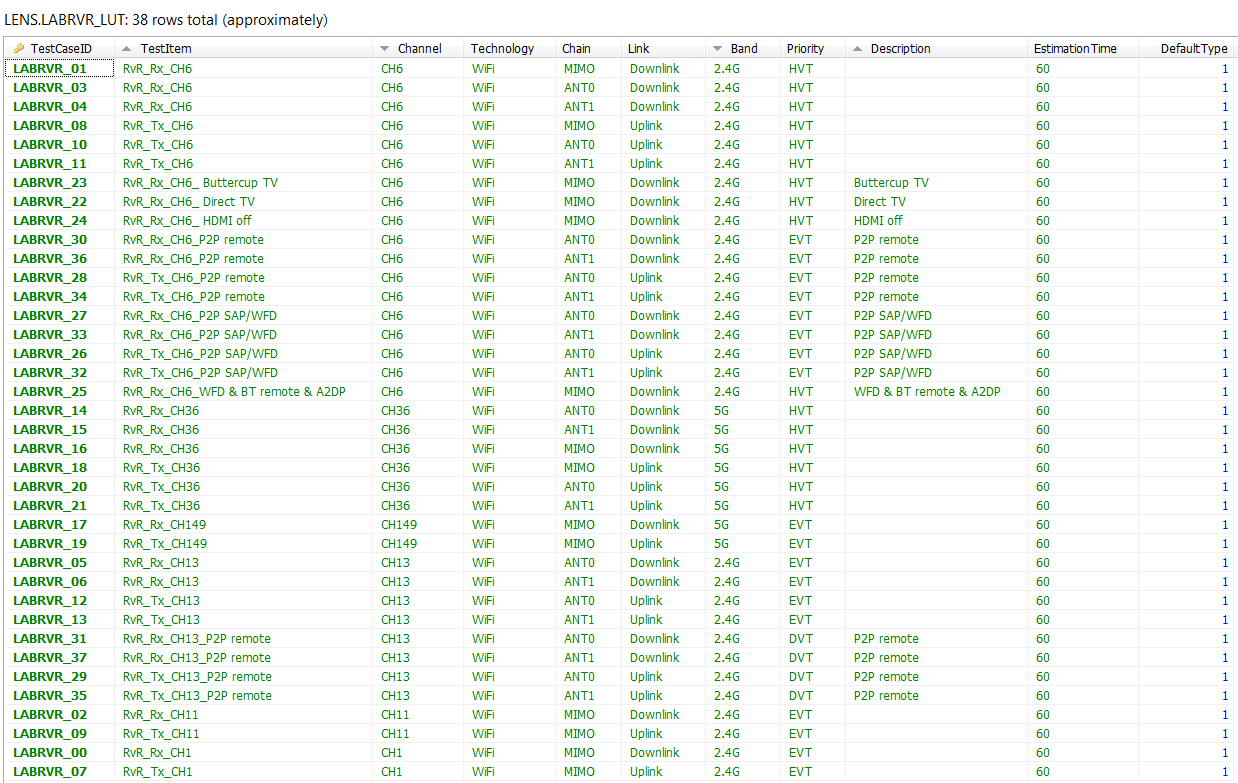
Track all metadata: 0 means unprocessed, 1 means processed.



## Test Function LUT DB

Look up for test case ID:

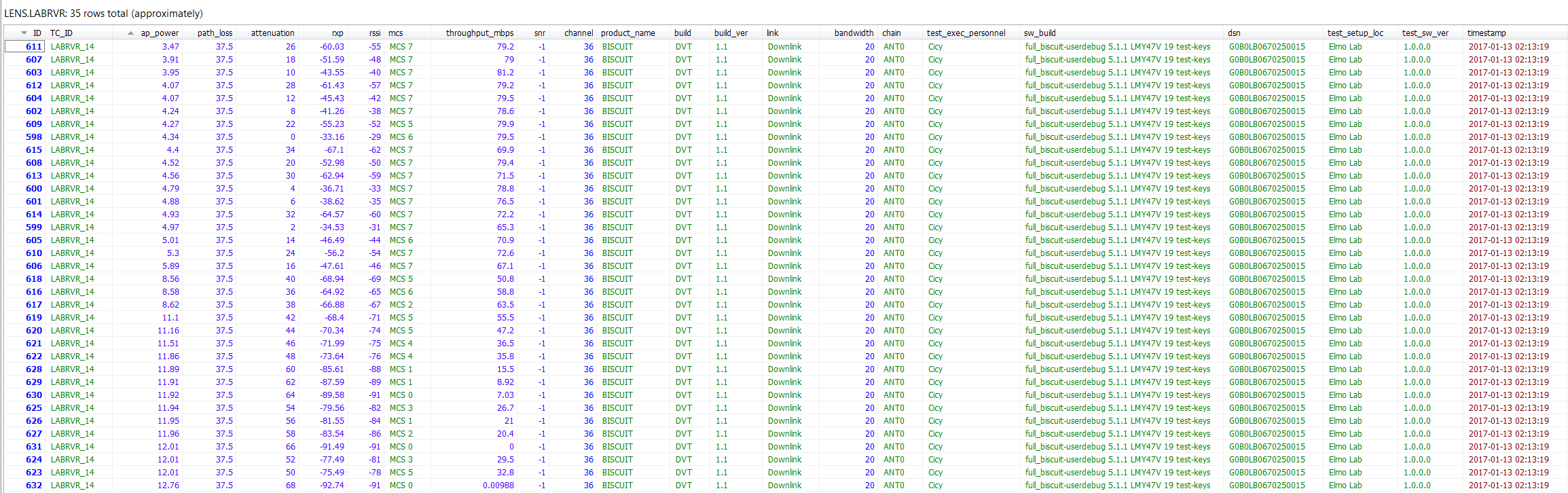
1. Name rule: TF\_LUT
2. Take LABRVR\_LUT for example:



## Test Function DB

Record all test data:

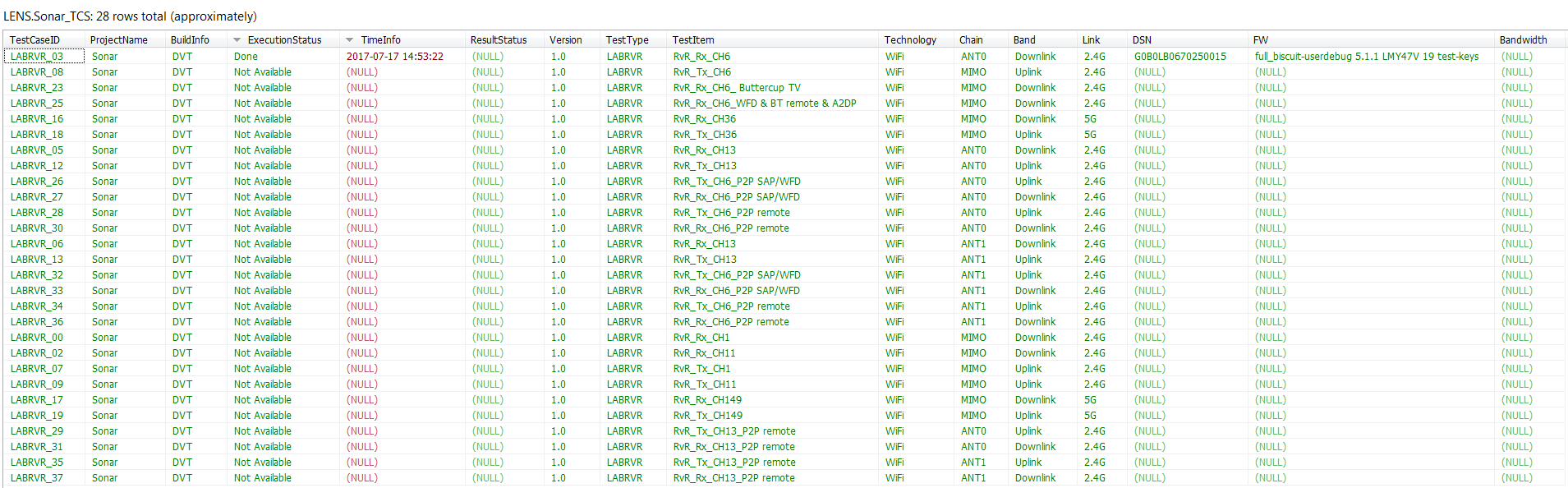
1. Name rule: TF
2. Take LABRVR for example:



## Test Case Status DB

Track test case status:

1. Name rule: ProjectName\_TCS
2. Take Sonar\_TCS for example:



# Authorization & Authentication

The upload utility will get information from database, so upload point IP needs to get approval for accessing database firstly.

# Time Estimate

## Table of Milestone:



|  |
| --- |
| PASS/FAIL TBD: because for RVR/Congestion/ANT, no spec settle down yet |
| For ANT: due to Linyang and Cicy are going to take PTO back to China, can’t guarantee it could hit the release date |

## Table of Detail Schedule:



|  |
| --- |
| JIRA TBD: because for NONSIG/SIG/ANT, no JIRA solution settle down yet |
| For ANT: due to Linyang and Cicy are going to take PTO back to China, can’t guarantee it could hit the release date |
| For CMLRVR & LABRVR, dut to adding 2 project overview page, timeline will delay 4 days |

## Table of WF/DP:

