

**SALCOMP CUSTOM MES (cMES) v1.5**

**cMES MACHINE INTERFACE SPECIFICATIONS**



****

**Revision History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date** | **Version** | **Changes** | **Created By** | **Modified By** |
| 25-May-20 | 1.1 | Base Version | Prakash Elumalai |  |
| 15-Sep-20 | 1.2 | Fixture/Jig Serial Number Linking related methods included (9.0,10.0,11.0 & 12.0) |  | Saravanan Janarthanan |
| 22-Sep-20 | 1.3 | New method GetStatusForSerialNumber(SerialNumber) added |  | Saravanan Janarthanan |
| 16-Oct-20 | 1.4 | New method GetSNLocation(SerialNumber) added |  | Saravanan Janarthanan |
| 02-Nov-20 | 1.5 | New method GetSNCurrentStatus(SerialNumber) added |  | Saravanan Janarthanan |

**Overview**

Salcomp has a custom MES Solution (cMES) which handles the SFCS and MES requirements for Salcomp. cMES supports Material Traceability, Product traceability and Routing control requirements for manufacturing.

cMES interfaces with Line equipment (Machines) for routing control and test & traceability data collection.

This document lists out the interfaces that cMES has with equipments and the methods or specifications for implementing these interfaces.

|  |  |  |
| --- | --- | --- |
| **Table of Contents** | | |
| 1.0 | **Machines Interfaces from cMES** | **03** |
| 2.0 | **Route Check** | **03** |
| 2.1 | **CMESAPI.dll & Methods** | **03** |
| 2.1.1 | **Using Only Serial Number for Route Check** | **03** |
| 2.1.2 | **Using Serial Number and & Equipment ID** | **03** |
| 2.2 | **Config File** | **04** |
| 2.3 | **How to Invoke CMESAPI API.dll in any C# application** | **05** |
| 2.3.1 | **Import the CMESAPI.dll as a reference to your project** | **05** |
| 2.4 | **Sample code for Route Check** | **05** |
| 2.4.1 | **Sample code to use method 1 with only Serial Number** | **05** |
| 2.4.2 | **Sample code to use method 2 - Using Serial Number and equipment** | **05** |
| 3.0 | **Test and Process Data Collection** | **06** |
| 3.1 | **Schema definition for the xml file (Xsd) for reference** | **06** |
| 3.2 | **Pass File format** | **06** |
| 3.3 | **Fail File format** | **08** |
| 4.0 | **Get Block Serial Numbers from Panel ID** | **09** |
| 4.1 | **Method Description** | **09** |
| 4.2 | **Calling Method from C# (Sample)** | **09** |
| 4.3 | **Method response and Handling** | **10** |
| 5.0 | **Get SerialNumber** | **11** |
| 6.0 | **Get SerialNumber from Carrier ID** | **11** |
| 7.0 | **Check Route with Alias Output** | **12** |
| 8.0 | **Generate Serial Number** | **12** |
| 9.0 | **GetSNsFromFixture** | **13** |
| 10.0 | **LinkFixturetoSerialNumber** | **14** |
| 11.0 | **DeLinkFixtureFromSerialNumber** | **14** |
| 12.0 | **ReLinkFixtureToSerialNumber** | **15** |
| 13.0 | **GetStatusForSerialNumber** | **15** |
| 14.0 | **GetSNLocation** | **16** |
| 15.0 | **GetSNCurrentStatus** | **17** |
| 15.0 | **Salcomp Contacts for Clarifications** | **17** |
| 16.0 | **Glossary** | **18** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

1. **Machines Interfaces from cMES**

Salcomp has 2 types of direct interfaces to collect data and provide information to equipment for the following,

1. **Route Check**
2. **Test and Process Data collection**
3. **Route Check**

Salcomp MES supports routing checks to ensure that only passed chargers are moving from one stage to the next. This is implemented using a dll that needs to be called by Tester software and invoking a method by passing the Serial number and the equipment ID.

* 1. **CMESAPI.dll & Methods**

CMESAPI.dll file has the route check methods embedded in the file. Please consider the file CMES RouteCheck API for getting the dll for new implementations.

For testers already integrated with MentorAPI.dll, you can take the CMES API with the Mentor Name package.

There are 2 methods available for doing the route check

The dll and associated files are in an attached repository(Currently MentorAPI.DLL was not atttached with this document due to make use of CMESAPI.DLL)



* + 1. **Using Only Serial Number for Route Check**

Method: api.CheckRouteForSerialNumber("SerialNumber")

This method requires only Serial number as an input. In this case, the equipment id would be taken from the config file and the method will check whether the serial number should be allowed to be processed in the equipment id available in the config file

* + 1. **Using Serial Number and & Equipment ID**

Method: api.CheckRouteForSerialNumber("SerialNumber",”Equipment ID”);

This method requires both Pack Serial Number and Equipment ID as a parameter. In this case, the equipment ID given in the config file is not considered and the system checks whether the serial number can be processed in the equipment ID that is passed to the dll.

* 1. **Config File**



The MentorAPI.dll uses the above config file to get the necessary information related to equipment in case the 1st method is used.

<?xml version="1.0" encoding="UTF-8"?>

[<Config>](file:///C:\Users\nairra\AppData\Local\Temp\SalcompMESConfig.xml)

[<MSSServer>](file:///C:\Users\nairra\AppData\Local\Temp\SalcompMESConfig.xml)

<!--Set Rest API Server IP or Name -->

<IPOrName>cngughuaapi</IPOrName>

<Port>6082</Port>

<!--Debug mode, not connect MSS -->

<Debug>No</Debug>

</MSSServer>

[<OperationPlan>](file:///C:\Users\nairra\AppData\Local\Temp\SalcompMESConfig.xml)

<!--Set Equipment number in MSS-->

<EquipmentNo>12</EquipmentNo>

</OperationPlan>

</Config>

A detailed explanation of the config file

<IPOrName>cngughuaapi</IPOrName> - MES Server name for route check- Mandatory

<Port>6082</Port> - API Port for connecting to WebAPI for route check

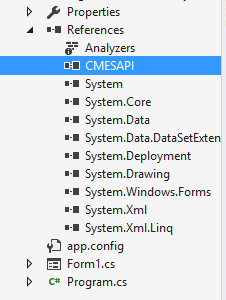
<Debug>No</Debug> - Whether to run in Debug mode or not. For development testing please keep the value as **YES** and route check will return “True” when invoked

<EquipmentNo>12</EquipmentNo> - The Equipment code can be defined here. If the first method is used for route check then the route check will consider the equipment code from this parameter for route check. If the 2nd method is used for route check then this equipment id will be ignored.

* 1. **How to Invoke CMESAPI API.dll in any C# application**

The steps to involve Mentor API.dll is any C# application is as follows

* + 1. **Import the CMESAPI.dll as a reference to your project**



* + 1. **Use dll to access the methods in the dll**
  1. **Sample code for using Route Check**
     1. **Sample code to use method 1 with only Serial Number**

private void CallMESRouteCheck()

{

cSelcompAPI api = new cSelcompAPI(); // Initialization

bool Response = api.CheckRouteForSerialNumber("SerialNumber");

// Calling API

string ErrorMsg = api.Message; //Error msg

}

Response will be

* True - If the Response of the method is OK
* False & message - in api.message if response is not ok
  + 1. **Sample code to use method 2 - Using Serial Number and equipment**

**code.** – This is used when connected equipment are there which share a config file and need to call separately for each equipment

**Method:** api.CheckRouteForSerialNumber ("SerialNumber",” Equipment ID”);

Response will be

* True - If the Response of the method is OK
* False & message - in api.message if the response is not ok

1. **Test and Process Data Collection**

cMES collects test status and test result data from machines by processing the log files created by the machines. cMES has a file processor service that looks up log file destination folders continuously and processes files as soon as the machines create them. The files need to be in a pre-specified format for the file processor service to parse and collect data from log files.

The files can have an extension of .log or .xml, but the structure/construction of the files should be the same.

The following data is mandatory for collecting data

* Work order (WO) (Optional, Mandatory for 1st station)
* Serial Number - Pack Serial Number (PSN) ( Mandatory)
* Cell Serial Number (CSN) ( Optional, Mandatory for loading Station)
* Time Stamp, Start time and End time of the test ( Mandatory)
* Fixture Number ( Optional, Mandatory, where required)
* Test header number, Program name, Version ( Optional)
* Image Path (Optional, Mandatory where images are stored)
* Status – Pass / Fail ( Mandatory)
* Test Results – based on test – format will be provided
* Symptoms – (Optional, Only for failed PSN) format will be provided
* Operator ID – (Optional)
  1. **Schema definition for the xml file (Xsd) for reference**



* 1. **Pass File format**

The XML or log file for a pass PSN should be in the format attached below



Sample data from the XML file with explanations of the fields.

<?xml version="1.0" encoding="UTF-8"?>

[<vCheckTester WorkOrderNumber="**02**" OperatorNumber="**op**" xmlns="**Valor.vCheckTester.xsd**" Version="**4.0**" xmlns:xsd="**http://www.w3.org/2001/XMLSchema**" xmlns:xsi="**http://www.w3.org/2001/XMLSchema-instance**">](file:///C:\Users\nairra\AppData\Local\Temp\Sample%20Pass%20(2).xml)

[<Unit xmlns="**Valor.vCheckTester.xsd**" StatusCode="**PASS**" DateTimeEnd="**2018-04-13T00:18:23**" DateTimeStart="**2018-04-12T22:16:59**" UnitNumber="**CellSerialNumber**" SerialNumber="**PNB001**" Timestamp="**2018-04-13T00:18:23**">](file:///C:\Users\nairra\AppData\Local\Temp\Sample%20Pass%20(2).xml)

<Measurement xmlns="**Valor.vCheckTester.xsd**" StatusCode="**PASS**" Name="**Read Voltage**" Value="**5.4**" ValueUnit="**Double**" UpperLimit="**5.5**" LowerLimit="**4.6**" MeasurementUnit="**V**" DateTime="**2018-04-13T00:18:23**"/>

<Header TestFixtureNumber="**PASS**" TestHeadType="**BR-GGT03**" TestProgramVersion="**18线**" TestHeadNumber="**C:\Logs\Pictuure.jpg**" TestProgramName="**X771-ATL(V1.22)**"/>

</Unit>

</vCheckTester>

The field descriptions are given below

* WorkOrderNumber="02" # Workorder number for product. Mandatory at first station. At other stations can be null
* OperatorNumber="op" # Operator Id logged in. Optional. Can be null
* StatusCode="PASS" # Status of the Pack after the test. Can be PASS or FAIL. Mandatory
* DateTimeEnd="2018-04-13T00:18:23" # End time of test – Mandatory
* DateTimeStart="2018-04-12T22:16:59" # Start time of test – Mandatory
* UnitNumber="CellSerialNumber" # Serial Number of Cell – Optional only for Cell loading
* SerialNumber="PNB001" # Pack Serial Number – Mandatory
* Timestamp="2018-04-13T00:18:23"> # Testing time stamp for tester – Mandatory
* <Measurement – Within Measurement all test values need to be given. Any number of test steps can be included based on the test being given. In case of tests like AOI where only PASS / FAIL is recorded and no test value is there, this can be skipped
  + StatusCode="**PASS**" – Status code of individual test step PASS /FAIL
  + Name="**Read Voltage**" – Test step name as provided by tester
  + Value="**5.4**" – Test result value for the test
  + ValueUnit="**Double**" – Data type for value
  + UpperLimit="**5.5**" - Upper Control Limit
  + LowerLimit="**4.6**" - Lower Control Limit
  + MeasurementUnit="**V**" - Measurement Unit (Volts, Amp, mAmp, Ohms, %, C, F)
  + ValueUnit = “double” - data type Value should be int or double only
  + DateTime="**2018-04-13T00:18:23**" – Time of measurement has to be in format yyyy-MM-ddTHH:mm:ss.fff ( milliseconds are optional)
  + Type = “Component” – no spaces allowed
* Header TestHeadNumber="**C:\YourPictureFilePath\File.jpg**" – Image Path -Optional
* TestFixtureNumber="**0003171968**" – Fixture ID to link to Pack SN - Optional
* TestHeadType="**BR-GGT03**" – Test head type information – Optional
* TestProgramVersion="**18线**" – Test Program Version information – Optional
* TestProgramName="**X771-ATL(V1.22) –** Name of test program – Optional
  1. **Fail File format**

Log format for fail files are slightly different from the pass log file formats. In case any of the measurements are fail, then the overall PSN status should be fail.



In addition to the details available in pass file, for failed test steps, symptoms details should be added as per attached format. Sample details in file shown below with field descriptions.

<?xml version="1.0" encoding="UTF-8"?>

[<vCheckTester OperatorNumber="**operator**" WorkOrderNumber="**WO123**" xmlns="**Valor.vCheckTester.xsd**" Version="**4.0**" xmlns:xsd="**http://www.w3.org/2001/XMLSchema**" xmlns:xsi="**http://www.w3.org/2001/XMLSchema-instance**">](file:///C:\Users\nairra\AppData\Local\Temp\Sample%20Fail.xml)

[<Unit StatusCode="**FAIL**" SerialNumber="**B27086019482S**" Timestamp="**2017-02-24T21:59:13.827+08:00**">](file:///C:\Users\nairra\AppData\Local\Temp\Sample%20Fail.xml)

[<Symptom Type="**Unknown**" Name="**B27086019482S**">](file:///C:\Users\nairra\AppData\Local\Temp\Sample%20Fail.xml)

<Message>C10:不出脚</Message>

<Measurement StatusCode="**FAIL**" Name="**C10:不出脚**" ValueUnit="**Double**" UpperLimit="**100**" LowerLimit="**0**" MeasurementUnit="**S**" Value="**-1**" DateTime="**2017-02-24T21:59:13.827+08:00**"/>

</Symptom>

<Measurement StatusCode="**PASS**" Name="**C10:不出脚**" ValueUnit="**Double**" UpperLimit="**100**" LowerLimit="**0**" MeasurementUnit="**S**" Value="**55**" DateTime="**2017-02-24T21:59:13.827+08:00**"/>

<Header TestFixtureNumber="" TestHeadNumber=""/>

</Unit>

</vCheckTester>

For failed test steps the measurement value should be inside the symptom tags. The following are the field descriptions

Symptom Name ="B27086019482S" - Mandatory – Can be any text provided by Tester

Type = "Unknown" - Mandatory – Should be Unknown / Component / Pin / Contact / Fixture / Short / probe

<Message>C10:不出脚 – Mandatory – Failure reason or some text

Measurement details to be added as usual in case of pass file. Please note if any of the measurements is failed then the overall result should be failed.

1. **Get Block Serial Numbers from Panel ID**

To collect All block serial numbers from the Panel ID(SN)

* 1. **Method Description**

**Method :** GetBlockSerialNumbers(string PanelSN,bool RouteCheck)

* PanelSN = Panel Serialnumber
* Route Check Enable/Disable
  1. **Calling Method from C# (Sample)**

**With Route check for Panel**

/// <summary>

/// Sample Testing Method

/// </summary>

/// <param name="PanelSN"></param>

public void GetBlockSNs(string PanelSN)

{

//Invoking API

/// <summary>

/// GetBlockSerialNumbers

/// </summary>

/// <param name="PanelSN"> Mandatory Input</param>

/// <param name="CheckRoute"> Optional Default true</param>

cSelcompAPI api = new cSelcompAPI();

Dictionary<string, string> dictBlocks =

api.GetBlockSerialNumbers(PanelSN);

// Checking Api Blocks Count

if (dictBlocks.Count() > 0)

{

foreach(string Key in dictBlocks.Keys)

{

// Printing Sequence and SN

Console.Write("Sequence :" + Key + " = SN :" + dictBlocks[Key]);

}

}

else

{

// Error Message

Console.WriteLine("Error Message :" + api.Message);

return;

}

}

**Without Route Check for Panel**

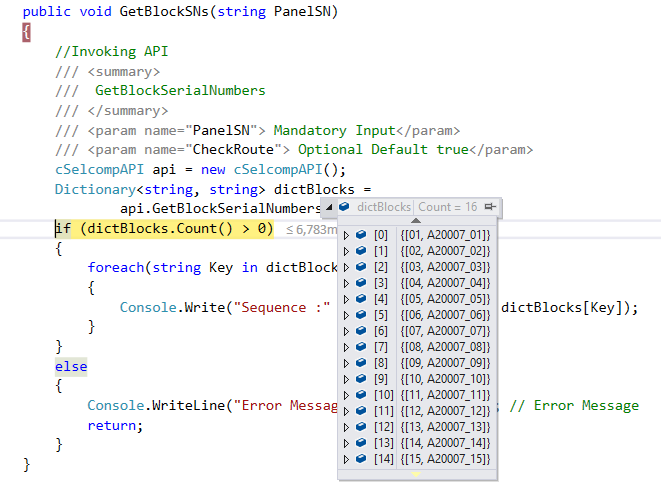
Route check Parameter is Optional 🡪 Default true

cSelcompAPI api = new cSelcompAPI();

Dictionary<string, string> dictBlocks = api.GetBlockSerialNumbers(PanelSN,false);

* 1. **Method response and Handling**
* After the successful invoking method it will return Dictionary type of result set it contains key and values group in this dictionary.
* To get the serial number we need to pass the sequence of the Block into the dictionary then dictionary will return the serial number.
* Incase if it not returned any list of serial number the api.Message will holding the exact error messagge to show in window like same as route check.
* For program convinient they can convert Dictionary in list of Key value pair it will another way to consume the data.
* Refer the below picture for sample response shown in visual studio response window.

API Response snapshot (Visual Studio)



1. **Get SerialNumber**

This method will help you To retrieve all three serialnumber from any one of the serialnumber

Method : GetSerialNumber(string SerialNumber)

* 1. Method Invoking

cSelcompAPI api = new cSelcompAPI();

Dictionary<string, string> Response = api.GetSerialNumber("SERIALNUMBER ");

* 1. Output Response and Handling

cSelcompAPI api = new cSelcompAPI();

Dictionary<string, string> dictSN = api.GetSerialNumber("SERIALNUMBER");

string SN1 = dictSN["SERIALNUMBER"].ToString();

string SN2 = dictSN["SERIALNUMBER2ND"].ToString();

string SN3 = dictSN["SERIALNUMBER3RD"].ToString();

string Message = dictSN["MESSAGE"].ToString();

Incase Dictionary output doesn’t return any serialnumber read the “MESSAGE” from the dictionary it will contains the message to display.

1. **Get SerialNumber from Carrier ID**

This method will gives the All three serialnumbers from last loaded carrier (If the carrier or Test Fixture number was mentioned in log)

Method :GetSerialNumberFromCarrier(string CarrierId)

* 1. Method Invoking

cSelcompAPI api = new cSelcompAPI();

Dictionary<string, string> Response = api.GetSerialNumber("SERIALNUMBER ");

* 1. Output Response and Handling

cSelcompAPI api = new cSelcompAPI();

Dictionary<string, string> dictSN = api.GetSerialNumber("SERIALNUMBER");

string SN1 = dictSN["SERIALNUMBER"].ToString();

string SN2 = dictSN["SERIALNUMBER2ND"].ToString();

string SN3 = dictSN["SERIALNUMBER3RD"].ToString();

string Message = dictSN["MESSAGE"].ToString();

Incase Dictionary output doesn’t return any serialnumber read the “MESSAGE” from the dictionary it will contains the message to display.

1. **CheckRouteWithAliasOutput**
   1. **Method invoking**

private void CallMESRouteCheckWithAliasOutput()

{

cSelcompAPI api = new cSelcompAPI(); // Initialization

bool Response = api.CheckRouteWithAliasOutput ("SerialNumber"); // Calling API

string ErrorMsg = api.Message; //Error msg

string AliasSN = api.SerialNumberAlias; //Alias SerialNumber

}

Response will be

* True - If the Response of the method is OK
* SerialNumberAlias – Alias Serialnumber will be present in this variable
* False & message - in api.message if response is not ok error message will be present in message variable

1. **GenerateSerialNumber**
   1. **Method Invoking**

private void CallingGenerateSerialNumber()

{

cSelcompAPI api = new cSelcompAPI(); // Initialization

bool Response = api.GenerateSerialNumber("workorder", "SeedSN"); // anyone param

string ErrorMsg = api.Message; //Error msg if Response is false

string GeneratedSN = api.Message; // SN if Response id true

}

Response will be

* True - If the Response of the method is OK
* Message – Message contains SN if the response is true
* False & message - in api.message if response is not ok error message will be present in message variable

1. **GetSNsFromFixture**

This method will return the SerialNumbers which is linked with the passed FixtureNumber

Method :GetSNsFromFixture(string FixtureNumber)

* 1. Method Invoking

cSelcompAPI api = new cSelcompAPI();

/// <param name="FixtureNumber"> Mandatory Input</param>

string result = api.GetSNsFromFixture(FixtureNumber);

* 1. Output Response and Handling

cSelcompAPI api = new cSelcompAPI();

result = api.GetSNsFromFixture(FixtureNumber);

The final result has true#Message#SN1,SN2,,SN4, …. or false#Message. First indicates the status & the second one will contains the success/error message to display and last one indicates the Serial Numbers separated with comma.

1. **LinkFixtureToSerialNumber**

This method will link the FixtureNumber with all SerialNumbers

Method :LinkFixturetoSerialNumber(string FixtureNumber, string SerialNumbers,string EquipmentCode)

* 1. Method Invoking

cSelcompAPI api = new cSelcompAPI();

/// <param name="FixtureNumber"> Mandatory Input</param>

/// <param name="SerialNumbers"> Mandatory Input </param>

/// <param name="EquipmentCode"> Mandatory Input</param>

string result = api.LinkFixturetoSerialNumber(FixtureNumber, SerialNumbers,EquipmentCode);

* 1. Output Response and Handling

cSelcompAPI api = new cSelcompAPI();

result = api.LinkFixturetoSerialNumber(FixtureNumber,

SerialNumbers,EquipmentCode);

The final result has “true#Message” or “false#Message”. First indicates the status & the second one will contains the success/error message to display.

1. **DeLinkFixtureFromSerialNumber**

This method will Delink the SerialNumbers from the associated FixtureNumber

Method :DeLinkFixtureFromSerialNumber(string FixtureNumber)

* 1. Method Invoking

cSelcompAPI api = new cSelcompAPI();

/// <param name="FixtureNumber"> Mandatory Input</param>

string result = api. DeLinkFixtureFromSerialNumber(FixtureNumber);

* 1. Output Response and Handling

cSelcompAPI api = new cSelcompAPI();

result = api. DeLinkFixtureFromSerialNumber(FixtureNumber);

The final result has “true#Message” or “false#Message”. First indicates the status & the second one will contains the success/error message to display.

1. **ReLinkFixtureToSerialNumber**

This method will Delink the SerialNumbers from the associated FixtureNumber & Link the FixtureNumber with all SerialNumbers without doing any unit & unit event entries

Method :ReLinkFixtureToSerialNumber(string FixtureNumber, string SerialNumbers,string EquipmentCode)

* 1. Method Invoking

cSelcompAPI api = new cSelcompAPI();

/// <param name="FixtureNumber"> Mandatory Input</param>

/// <param name="SerialNumbers"> Mandatory Input </param>

/// <param name="EquipmentCode"> Mandatory Input</param>

string result = api.ReLinkFixtureToSerialNumber(FixtureNumber, SerialNumbers,EquipmentCode);

* 1. Output Response and Handling

cSelcompAPI api = new cSelcompAPI();

result = api.ReLinkFixtureToSerialNumber(FixtureNumber,

SerialNumbers,EquipmentCode);

The final result has “true#Message” or “false#Message”. First indicates the status & the second one will contains the success/error message to display.

1. **GetStatusForSerialNumber**

This method will return the status of the input SerialNumber for the input EquipmentCode

Method :GetStatusForSerialNumber (string SerialNumber, string EquipmentCode)

* 1. Method Invoking

cSelcompAPI api = new cSelcompAPI();

/// <param name="SerialNumber"> Mandatory Input </param>

/// <param name="EquipmentCode"> Mandatory Input</param>

string result = api. GetStatusForSerialNumber(SerialNumber,EquipmentCode);

* 1. Output Response and Handling

cSelcompAPI api = new cSelcompAPI();

result = api. GetStatusForSerialNumber(SerialNumber,EquipmentCode);

The final result has “true#PASS#<DateTime>” or “false#Message”. First indicates the status & the second one contains the success/error message to display.

1. **GetSNLocation**

This method will return the linked FixtureNumber and the position of input SerialNumber.

Method :GetSNLocation(string SerialNumber)

* 1. Method Invoking

cSelcompAPI api = new cSelcompAPI();

/// <param name=" SerialNumber "> Mandatory Input</param>

string result = api. GetSNLocation(SerialNumber);

* 1. Output Response and Handling

cSelcompAPI api = new cSelcompAPI();

result = api.GetSNLocation(SerialNumbers);

The final result has “true#<FixtureNumber>,<position>” or “false#SerialNumberNotFound”. First indicates the status & the second one will contains the success/error message to display.

1. **GetSNCurrentStatus**

This method will return operation code & current status of input SerialNumber.

Method :GetSNCurrentStatus(string SerialNumber)

* 1. Method Invoking

cSelcompAPI api = new cSelcompAPI();

/// <param name=" SerialNumber "> Mandatory Input</param>

string result = api. GetSNCurrentStatus(SerialNumber);

* 1. Output Response and Handling

cSelcompAPI api = new cSelcompAPI();

result = api.GetSNCurrentStatus(SerialNumber);

The final result has “true#<Operation Code>#<Status>” or “false#SerialNumberNotFound”. First indicates the status & the second one will contains the success/error message to display.

1. **Salcomp Contacts for Clarifications**

The following Salcomp IT person can be contacted for any technical clarifications.

* Ranjeev Nair – General Manager – Global IT Applications– [ranjeev.nair@salcomp.com](mailto:ranjeev.nair@salcomp.com)

1. **Glossary**

* PSN – Pack Serial Number
* CSN – Cell Serial Number
* Fixture Number – Fixture ID
* cMES – Salcomp Custom MES Solution
* log file – Tester output file in predefined format .log or .xml extension
* Carrier Number – Pallet number(ID)
* Test Fixture – Jig/Carrier/Any stationary support