NEDA nSTD Writer
开发手册

A Light and Intelligent Solution

NEDA nSTD Writer
开发手册

©2021, Nornion, Co. Ltd.

All rights reserved.

First Printing, September 2024

Document Number: NL-003-01 Rev. D

http://www.nornion.com

目录

1 开始

NEDA nSTD Writer 是什么 1-1
开发环境说明.. 1-2
nSTD.dll 1-3
创建 Visual Studio 项目工程... 1-4

2 手册

命名空间... 2-1
nSTD 类 2-2
初始化和标志位 2-3
内置数据结构... 2-4
创建和写入 STDF 的流程.. 2-5
方法详细介绍... 2-6

1
开始

- NEDA nSTD Writer 是什么.
- 开发环境说明.
- nSTD.dll.
- 创建 Visual Studio 新项目工程

NEDA nSTD Writer 是什么？

NEDA nSTD Writer (nstd.dll) 是一个 STDF 创建控件，可以用来生成 STDF 文件，此控件主要应用于 Windows 平台测试机上，可供测试机环境软件生成 STDF 测试数据。阅读此文档时，需要您对 STDF 的基本记录有一些了解。

我们提供过了一个用 nSTD 创建 STDF 的完整的 Visual Studio (C#)工程，请下载参考。

开发环境说明

开发语言和环境:

- Microsoft .NET 4.0 或者以上.
- Visual Studio 2017 或者更高版本.
- 支持.NET 环境下的编程语言 C#, VB, C++等.

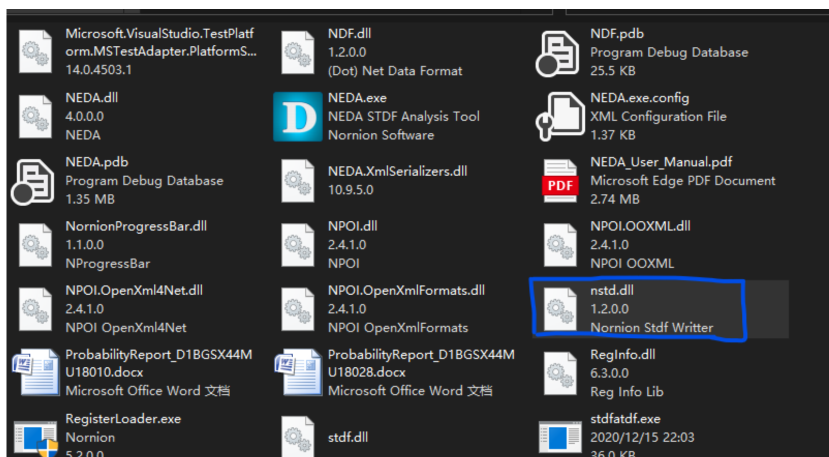
NEDA 控件环境:

- 需要安装 NEDA Desktop Edition 最新版本并具有有效授权 (可以申请 1 个月的试用授权来开发或者学习

nSTD.dll

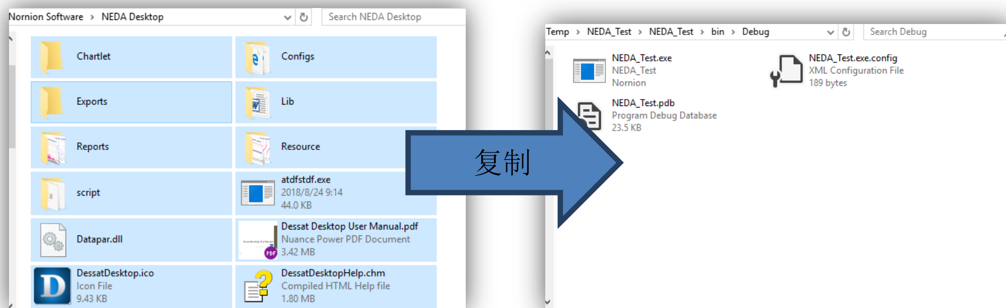
在安装好 NEDA Desktop Edition STDF 分析工具之后，你可以在安装目录下找到 nstd.dll，这个就是我们需要的 NEDA nSTD Writer 控件，我们可以用它来做二次开发，通过它把测试数据写入 STDF 文件。

如：C://Program Files(x86)/Nornion Software/NEDA Desktop/

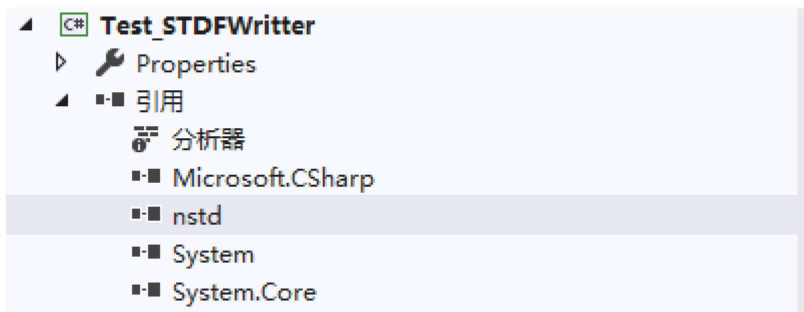


创建 Visual Studio 项目工程

创建一个 Visual Studio 新项目，并把 NEDA Desktop Edition 安装目录下的所有文件复制到新建工程的 bin/Debug 目录下。



在项目中添加对 nstd.dll 对的引用



2 手册

- 命名空间.
- nstd 类.
- 初始化和标志位
- 内置数据结构
- 创建和写入 STDF 的流程
- 方法详细介绍

命名空间

引入对 nstd.dll 的引用后，我们的 nstd 类在 Nornion 命名空间里面。

```
string file = "c:\\temp\\sample.std";
nstd std = new nstd(file);
```

nstd 类

其中类 nstd 就是我们需要调用的主要的类，用 nstd 类完成我们把测试数据写入 STDF 的操作

初始化和标志位

Bool ErrorFlag, string ErrorMsg: 这是一个重要的 flag，每次操作之后需要检查是否被置位(ErrorFlag=true),如果被置位表示操作过程中有异常发生，错误信息保存在 ErrorMsg 中。

我们主要的操作有 2 个：初始化和解析，所以在初始化和解析之后都需要检查 ErrorFlag。

在我们创建 nstd 对象后，在开始写入 STDF 之前，需要初始化一下 nstd 对象，并检查标志位，只有初始化成功后才能开始 STDF 写入操作

```
if (std.InitFile())
{
    //向 STDF 写入数据的所有操作
}
```

内置数据结构

为了便于在二次开发的时候从客户编程环境向 nSTD 对象传递数据，我们定了一些数据存储对象，这里简单介绍一下这些对象的作用和使用。

1. **Header**：用于存储 STDF 头信息的所有字段，并传给 nSTD 完成开批操作。在 StartLot()的时候使用

   构造函数：public Header(List<byte> SiteNumList)，在创建 Header 对象的时候，需要把 Site 的 List 传入。

   属性：byte STAT_NUM 测试头号，默认 STAT_NUM=1

   char MODE_COD Mode Code，默认为一个白空格

   char RTST_COD Retest Code，默认为一个白空格

   char PROT_COD Protect Code，默认为一个白空格

   UInt16 BURN_TIM 老化时间，默认 65535，一般不需要设置

   char CMOD_COD Command Code，默认为一个白空格

string LOT_ID 生产的批次号，一般放客户批次号，**必须设置**

string PART_TYP 产品型号，设置为 Device Name, **必须设置**

string NODE_NAM 测试机编号，测试机的具体编号，**必须设置**

string TSTR_TYP 测试机型号，比如 J750/UltraFlex/HP93K/DimondX

string JOB_NAM 测试程序名，**必须设置**

string JOB_REV 测试程序版本

string SBLOT_ID 子批号，一般放工厂批次号

string OPER_NAM 操作员工号

string EXEC_TYP 测试机系统类型，比如 SmarTest/IGXL/Unison/Image

string EXEC_TYP 测试机系统环境的版本

string TEST_CODE Test Code, 放一些编码表示当前是哪个测试环节，如 FT, RT1, QC1

string TST_TEMP 测试温度

下面还有一些字符串类型的字段，这里就不一一解释了

USER_TXT，AUX_FILE,PKG_TYP, FAMLY_ID, DATE_COD, FACIL_ID, FLOOR_ID, PROC_ID, OPER_FRQ, SPEC_NAM, SPEC_REV, FLOW_ID, SETUP_ID, DSGN_REV, ENG_ID, ROM_COD, SERL_NUM, SUPR_NAM

还有一些其他和硬件相关的字段需要注意的，这些都是用来存储相关硬件的编号和类型的。

HAND_TYP, HAND_ID, CARD_TYP, CARD_ID, LOAD_TYP, LOAD_ID, DIB_TYP, DIB_ID, CABL_TYP, CABL_ID, CONT_TYP, CONT_ID, LASR_TYP, LASR_ID, EXTR_TYP, EXTR_ID

2. **Pin**: 用于存储 Pin Definition 的数据，以便传入 nSTD 写入 STDF 的 PMR， 这个在 MPR（多 Pin 测试记录）解析的时候会被调用。

   属性：UInt16 PinIndex Pin 的索引，这个需要是唯一的索引

   UInt16 PinType Pin 的类型，这个测试机软件自定义的数字，用于区别不同类型硬件资源

   String ChanNum Device Pin 对应的测试机硬件资源的 Channel 编号

   String PinName Device 的 Pin 脚的名称

   Byte HEAD_NUM 测试头号

   Byte SITE_NUM 测试工位号

3. **PinGroup**: 用于存储 Pin Group 数据，并传入 nSTD 写入 STDF, PinGroup 不是必要信息

   属性：

String GroupName 组名称，比如 PowerPins, DigitalPins 等

List<UInt16> PinIndexList 组所包含的 Pin 的 Index 的 list

4. PartResult:每个 Die/Unit 测试后的结果信息，包含 HBin, SBin, X，Y 坐标等，在 EndPart()的时候使用

属性：

Byte HEAD_NUM 测试头号，默认为 1

Bool PassFlag 这个 Die/Unit 测试是 Pass 还是 Fail，默认是 True (Pass)

UInt16 HARD_BIN Hardware Bin 编号，当前这个 Die/Unit 被分到哪个硬件 Bin

UInt16 SOFT_BIN Software Bin 编号，当前 Die/Unit 被分到哪个软件 Bin

Int16 X_COORD 当前 Die 的 X 坐标，Wafer Sort 的时候需要设置

Int16 Y_COORD 当前 Die 的 Y 坐标，Wafer Sort 的时候需要设置

UInt32 TEST_T 当前 Die/Unit 的测试时间信息，（单位毫秒）

String PART_ID 当前 Die/Unit 的测试序列号

String PART_TXT 当前 Die/Unit 的额外信息，一般存储 DIE ID

5. ParaTest: 参数测试项对应的数据结构，它和 sParaTest 是一对。在测试项结果第一次写入 STDF 是需要指定 Limit 和 Unit 信息，再次写入的时候就不需要了（用 sParaTest）。

属性：

Byte HEAD_NUM 测试头号，默认为 1

UInt TEST_NUM 测试项编号，不同测试项的 TEST_NUM 必须不一样，否则会导致解析系统混乱

Bool PassFlag 测试项结果是否 Pass, 这个信息不是很重要，可以不指定，一般分析软件可以通过 Spec Limit 来判断最终的 fail

Double RESULT 测试结果的值，必须赋值

String TEST_TXT 测试项的名称。如果在这里面需要包含测试函数名称，请遵循这个格式：Test_Name <> Test_Function

Double LO_LIMIT 测试下限，如果没有下限可以不指定

Double HI_LIMIT 测试上线，如果没有上限可以不指定

String UNITS 测试项数值的单位，如 mV, uA, ohm, dB, VOLTS, KHz 等，前面的系数必须遵循下面的规则：T (Tera), G (Giga), M (Mega 兆), K (kilo 千), m (mili 毫), u (micro 微), n (nano 纳), p (Pico 皮)

6. sParaTest: Short ParaTest, 短的参数测试项信息， 省略了 Limit 和 Unit 信息

属性：TEST_NUM, HEAD_NUM, PassFlag, RESULT, TEST_TXT

7. **FuncTest**: 数字测试项的结果，数字测试项可以存储比较详细的 Digital 结果，但是一般使用的时候只是放 Pass/Fail，用来表示 Pattern 执行是否 pass。

   属性：

   Byte HEAD_NUM　　测试头号，默认 1

   UInt32 TEST_NUM 测试项编号，请保证 TEST_NUM 的唯一，即不同测试项有不同的 TEST_NUM

   Bool PassFlag:　Pass/Fail flag, 默认 true(pass), 测试项 fail 的时候需要设置为 false

   String VECT_NAM Vector Name, Pattern 的名称

   String TIME_SET　Timing set 的名称

   String TEST_TXT　测试项名称，如果在这里面需要包含测试函数名称，请遵循这个格式：Test_Name <> Test_Function

8. **MultiPinTest**: 多 Pin 测试项，这种测试项结果类似于 Parametric 测试项，但是 MultiPinTest 的一个测试项中包含多个 Pin 的结果(多个测试结果)，但是用同样的 Limit 和 Unit。

   Note: 这个在当前大型测试机中（HP93K,UltraFlex）中用的比较多。当今很多芯片多达几百个甚至上千个 pin 脚。在测试 Leakage 的时候，一般都是很多 pin 同时测试的，Spec limit 也一样。所以一个测试项测完时，每个 pin 都会有一个测量结果。

   属性：

   Byte HEAD_NUM　　测试头号，默认为 1

   UInt32 TEST_NUM 测试项编号，不同测试项请用不同的 TEST_NUM

   Bool PassFlag　　测试结果 Pass 还是 fail，一般不需要设置

   Double[] ResultArray　　测试结果的数组，每个 pin 一个测试结果

   UInt16[] PinIndexArray　PinIndex 的数组，PinIndex 是 Pin 对象的 Index，必须和 ResultArray 数组一一对应

   String TEST_TXT　测试项名称，如果需要包含测试函数名称：Test_Name <> Test_Function

   Double LO_LIMIT　测试项的下限，如果没有可以不设置

   Double HI_LIMIT　测试项的上限，如果没有，可以不设置

   String UNITS　　测试结果的单位，参看 ParaTest 的 UNITS

9. **sMultiPinTest**: Short MultiPinTest, 短的 MultiPinTest 结构，省略了 Limit， Unit 和 PinIndexArray 信息

   属性：TEST_NUM, HEAD_NUM, PassFlag, ResultArray, TEST_TXT

10. **Bin**: 用于存储 HBin 和 SBin 数据，用于在 STDF 末尾写入 Summary 信息，需要为每个 Site 的每个 Bin 写入一条信息。nSTD 会自动为每个 Bin 创建一个 All Site 的数据并写入

    属性：

    Byte HEAD_NUM 测试头号，默认为 1

    Byte SITE_NUM 工位号

    UInt16 BIN_NUM Bin 的号码

    UInt32 BIN_CNT Bin 的数量

    Char BIN_PF 用以区别当前 Bin 是 Pass Bin 还是 Fail Bin, 请使用'P'和 'F'

    String BIN_NAM Bin 的名称，如 GOOD, DC_FAIL, Digital_FAIL 等

11. **TestCount**: 用于在 STDF 末尾写入 Summary 的测试项数量统计信息，每个 site 的每个测试项需要一条信息，nSTD 会自动为每个测试项创建一个 All Site 的数据并写入

    属性：

    Byte HEAD_NUM 测试头号

    Byte SITE_NUM 工位号

    Char TEST_TYP 测试项的类型：'P'- Parametric, 'F'- Function, 'M'- MultiPin

    UInt32 TEST_NUM 测试项编号

    UInt32 EXEC_CNT 执行过当前测试项的 Unit 的数量

    UInt32 FAIL_CNT Fail 过当前测试项的 Unit 的数量

    String TEST_NAM 测试项名

    String SEQ_NAM Sequence Name, 可选

12. **PartCount**：记录每个 site 的测试数量信息，每个 site 一条记录，nStd 会自动添加一个 All Site 的记录

    Byte HEAD_NUM 测试头号

    byte SITE_NUM 测试工位号

    UInt32 PART_CNT 当前工位的测试总数，**必须设置**

    UInt32 RTST_CNT 当前工位的复测的数量

    UInt32 ABRT_CNT 当前工位每个完成测试的数量，可以不设置

    UInt32 GOOD_CNT当前工位的好品数量，**必须设置**

    UInt32 FUNC_CNT 当前工位的 Fail 在 Function 测试的数量，可以不设置

13. WaferConfig：存放 Wafer 的相关信息，在 Start Wafer 的时候写入 STDF

Byte	HEAD_NUM	测试头号，默认 1
String	WAFER_ID	Wafer ID 必须设置
Float	WAFR_SIZ	Wafer Size
Float	DIE_HT	Die 高度
Float	DIE_WID	Die 宽度
Byte	WF_UNITS	0 – unknown, 1 – inches, 2 – centimeter, 3 – millimeter
Char	WF_FLAT	U – Up, D – Down, L – Left, R – Right
Int16	CENTER_X	中心 Die X (一般不需要设置)
Int16	CENTER_Y	中心 Die Y (一般不需要设置)
Char	POS_X	X 增加的方向，一般为 R, 也可以不设置
Char	POS_Y	Y 增加的方向，一般为 D,也可以不设置

14. WaferResult: 存放 Wafer 结果相关的信息，在 End Wafer 的时候会被写入 STDF。

Byte	HEAD_NUM	测试头号，默认 1
UInt32	PART_CNT	当前 wafer 的 die 总数量，必须设置
UInt32	RTST_CNT	当前 wafer 的复测数量
UInt32	ABRT_CNT	Abort 数量，测试中断数量
UInt32	GOOD_CNT	良品数量， 必须设置
UInt32	FUNC_CNT	Function fail 的数量
String	WAFER_ID	Wafer ID, 必须设置
String	FABWF_ID	Fab Wafer ID
String	FRAME_ID	框架编号
String	MASK_ID	MASK 编号
String	USR_DESC	用户自定义的描述
String	EXC_DESC	系统描述

创建和写入 STDF 的流程

STDF 是一种结构化的数据，这里需要开发者对于芯片测试和测试数据有一定的了解，才能更加准确地写入相应的数据。这里简单介绍一些芯片测试数据的相关知识。

STDF 的一个文件一般情况对应于一个 LOT，一个 lot 中可能包含多片 Wafer (一般 25 片)，一片 wafer 又包含很多颗 Unit，每个 Unit 的数据又包含很多个测试项的数据。

所以用 nSTD 创建和写入 STDF 的流程如下 (其中蓝色部分需要循环执行，每个 Unit/Die 执行一次)

Start Lot -> [Start Wafer] -> Start Part -> Write Data -> End Part -> [End Wafer] -> End Lot

注：其中 Start Wafer 和 End Wafer 在 FT 的数据中是不需要的

方法详细介绍

1. 创建 nSTD 对象

调用构造函数的时候传入将要创建的 STDF 文件的详细路径。

```
nstd std = new nstd(file);
```

紧跟着需要用 InitFile()方法来检验是否创建 STDF 文件成功，如果不成功可以显示 ErrorMsg 的错误信息。

```
if (std.InitFile())
{
//所有操作
}
else
{
 Label1.Text = std.ErrorMsg;//显示错误信息
}
```

2. 开批操作

调用 StartLot()方法来实现开批操作，同时通过 Header 对象把所需要写入的信息传如 nSTD。详细信息请参考 Header 对象的介绍。

```
std.StartLot(new Header(new List<byte>() { 0, 1, 2, 3 })
 {
     STAT_NUM = 1,
     HEAD_NUM = 1,
     LOT_ID = "xLOT",
     PART_TYP="XP669901",
     FACIL_ID="Nornion",
     JOB_NAM = "FT_QS_XP669901_A01",
     NODE_NAM="J750-01",
     //... Any MIR/SDR fields you want to set ...
 });
```

写入 Pin Definition 信息，把 Pin Map 的相关信息写入 STDF，记得 PinIndex 唯一，在 MultiPinTest 数据中会使用到 PinIndex 信息。

```
std.WritePinMaps(new List<Pin>()
{
    new Pin(){PinIndex=1, PinName="VDD", PinType=100, ChanNum="101", HEAD_NUM=1, SITE_NUM=0},
    new Pin(){PinIndex=2, PinName="VDD", PinType=100, ChanNum="102", HEAD_NUM=1, SITE_NUM=1},
    new Pin(){PinIndex=3, PinName="VDD", PinType=100, ChanNum="103", HEAD_NUM=1, SITE_NUM=2},
    new Pin(){PinIndex=4, PinName="VDD", PinType=100, ChanNum="104", HEAD_NUM=1, SITE_NUM=3},
    new Pin(){PinIndex=5, PinName="SCLK", PinType=200, ChanNum="201", HEAD_NUM=1, SITE_NUM=0},
    new Pin(){PinIndex=6, PinName="SCLK", PinType=200, ChanNum="202", HEAD_NUM=1, SITE_NUM=1},
    new Pin(){PinIndex=7, PinName="SCLK", PinType=200, ChanNum="203", HEAD_NUM=1, SITE_NUM=2},
    new Pin(){PinIndex=8, PinName="SCLK", PinType=200, ChanNum="204", HEAD_NUM=1, SITE_NUM=3},
});
```

3. 开始一片新的 Wafer， **StartWafer()**

对于 Wafer Sort 的数据，在每一片 Wafer 测试开始前需要调用一下 StartWafer()函数来写入一些 wafer 相关的信息，在每片 Wafer 测试结束之后，需要调用 EndWafer()来结束当前 Wafer 的数据。

Start Wafer 的时候需要传入 WaferConfig 对象，这里面的信息很多，但是一般最重要的是指定 Wafer ID 即可，Wafer Flat 信息最好也指定一下。

```
std.StartWafer(new WaferConfig()
{
        HEAD_NUM = 1,
        WAFER_ID = "WFSAMPLE.1",
        WF_FLAT = 'D',
});
```

4. **写入一颗 Unit 的测试数据** (在程序中循环为每颗 Unit 执行下面 3 个步骤)

   1) 开始一颗 Unit,用 StartPart()函数，把 Site 号作为参数传入

   ```
   std.StartPart(0);
   ```

   2) 写入数据， 通过下面 5 中数据写入函数写入相应的数据。其中 WriteParaTestResult()和 WriteShortParaTestResult()为一对， WriteMultiPinTestResult()和 WriteShortMulitPinTestResult()为一对。每个测试项都需要

std.WriteParaTestResult(0, new ParaTest(){TEST_NUM = 1001, TEST_TXT = "IVDD_LP", HEAD_NUM = 1, RESULT = 34.598, UNITS = "uA", LO_LIMIT = 20.000, HI_LIMIT = 60.000});

std.WriteFuncTestResult(0, new FuncTest() { TEST_NUM = 5000, HEAD_NUM=1, PassFlag=true, VECT_NAM = "MIPI_Functional", TIME_SET = "Digital_Timming", TEST_TXT = "MIPI Test", });

std.WriteMultiPinTestResult(0, new MultiPinTest() { TEST_NUM = 1004, TEST_TXT = "IIL", PassFlag=true, ResultArray = new double[] {42.567, 38.345 }, PinIndexArray= new UInt16[] { 5, 10 }, LO_LIMIT=30.0, HI_LIMIT=210.0, UNITS="uA"});

std.WriteShortParaTestResult(1, new sParaTest(){TEST_NUM = 1002,TEST_TXT = "RF_Gain", RESULT = 12.4,});

std.WriteShortMultiPinTestResult(2, new sMultiPinTest() { TEST_NUM = 1004, TEST_TXT = "IIL", ResultArray = new double[] { 42.567, 38.345 } });

注：在每个测试项第一次写入的时候需要设置Limit和Unit等信息，后续只需要用WriteMultiPinTestResult()和WriteShortMulitPinTestResult（）写测试结果而不需要Limit和Unit信息。另外HEAD_NUM值默认为1，如果不是多个测试头的测试机，HEAD_NUM不需要指定。PassFlag默认为True, 一般也可以不用设置。

3) 结束一个Unit，调用EndPart()方法，把PartResult对象数据传入

std.EndPart(2, new PartResult(){ HARD_BIN = 1, SOFT_BIN = 1, PassFlag = false, PART_ID = "3", TEST_T = 561,});

5. 结束Wafer, EndWafer()， End Wafer操作需要传入WaferResult对象，把Wafer的测试数量结果写入STDF, 这里面最重要的是Wafer ID, PAFT_CNT（总数量）和GOOD_CNT（良品数量）.

```
std.EndWafer(new WaferResult()
{
    HEAD_NUM = 1,
     PART_CNT = 1000,
     GOOD_CNT = 989,
     WAFER_ID = "WFSAMPLE.1",
 });
```

6. **写入Summary信息**，在STDF的末尾都会有Summary信息相关记录，所以可以从STDF中恢复/转出Summary信息。Summary中包含Hardware Bin, Software Bin和Test Count信息。另外还有Part Count信息（不用于Summary）

```
//HBin

std.WriteHBins(new List<Bin>()
{
    //Site0
    new Bin(){ SITE_NUM = 0, HEAD_NUM =1,BIN_NUM = 1, BIN_PF='P', BIN_NAM="Good", BIN_CNT=997},
    new Bin(){ SITE_NUM = 0, HEAD_NUM =1,BIN_NUM = 2, BIN_PF='F', BIN_NAM="Continuity", BIN_CNT=5},
    new Bin(){ SITE_NUM = 0, HEAD_NUM =1,BIN_NUM = 3, BIN_NAM="DC_Fail", BIN_CNT=12},
    new Bin(){ SITE_NUM = 0, HEAD_NUM =1,BIN_NUM = 4, BIN_NAM="RF_Fail", BIN_CNT=3},
});

//SBIN
std.WriteSBins(new List<Bin>()
 {
    //Site0
    new Bin(){ SITE_NUM = 0, HEAD_NUM =1,BIN_NUM = 1, BIN_PF='P', BIN_NAM="Good", BIN_CNT=997},
    new Bin(){ SITE_NUM = 0, HEAD_NUM =1,BIN_NUM = 201, BIN_NAM="OS_Open", BIN_CNT=1},
    new Bin(){ SITE_NUM = 0, HEAD_NUM =1,BIN_NUM = 202, BIN_NAM="OS_Short", BIN_CNT=2},
    new Bin(){ SITE_NUM = 0, HEAD_NUM =1,BIN_NUM = 301, BIN_NAM="IVDD_Typ", BIN_CNT=5},
    new Bin(){ SITE_NUM = 0, HEAD_NUM =1,BIN_NUM = 302, BIN_NAM="SCLK_IIL", BIN_CNT=1},
    new Bin(){ SITE_NUM = 0, HEAD_NUM =1,BIN_NUM = 303, BIN_NAM="SDATA_IIL", BIN_CNT=6},
});

//Test Count
std.WriteTestCounts(new List<TestCount>()
 {
    //Site 0
    new TestCount(){ SITE_NUM=0, HEAD_NUM = 1, TEST_NUM =1001, TEST_NAM="IVDD_LP", TEST_TYP='P',
EXEC_CNT=1000, FAIL_CNT=3},
     new TestCount(){ SITE_NUM=0, TEST_NUM =1002, TEST_NAM="RF_Gain", TEST_TYP='P', EXEC_CNT=997,
FAIL_CNT=3},
     new TestCount(){ SITE_NUM=0, TEST_NUM =1003, TEST_NAM="Noise Figure", TEST_TYP='P', EXEC_CNT=987,
FAIL_CNT=2},
     new TestCount(){ SITE_NUM=0, TEST_NUM =5000, TEST_NAM="MIPI_Functional", TEST_TYP='F',
```

```
EXEC_CNT=985, FAIL_CNT=1},
    new TestCount(){ SITE_NUM=0, TEST_NUM =1004, TEST_NAM="IIL", TEST_TYP='M', EXEC_CNT=976,
FAIL_CNT=1},
});

//Part Count
std.WritePartCounts(new List<PartCount>()
 {
    new PartCount(){SITE_NUM=0, HEAD_NUM = 1, PART_CNT=1000, GOOD_CNT=997, RTST_CNT=0},
    new PartCount(){SITE_NUM=1, PART_CNT=1000, GOOD_CNT=987, RTST_CNT=0},
    new PartCount(){SITE_NUM=2, PART_CNT=1000, GOOD_CNT=657, RTST_CNT=0},
    new PartCount(){SITE_NUM=3, PART_CNT=1000, GOOD_CNT=999, RTST_CNT=0},
});
```

7. 结批，EndLot()结批并关闭 STDF 文件

```
std.EndLot();
```

注：STDF 写入完成后记得调用 Dispose()函数释放 nSTD 对象。

```
std.Dispose();
```

注意试项：