WDF 控件开发手册

- Map 读取控件开发
- Map 导出控件开发

A Light and Intelligent Solution

WDF 控件
开发手册

First Printing, January 2025

Document Number: NL-014-01 Rev. A

目录

1
开始

- WDF 控件是什么.
- 开发环境说明.
- WDF.dll.
- Map 控件如何工作

WDF 控件是什么？

NEDA WDF 控件 (WDF.dll) 是 NEDA 自带的一个 Wafer Data Format 数据结构控件，这个控件是 NEDA 和 Wafer Map Viewer 之间传递 Map 数据的接口，也是 Wafer Map Viewer 和其他 map 格式读取控件以及 map 导出格式控件的数据结构。

这里我们详细介绍一下 WDF 控件，以便用户可以开发更多的 Map 格式控件用来把不同格式的 map 导入 Wafer Map Viewer 进行查看/变换/Ink/格式转换，或者让 Wafer Map Viewer 支持导出更多的 map 格式。

Map 读取控件开发：读取特定格式的 map 文件内容，并把数据存到 WDF 对象中，然后通过指定的方法返回。

Map 导出控件开发：把传入的 WDF Map 数据写入到指定格式的 map 文件中。

开发环境说明

开发语言和环境:

- Microsoft .NET Framework 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.

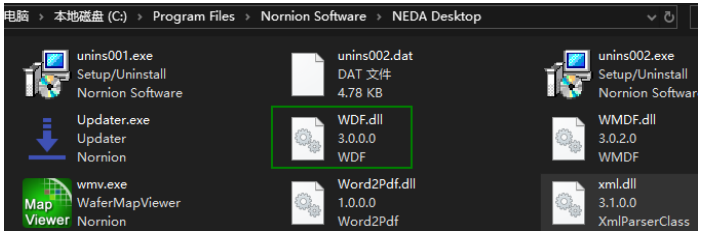
WDF 控件环境:

- WDF.dll 控件没有授权控制，可以直接调用进行开发。但是如果需要验证控件是否能正常工作，则需要安装 NEDA Desktop Edition 最新版本并有有效授权 (可以申请 1 个月试用授权)，然后在 Wafer Map Viewer 里面调用验证。

WDF.dll 在哪里

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 WDF.dll，一般情况会在下面的路径。

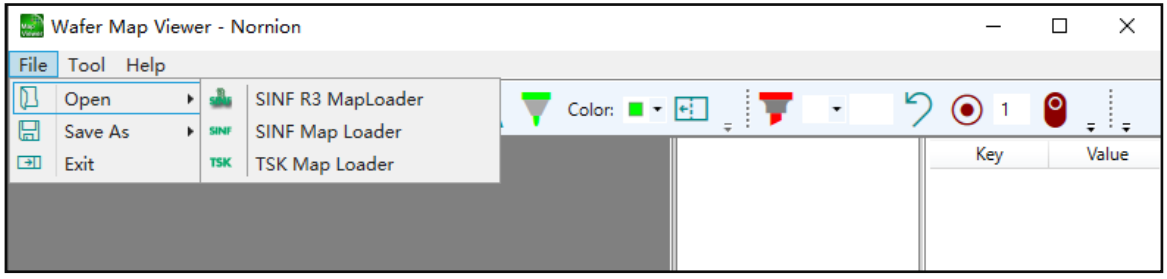
C://Program Files(x86)/Nornion Software/NEDA Desktop/WDF.dll



Map 控件如何工作

Map 格式读取控件，我们称之为 MapLoader 控件，开发好之后需要放到 NEDA 安装目录下的 PlugIn/MapLoader, 并且命名必须为 xxxxMapLoader.dll，以便 Wafer Map Viewer 发现它们并显示在菜单 [File] – [Open]下面供用户调用。

Map 格式导入控件，我们称之为 MapExporter 控件，开发好之后需要放到 NEDA 安装目录下的 PlugIn/ MapExporter 目录，并且命名必须为 xxxxMapExporter.dll, 以便 Wafer Map Viewer 发现它们并显示在菜单 [File] – [Save As] 下面供用户调用。



2
WDF 控件手册

- 命名空间.
- 枚举
- 属性
- 字段
- 方法
- 开发流程

命名空间

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

枚举

在 WDF.dll 中我们定义了一些和 Map 相关的常用枚举类型，以供在数据交互过程中提供确定的数据。

NotchDirection 枚举定义了Wafer Notch的方向

```
{
        DOWN,
        LEFT,
        RIGHT,
        UP,
}
```

OriginLocations 枚举定义了坐标原点的方向

```
{
        Center,
        UpperLeft,
        LowerLeft,
        LowerRight,
        UpperRight,
}
```

MapTypes枚举定义了Map类型

```
{
        Bin,            // Wafer Map
        Count,          // Stacked Map
        Parametric,     // Parametric Map
}
```

属性

WDF 的属性中存储了很多重要的 Map 相关信息，在读取 Map 文件的时候，需要给特定的属性赋相应的数据。

1. **MapType**: MapTypes 枚举类型，用来指定 Map 的类型，WaferMap 需要设置为 MapTypes.Bin 类型， StackMap 和 ParametricMap 使用不同的类型。
2. **LOT_ID**: string 类型， 用来设置 Lot ID.
3. **WAFER_ID**: string 类型，用来设置 Wafer ID
4. **WaferNotch**: NotchDirection 枚举类型，设置 Wafer Notch 的方向
5. MapOrigin: OriginLocations 枚举类型，设置坐标原点的方向
6. **LotInfoDict**: Dictionary<string, string>字典类型，用来存储 lot level 的相关信息，key 关键字命名参照 STDF Spec 的 MIR/SDR/MRR 的相关字段。如果需要添加自定义的关键字，直接赋值即可：LotInfoDict[myKeyWord] = myValue;

   常见字段有：LOT_ID (批号), **PART_TYP** (产品名), **JOB_NAM** (测试程序名), JOB_REV (测试程序版本号), NODE_NAM (测试机名)，**START_T** （lot 测试开始时间）， FINISH_T (测试结束时间)
7. **WaferInfoDict**: Dictionary<string, string>字典类型，用来存储 wafer level 的相关信息，key 关键字命名参照 STDF Spec 的 WIR/WCR/WRR 的相关字段。如果需要添加自定义的关键字，直接赋值即可：WaferInfoDict [myKeyWord] = myValue;

   创建字段有：**WAFER_ID** (Wafer 编号)，**WF_FLAT** (Notch 方向)，**START_T** (wafer 测试开始时间),WAFR_SIZ (wafer size)，WF_UNITS (wafer size 的单位)，DIE_HT (die height)，DIE_WID (die width)，CENTER_X (reference die X), CENTER_Y (reference die Y)，**POS_X** (X 坐标增加的方向)，**POS_Y** (Y 坐标增加的方向)
8. MapFileName: string 类型，存储原 map 文件的文件名(不包含路径)
9. PassBinList: List<UInt16>类型，用来存储所有 Pass Bin number
10. TotalCount: UInt32 类型，die 总数量
11. GoodCount: UInt32 类型，good die 的数量

12. FailCount: UInt32 类型，fail die 的数量
13. RetestCount: UInt32 类型，复测的 die 的总数量
14. RetestGood : UInt32 类型，复测 die 中 pass 的总数，用来计算 recover rate
15. MinXCoord: Int16 类型，最小的 X 坐标值
16. MaxXCoord: Int16 类型，最大的 X 坐标值
17. MinYCoord: Int16 类型，最小的 Y 坐标值
18. MaxYCoord: Int16 类型，最大的 Y 坐标值
19. BinSumDict: Dictionary<UInt16, int>字典类型，用来存储每个的数量 （BinSummary 数据）

方法

WDF 类有两个重要的方法，一个是添加 die 数据的方法，一个是设置 map origin 的方法。

1. UpdateDie()方法用来添加一个 die 到 WDF 类中，读取其他 map 格式数据的时候，只要通过这个方法循环添加最有读取到的 die 信息就可以了。这个方法会自动更新 die count 相关属性 (TotalCount, GoodCount 等) 和 PassBinList 以及 BinSumDict。UpdateDie()方法有多个重载，请务必用下面标记的类型的变量来调用这个方法，X/Y 坐标用 Int16 类型，bin number 用 UInt16 类型，part_flat 用 char 类型 (取值: P 或 F)。

   public void UpdateDie(Int16 xcoord, Int16 ycoord, UInt16 bin, char part_flag)
2. UpdateOriginLocation()方法用来设置 Map 的坐标原点位置，这个方法会同时自动设置 WaferInfiDict 的 POS_X 和 POS_Y 关键字的值。

   public void UpdateOriginLocation(OriginLocations orig_loc)
3. InitializeMapTable() 方法用来初始化 MapTable Schema, 创建行和列

   public void InitializeMapTable(string waferId, Int16 maxXCoord, Int16 maxYCoord)

字段

WDF 类中有一个重要的字段 MapTable (DataTable 类型)， map 数据被存储在这个 DataTable 中，在每个 Cell 中存储了对应 die 的 Bin #， 如果对应坐标没有 die 或者 die 没有测试则对应的 cell 留空即可。

MapTable 的 row_id,col_id 和坐标的关系

我们都知道 DataTable 对象行和列的编号都是从 0 开始，但是 Map 的坐标却有可能从(1,1)或者其他更大的坐标开始，也会有负数的坐标值。这就需要了解 Map 的行标/列表和坐标的对应关系，这里就需要用 MinXCoord 和 MinYCoord，MapTable 中的某个 cell 的坐标计算公式如下。

X = MinXCoord + col_id

Y = MinYCoord + row_id

其实这里就可以知道 MinXCoord 和 MinYCoord 其实就是 MapTable[0][0]那个 cell 对应的坐标。不过在开发 MapLoader 控件的时候并不需要考虑这么多，只需要调用 UpdateDie()方法循环添加每个 die 的数据即可，MinXCoord, MinYCoord, MaxXCoord 和 MaxYCoord 都会被自动更新的。

另外改变坐标原点位置其实并不会改变 MapTable 的结构，仅仅是改变了 POS_X 和 POS_Y 的设定，然后 Wafer Map Viewer 在渲染 map 的时候会根据设定来调整坐标系和 die 的渲染。

这里我们再总结一下创建 WDF 对象的步骤和关键点：

1. 创建 WDF 对象，设置关键属性的值: LOT_ID, WAFER_ID, WaferNotch
2. 添加 LotInfoDict 和 WaferInfoDict 中的重要的关键字和其应的值 (参看上面属性介绍中标**黑体**的属性)
3. 调用 UpdateOriginLocation()方法设置坐标原点位置
4. 循环调用 UpdateDie(Int16 xcoord, Int16 ycoord, UInt16 bin, char part_flag) 方法添加 die 的数据到 WDF 对象中

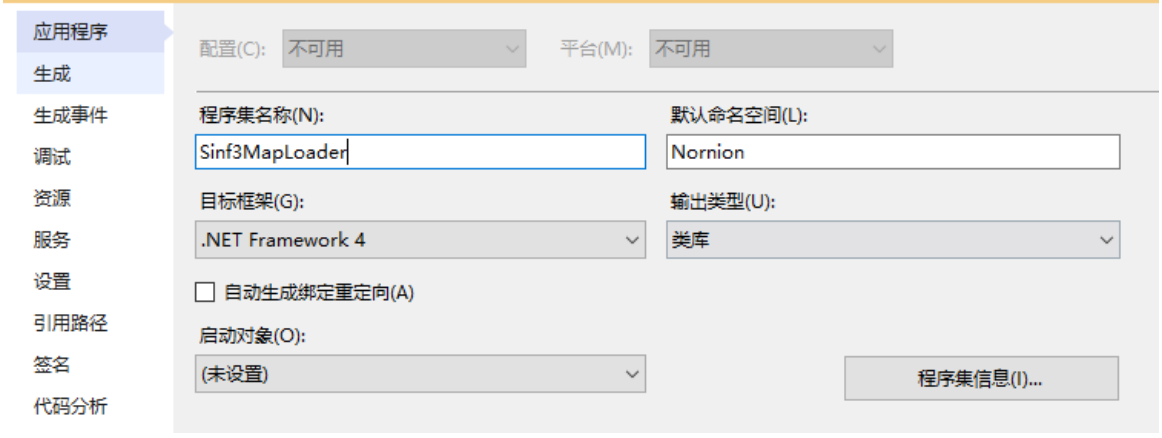
3
Map 控件开发模板

- 控件命名规则
- 命名空间和属性
- 委托和事件
- 方法

控件命名规则

为了让开发的 Map 控件能够被 Wafer Map Viewer 调用成功，控件的文件名，命名空间，程序集名称以及关键属性和方法名称必须遵照一定的规则。其中命名空间必须是 Nornion。

控件的文件名，程序集名称和 ClassName 三者必须一致，且必须符合下面的命名规则。对于 MapLoader 控件必须命名为: xxxxMapLoader (文件名为:xxxxMapLoader.dll)，而 MapExporter 控件的命名须为: xxxxMapExporter (文件名为: xxxxMapExporter.dll)



属性

Map 控件的有几个重要的属性必须定义并设置为 public，数据类型必须一致。这些属性必须设置合适的值。

public string Title：控件的名称属性，如 "SINF R3 MapLoader"。

public string Desc：控件描述属性，如 "Load SINF R3 Map"。

public string Icon：控件的图标，如 "SINF3.png"，只需要图标文件名，不需要完整路径，图标文件需要和控件 dll 文件放到同一个目录，会显示在 Wafer Map Viewer 的菜单栏前面。

public string FileExtFilter：加载的 map 文件格式的后缀名，如 "Map Files|*.*"，Wafer Map Viewer 打开文件的时候会自动筛选。

委托

在 map 控件中需要设置指定的委托和事件，它们定义了和 Wafer Map Viewer 交互的接口。

- public delegate void ProgressChangedEventHandler(int i); 这个委托用来向 Wafer Map Viewer 报告数据解析进度。
- public delegate void ErrorHappendEventHandler(string msg); 这个委托用来向 Wafer Map Viewer 报告错误。
- public delegate void LoadCompleteEventHandler(Dictionary<string, List<WDF>> MapListDict，Dictionary<string, List<string>> MapHeaderTextDict); 这个委托用来在解析完成之后向 Wafer Map Viewer 返回数据。

  Dictionary<string, List<WDF>> MapListDict 这个字典是存储 map 数据的结构 key 为 LOT_ID，每个 lot 可能有多片 wafer 的数据，对应 List<WDF>。每个 WDF 对象对应一片 wafer 的数据

  Dictionary<string, List<string>> MapHeaderTextDict 这个字典用来保存原 txt map 文件 header 的所有数据行，行之间用换行符 \r\n 来分隔。**这个参数可以为空**。

事件

在 map 控件中需要定义 3 个事件，事件为对应委托类型，事件名称必须为规定的名称。委托是用来定义事件的，事件是真正在 Load()方法中被调用的。

- public event ProgressChangedEventHandler ProgressChanged;
- public event LoadCompleteEventHandler LoadComplete;
- public event ErrorHappendEventHandler ReportError;

方法

有一个规定的方法需要实现，那就是 **public void Load()** 方法，在这个方法中需要解析传入的文件名列表的所有 map 文件，并在最后通过下面这个**事件**返回解析的 Map 结果：Dictionary<string, List<WDF>> 和 Dictionary<string, List<string>>。

```
public void Load()
{
    Dictionary<string, List<WDF>> mapDataDict =  new Dictionary<string, List<WDF>>();
    … …
    //解析 map 数据文件，并创建 WDF 对象，然后加入到 mapDataDict 中
    … …
    LoadComplete(mapDataDict, null);
}
```

更多细节请参考 Nornion 提供的示例代码. http://www.nornion.com/dev_extend.aspx