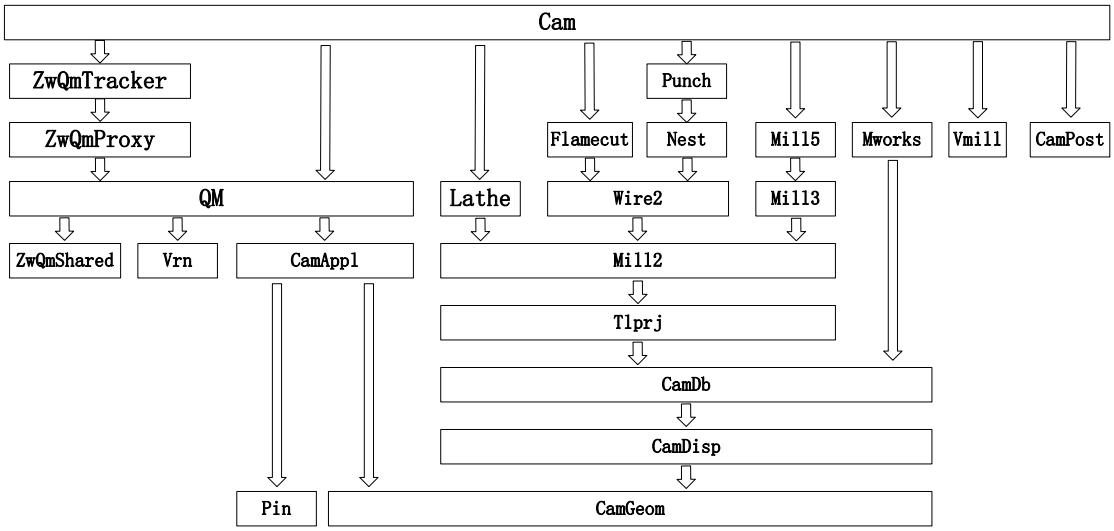
1.CAM 架构现状与未来

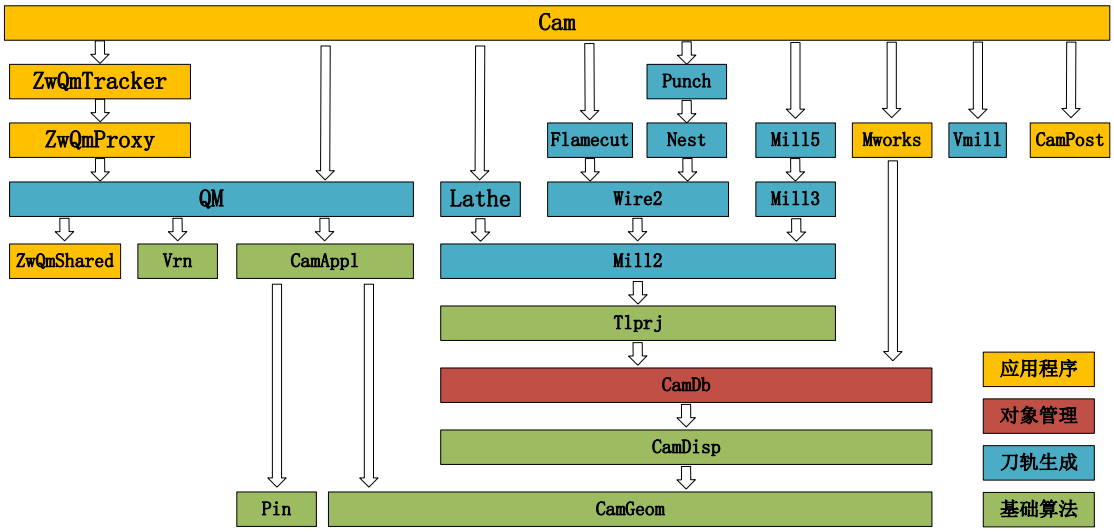
1.1 CAM 当前架构



工程名称	职责
Cam	界面 ui、交互命令、计算流程控制、参数获取、仿真与后处理控制等
CamPost	解析 ZNC，将 CL 转换成 NC
Vmill	封装第三方库 Volumill，以适配 ZWCAM
Mworks	封装第三方库 MachineWorks，提供实体仿真功能
Mill5	5 轴刀轨算法相关
Mill3	3 轴 Nurbs 刀轨算法相关
Mill2	2 轴刀轨算法相关
Lathe	车削刀轨算法相关
Punch、Nest、Flamecut、Wire2	冲压、排料、火焰加工与线切割算法，目前客户使用不多，维护很少
Tlprj	刀具投影算法相关
ZwQmTracker	QM 批处理计算器
ZwQmProxy	QM 代理器，用于远程计算
ZwQmShared	定义 QM 有关 message，用于批处理计算
QM	QuickMill 刀轨算法相关
Vrn	Voronoi 中轴线与偏置算法
Pin	包括几何运算，也包括文件管理
CamAppl	刀轨基础算法相关
CamDb	Cam 数据库管理

CamDisp	提供线框仿真所需的刀具轮廓
CamGeom	Clipper 封装、Tsp、arc 与 crv 与 srf、刀具轮廓相关

1.2 当前存在的不足



1）模块功能不够清晰：

- Cam 过于臃肿杂乱，不仅包括界面与控制，而且包括部分刀轨算法的前置处理；
- Tlprj 是过渡模块，当时为了快速模块化，而把部分函数直接放入其中；
- CamGeom 包括太多内容，需要重新规划，比如刀具轮廓是否应该移出去；
- CamDisp 目前仅剩下线框仿真提供铣刀轮廓的功能，需要重新规划；
- CamAppl 是新增模块，需明确其定位（只提供刀轨计算相关的算法，跟几何相关）。

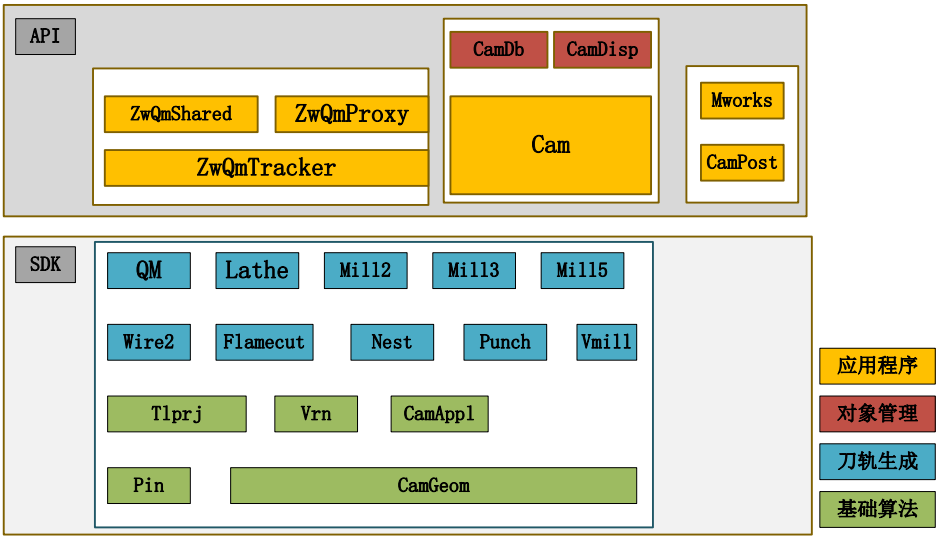
2）模块耦合度过高

- CamDb 被太多模块不合理地调用（ZwGetPubVar(VgCmActiveOpdef)，其本身的对象管理功能，不应该被任何算法模块直接调用，这是因为以前缺乏层次划分概念，一切以快速实现为首位；
- Tlprj、mill2 被 lathe、mill3、mill5 调用，应该提取基础功能函数到另外的底层模块。

3）无法支撑独立发售与外包服务

- 缺乏统一的输入输出数据定义；
- 虽然刀轨算法模块已 DLL 化，但必须挂靠众多的 ZW3D 平台功能。

1.3 未来愿景



1） 算法库独立化
- 每个刀轨算法库都只依赖几何库与基础算法库；
- 算法库有统一的输入与输出数据结构定义；
  刀轨定义；（目前在 WebCAM 项目中已定义）
  工序参数；
  刀具参数；
  几何使用 ZW3D geom；
  加工特征；
  。。。。
- 算法库之间耦合度低，可单独提供。

2） MVC 分层（Model-View-Control）
- 对象管理、流程控制与界面显示明确划分；
- 既可提供依赖 ZW3D 平台的显示管理模块（cam），也可提供不依赖 ZW3D 平台的显示管理模块（ZwQmTracker，ZwMill2Tracker，…）。

3） 后处理模块拆分与替换
- 提供自主仿真平台，提升产品安全性；
- CamPost 模块拆分重构，如从 cam 剥离生成 CL 功能(CamCL 库)；重构 ZNC 解析与 NC 生成；提供表达式支持等。(在 NC 里面提供 IF、ELSE、FOR、加法减法)

4） API 与 SDK 支持
- 持续完善 API 生态与规范二次开发接口；
- 依托算法库，提供 SDK 算法组件。

5） 云平台支持
- 依托算法模块独立、SDK 支持与跨平台支持；
- 提供 WebCAM 与 Cloud3D 两种支持方式。

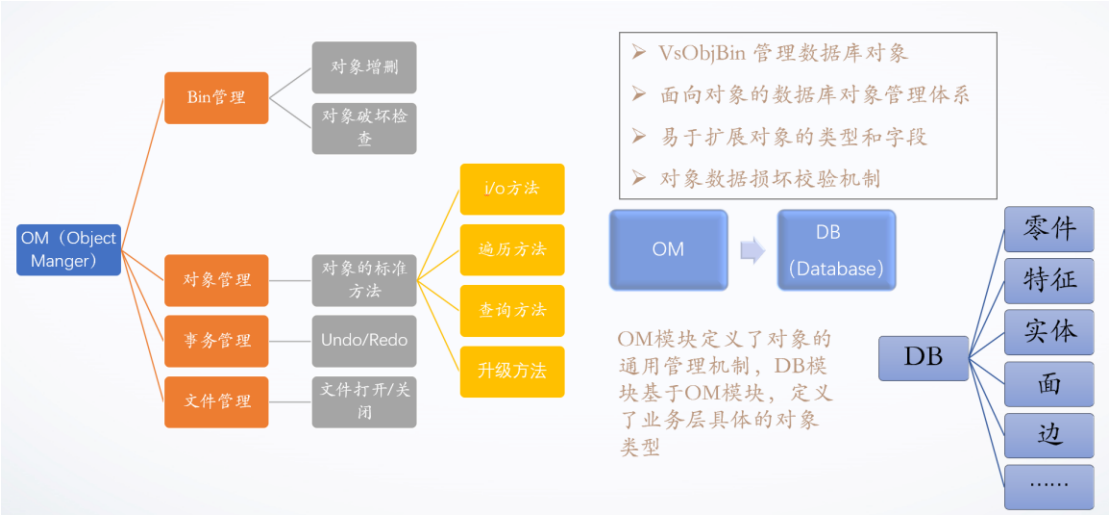
1.4实施路线

1） 长期主义
- 统一思想，明确目标；
- 做项目时考虑模块化与分层，考虑复用与扩展性；
- 持续重构代码。

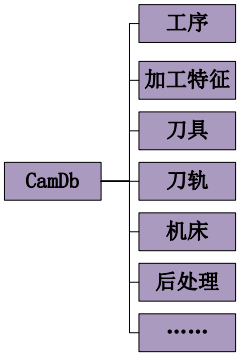
2） 编码规范
- 禁止低层模块调用高层模块；
- 禁止 include 其他模块的 VxXXXPr.h/ZwXXXPr.h 头文件。

2.CAM 数据库现状与未来

2.1 ZW3D 数据管理与存储技术

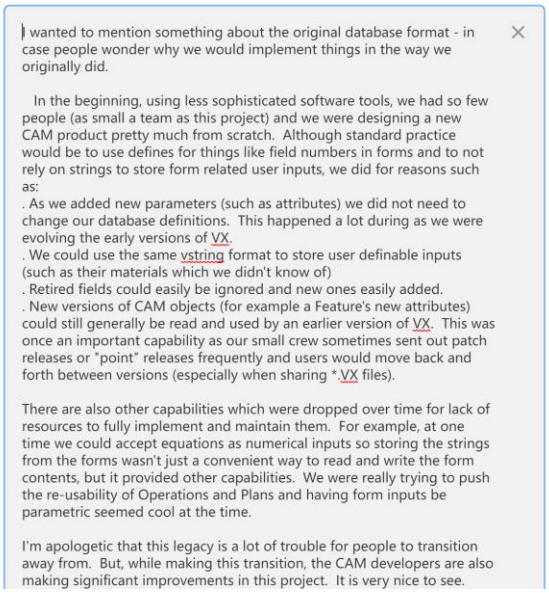


1） OM 与 DB 模块提供管理机制与抽象机制；
2） CamDb 根据上述模块的要求定义 Cam 自身的对象与对象方法；



2.2 GUI Vstring

1） 历史



I wanted to mention something about the original database format - in case people wonder why we would implement things in the way we originally did.

In the beginning, using less sophisticated software tools, we had so few people (as small a team as this project) and we were designing a new CAM product pretty much from scratch. Although standard practice would be to use defines for things like field numbers in forms and to not rely on strings to store form related user inputs, we did for reasons such as:
. As we added new parameters (such as attributes) we did not need to change our database definitions. This happened a lot during as we were evolving the early versions of VX.
. We could use the same vstring format to store user definable inputs (such as their materials which we didn't know of)
. Retired fields could easily be ignored and new ones easily added.
. New versions of CAM objects (for example a Feature's new attributes) could still generally be read and used by an earlier version of VX. This was once an important capability as our small crew sometimes sent out patch releases or "point" releases frequently and users would move back and forth between versions (especially when sharing *.VX files).

There are also other capabilities which were dropped over time for lack of resources to fully implement and maintain them. For example, at one time we could accept equations as numerical inputs so storing the strings from the forms wasn't just a convenient way to read and write the form contents, but it provided other capabilities. We were really trying to push the re-usability of Operations and Plans and having form inputs be parametric seemed cool at the time.

I'm apologetic that this legacy is a lot of trouble for people to transition away from. But, while making this transition, the CAM developers are also making significant improvements in this project. It is very nice to see.

我想提到一些关于原始数据库格式的东西——以防有人想知道为什么我们要以我们原来的方式实现东西。

在一开始，我们使用不那么复杂的软件工具，只有很少的人(和这个项目一样小的团队)，我们几乎是从零开始设计一个新的CAM产品。虽然标准的做法是使用定义来处理表单中的字段编号之类的东西，并且不依赖字符串来存储与表单相关的用户输入，但我们这样做的原因如下：
. 当我们添加新的参数(例如属性)时，我们不需要更改数据库定义。在我们研制VX早期版本的过程中，这种情况经常发生。
. 我们可以使用相同的vstring格式来存储用户可定义的输入(比如我们不知道的材料)
. 退休字段很容易被忽略，而新字段很容易被添加。
. CAM对象的新版本(例如Feature的新属性)通常仍然可以被早期版本的VX读取和使用。这曾经是一个重要的功能，因为我们的小团队有时会频繁地发布补丁版本或“点”版本，用户会在不同版本之间来回移动(特别是在共享*时)。VX文件)。

由于缺乏充分的资源来实现和维护它们，还有一些其他的功能随着时间的推移而被放弃了。例如，曾经有一段时间，我们可以接受方程作为数值输入，因此存储来自表单的字符串不仅是一种读取和写入表单内容的方便方式，而且还提供了其他功能。我们真的在努力推动操作和计划的可重用性，让表单输入是参数化的在当时看起来很酷。

我很抱歉，这个遗产给人们带来了很多麻烦。但是，在进行这种转换的同时，CAM开发人员也在这个项目中进行了重大改进。很高兴看到它。

2） 依托于 GUI form

- 在 ui 文件中设置 callback/event 属性，以实现控件操作与业务函数之间的绑定；
- 在业务逻辑代码中对控件的操作则通过调用 uiFormItemSet 函数实现；
- 使用 uiFormItemGet 函数来获得每一个 field 的内容，并组合起来；
- 参考文档《FrameWork_of_Ui.docx》。

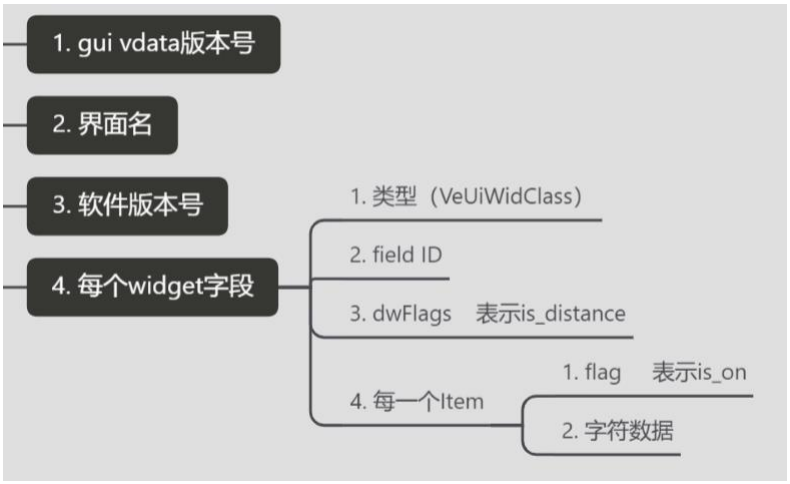
3） Vstring 如何组成

- 关键函数：CmGuiToTxt( )/ CmGuiFromTxt( );
- 遍历 form 的所有 widget，为不同类型的 widget 调用不同的 Loc_AddXXXField 函数，将 widget 的感兴趣的内容保存下来；
- 遍历 Vstring 里面的每一个 item，为不同类型的 item 调用不同的 Loc_SetXXXField 函数，将所保存的内容设置到对应的 widget。

控件类型	保存内容
lineEditBtn	checkBox 状态、lineEdit 里面的文本
checkableComboBox	checkBox 状态、当前 comboBox 的文本
list	每一行的文本
table	每一个 cell 的文本；记录行号与列表头，实现兼容与扩展性
color	用字符串记录 RGB 值
……	……

4） Vstring 包括哪些内容

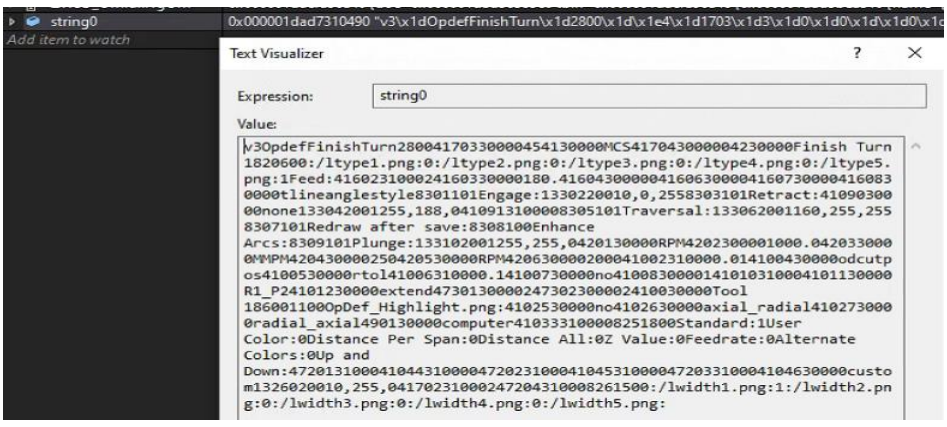
- 组成结构



- 数据结构

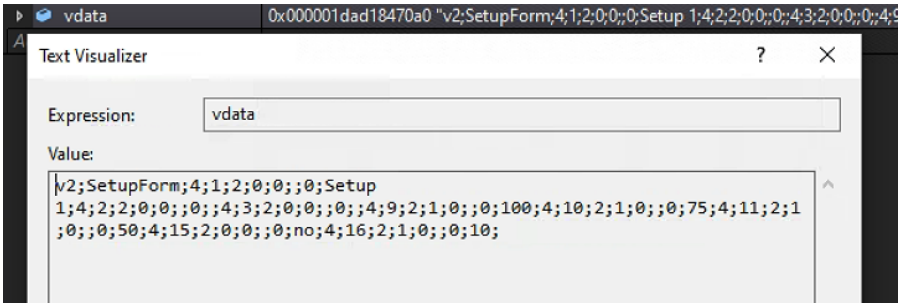
```
typedef struct tagItemData
    {
    DWORD dwFlags; //data item status
    std::string strData; //data item value
    std::string strKey; //table header
    int iRow;          //table Row;
    int tblFlag;       //table flag
    tagItemData::tagItemData() { ... }

    void reset() { ... }
    } ItemData;

/*
**STRUCTURE DESCRIPTION:
**    Field Items inside an UI form, used by FormInfo.
*/
typedef struct tagFieldItem
    {
    int nKlass; //the field control class
    int nFieldId; //the field id
    int nItemCount; //number of data item
    DWORD dwFlags; //field status, e.g. is distance and need to convert with scale factor
    std::vector<ItemData> azItemData;
    tagFieldItem::tagFieldItem() { ... }
    } FieldItem;

/*
**STRUCTURE DESCRIPTION:
**    Store the info of an UI form, it contains an array of control field and the item data.
*/
typedef struct tagFormInfo
    {
    std::string strVersion; //form vdata version, v3 currently
    std::string strForm; //form name of the vdata
    std::string strCadVersion; //SOFTWARE version
    std::vector<FieldItem> azFieldItem; //field collection
    tagFormInfo::tagFormInfo() { ... }
    } FormInfo;
```

- 案例

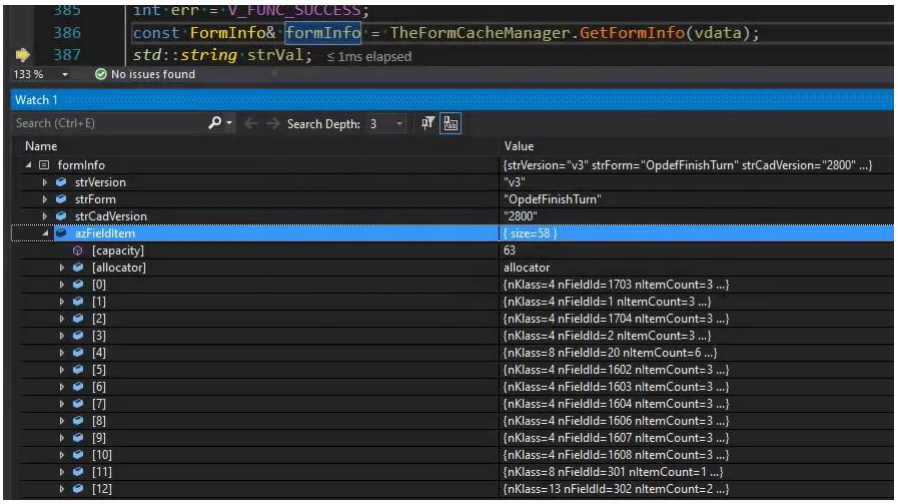


5） R2400_CamV3_Improve 是一个分水岭

- 之前使用分号作为分隔符，容易被用户的误输入导致解析失败，而且严格限定读取顺序，造成维护困难；（VsStrstr）

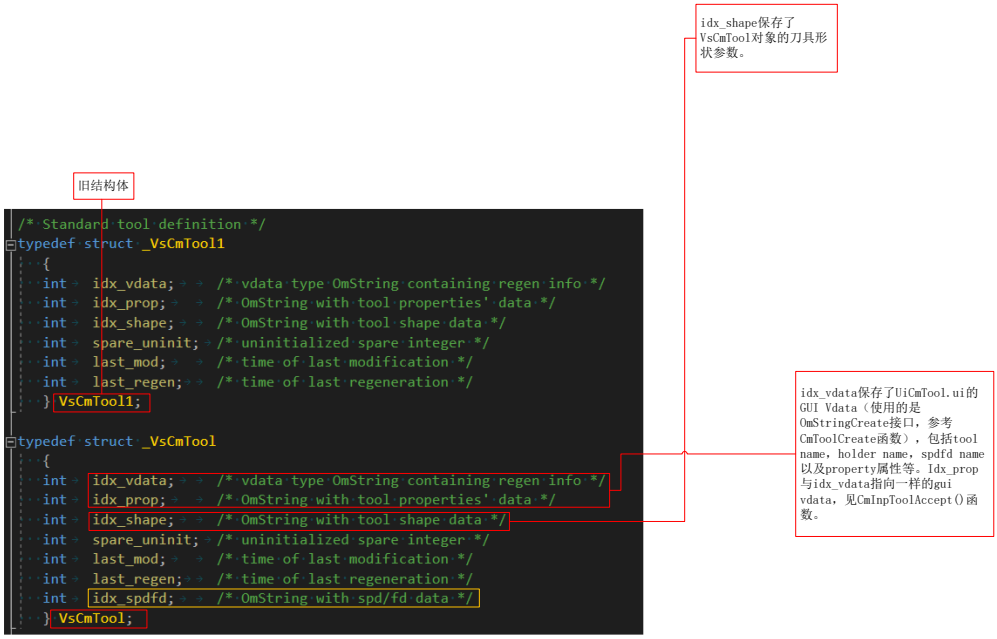


- 之后使用不可见字符作为分隔符（\x1D, \x1E），新建一个中间层数据结构，方便调试、提高安全性。(CmVdataGetItem)



2.3 CamDb 现状

- 数据库结构体存储的大部分是某个 form 的 GUI vstring 字符串对象



- 具体参考文档《Cam 全流程_20211019.pdf》

2.4 当前存在的不足

2.4.1 GUI form 缺乏交互能力

- 没有 ECHO 与 DDD 能力；
- 为了实现简单的关键点显示，不得不使用“无休止绘制”的笨办法，导致渲染效率低下；
- 想获取显示区域的 picking 困难重重；
- 缺少输入检查机制。

2.4.2 GUI Vdata 维护成本高

1） 一体成型，无法拆分
   - 所有工序都有的参数难以进行抽象，统一封装；CmOpdefGetParam()
   - 没有参数枚举或接口，只有 field ID。(API)
2） 文本存储的缺陷
   - 多语言版本的翻译问题；(cam_config,comboBox,)
   - 多语言版本的跑宏问题；(宏语句记录的是 combobBOX 的显示字符串)
   - 数值精度转换。
3） 调试困难
   - 阅读困难，必须时刻与 ui 对应查找；
   - 无法提供可视化调试工具。(ZwVisualizer, VgOm，)
4） 兼容能力不足
   - 与 form 深度绑定，一旦 form 变更，会导致大量兼容代码；

- 兼容代码广泛分布在应用层；
- 未能在 Revise 函数内进行兼容；
- V2 与 V3 没有转换函数，导致老文件必须先打开界面，然后再保存来实现兼容。

5) 将 object 保存在字符串中，缺少相互关系，导致对象管理困难，经常要做遍历查询对比，导致效率低
   - 数据库对象关系管理器项目
     https://zwiki.zwcax.com/pages/viewpage.action?pageId=20480729
   - 工序关键点（#PntXXX）；
   - zlevel 工序的 drive curve（#CurveXXX）；
   - 加工特征所对应的 geom 对象（用 index 字符串表示 pick path）；
   - frame 里面的 datum 对象（存储 datum 对象名）。

2.4.3 CAM DB 架构不足

1) 存在重复存储问题（Setup 与 Plan 里面的 gcomp）
2) DB 版本号意义不大，因为兼容基本不发生在 DB 层（V_CM_TOOL_NUM 2）
3) 颗粒度太大，只有大一统结构体
   - 只有 VsCmOpLaceData()工序结构体，没有 spiralcut 工序结构体；
   - 只有刀具结构体，没有铣刀结构体；
   - ……
4) 对象关系完全被隐藏
   - 如工序使用某把刀，通过将刀具名称记录在 gui vdata 里面来实现；
   - 查询有关对象时需要进行大量遍历；
   - 对象拷贝与删除时必须特殊处理；
   - 对象关系管理器项目（Wiki 链接）。

2.5未来愿景

1) 逐步抛弃 GUI form，全面拥抱 option form，实现 CAD/CAM 一体化；
2) Cam 与 CamDb 模块分层，实现 MVC 分离；
3) 为每一个工序/刀具/特征等设计对应的 tcmd、ECHO 交互、VDATA 化数据结构。

2.6实施路线

1) 在应用层（cam.dll）隐藏所有 gui vdata 操作（希望大家每天都在做）
   - 不允许在 CamDb 之外的任何地方调用 CmXXXInqVdata( )/CmXXXUpdVdata( )函数；
   - 提供所有 CAM 对象的细粒度操作，直接使用对象 index 作为参数；
     CmXXXInqData() CmXXXUpdData( )　CmXXXInqStep/InqToolIndx()
   - 提供 gui vdata 的 AddItem、DelItem 等接口，实现在 DB 层的兼容能力；
   - 提供 CmPlanUpdate( )版本兼容机制，确定统一兼容接口，避免兼容代码满天飞。
2) 拆分大一统结构体（希望大家做项目的时候考虑）
   - VsCmOpTurn2DData/VsCmOpLacecutData/MillStruct/~~VsTurn2Cutter~~；

- 为不同的工序、不同的刀具、不同的特征设计对应的应用层数据结构。

3) CAM DB 层重新设计（以后立项做）
   - 基于 VDATA 来重新设计数据库结构；
   - 版本管理；
   - 理清对象之间的关系。
4) 界面革新(ZMP-620 CAM 适配新渲染引擎)
   - ui 替换；
   - tcmd 编写；
   - echo 实现。

3.CAM 界面案例

4.1 怎么新增一个 gui form

1) copy 一个 ui 文件，修改 object name，修改 form function；
2) 提供一个新 cpp，在里面定义 form function，并将其在 VxCamInitSym( )里注册；
3) form function 需要定义 init、okay、apply、reset 与 cancel 等类型消息处理；
4) 为其新增一个 cfg 文件，根据需要，为指定的 widget 设置公英制下的默认值；
   - cfg 文件都放在/driver/cam_config/目录下
   - cfg 的规则都在文件注释上面

```
################################################################################
# ZWSOFT CAM Config Version 4.0
#
################################################################################
#
# BEGIN_FILE : OpdefSpiralcut.cfg
#
################################################################################
#
# Form Parameter Configuration File
#
################################################################################
#
# Form:FieldId:FieldDesc:FieldType:FieldValueMM:FieldValueIN:OptionsFile:Keyword
#
################################################################################
#
# ** CAUTION :
# ** 1. Items should be sorted by FieldId ascending order, i.e. small to large.
# **
#
################################################################################
#
OpdefSpiralcut:1:Frame:S::::OPDEF_FRAME:
#
OpdefSpiralcut:2:Speeds,Feeds:S:::::
#
OpdefSpiralcut:3:Tolerance:S:0.01:0.0004::OPDEF_TOL:
#
```

5) ui 上面的 widget 所需的 callback 函数，同样需先注册，建议放在同一个 cpp。

4.2 怎么往 ui 新增一个 widget

1) 从本 ui 或者其他 ui 文件中，copy 一个所需的 widget 过来；
2) 放入指定位置，注意 layout 是否合适（区分 HLayout，VLayout 与 GridLayout）；
3) 修改 widget 的 id 为文件唯一的；
4) 修改 widget 的 object name 为“idXXX”；
5) 修改 widget 的 callback 或者 command；
6) 修改 widget 的 labelText（按需，是否翻译）；
7) 修改 widget 的 toolTip（按需，是否翻译）。

4.3 为何不能删除 widget

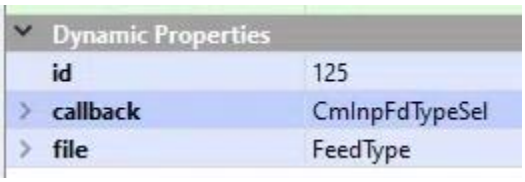
- 对于废弃的 widget，通常的做法是为其增加一个 hide 属性

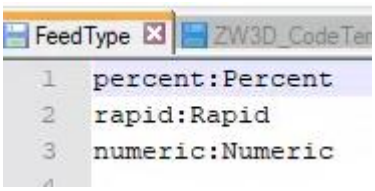

```
<property name="hide" stdset="0">
 <string>hide0</string>
</property>
```

- 因为 GUI Vstring 在设置到 ui 时，会遍历 Vstring 里的每一个 item，然后直接调用 uiFormItemSet( )；若找不到，则会报错退出
- GUI Vstring 里面不能记录 form 的版本号，form 本身也没有版本号，因此无法做机制层面的兼容

4.4 常用的 widget 及其用法

1) **ZsCc::LineEditEx**
   - 支持 check 的显示与回调
   - 支持 lineEdit 的 changed 回调（eventsEnabled 属性设置为 on_char）
   - 具体 item 的定义见 ZsUi::LineEditExHandler 类
2) **ZsCc::LineEditBtn**
   - 支持 check 的显示与回调
   - 支持 button 的点击回调（callback 属性）
   - 支持 lineEdit 的 changed 回调（eventsEnabled 属性设置为 on_char）
   - 具体 item 的定义见 ZsUi::LineEditBtnHandler 类
3) **ZsCc::CheckableTextComboBox**
   - 需要设置 file 属性，指定一个翻译文件作为每一个 item 的显示值





   - 冒号前是 db string；冒号后是 gui string，且与语言版本相关，中文版本显示中文，

德语版本显示德文

- 核心函数是 CmGuiXTable( )
- 在 cam 里面，必须使用 CmInpGetItem( )来获取 db string 并使用，同时必须使用 CmInpSetItem( )来将 db string 设置到 ui, 其会自动转换成对应语言版本的 gui string
- 在 GUI form 的 cfg 文件里，该 widget 的默认值也必须使用 db string
- 必须定义 callback 属性，哪怕是一个空的回调函数，否则无法设置 item
- 具体 item 的定义见 ZsUi::CheckableTextComboBoxHandler 类
- 同时提供了 uiComboXXXX( )等便利函数

4) ZsCc::ListWidget

- 具体 item 的定义见 ZsUi::ListWidgetHandler 类
- 提供了 uiListXXX( )系列便利函数
- 对于 ListWidget 的内容是否存储，需要在 Loc_AddListField( )指定

5) ZsCc::TableWidget

a) 考虑到未来扩展，使用键-值对的存储方式
   - 以表头的 column 为 key，存储每一个 cell 内的字符串
   - 必须定义 file 属性，指定 column 的 db string

GUI Vstring 操作函数
CmVdataTableAppendRow( )
CmVdataTableDelRow( )
CmVdataTableClear( )
CmVdataGetTableHeaderList( )
CmVdataSetTableFlag( )
CmVdataTableGetItem( )
CmVdataTableSetItem( )
CmVdataTableGetPoints( )

b) 支持 cell 为 checkbox、number 或者 comboBox 类型
   - 调用 CmVdataSetTableFlag( )去设置
   - 对于 number，只存储公制数值，因此显示时需要进行公英制切换
   - 对于 checkbox，记录 isOn
   - 对于 comboBox，记录 item

c) cam 提交 CmTableXXX( )与 CmTblXXX( )等 ui 操作函数

d) 具体 item 定义，见 ZsUi::TableWidgetHandler 类

e) 提供 uiTableXXX( )系列便利函数

6) ZsCc::MatrixPushButtons

- 可以设置整体的 callback 函数
- 也可以双击打开控件, 对每一个 button 单独设置, 包括 text、icon、toolTip 与回调 action
- 具体 item 定义，见 ZsUi::AbstractMatrixOptionsHandler( )

7) ZsCc::MatrixCheckBoxes

- 可以根据需要设置 exclusive 选项，是否互斥
- 目前只支持 formItemSet( )，见 ZsUi::MatrixCheckBoxesHandler( )
- 具体 item 定义，见 ZsUi::AbstractMatrixOptionsHandler( )

8) ZsCc::ColorButton

- 设置 callback 函数为 CdColorTCb( )
- 提供 uiColorRGBXXX( )系列便利函数

- 在 GUI Vstring 中以逗号为分隔符，存储 RGB 值
- cam 提供专门的 CmVdataGetColor/SetColor 函数

9) **ZsCc::DrawbackBoxGroup**
   - 在 GUI form 里应用较少，目前仅在 UiCmOutSetting.ui 里面
   - 设置 checkable 的初始值，这样才能显示 check item
   - 设置 callback 函数，响应 check state 的改变

4.5 半命令化下的工序 form

1) 工序 form 的初始化、ok 等过程都独树一帜，没有使用平台本身的逻辑
2) 每个工序最重要的 3 个函数
   - CmInpXXXReset( ) 负责界面初始化
   - CmInpXXXAccept( ) 负责从界面上保存改动到数据库
   - CmInpXXXCalc( ) 负责先保存改动，然后计算刀轨，最后保存刀轨对象
3) 为什么选择 GUI form 激活 tcmd 的半命令化路线?
   - 见文档《半命令化路线对比.xlsx》
4) 所有工序 form 提供各自的 form function
5) 在 form function 里面进行启动与终止 command
   PmOpStartCmd( )/ PmOpStopCmd( )
6) 当前 command 与 form 不是强关联，因此终止时需要考虑许多因素
   - 是否正在执行 frame 有关命令
   - 是否正在执行其他子命令，包括线框仿真、点选择、跟踪点选择等
   - 是否正在添加或者编辑加工特征
   - 是否正在执行测量命令

4.CAM 界面交互开发的规则与技巧

4.1 规则

1. 尽量使用 CmInpGet/SetItem( )函数，避免多语言的问题
2. 尽量使用 CmInpGet/SetNum( )函数，避免公英制问题
3. 出现在 ui 文件上面的资源，无论是 callback、function 还是 label，都必须确认是否需要翻译，且必须符合资源代码规范
4. 文件命名方式（UiCmXXX.ui、OpdefXXX.ui、CmXXX.ui）
5. 函数命名方式（CmInpXXX( )，PmOpXXX( )， PmFtrXXX, PmFrmXXX()）
6. 尽量避免全局变量(如果你不行，你就用文件域内的全局变量)

4.2 技巧与经验

1. 若难以定位何处修改 form，可以在 uiFormItemSet( )里面打条件断点
2. 若想调试 GUI Vstring V3，建议在 GetFormFieldItem( )里面打断点

3. 每个工序的 reset、accpet 与 calc 函数都是很好的断点位置
4. 尽量查看 NUI 源码（ZsCc::XXX 与 ZsUi::XXX）