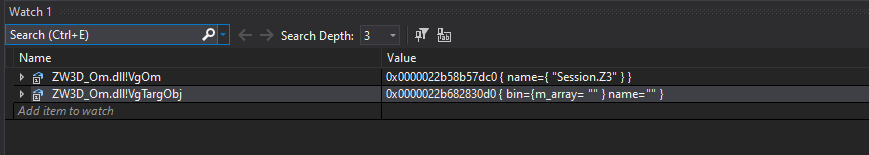
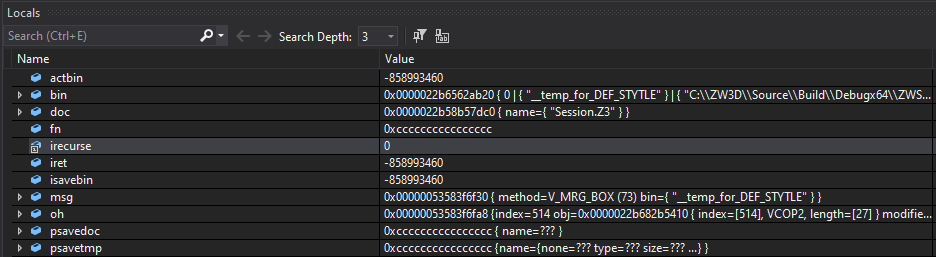
# Visual Studio常用调试窗口

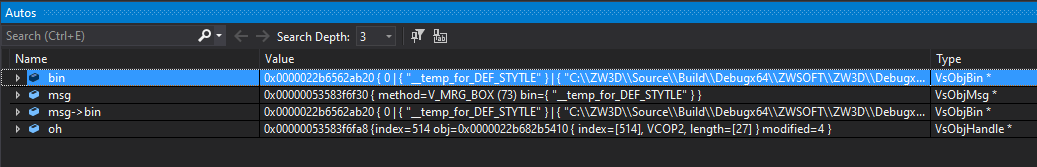
Watch窗口



Local窗口

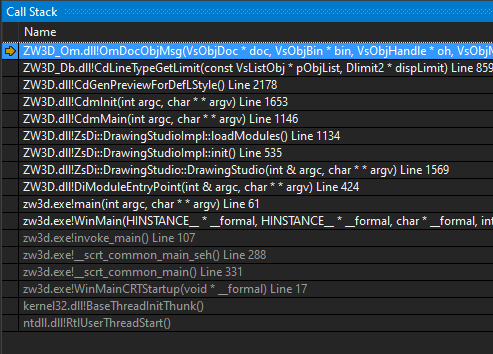


Auto窗口

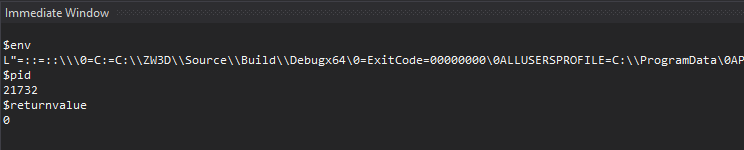


在Auto窗口里可以看到函数返回值，在含有多个函数的复杂条件语句里比较有用。

Callstack窗口



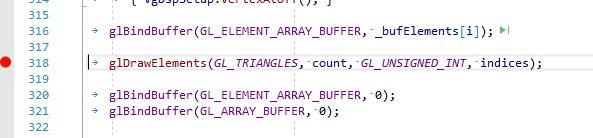
Immediate窗口



# 断点

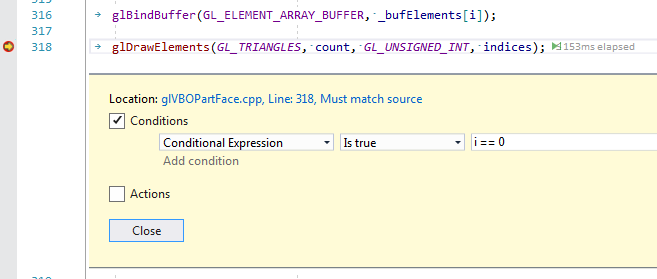
## 在代码处设置断点

开发人员常常需要在代码处中断程序，以便调试。



## 条件中断

某些代码会被多次调用，但是只有在特定的场景才是开发人员想要的，这时，可以给断点设置条件，只有满足条件时，程序才会被中断。

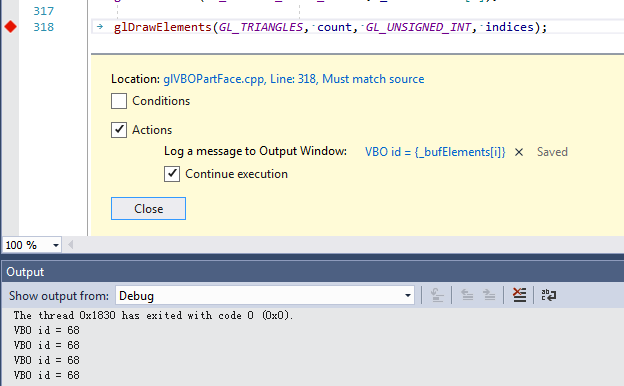


这样可以避免用户频繁进行这样的过程：中断程序、观察当前场景是否为想要的场景、继续执行。

常用的条件是条件表达式和Hit Count、ThreadId。

## 断点处输出信息

有时程序不方便中断，但是又希望可以在断点处得到一些信息，这时可以在断点处设置action，这样可以在不修改代码不重新编译的情况下输出信息。字符串中被{}包裹的表达式将会被求值输出。



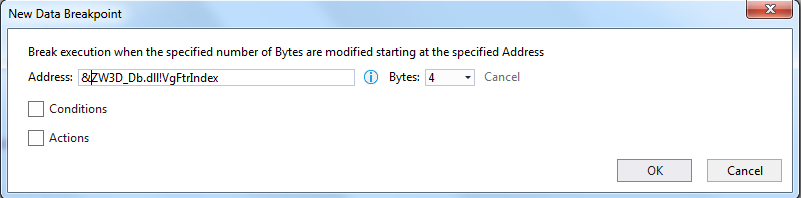
Actions和Conditions可结合使用，同时生效。

## 数据断点

开发人员调试时有时希望知道某个变量什么时候被更改，这时，可以设置数据断点。

方法一：在变量上右键选择Break When Value Changes

方法二：New Data Breakpoint

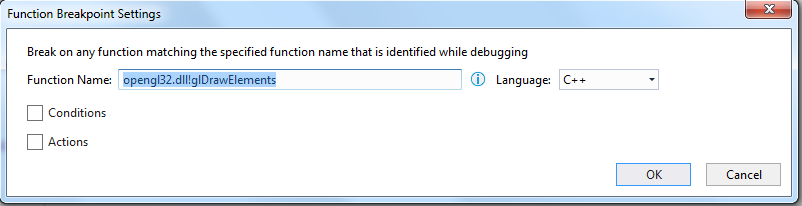


创建断点时输入数据的地址以及以及大小（Bytes），当地址上的数据发生改变时，程序会中断。

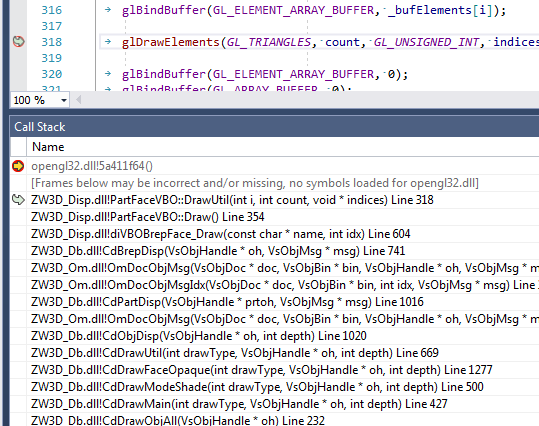
需要注意的是，只有64位的程序才设置8Bytes的数据，32位程序设置double型数据的数据断点比较麻烦。

## 函数断点

开发人员在调试时有时希望在某个函数中断，但是该函数又不是在当前solution生成的，没有对应的源文件，无法直接中断，这时可以使用函数断点。



比如说，开发人员知道ZW3D里渲染面都是使用glDrawElements，但是不知道具体在哪里使用，那么可以直接通过模块名opengl32.dll和函数名glDrawElements设置函数断点，当执行到glDrawElements时，程序会中断。这样，开发人员无需搜索所有ZW3D使用glDrawElements的位置并设置断点。



## 调试启动时中断程序

使用F11启动程序时，程序会在执行main函数的第一行中断。

## 非调试启动时中断程序

在代码调用函数\_\_debugbreak可以使程序中断。

使用场景：ZW3D的配置发生改变时需要重启。重启的ZW3D进程是原进程ZW3D直接调用CreateProcess产生的，没有附着调试器，这时，如果需要调试ZW3D初始化的过程，可以在有关代码处加入\_\_debugbreak。

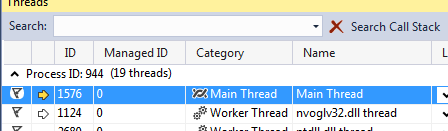
当ZW3D的output框打印错误信息时中断：$break=报错等级。

## 中断程序

点击按钮可以直接中断程序。

中断程序

然后切换到主线程，就能看到程序当前的堆栈。



使用场景：

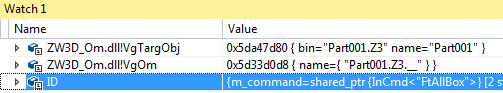
1. 当ZW3D卡住时，可以中断程序，观察程序卡住的地方，以便优化性能。
2. 当ZW3D弹出模态对话框时，中断程序，可以方便找到模态对话框弹出的位置。

## 拖动当前指令

可拖动断点前的箭头改变下一句被执行的代码，常用于回头用F11重复执行某个函数查看具体逻辑。本操作对程序结果可能产生副作用（例如函数里有计数器，重复执行会影响计数结果），调试时需要注意。

# 观察变量

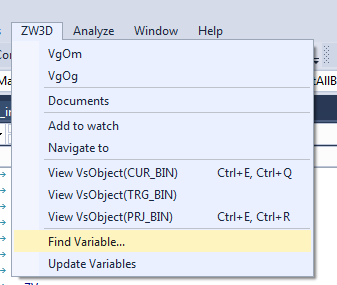
## 添加观察

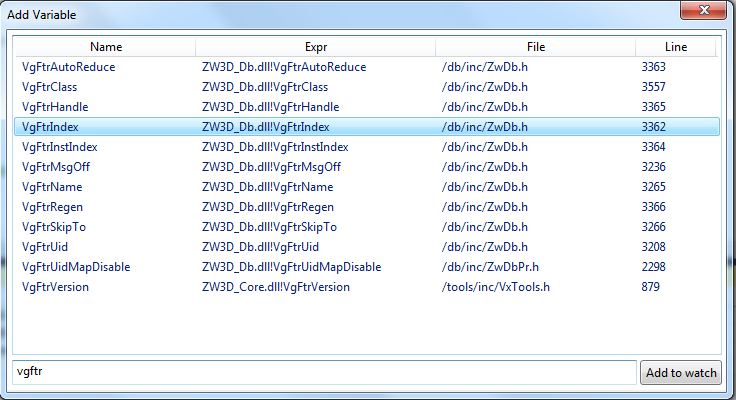


ZW3D是由多个模块（.dll）组成的软件，中断程序时，查看与中断点模块不同的模块的全局变量是需要指定模块名。

## 使用插件添加观察

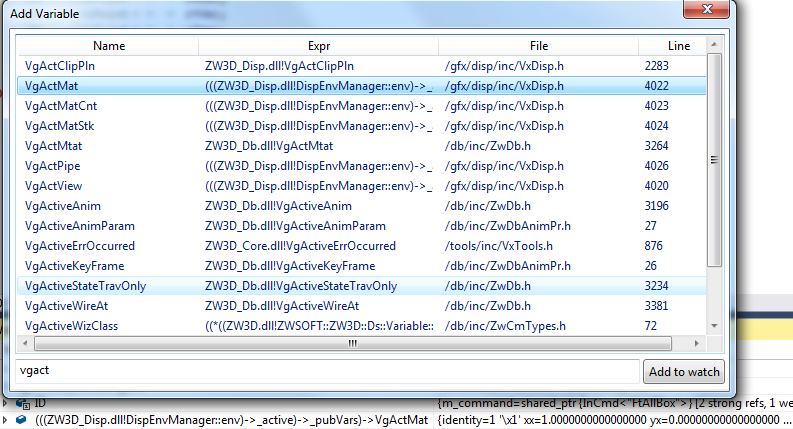
每次添加一个变量都需要给指定模块名，非常繁琐，开发人员也不一定记得全局变量所属模块。因此，开发人员开发了插件，自动补全模块名。





这个插件可以方便开发人员查看全局变量。

尤其是某些变量的值时通过比较复杂的表达式得到，手工输入非常复杂。比如，想要查看当前的模型矩阵：



很多跟显示有关的全局变量是通过ZwDispPubVars::get获得，其表达式比较复杂，难以理解，用插件来添加观察会非常方便。

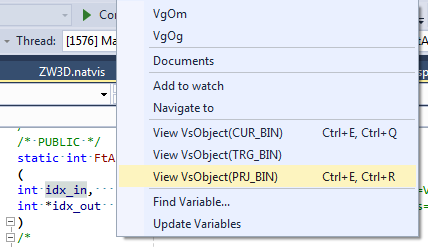
开发人员对全局变量做了增删改时，应该同步更新ZW3D.xml文件配置，保证插件能正常检索到各个全局变量。

## 数据库对象

开发人员在调试时常常需要查看特定index的数据库对象，根据有关数据结构的定义，可以输入如下的表达式：

变量_数据库对象

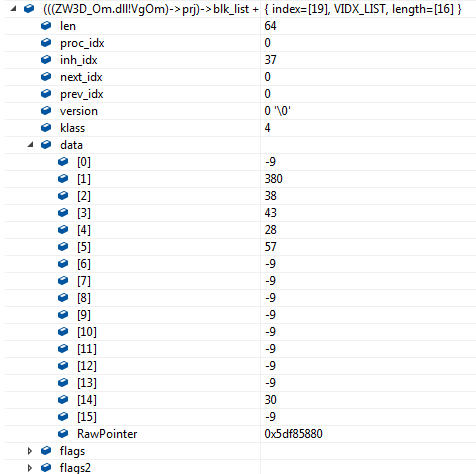
这时，开发人员也可以使用插件：选中变量，点击插件



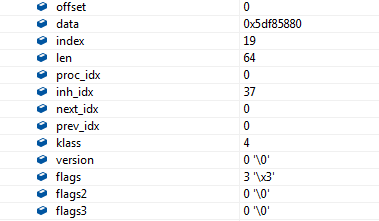
## 调试可视化文件

编写调试可视化文件可以根据开发人员想要的方式查看变量。

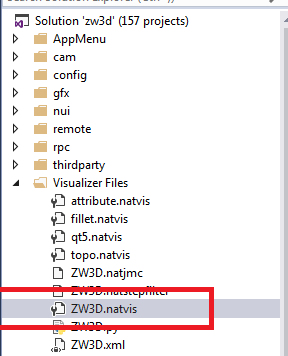
比如说，我们想看一个VDATA类型的VsObjHandle，我们知道obj->data指向了一个int型的数组，我们希望看到的是：



而非



实现这样的效果，主要是在ZW3D工程中加入了调试可视化文件（.nativs）



有关内容为：

VDATA解析

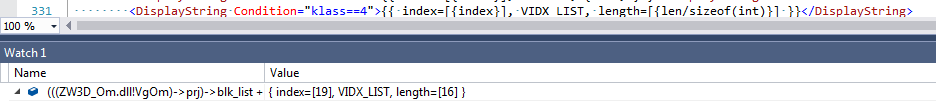
意思是当这个变量展开时，把data按int型指针展开，展开成[len/sizeof(int)]项。

在调试状态下修改ZW3D.nativs文件，点击保存时立即生效的，不需要重新启动调试。

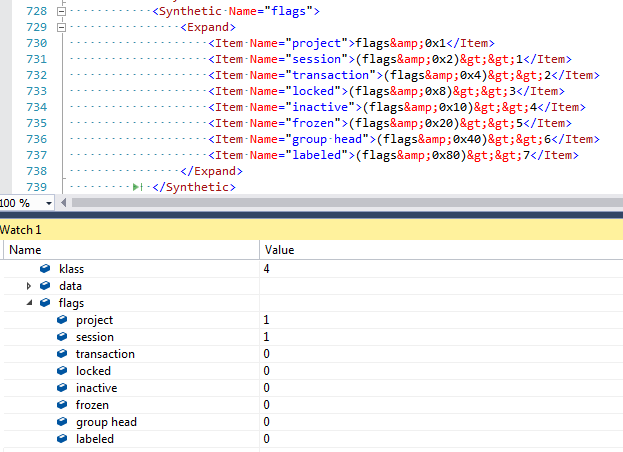
调试可视化文件的语法具体可以参看

比较常用的语法有:

**\*** DisplayString 来展示对象的值



**\*** Expand来展示对象展开的状态



**\*** 用Condition来限制条目是否显示

甚至可以在调试可视化文件里添加变量和控制流，来计算出要展示的值。下面的代码用于展示对象的永久命名。



开发人员在添加新的比较重要的数据结构，比如添加了新类型的数据库对象，需要更新ZW3D.natvis，以便调试。

如果发现Watch里的结果有异常，可参考这篇文档做调试：[ZW3D变量在watch中显示异常的调试方法 - 3DCAX跨部门技术共享 - ZWiki (zwcax.com)](https://zwiki.zwcax.com/pages/viewpage.action?pageId=80550030)

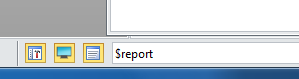
# 命令

## 命令窗口输入命令

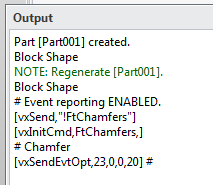
ZW3D的命令都可以通过命令窗口输入命令来执行。

## $report

在ZW3D 命令窗口输入$report



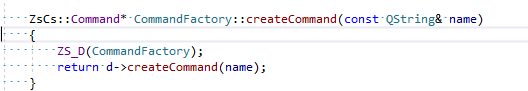
Output窗口会打印用户的操作，由此可以观察，用户执行了什么命令。



开发人员可以由这些信息快速找到对应的代码。

## 命令创建和执行

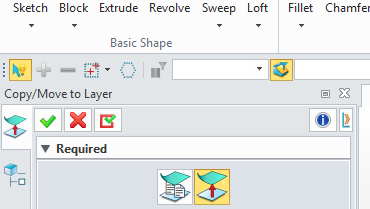
ZW3D创建命令的统一接口为：



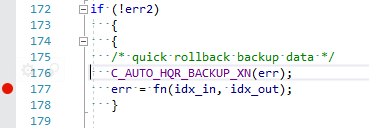
创建的命令对象为ZsCs::Command，其为基类，执行函数为各自的involve函数。

对于模板命令和GUI命令，命令激活后不会立即返回，点击确定命令才完成执行。

模板命令就是在input manager中打开的命令，如倒角等。

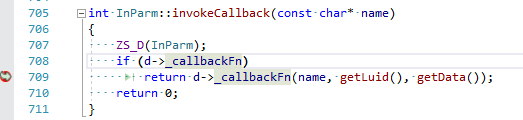


模板命令的执行函数为ZsInCmdReturn，但是这个函数离用户自定义的主回调函数比较远，可以把断点打在InEvalErr中。



命令Echo函数入口为InCmd::echo。

控件的回调函数入口为：

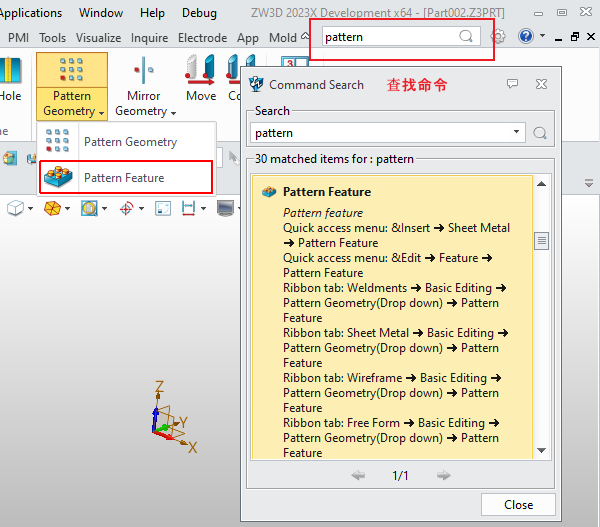


GUI命令的主回调函数会被FormForwarder::forward调用，命令initialize、OK、cancel、apply都会调用这个函数。

控件回调函数入口为BaseFieldHandler::invokeFieldCallback，这个函数捕获所有的事件，甚至鼠标滑过都会触发这个函数，因此最好设置条件断点以便调试。

## 命令查找

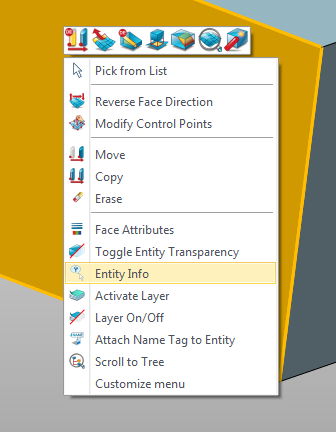
有时候开发人员知道了命令的名字，希望以此找到命令在ZW3D界面上的位置，方便激活。可以在右上方的输入框输入命令的名字，ZW3D跳转到相应的页面，并展开，展示命令的位置

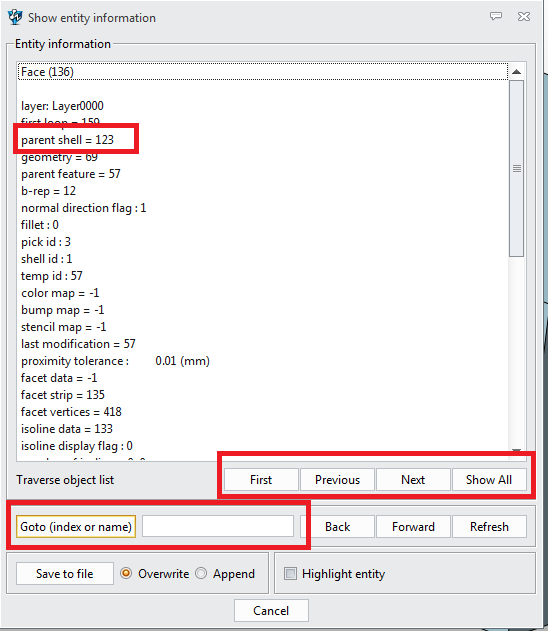


# 数据库

## 打印对象信息

ZW3D提供了显示数据库对象信息的功能。选中实体，点击右键弹出菜单的Entity Info项即可查看选中实体的信息。这方便开发人员查看数据库对象各个字段是否正确，同时也有助于测试人员对问题判断。



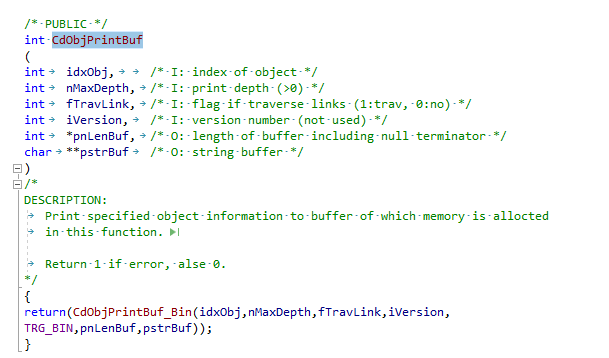


用户可以直接输入数据库对象的index来查看对象，也可以输入名字来查看对象。

点击存在“=”号的字段，如果其值大于-1，那么会跳转到index为其值的数据库对象。

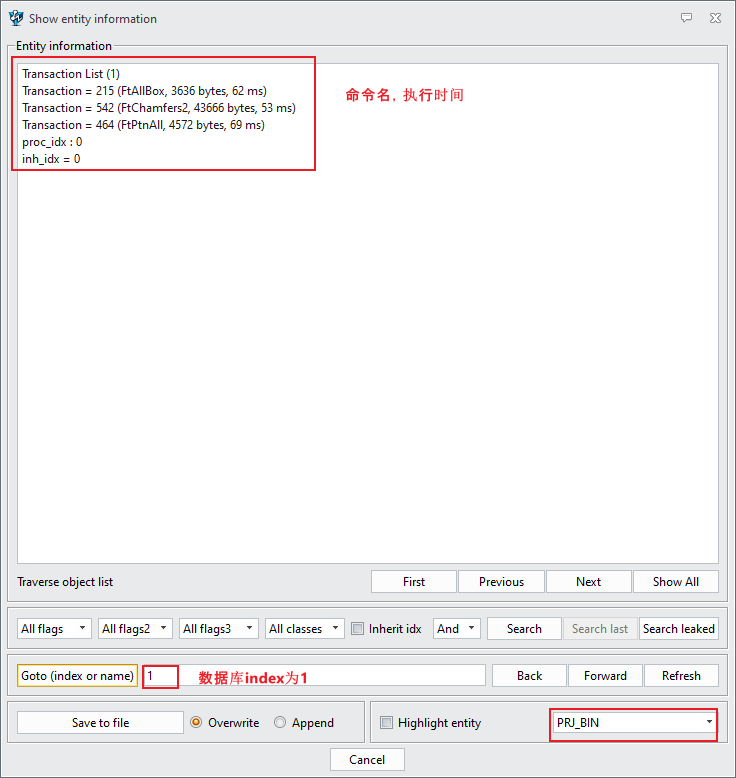
点击Show All可以显示当前对象所属链表的所有对象。

这个功能是调用了各自对象的V\_PRINT方法。其通用入口为CdObjPrintBuf，当开发人员不知道界面上的对象是什么类型的对象是，可以把断点打在这个函数上面，选中实体，点击右键弹出菜单的Entity Info，即可中断程序来查看。



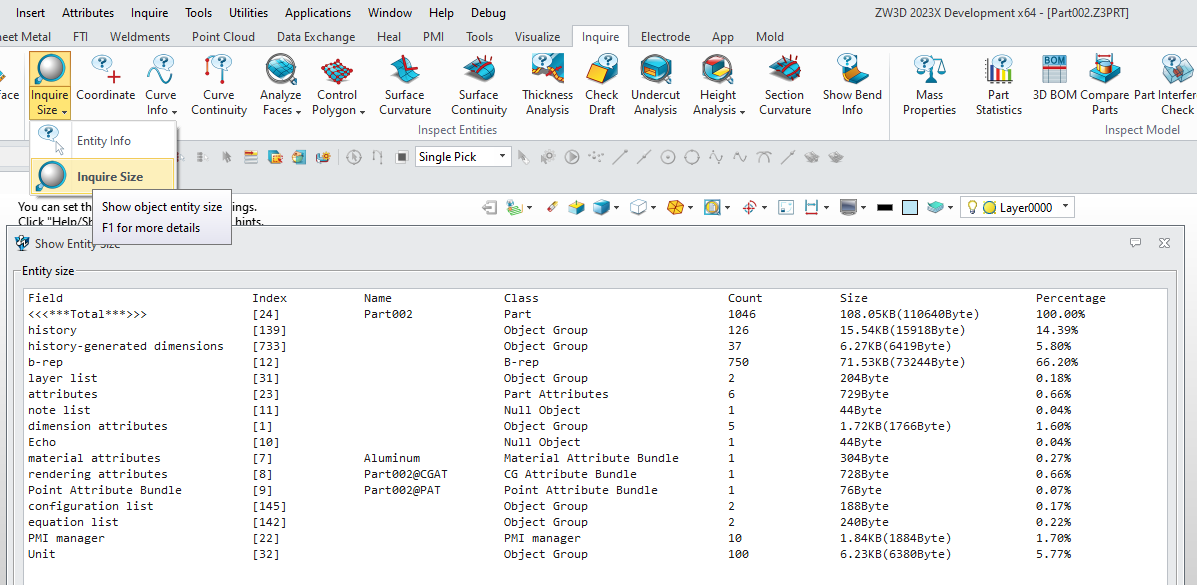
## 打印事务

ZW3D的事务存在于PRJ\_BIN的index为1的对象里，这里可以看到各个命令执行的时间



## 打印对象大小

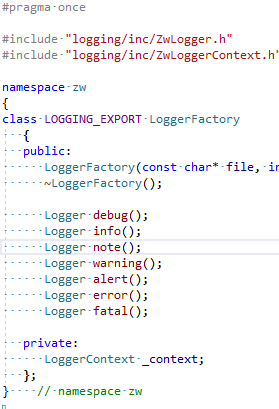
当ZW3D的图纸的大小出现异常，变得特别大时，可以使用Inquire Size来打印一个对象及其所属子对象的大小，还包括子对象的个数，包括大小的占比



# 定位问题

## Log系统

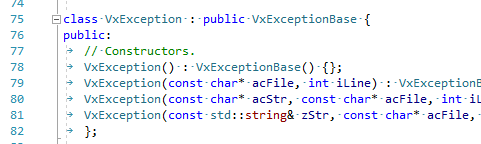
开发人员在调试问题时常常需要定位错误，当发生错误时，一般会在output窗口打印红色错误信息，或者弹出警告气泡，这都会触发ZW3D的log系统。



不同输出类型对应不同的执行函数，黄色警告为warning，红色警告为alert。把断点打在这些位置，可以方便定位问题。

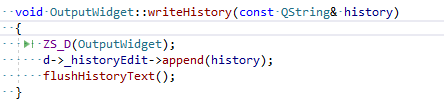
## 异常

ZW3D的代码中，发生错误有时会抛出异常vxException()，开发人员在调试问题时把断点打到异常类的构造函数中，以便定位问题。



## Output窗口输出

开发人员调试时希望找到输出信息代码的位置，而输出的信息不一定是错误。可以把断点打到output窗口输入的底层代码中。



# 查找

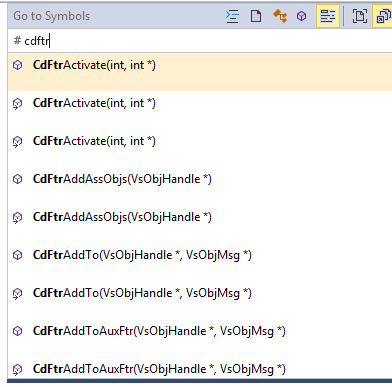
## VAssistX

Ctrl+Shift+s 查找符号

Ctrl+Shift+o 查找文件

## 内置的符号查找工具

Visual Studio内置了查找工具，同样强大。通过”Ctrl+,”（Ctr键加逗号键）来激活。

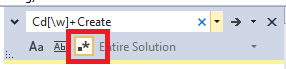


可以指定查找符号、文件、类型等，可以输入”?”来寻求帮助。

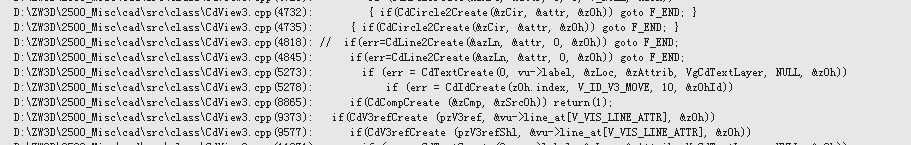
## 正则表达式查找

开发人员有时想找的不是固定的字符串，而是某种格式的字符串，这个时候就需要使用正则表达式匹配查找。

比如开发人员想找到数据库对象创建的函数，格式如Cd???Create，那么在切换成正则表达式模式，并输入正则表达式。



查找的结果为：



查找的话也可以用VsCode，因为VS经常处于编译或调试的状态，响应速度不够快。

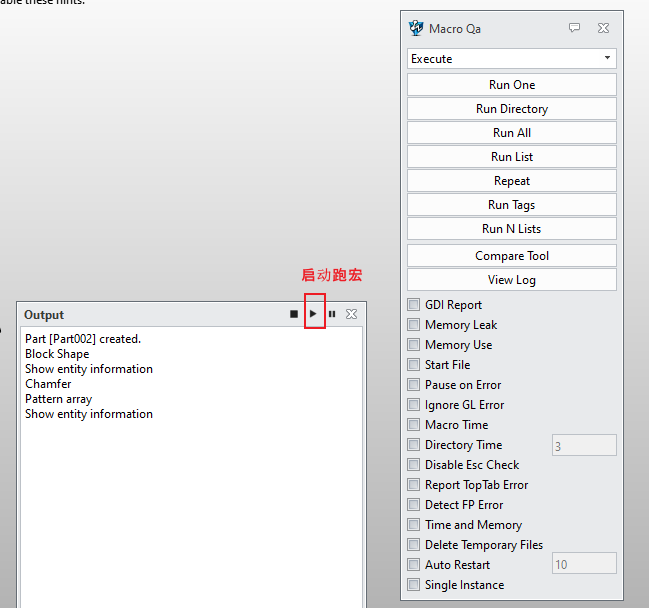
# 宏系统

宏系统是ZW3D的自动化脚本测试系统，开发人员通过跑宏调试问题

## 运行宏

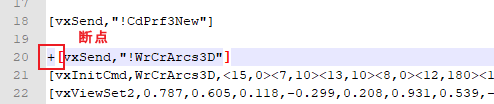
点击output窗口的小箭头即可打开运行宏的界面。Production版本可以通过$qa来启动。

通过不同的选项，可以进行内存泄漏、性能等测试

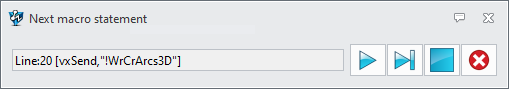


## 调试宏

可以在宏文件里的一行前加入+号，宏执行到这一行会中断



展示下面界面

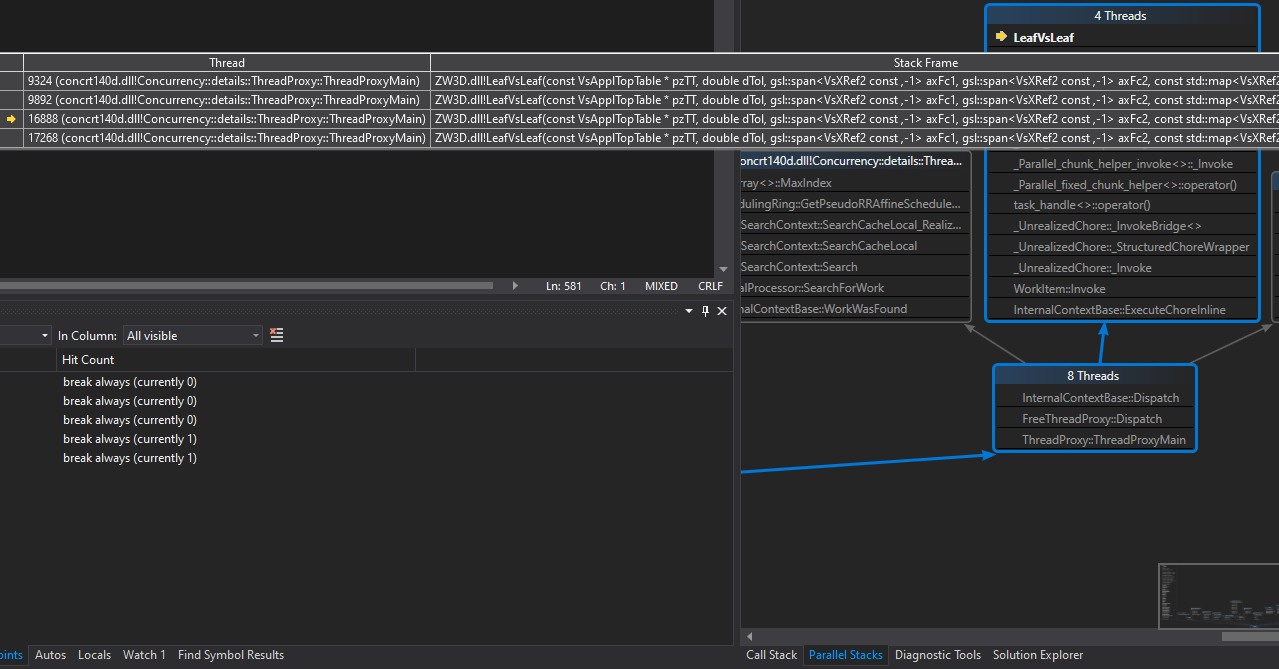


可以单步执行，连续执行，和中断。这里还显示了宏的行号

# 多线程调试

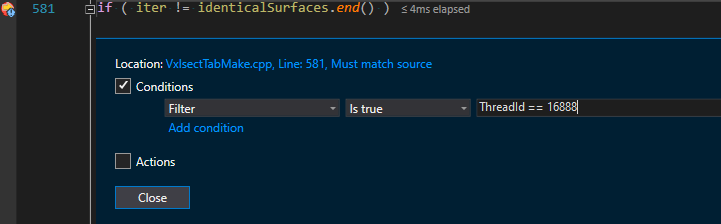
ZW3D的多线程业务场景逐渐增多，常见的多线程并发入口是VxParallelFor

## 查看线程状态



从Parallel Stacks窗口里可以看到，在LeafVsLeaf断下来时，共有4个线程当前正运行到这个接口。

## 针对特定线程下断点

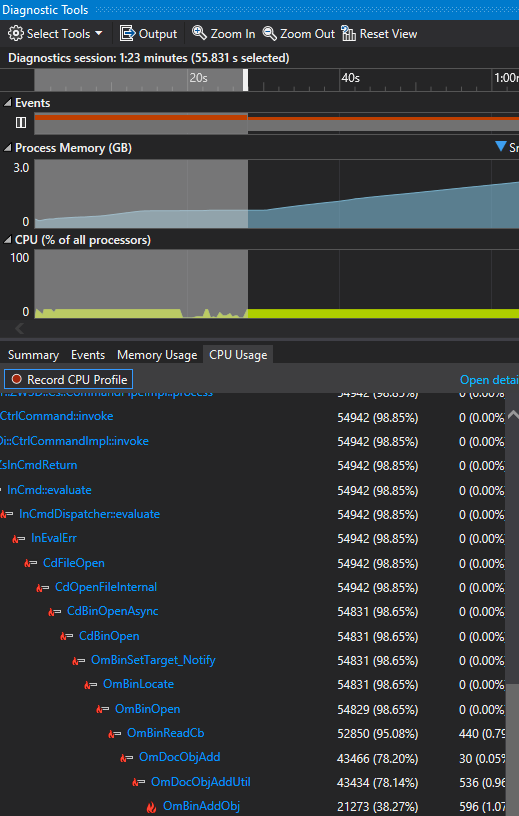


使用上文提到的条件断点里的Filter模式过滤特定的ThreadId

# 性能调试

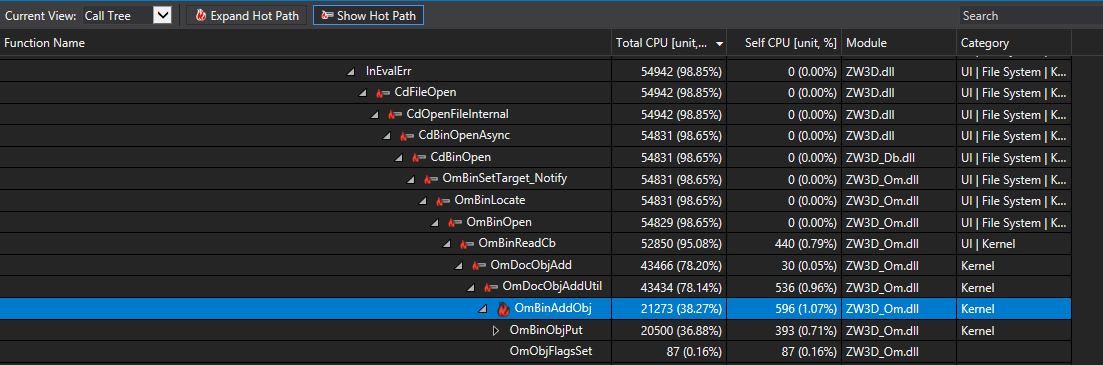
本文只介绍了VS自带的性能调试工具，开发人员可以选择其它趁手的工具。

## Diagnostic tools



VS自带的工具，可以监视CPU开销、内存开销、事件。

## 分析效率瓶颈



以读取文件为例，这个流程里OmBinReadCb占据了95%以上的CPU时间，如果要优化文件读取到内存里的速度，可以从这个函数入手。