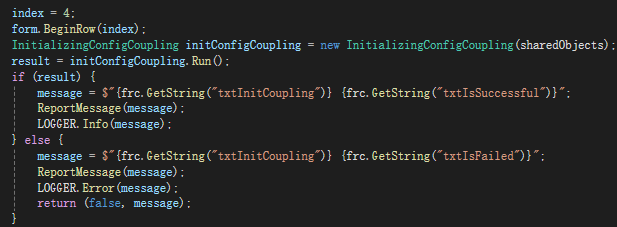
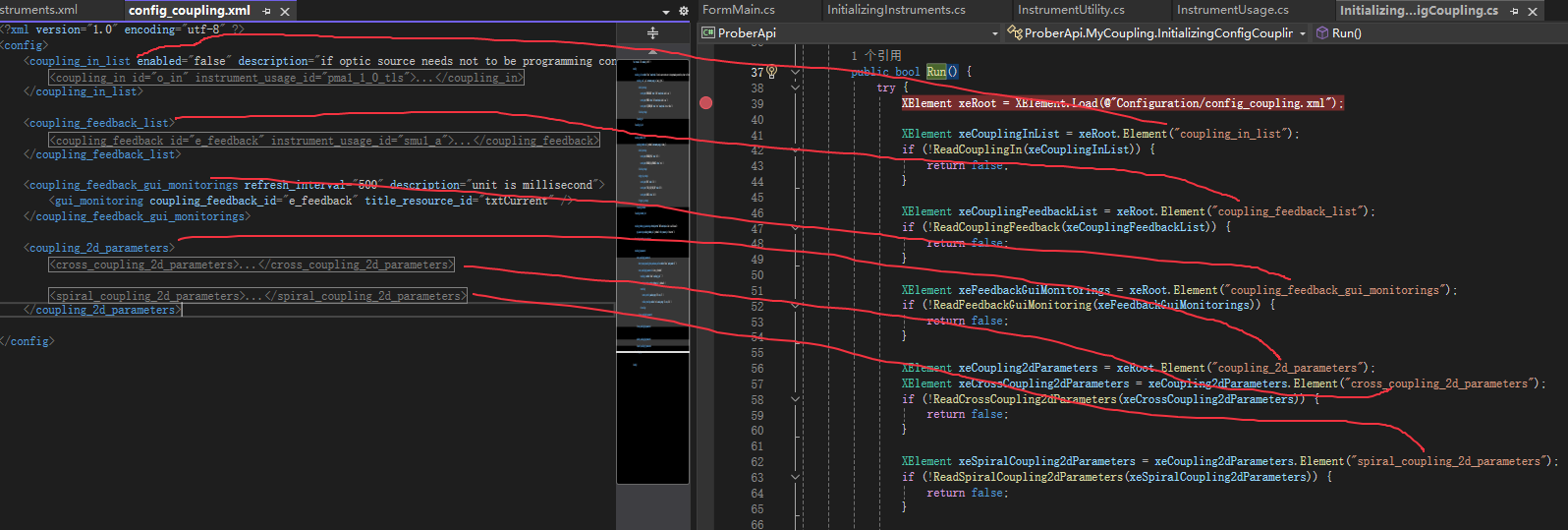
分析config\_coupling.xml的解析和使用过程。

根据分析config\_insturments.xml的经验，config\_coupling.xml应该也是在初始化流程中解析出来的。所以这次我们顺着程序的执行来分析，首先看到初始化的过程：



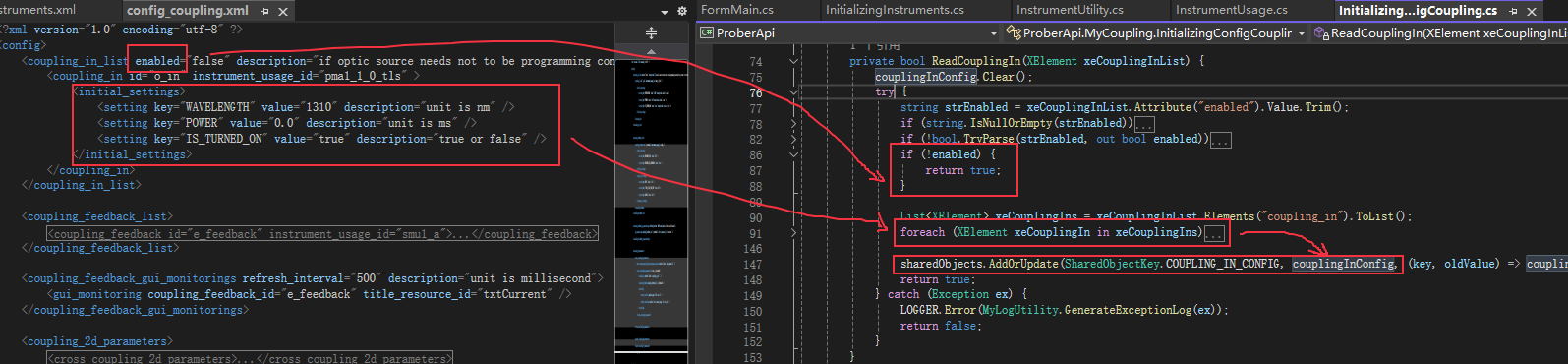
看Run:



架构暂时不支持3D.

依次看看这5个具体的解析动作：

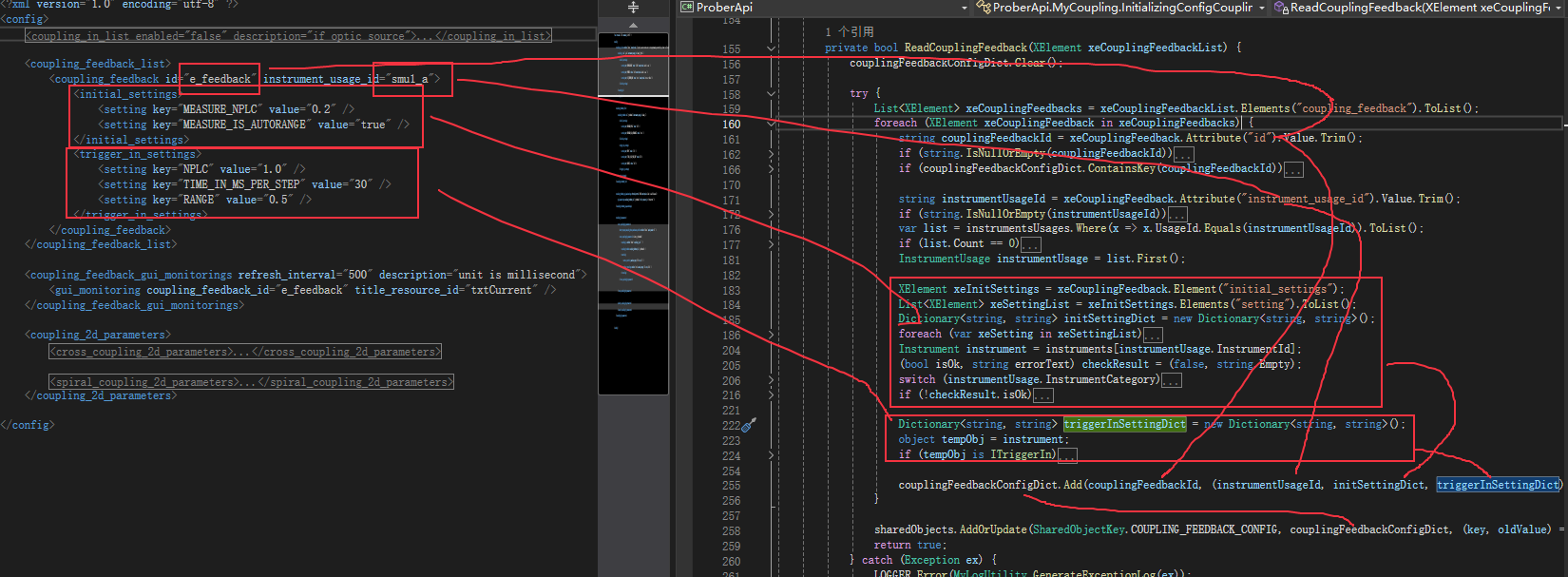
ReadCouplingIn:



可选择是否enable，如果不需要程序控制光源，则设为false。

解析出来放入sharedObjects

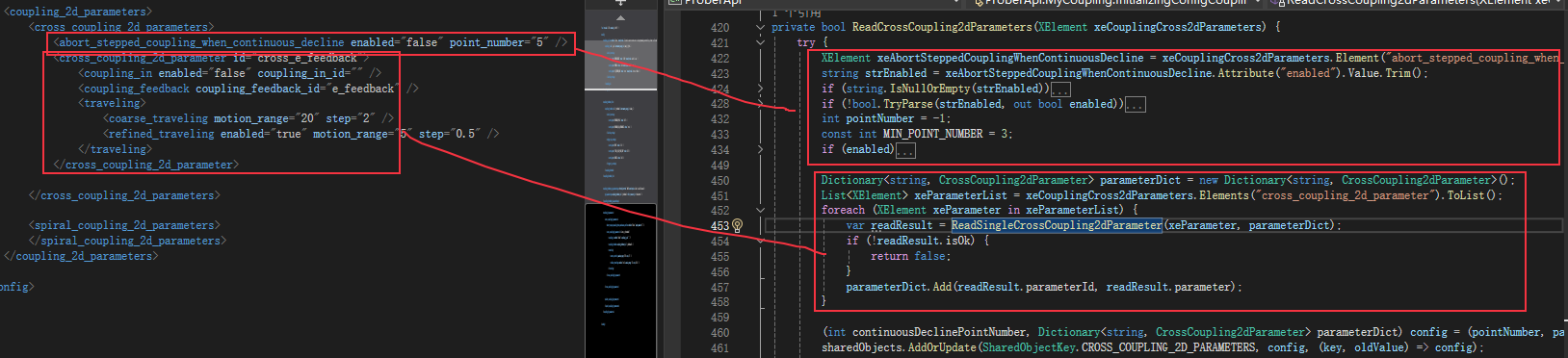
ReadCouplingFeedback



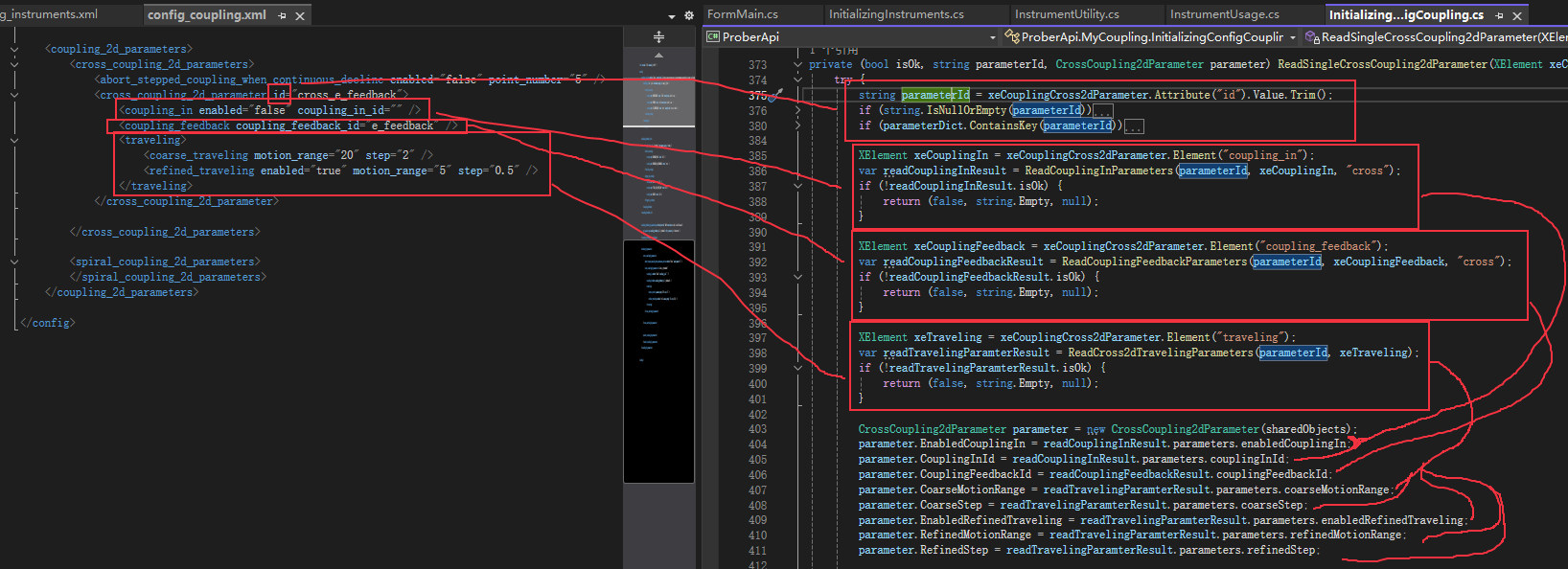
解析出来放入sharedObjects

ReadFeedbackGuiMonitoring也类似。

ReadCrossCoupling2dParameters：

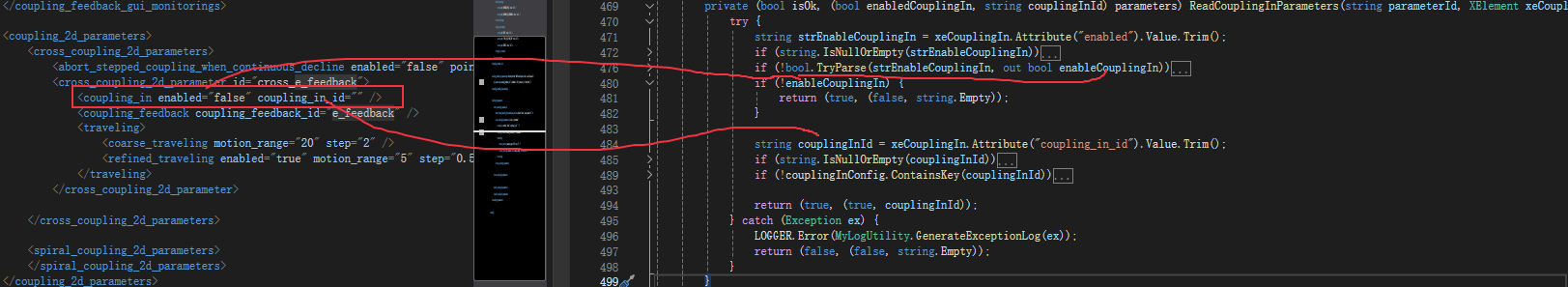


多套了一层ReadSingleCrossCoupling2dParameter：



又多了一层：

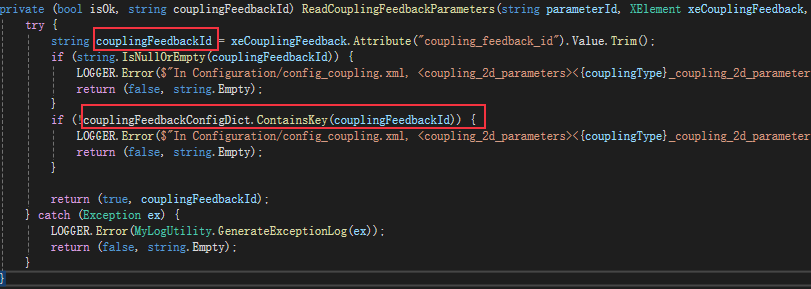
ReadCouplingInParameters：



检查couplingInConfig中有没有couplingInId,如果有，而且Enable了，则关联起来。

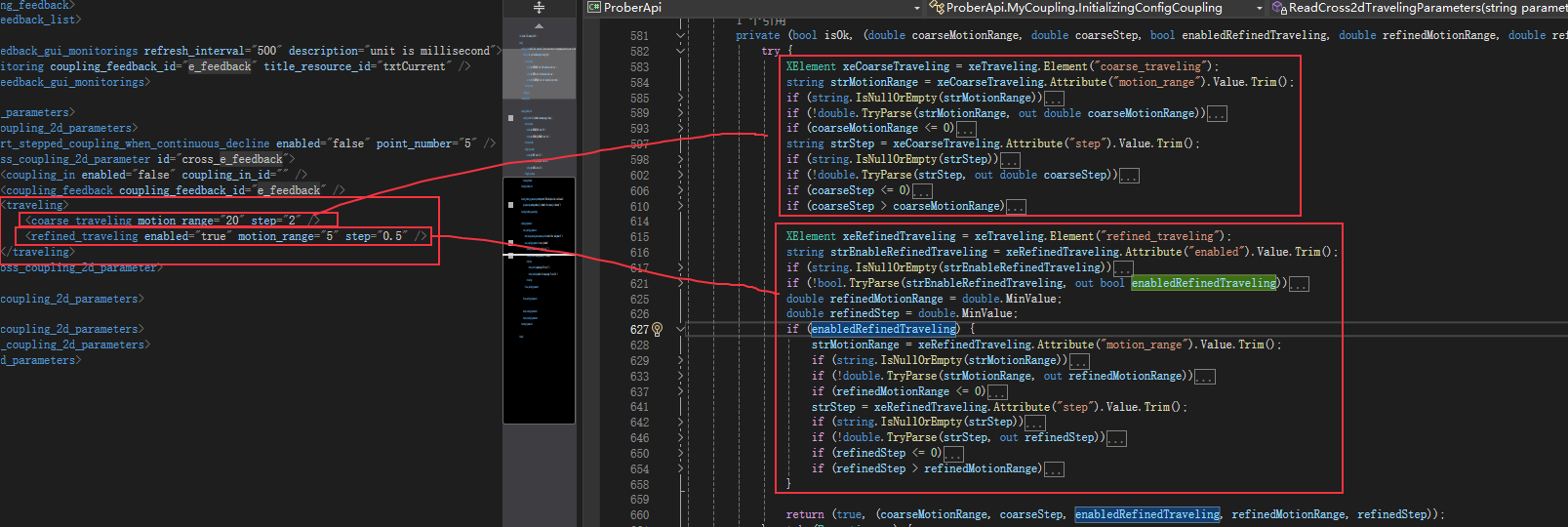


ReadCouplingFeedbackParameters：



只是检查前面的couplingFeedbackConfigDict里面有没有couplingFeedbackId。

ReadCross2dTravelingParameters：

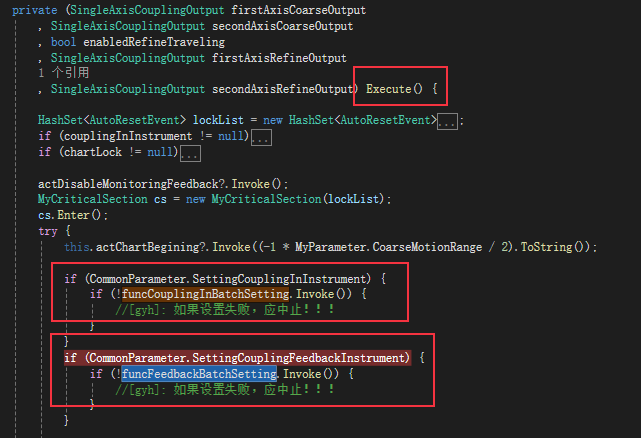


ReadCrossCoupling2dParameters看完了

再来是ReadSpiralCoupling2dParameters。这个暂时就不看了。

到这里，初始化的流程看完了，那么实际耦合时的钩子函数到底跟这里有什么关联呢？再来看看耦合代码。

钩子函数：



先来看看这个CommanParameter.SettingCouplingInInstrument



数据结构：

btnCoupling2D\_Click中的临时变量

RequestCrossCoupling2d request = new RequestCrossCoupling2d(sharedObjects);

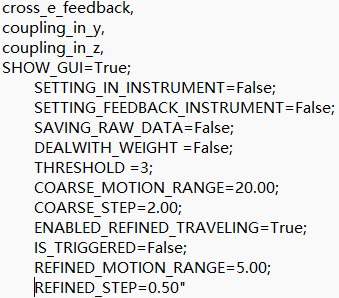
此时创建了coupling2DList

在request.TryUpdateParameters()时，创建了CrossCoupling2DStepped实例，并添加到了coupling2DList里面去。

CommanParameter是CrossCoupling2DStepped的基类的属性，在创建CrossCoupling2DStepped时由构造函数传入。

传入的参数，是在创建CrossCouping2DStepped实例前完成的，同样是在TryUpdateParameters函数中。

这个参数是从一段字串里解析出来的，这个字串是传入TryUpdateParameters的参数，大概长这样：



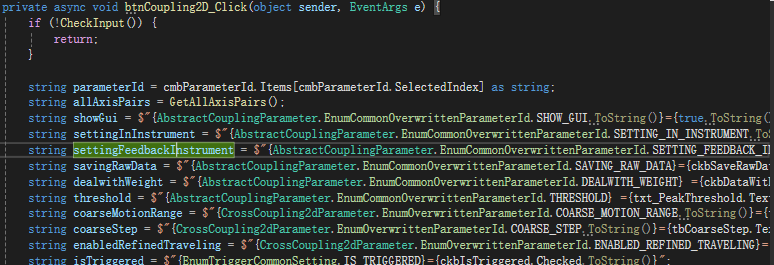
这个cross\_e\_feedback就是parameterId，

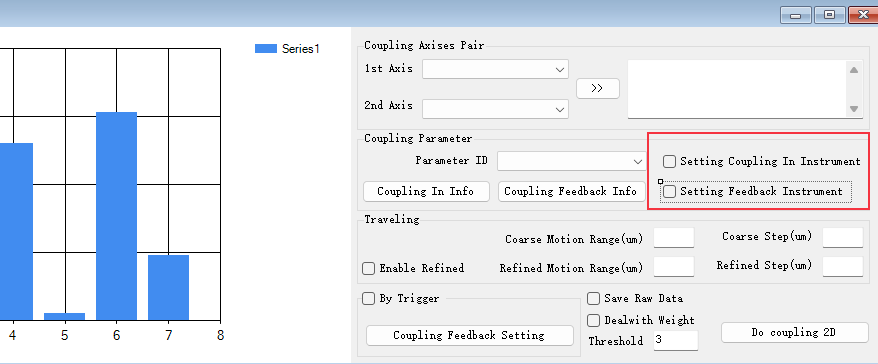
CrossCoupling2dParameter defaultParameter = coupling2dParameterDict[parameterId];

第四段，也就是用分号区隔的那一部分，都是overwritingPart。这部分会被解析出来用来更新currentParameter.

简单来说，就是这里传入的参数就是后面用到的值。

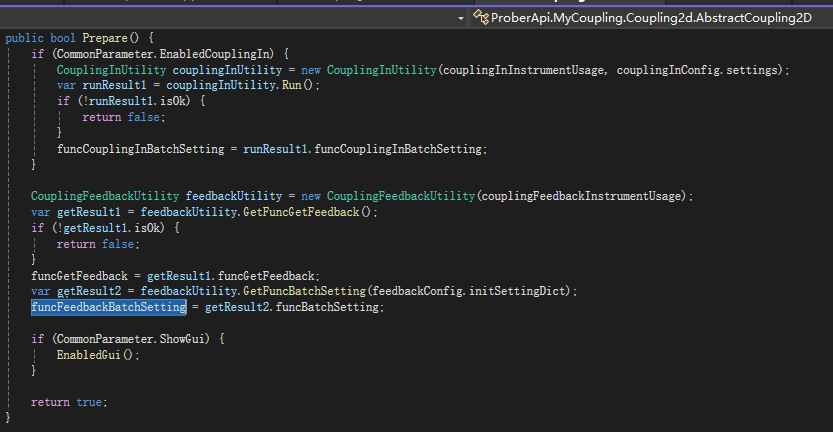
而这里传入的值，源头是UI:

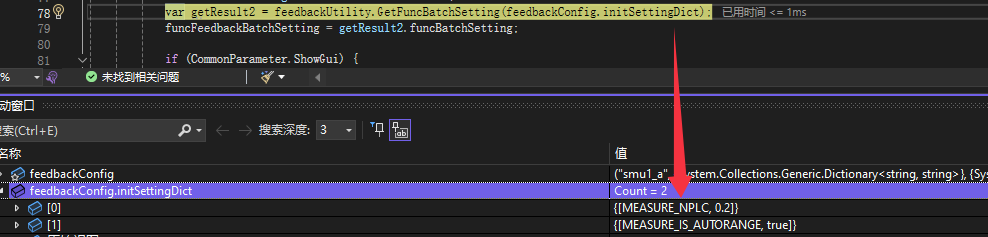


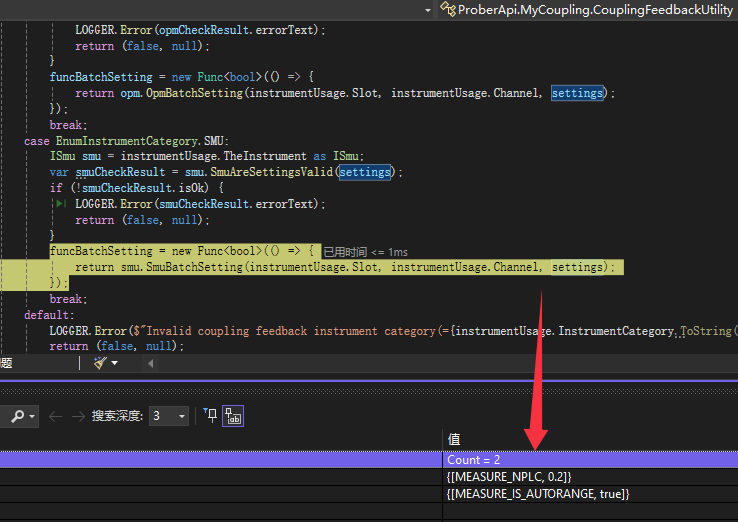


再来看两个钩子函数：

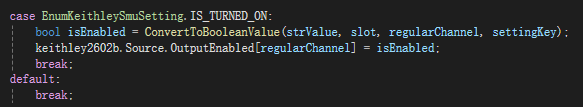
它们是在prepare这里被创建和初始化的：

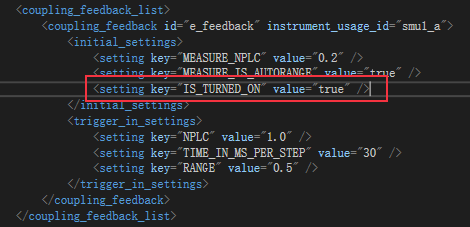






SmuBatchSetting：





所以如果xml中这样配，就会将这个设定也加到流程中去了。