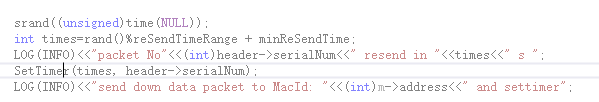


MAC层的重传定时器事件：



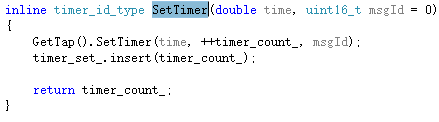
在**Module基类中**，有SetTimer函数，可以供协议写下定时器事件：

1. Mac层传入的是数据包的串号，最后定时器事件触发时，通过串号获得对应的数据包。

2. 协议基类中已经有Sap、Tap和状态机，协议构造的时候就把他们的owner设置成了协议自身，因此需要用到的时候只要GetXXX()，即可获得属于协议的成员对象。

3. 这里另外生成了一个数timer\_count\_，最后当定时器事件触发的时候，调用Module基类中的TimeOut函数，会检查是否有这个16位的数字，如果有就发布MsgTimeOut事件。

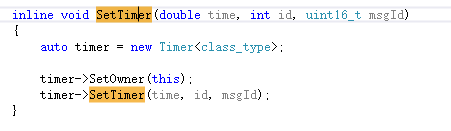
4. timer\_count\_和msgId的区别：msgId是提供给写协议者的，是可以重复的，协议可以判断MsgTimeOut事件中的msgID来判断到底是什么定时器事件发生了。而timer\_count\_是提供给Module的，timer\_set\_中保存了这些数字，可以通过time\_set\_中的数字来唯一确定这个当时设定的这个事件。



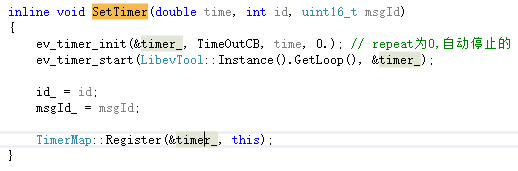
然后调用**Tap类中**的函数。首先，Tap属于传入模板OwenrType，生成了一个Tap模板类；然后，把这个Tap模板类作为Timer的模板参数传入Timer，由此生成了一个“owner”是这个Tap模板类的Timer。

得到了这各Timer对象之后，就调用其中SetTimer函数。

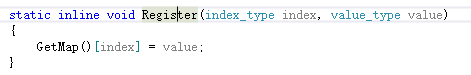




在**Timer类中**，调用函数，把定时器事件插入循环中，回调函数是TimeOutCB。同时，向TimerMap注册，事实上是一个在TimerMap里得到该定时器事件和该timer的一个映射关系。

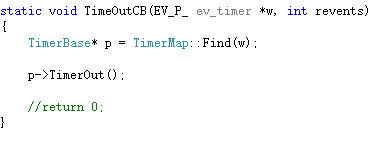


**TimerMap中**的注册函数：（事件和Timer对象的映射）

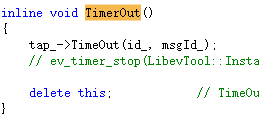


回调函数是**在类外**的，当定时器事件触发，就会在TimerMap中找到该Timer对象，然后调用其中的超时函数。

（通过在TimerMap中注册的事件，找到Timer对象，然后通过Timer对象找到Tap对象，再找到Owner，最后把MsgTImeOut事件压入该协议的事件队列中。）



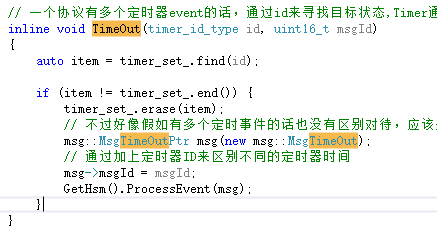
**Timer类中**的超时函数：（tap\_即Timer类对象的owner，Timer类中的SetOwner函数来设置）



**Tap中**的超时函数：



**Module基类中**的超时函数：（最后会把原来传进来的串号作为事件结构体MsgTimeOut中的一个16位数据msgID。注意：这里会先检查一下，有没有一开始设置好的timer\_count\_这个数，如果有才执行操作。）



小结：

每个协议对象有一个Tap，需要设置定时器时调用协议中的SetTimer函数。这样就会在Tap生成一个Timer对象。然后，向TimerMap注册这个事件Timer对象的对应关系。

当这个定时器事件发生，就会调用回调函数，在TimerMap中找到这个事件对应的Timer，然后找到Tap，找到Owner，从而找到最终的协议。

timer\_count\_和msgId，这两个数字是保存在Timer对象中的，因此插到循环中的定时器事件发生时，需要通过TimerMap找到Timer对象，然后再调用超时的函数。（个人认为，Timer的存在就是为了保存这两个数字，定时器事件发生时传给协议，协议才能唯一找到该事件。）

然而协议Module基类中有一个函数CancelTimer，需要根据timer\_count\_来消除这个定时器事件。事实上协议那边知道的只有msgId，除非协议那边另外保存了这个数字，否则不能通过调用这个函数来取消这个事件。