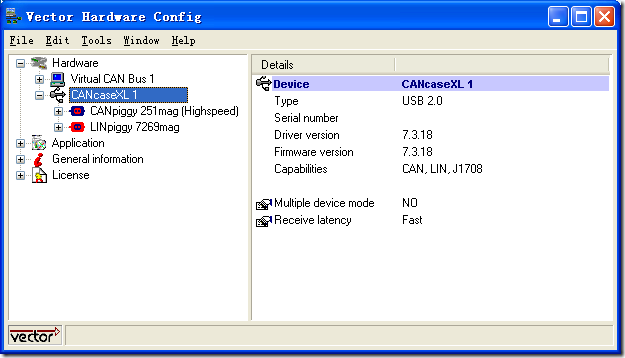
[**基础应用**](http://www.cnblogs.com/dongdonghuihui/archive/2012/09/26/2704611.html)

CANoe是Vector公司的针对汽车电子行业的总线分析工具，现在我用CANoe7.6版本进行介绍，其他版本功能基本差不多。

硬件我使用的是CAN case XL.

1,CANoe软件的安装很简单，先装驱动，再装软件。安装完成，插上USB，连接硬件，这样在控制面板中,Vector Hardware 进行查看

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261923299490.png)

通过查看信息可知，CANcaseXL中的两个piggy，一个是251（高速CAN），一个是7269（LIN），另外常用的还有1054（低速CAN，或称容错CAN），因为CANcaseXL中只能支持两路通讯，这样piggy可以自由组合

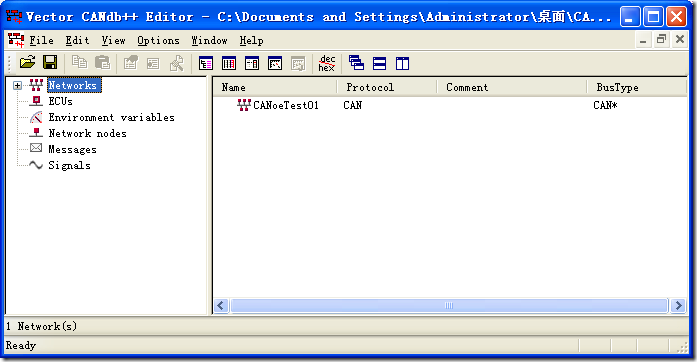
2,硬件连接正常，打开CANoe软件

File->New Configuration 可以选择新建工程的模版，我们这里选择CAN\_500kBaud.tcn,这样新建了波特率为500K CAN工程，可以File->Save Configuration,进行保存

3,接下来就要使用CAN db++ Editor工具对总线网络节点，消息，信号，进行定义了。

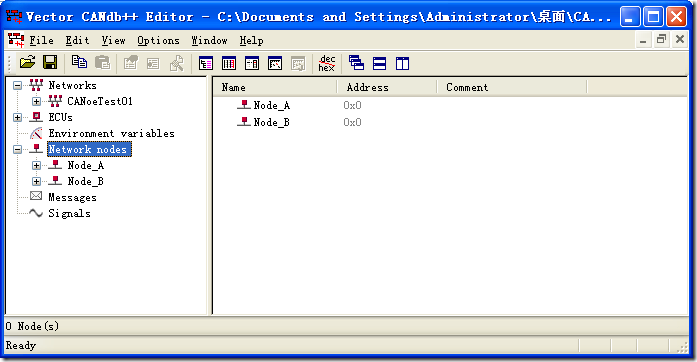
点击工具栏的这个图标[tmp5B](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/20120926192332834.png)，或开始菜单中找这个工具启动

启动后，File->Create Database,选择CANTemplate.dbc,选择目录及文件名，进行保存

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261923471895.png)

右键 Network nodes->New ，进行网络节点的定义,这里只需要填写Name即可，例如：Node\_A

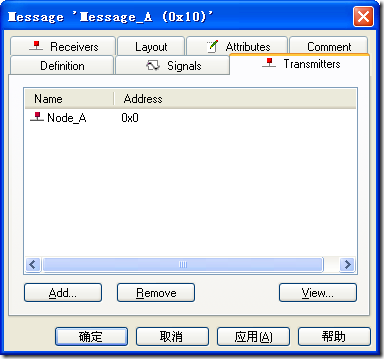
然后添加Node\_B，完成后如下图，这样在Network nodes目录下面添加出来两个节点

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261923538934.png)

节点添加完成后，下一步添加CAN消息，右键Messages->New,这是需要定义名称，ID，DLC等信息，如下：

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261924003249.png)

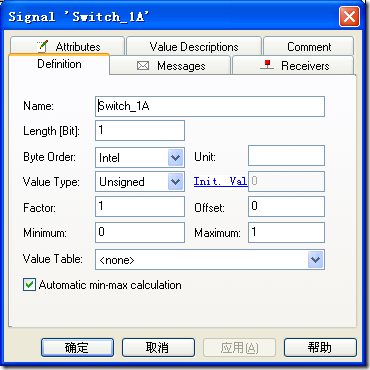
然后在Transmitters页面，点击Add按钮，添加Node\_A为发送节点，意思就是说，此消息是从Node\_A节点发送出来的

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261924071533.png)

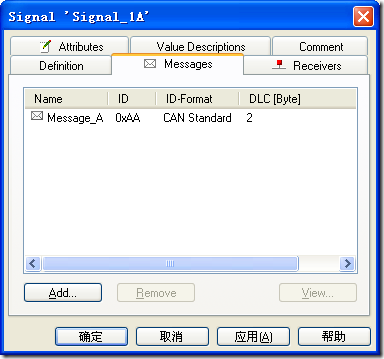
其实还有一种方法就是，此时暂时不定义发送节点，然后直接以拖曳的方式拖曳到发送节点上，功能上是一样的

有了消息，消息里携带的东西自然是信号咯，那么我们开始创建一个信号

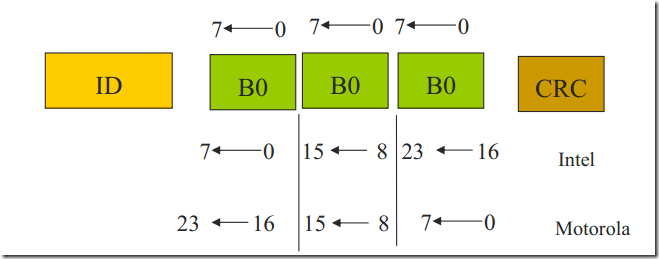
右键Signals->New,填写如下信息

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261924097030.png)

信号当然要放到消息中咯，切换到Messages页面，Add 我们刚刚建立的Message\_A,当然和上面一样，采用拖曳的方式从Signal到Message中建立关联也是可以的。

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261924127078.png)

上面信息中Intel模式和Motorola格式的区别如下：

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261924144768.png)

用0和1来表示开关信号状态似乎还不是很直观，我们可以使用on和off来表示就好了，于是View->Value Tables,右键空白处->New，键入如下信息：

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261924166361.png)[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261924189973.png)

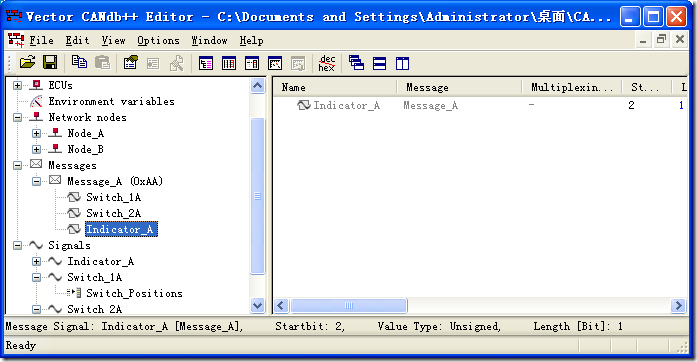
定义好了之后，还需要跟信号进行关联。右键需要关联的信号->Edit Signal,Value Table 中关联刚才建立的那个Value Talbe项

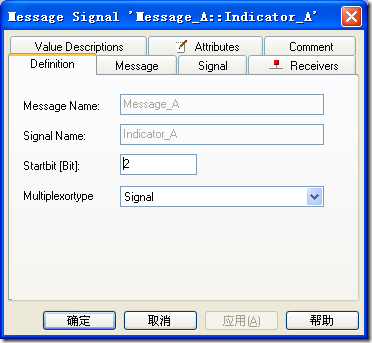
[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261924205221.png)

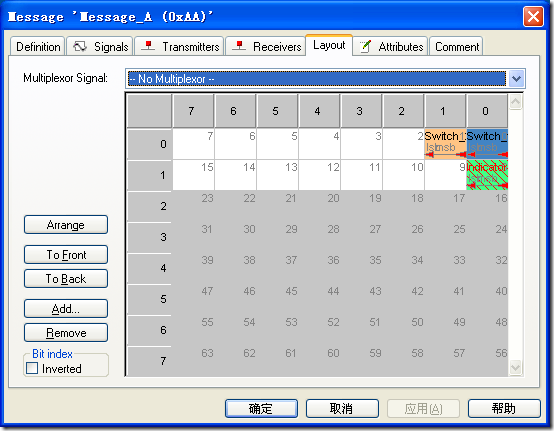
这样一个信号就完成了，另外还需要加一个类似的信号，可以点击Signal根目录，在右边选择Signal复制，及粘贴，然后双击复制好的Signal后，进行一些编辑，然后用上面方法关联Message即可快速建立第二个信号了，是不是很方便

继续添加第三个信号，Indicator\_A，跟上面一样，但是我这个信号想放在Message的第二个字节的位置，怎么办？

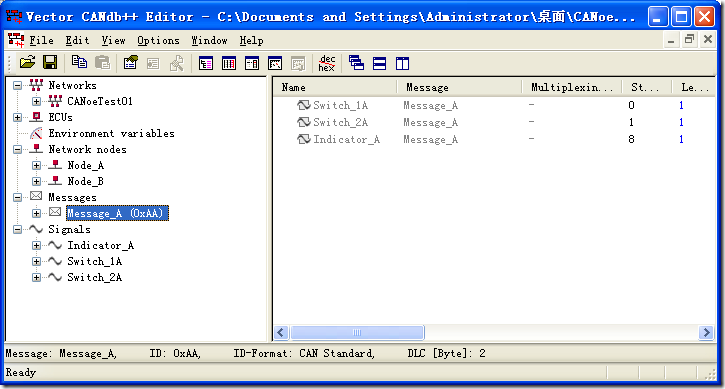
其实很简单，有两种方法：第一种，右键已经关联好的，即在Message目录下的信号，Edit mapped signal，在弹出的对话框中的Startbit中填8，这样就这个信号就跑到该消息的第二个字节中了。第二种方法，右键该消息->Edit Message,在Layout页面中，拖动信号的色块进行移动

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261924277060.png)

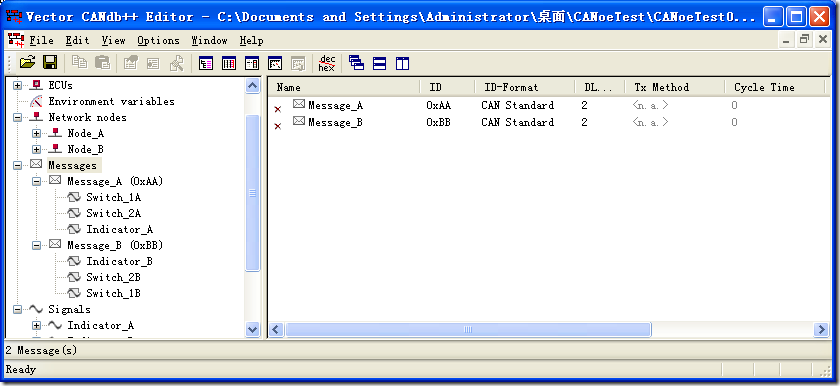
[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261924332114.png)

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/20120926192434294.png)

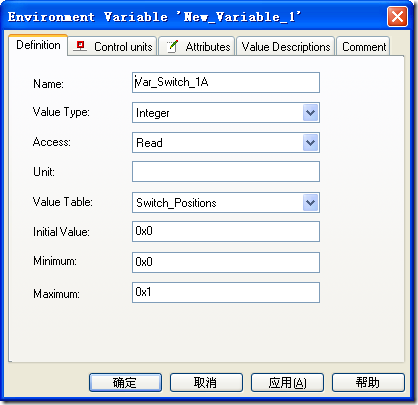
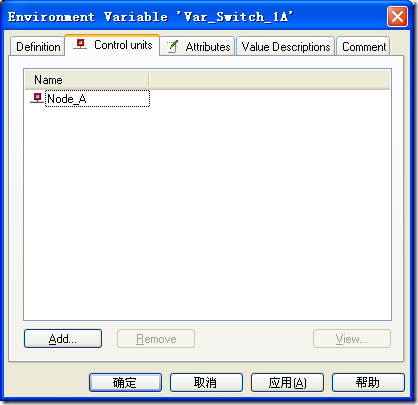
以上Message\_A消息及消息下所定义已经定义完成

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261924466500.png)

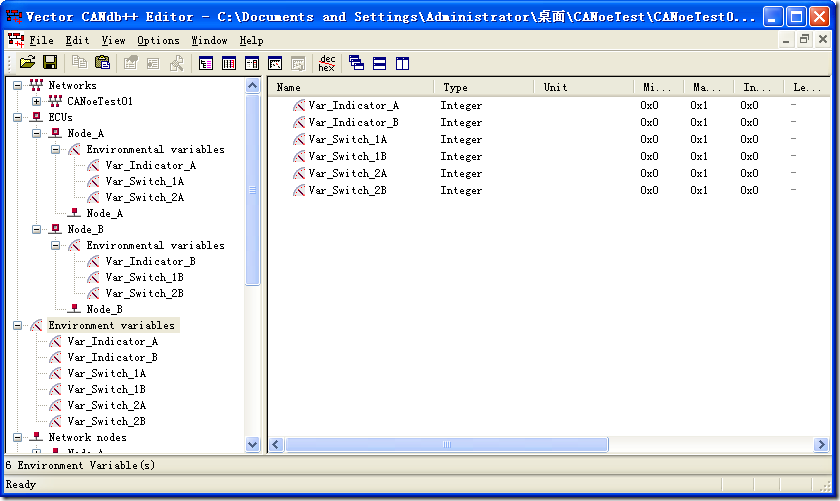
如法炮制另外一个消息Message\_B，完成后如下图。

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/20120926192455805.png)

下面创建环境变量，右键Environment variables

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261924597363.png)[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261925008052.png)

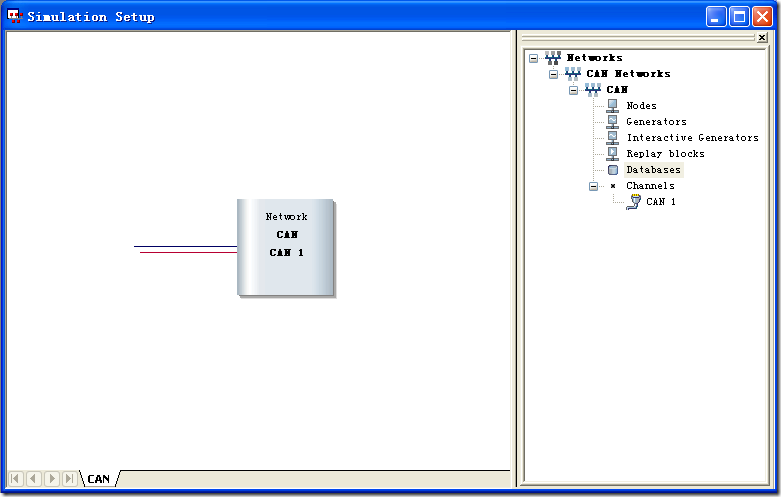
陆续把其他环境变量加好，Indicator的Access属性为Write哦，加完如下：

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261925054926.png)

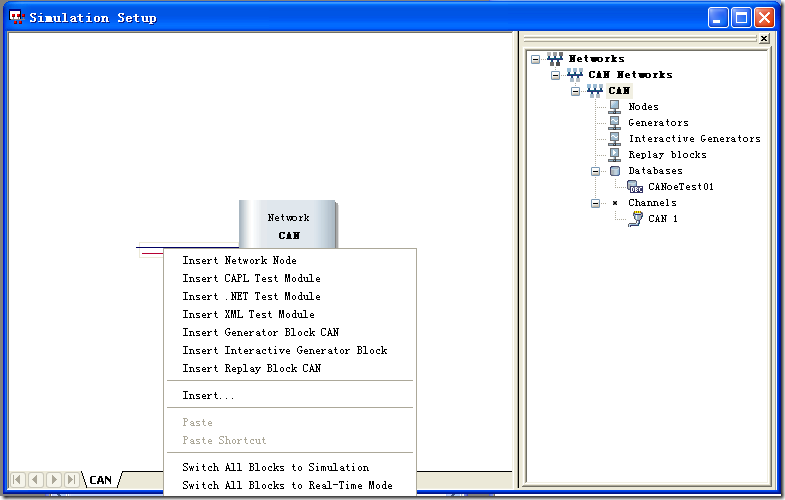
数据库的定义这款基本上就OK了。保存后关闭CANdb++ Editor,回到CANoe主程序中。

请在Configuration->Options->Configuration Settings->Channel Usage 设置使用的通讯通道，因为这个只需要1路CAN，其他都设置为0就可以。

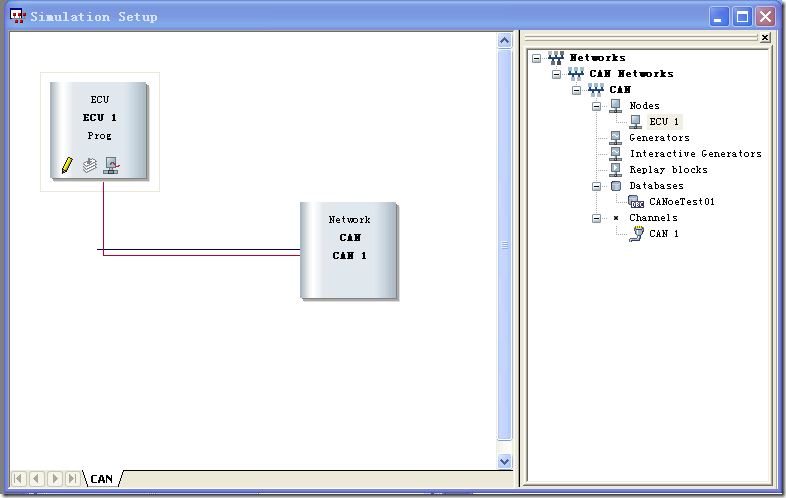
点这个选项页面[tmpB9](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261925071286.png)可出现Simulate Setup 界面

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261925099815.png)

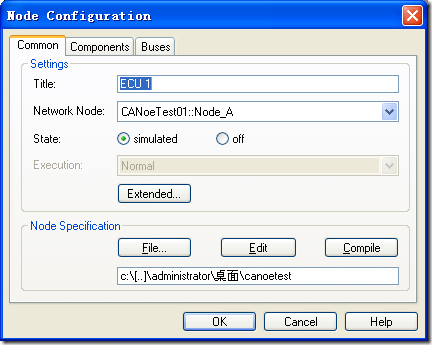
右键Databases->Add,把刚刚创建好的dbc给加载进来，之后再连线上右键->Insert Network Node

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261925111159.png)

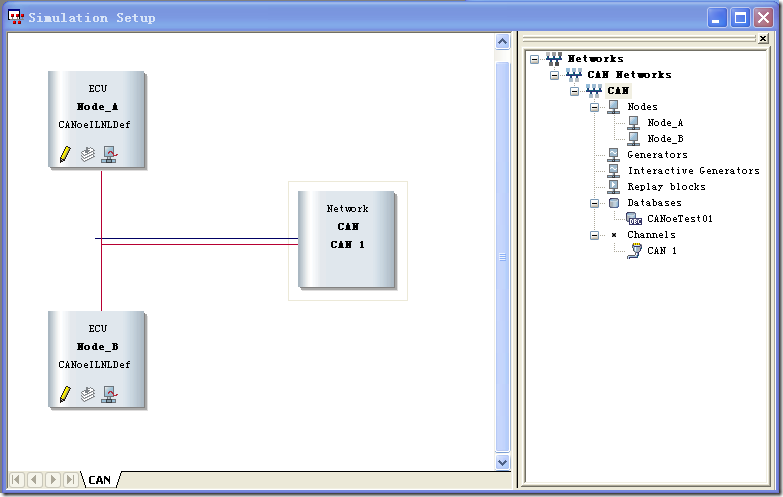
结果如下：

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261925161098.png)

右键这个ECU模块->Configuration,在Network Node 选择Node\_A,完成，然后添加Node\_B

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261925183837.png)

完成后如下图，这样dbc与CANoe的关联就好了。

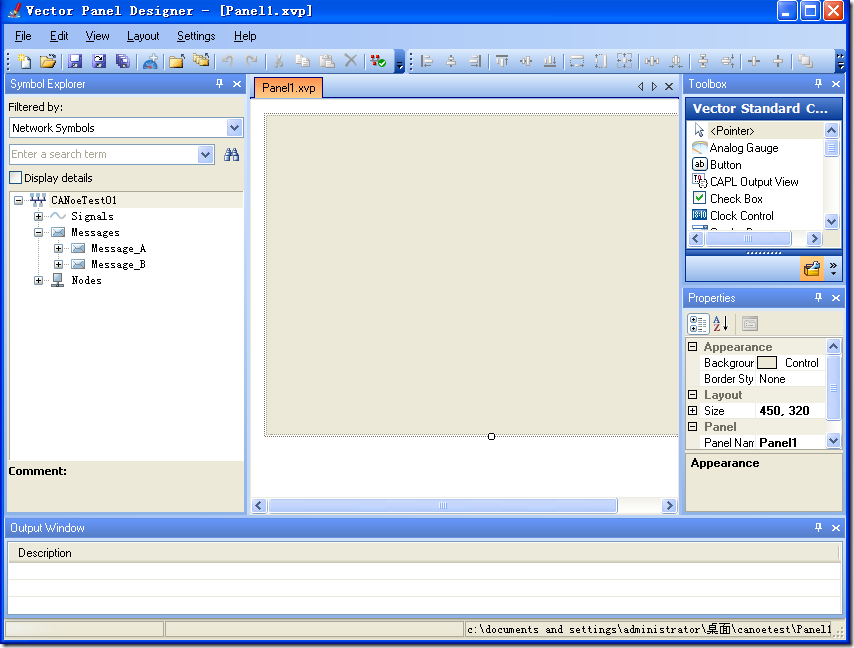
[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261925204069.png)

4，界面设计

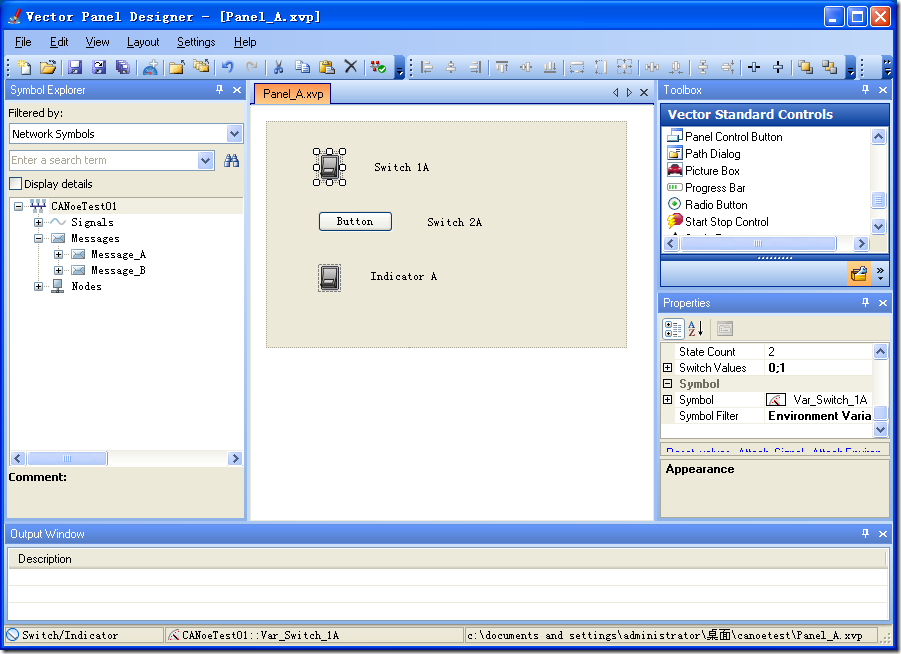
网络及相应的关系设置好之后，应该设计界面了，一个良好的界面可以直观的显示及控制各种信号。

界面设计工具有两种：一种是Panel Designer,新的界面设计工具，另外一种是Panel Editor是老的界面设计工具，无论采用何种工具设计的界面，都可以和CANoe很好的兼容，没有问题。下面我们先用第一种界面设计工具来完成界面的设计

点击工具栏的这个图标[tmpE](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261925233809.png)或File->Open Panel Designer,打开界面编辑器，界面如下：

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261925307044.png)

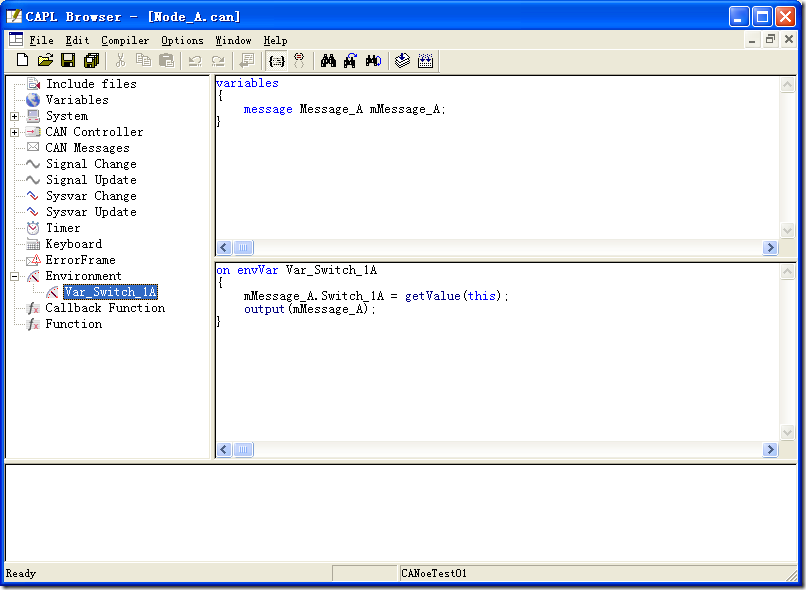
保存一下该界面，起名为Panel\_A,一个空的界面上可以放置右上边若干组件。放置好之后，选中界面上的组件的时候，右下侧回显示当前选中的组件的属性，可以更改相应的大小位置等，但最重要的是Symbol属性，这里要说一下，每个界面组件也都需要与CANoe关联，否则单纯的界面是没有任何意义的，关联主要是信号和环境变量，在Symbol属性的Symbol Filter属性中可以进行选择，我们这里关联的是环境变量，所以选择Environment Variable，然后再点击上面的Symbol属性，然后在弹出的页面中选择相应关联的变量

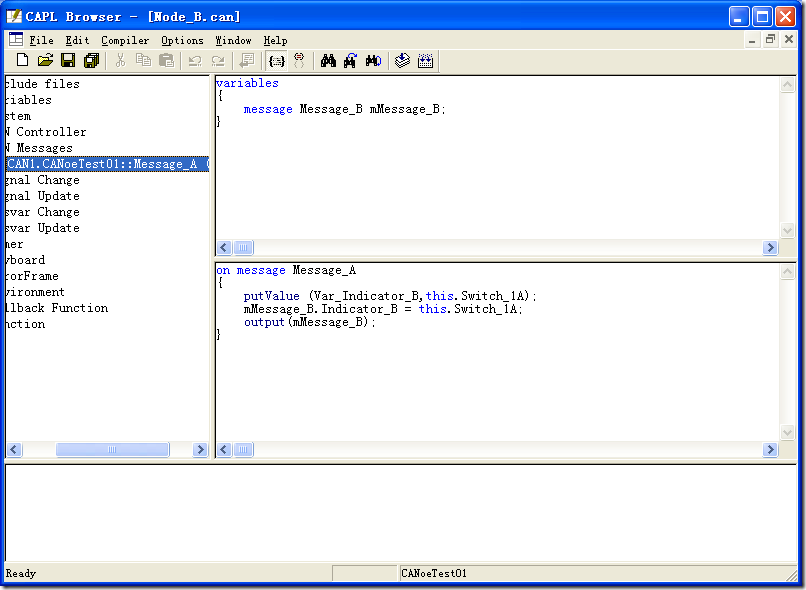
[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261925402594.png)

完成后可以按同样的方法设计Panel\_B,也可以Save As 的方法快速建立同样的Panel但是与环境变量的关联和界面的文字需要进行微调。这样两个Panel都已完成。也已经建立了与CANoe之间的关联。

5，CAPL编程

所有的前期准备工作和组装工作都已完成，现在需要编写程序使整个系统有机的运行起来。回到CANoe的Simulation Setup界面，然后点击Node\_A模块的，像铅笔一样的图标[tmp22](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261925442641.png)，填写程序名称，这里Node\_A,然后打开编程界面，完成以下程序。然后完成Node\_B程序。小技巧：如果背不出具体的消息名称，信号名称或环境变量，可以右键程序空白处，你会得到意外的惊喜，试试看。

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261925488370.png)

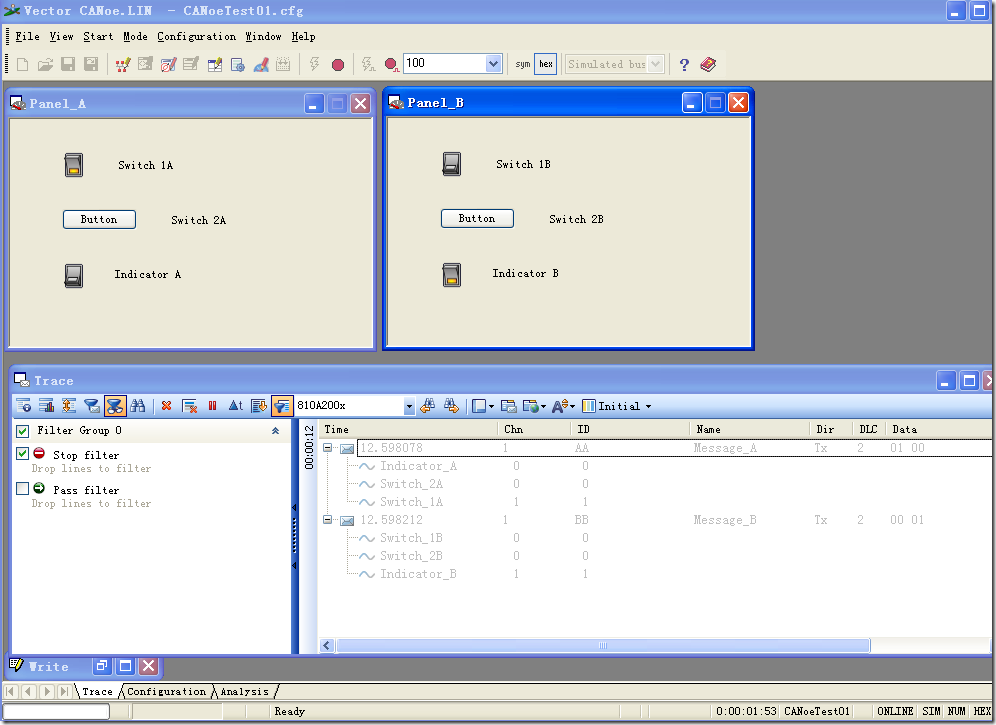
[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/20120926192553916.png)

OK大功告成

6，运行

现在点击工具栏[tmp40](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/20120926192557964.png)，运行程序。注意：[tmp41](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261925574344.png)这里表示实际的网络，所有的消息是发送到物理的网络上的，如果只是验证程序，设置成这样就可以了[tmp43](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261925579360.png)

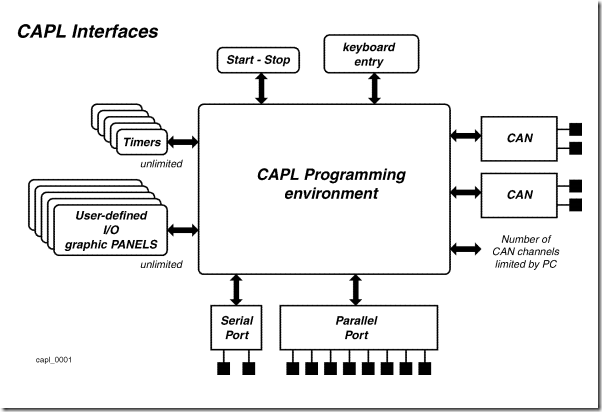
完整运行结果如下：

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261926005197.png)

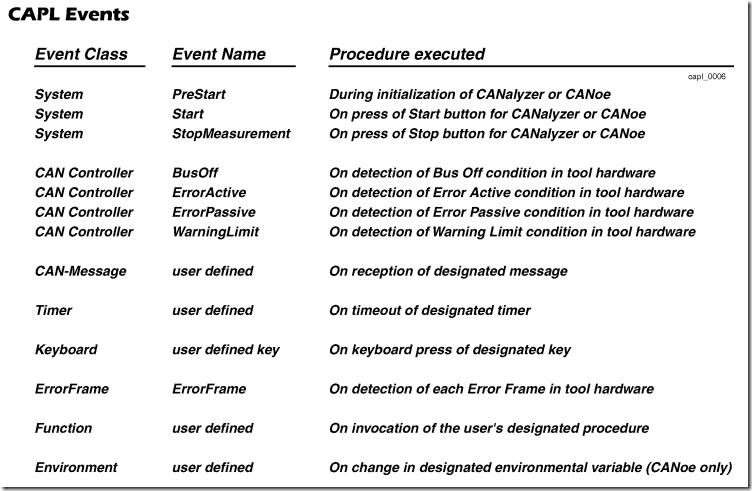
就是这样，一步一步的，我们完成了第一个完整的CANoe应用工程，虽然这只是入门第一步，但整个CANoe的开发流程基本涵盖，如果能跑出上面的运行结果，依然能够给我们很多的惊喜。我们需要这样的惊喜，因为这是我们继续前行的动力。

[**CAPL编程**](http://www.cnblogs.com/dongdonghuihui/archive/2012/09/26/2704619.html)

CAPL就是Communication Application Programming Laguage的缩写，CAPL类似于C语言的语法，因此所有的语法请参考C语言教程，这里不在这里进行详述，关于C语言，毫无疑问的，首先推荐大家看谭浩强老师的书，经典中的经典，看完这本C语言应该没问题了。CAPL在CANoe中起到重要的作用，他将所有的部分联系起来（请看下图）现在我们给予上一节的基础上，来着重介绍CAPL编程。

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261929294920.png)

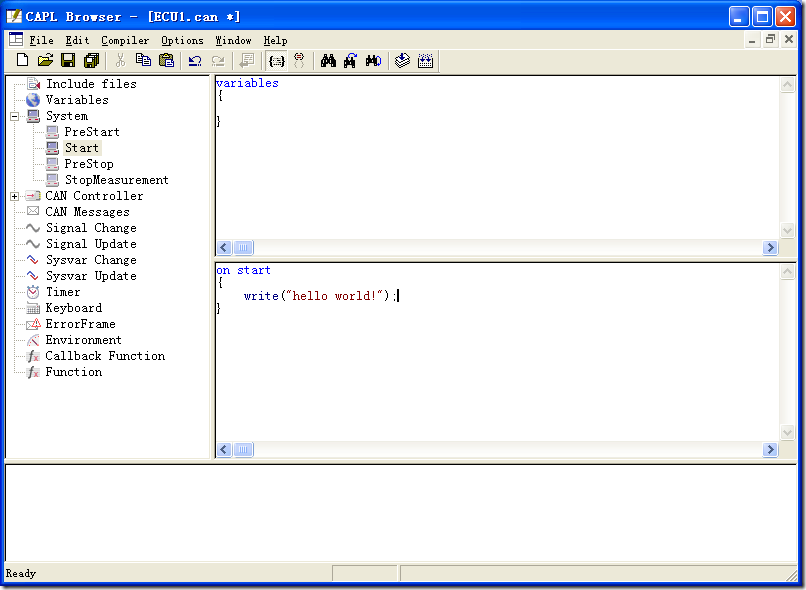
CAPL语法是C语言的，又有一些C++的特性，this指针，事件等，对于事件的理解可以学习任意一种面对对象语言，首推C#.NET,可以参考我的博客学习，当然这个似乎比CAPL本身更加难，事件也更加多，但这并不妨碍对事件的理解，其中的事件类型如下图：

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261929324902.png)

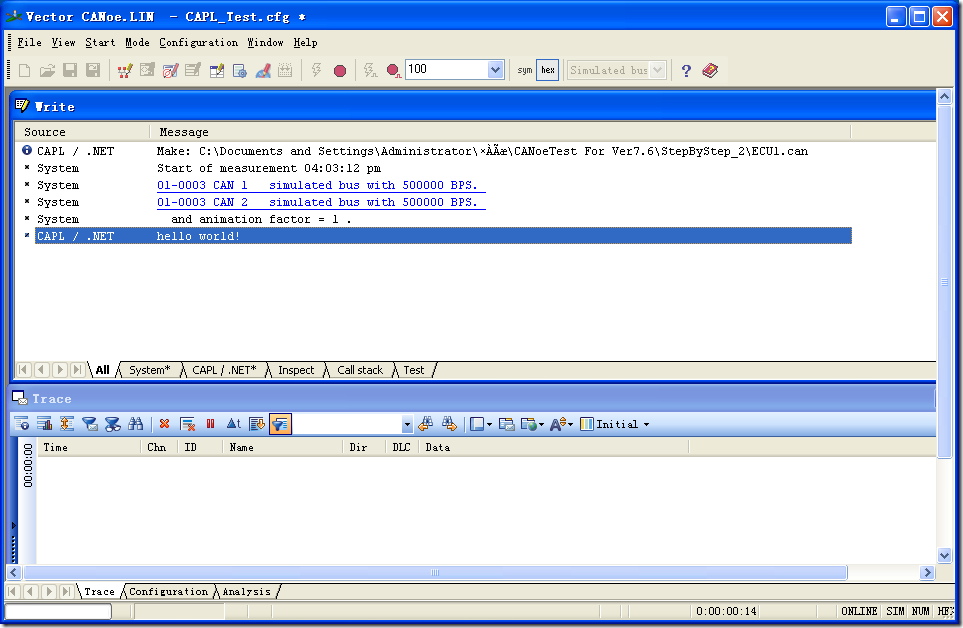
下面来几个小例子帮助理解

来个CANoe版本的hello world！来兴奋一小下吧。开整~

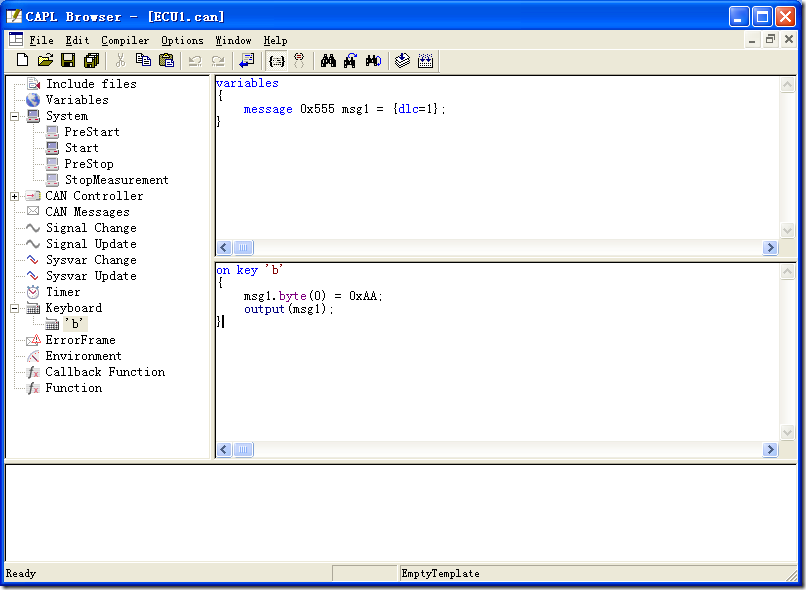
在打开CANoe，新建个工程，在Simulation Setup中加个Network node ,点铅笔，忘了吗，上一节刚讲过，右键Start->New,键入如下代码

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261929385006.png)

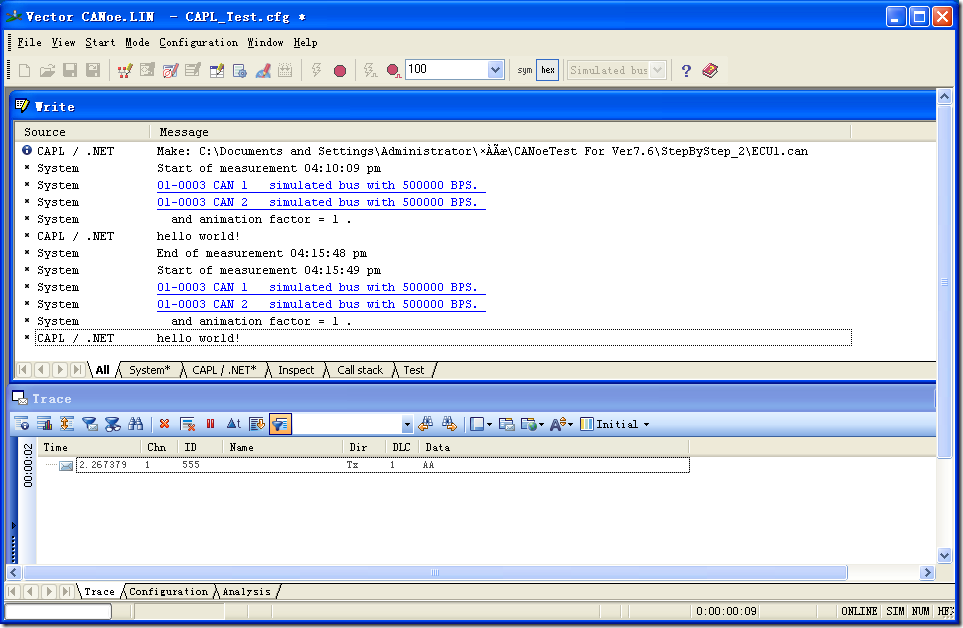
编译后，关闭CAPL Brower。运行CANoe工程，结果如下,哇塞成功咯，好有成就感啊。

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261929467293.png)

这个例子似乎跟CAN通讯没啥太大关系啊，好，我们接下来再做一个例子

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261930059797.png)

运行结果如下，按键盘b键，将发送一个CAN消息，连dlc是啥都不知道的同学，推荐先学习一下CAN基础知识，推荐瑞萨公司的《CAN入门书》，讲的非常的好。

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261930204796.png)

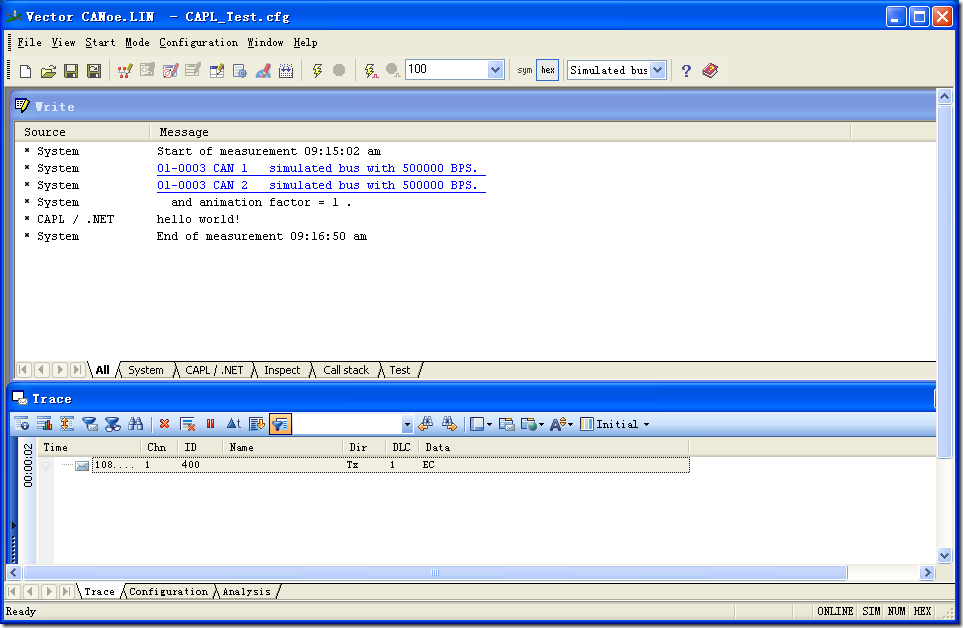
运行刚刚的那个例子你会发现，按一下b，只发送一条消息，但在实际应用中CAN消息都是循环连续发送的，我们要对刚刚的程序进行一些修改。完成这样的功能。程序如下：

variables  
{  
message 0x400 msgA = {dlc=1};  
mstimertimerA;  
intconditionA = 0;  
}

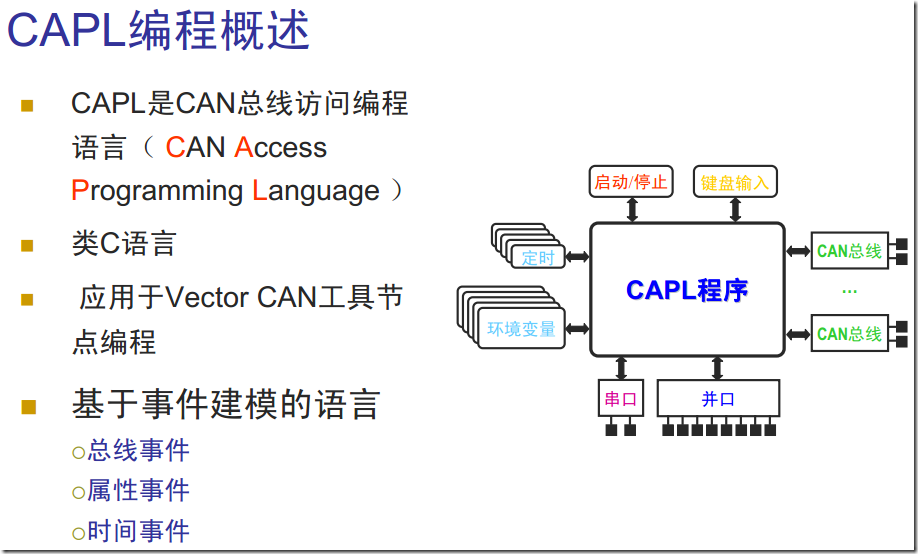
on key 'a'  
{  
conditionA = !conditionA;  
if(conditionA ==1)  
{  
setTimer(timerA,200);  
}  
}

on timer timerA  
{  
if(conditionA==1)  
{  
setTimer(timerA,200);  
}  
msgA.byte(0) = msgA.byte(0)-1;  
output(msgA);  
}

运行结果如下：按A键，Timer启动，发送消息

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261930544491.png)

接下来我们一起来看总结一下CAPL编程的要点：

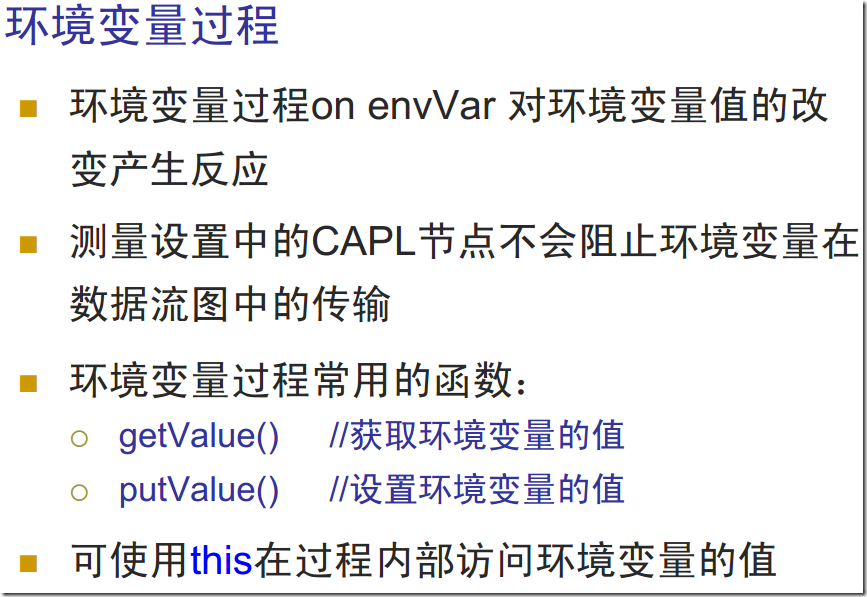
[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261930575693.png)

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261931099947.png)

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261931208104.png)

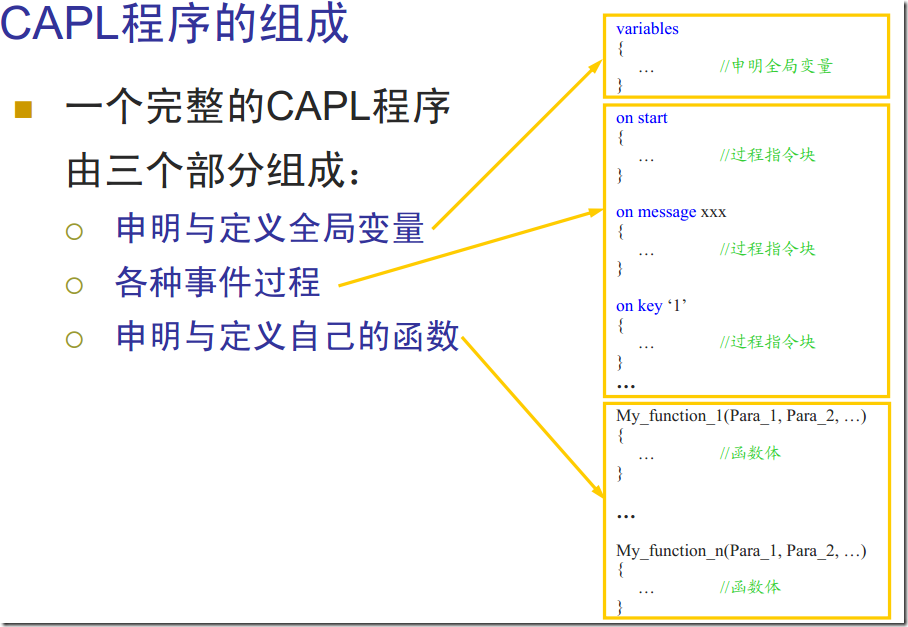
[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261931286977.png)

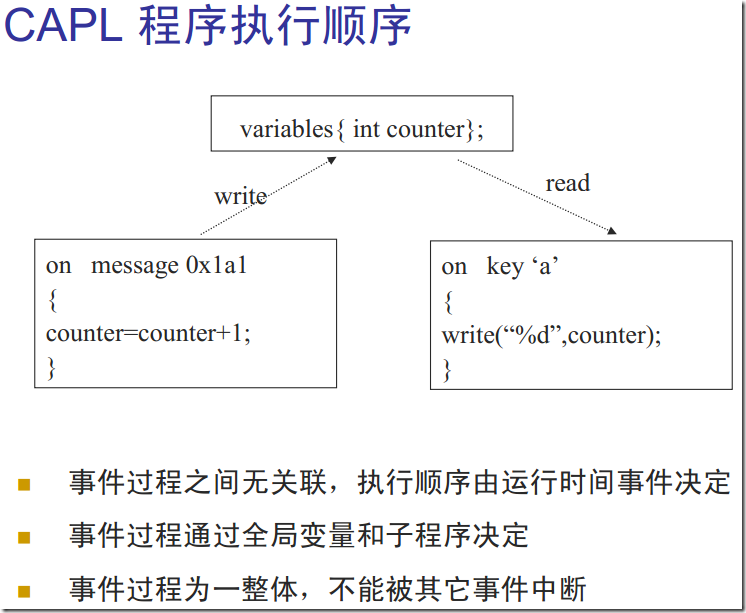
[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261931383084.png)

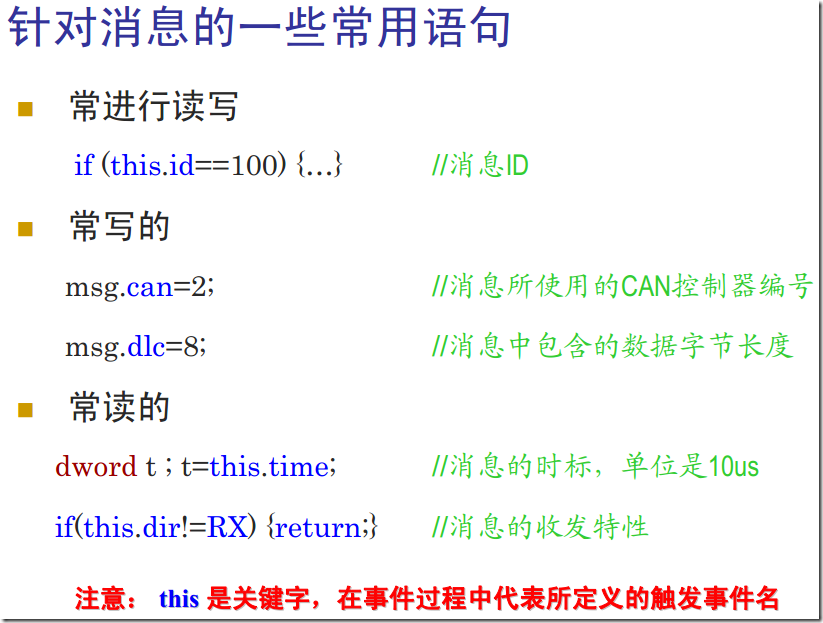
[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/20120926193145745.png)

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261931534046.png)

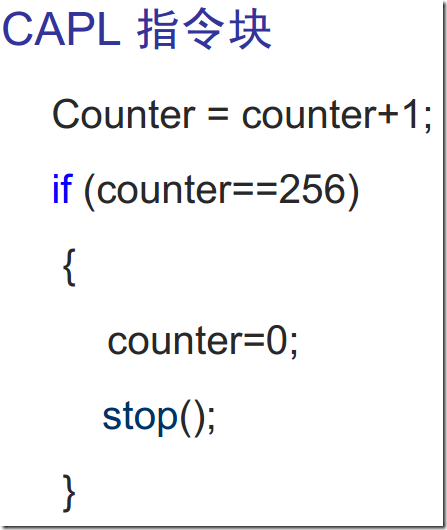
[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261931584673.png)

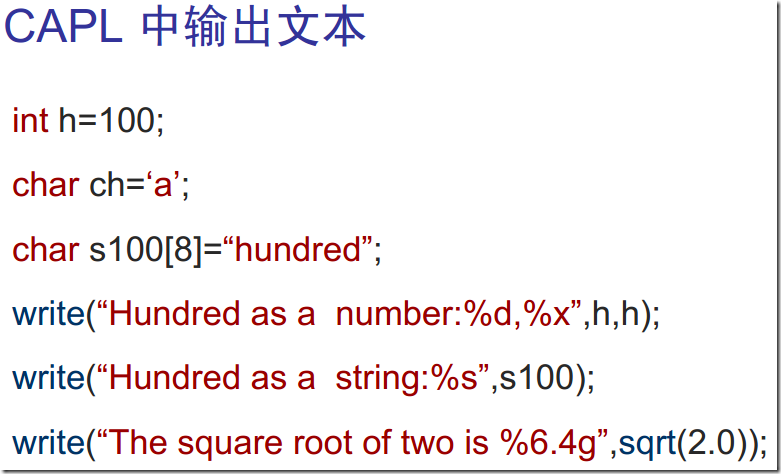
[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/20120926193206449.png)

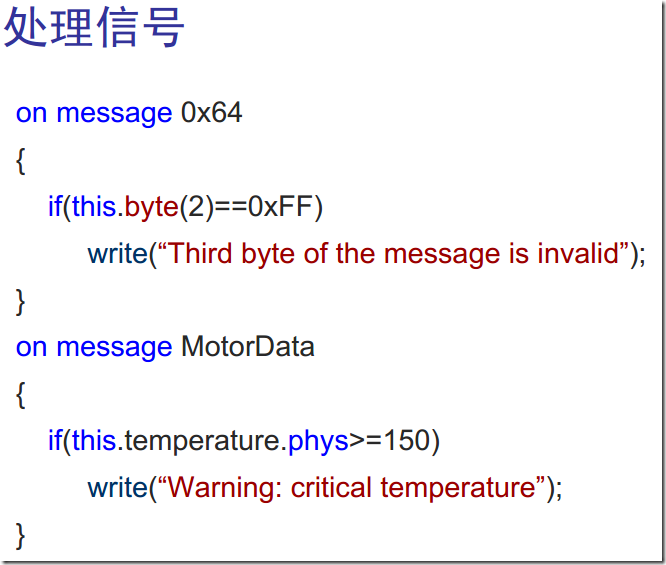
[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261932131209.png)

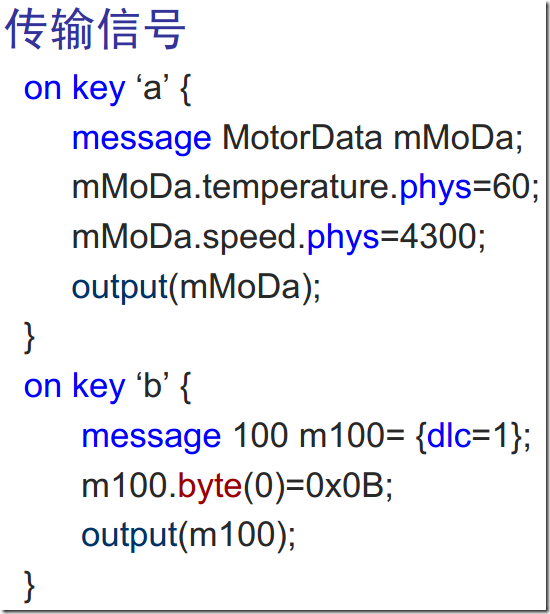
[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261932216461.png)

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261932282204.png)

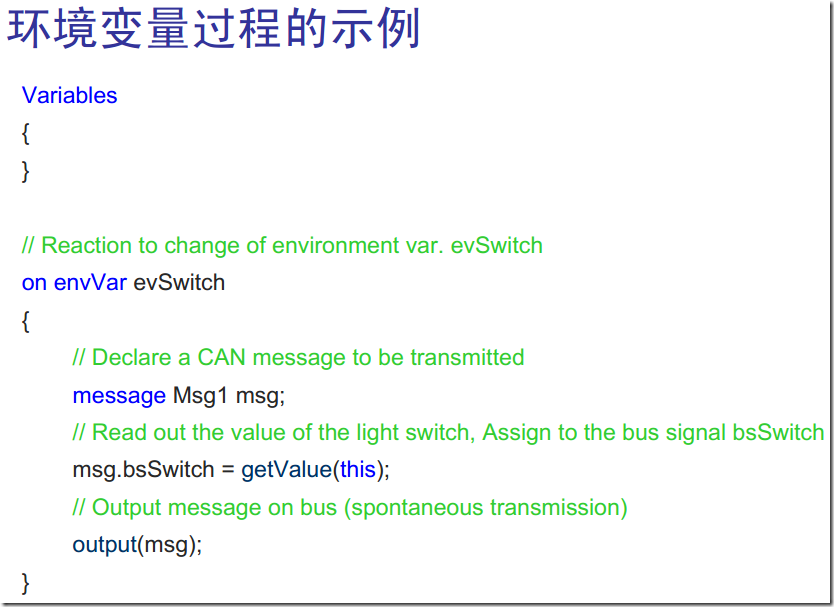
[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261932323779.png)

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261932388311.png)

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261932551555.png)

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261932584917.png)

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261933041923.png)

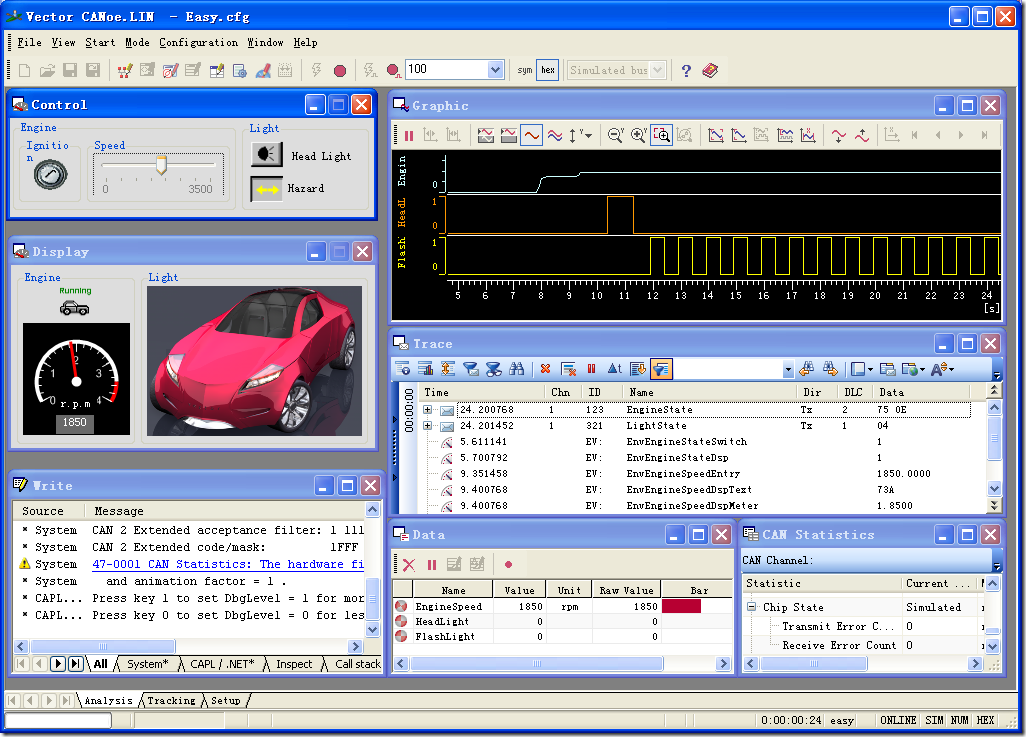
[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261933118223.png)

CAPL编程的学习，需要通过不断在实践中的积累，此外别无他法。以上真真儿的只是入门，如果你真心看过，不如你真心写过。

[**简单例子的剖析**](http://www.cnblogs.com/dongdonghuihui/archive/2012/09/26/2704623.html)

最好的学习方式是什么？模仿。有人会问，那不是山寨么？但是我认为，那是模仿的初级阶段，当把别人最好的设计已经融化到自己的血液里，变成自己的东西，而灵活运用的时候，才是真正高级阶段。正所谓画虎画皮难画骨。但初级阶段仍然是必须经历的过程，他会使你在达到高级阶段的过程中少走很多弯路，下面我们来迈出这一步。先研究一下别人的简单例子。

最好的例子莫过于Vector本身的Demo了，这个在安装完CANoe之后就会被自动安装。先看最简单的一个，名字叫Easy，但并不简单哦，比我们之前介绍的所有的东西都整合再一起了，很简单，但很全面。但是假如你说，这个我自己也可以完全自己写出来（并不是仅仅是看懂哦），那么我可以肯定的说，在工作中，你完全可以胜任一般的任务要求哦~，剩下的只是工作量的问题了。但我相信到现在为止，你们很多人，都无法写出这样的程序，所以我建议你们把这个程序好好的研究明白，这点很重要。废话不多说，上图，下面是打开运行后的界面。

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261936237242.png)

通过面板可以控制，及显示很多动画效果，做的非常的漂亮。在其余的窗体也将主要的数据以图表等表现方式呈现出来。

我们先看一下DBC的内容吧

Signals：

EngineSpeed车速信息

FlashLight双跳灯

HeadLight大灯

OnOff引擎状态

Messages:

EngineState引擎状态：包含的信号有OnOff，EngineSpeed

LightState灯光状态：包含的信号有FlashLight，HeadLight

Network nodes:

Display 显示节点，接收所有消息

Engine 引擎节点，发送EngineState消息

Light 灯光节点，发送LightState消息

Environment variables: 环境变量，一般与界面的组件相关联，这样就实现了图形化界面的控制与显示，下面就是关联的界面组件

EnvEngineSpeedDspMeter[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261936302080.png)

EnvEngineSpeedDspText[tmp61](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/20120926193632119.png)

EnvEngineSpeedEntry[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261936323183.png)

EnvEngineStateDsp[tmp5E](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261936329836.png)

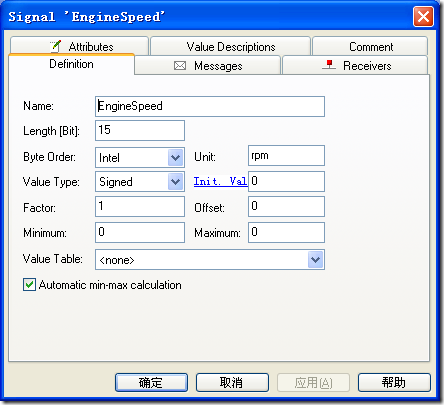
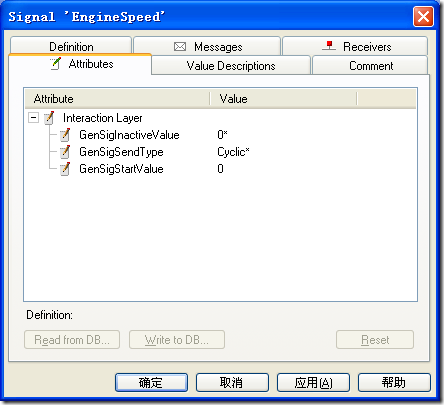
EnvEngineStateSwitch[tmp56](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261936325725.png)

EnvHazardLightsSwitch[tmp57](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/20120926193633741.png)

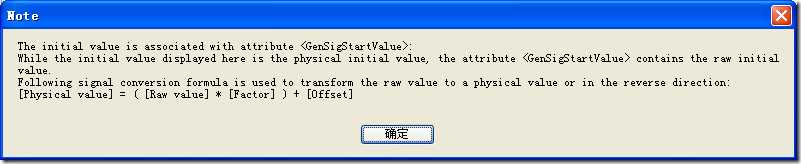
EnvHeadLightSwitch[tmp58](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261936339346.png)

EnvLightDsp[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261936335234.png)

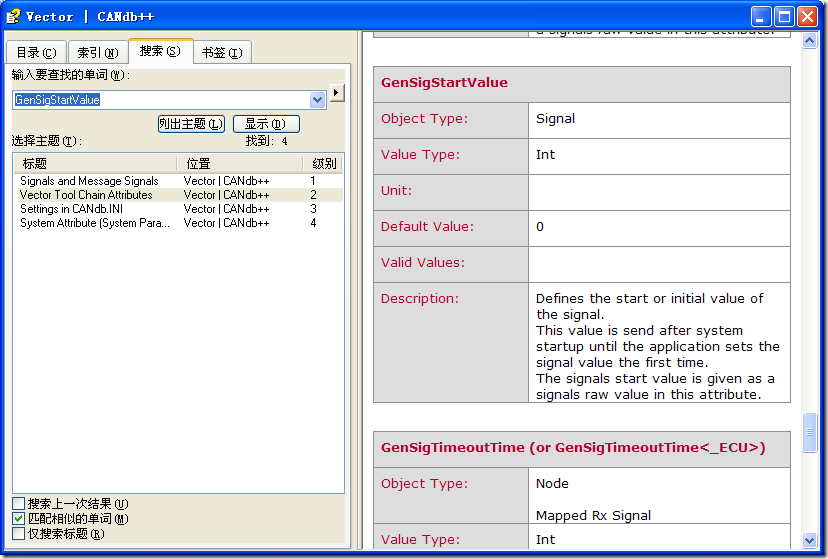
注意一下信号的信息：

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261936341463.png)[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261936363612.png)

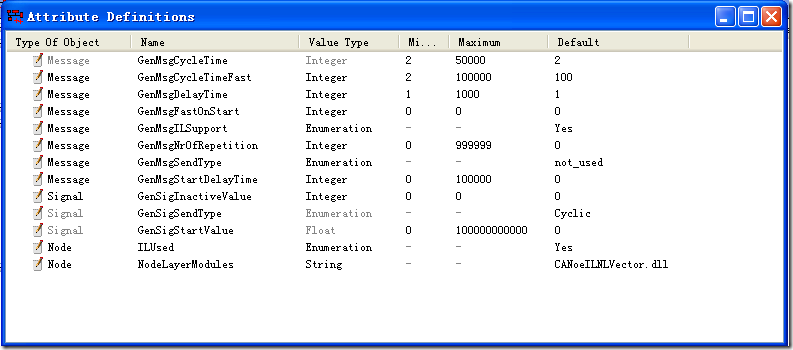
Definition页面的，Init.Val的输入框使能了，之前是灰色的状态，为什么呢？点击一下蓝色的带下划线的连接，弹出窗台如下：

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/20120926193641794.png)

意思是说这个值的设置，必须要定义的属性才能有效，之前一直没有提到信号的属性，这次还是第一次遇到哦。个人理解信号属性是表明信号的特点的一系列参数，当然消息和节点也都有对应的属性。为了更加详细的了解这个属性，我们求助于帮助。

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261936474245.png)

哦，明白了，原来是用来初始化数据的哦。其实在Definition表示的是物理值，都要转换成Raw值保存到GenSigStartValue属性中。在属性的创建我们之前也没有提到过，这里讲一下，请在CANdb++ Editor菜单中，View->Attribute Definitions

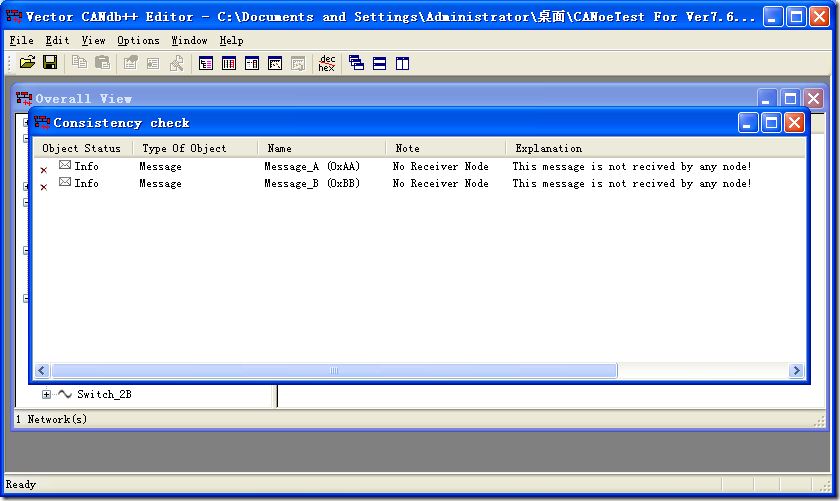
[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/20120926193651148.png)

右键，New，填写好信息即可。属性背后跟行为是密切相关的，甚至跟底层dll，其他的一些属性请参考Help文档，当然重要的属性我们也会跟大家在后面提到。

dbc还有一些细节，就是接受的消息的定义，之间也没介绍过，例如Display节点只接收消息，那么你就应该在节点的属性上进行配置，方法是右击节点然后点Edit Node，在Mapped Rx Sig.中就可以定义接收的信号了，Add…

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261936523345.png)

其实不定义接收消息也是可以的，但会在File->Consistency check 的检查中中显示出无接收节点等的报警。例如前面第一讲例子的dbc的检查如下：

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/20120926193657285.png)

再看一下CAPL程序。

engine.can程序如下：

variables  
{  
}

on envvarEnvEngineStateSwitch //当拨动开关的时候，会更改发动机发出的信号  
{  
$EngineState::OnOff = @this; //注意信号和环境变量直接赋值时的符号，信号用$,环境变量用@  
if(@this)  
$EngineState::EngineSpeed = @EnvEngineSpeedEntry;  
else  
$EngineState::EngineSpeed = 0;  
}

on envvarEnvEngineSpeedEntry //当移动车速滑条时，会更改发动机发出的信号  
{  
if(@EnvEngineStateSwitch)   
{  
$EngineState::EngineSpeed = @this;  
}  
}

on start //程序开始运行的时候，将调用所有的环境变量的事件  
{  
CallAllOnEnvVar(); // call all envvar procedures of this model and  
// thus consider the START VALUES of all environment  
// variables for:   
// - initialization of all message variables  
// - starting of any timers  
// - sending messages (output) with start values  
}

light.can的程序如下：

variables   
{  
msTimertFlashLightFrequency; //定义闪灯定时器  
constintgFlashLightFrequency = 500; //定义闪灯频率，初始化为500ms  
intgHazardLightsStatus = 0; //定义危险灯信号

intgDebugCounterTX = 0; //用于调试，记录TX报文个数  
intgDebugCounterTXRQ = 0; //用于调试，记录TXRQ报文个数  
intgDebugCounterRX = 0; //用于调试，记录RX报文个数  
}

on envvarEnvHeadLightSwitch //大灯开关状态更改时，更新灯光消息的信号  
{  
// assign EV value to the message signal  
$LightState::HeadLight = @this;  
}

on start  
{  
CallAllOnEnvVar(); // call all envvar procedures of this model and  
// thus consider the START VALUES of all environment  
// variables for:   
// - initialization of all message variables  
// - starting of any timers  
// - sending messages (output) with start values

setWriteDbgLevel(0); // set DbgLevel = 1 to get more information in Write-Window  
}

on message LightState //调试用，打印相关信息  
{  
if (this.dir == TX)  
{  
gDebugCounterTX++;  
if(gDebugCounterTX == 10)  
{  
writeDbgLevel(1,"LightState TX received by node %NODE\_NAME%");  
gDebugCounterTX = 0;  
}   
}  
if(this.dir == TXREQUEST)  
{  
gDebugCounterTXRQ++;  
if(gDebugCounterTXRQ == 10)  
{  
writeDbgLevel(1,"LightState TXREQUEST received by node %NODE\_NAME%");  
gDebugCounterTXRQ = 0;  
}   
}  
if (this.dir == RX)  
{  
gDebugCounterRX++;  
if(gDebugCounterRX == 10)  
{  
writeDbgLevel(1,"Error: LightState RX received by node %NODE\_NAME%");  
gDebugCounterRX = 0;  
}  
}  
}

on envVarEnvHazardLightsSwitch //危险警示灯开关变化时，更新灯光消息的闪灯信号  
{  
if (@this)  
{  
gHazardLightsStatus = 1;  
setTimer(tFlashLightFrequency, gFlashLightFrequency);  
}  
else  
{  
cancelTimer(tFlashLightFrequency);  
gHazardLightsStatus = 0;  
}

$LightState::FlashLight = gHazardLightsStatus;  
}

on timer tFlashLightFrequency //危险报警灯间隔闪烁的控制  
{  
gHazardLightsStatus = (gHazardLightsStatus == 1 ? 0 : 1);  
$LightState::FlashLight = gHazardLightsStatus;  
setTimer(this, gFlashLightFrequency);  
}

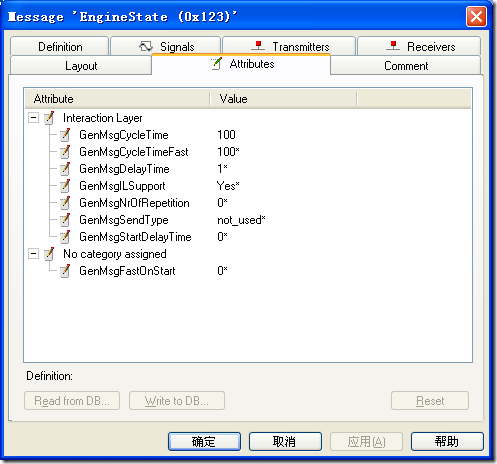
on key '0' //按键事件，定义打印调试信息的等级  
{  
setwriteDbgLevel(0);  
}

on key '1' //按键事件，定义打印调试信息的等级  
{  
setwriteDbgLevel(1);   
}

以上程序，有C语言基础的同学应该都可以看得懂，这里不用详细介绍了。

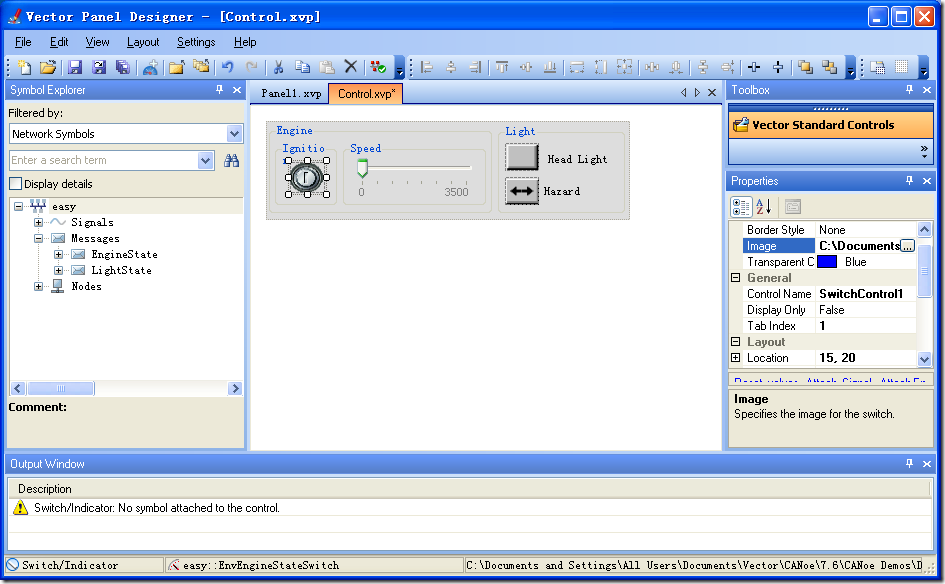
看完程序大家可能有个疑问，没有调用任何发送CAN消息的函数（只是更改其中的信号），但报文却真的发出去了，这是为什么呢？

这是因为周期发送消息的工作，已经在消息的属性中定义了，这样消息会自动周期的发送。如下：

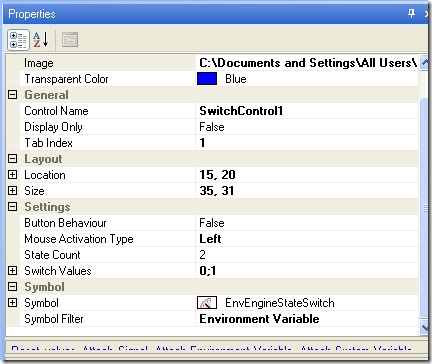
[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/20120926193704455.png)

这个在消息的属性查看中的界面，当然也可以在上面我们介绍的View->Attribute Definitions，进行修改和查看，但区别是，这个只是针对个别消息的，View->Attribute Definitions，是针对所有的情况。还有消息属性中，对此进行归类，以上归类到Interaction Layer这个是CAN通讯的交互层。上面的各个属性的具体含义，请参考帮助文档，都有详细的说明。

下面说一下界面。

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261937202139.png)

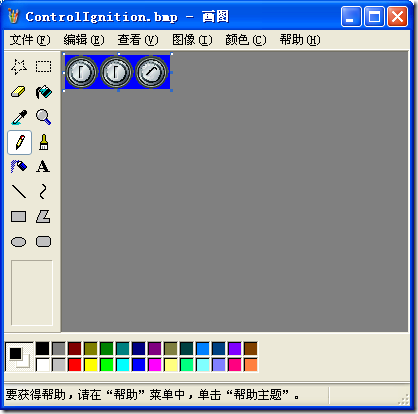
选中一个界面组件，在状态栏中可显示他的类型，关联的对象等信息。右边为属性窗口，定义选中组件的属性

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261937237453.png)

这个组件类型为：Switch/Indicator

属性栏中：

Image 表示该组件使用的图片，因为要表示几种状态，所以做成这样，尺寸105x34 pix

[](http://images.cnblogs.com/cnblogs_com/dongdonghuihui/201209/201209261937257127.png)

State Count 表示状态的个数

其他的属性不一一介绍了，自己试一下基本可以知道，实在不行求助帮助文档，这里不一一介绍了。

到现在整个工程的剖析基本上结束了，但说过的这些不足以覆盖所有的细节，但基本脉络已经很清晰了，剩下的可以自己研究，都不难理解。个人建议，在实际工作中创建自己的工程，当遇到问题是，参考例子中的实现方式，这样更加帮助理解。进步也最快。