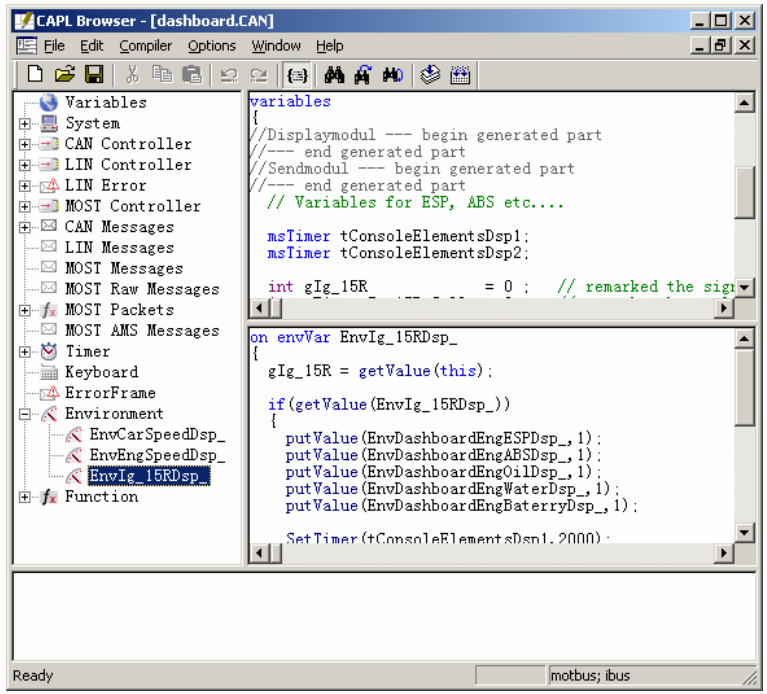
Vector 产品培训

CAPL编程

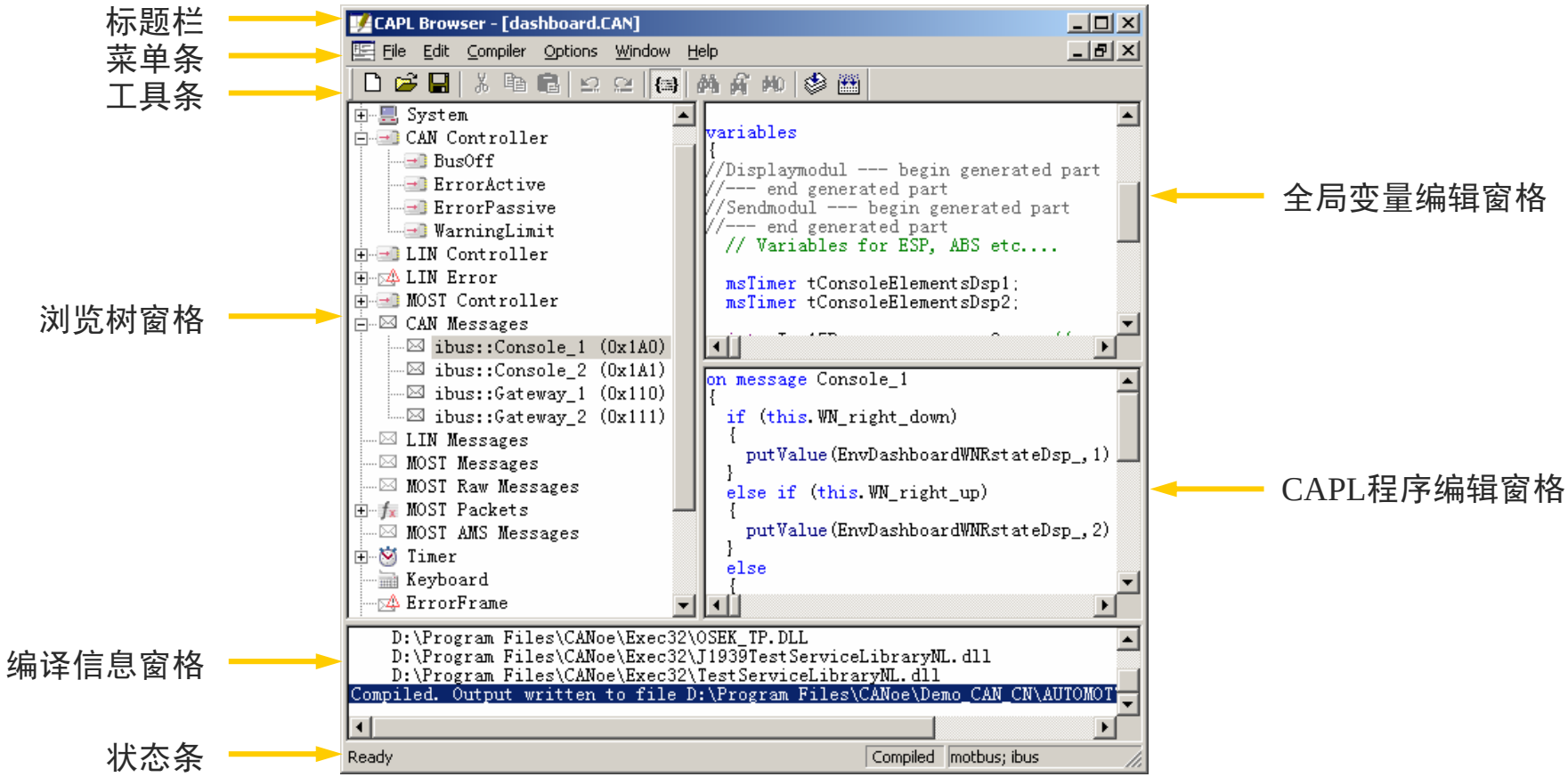
CAPL浏览器——概述

- 创建、修改、编译CAPL程序
- 以结构形式显示变量、事件程序和函数
- CAPL浏览器可同时打开多个CAPL程序窗口
- 快速编译
- 错误自动定位



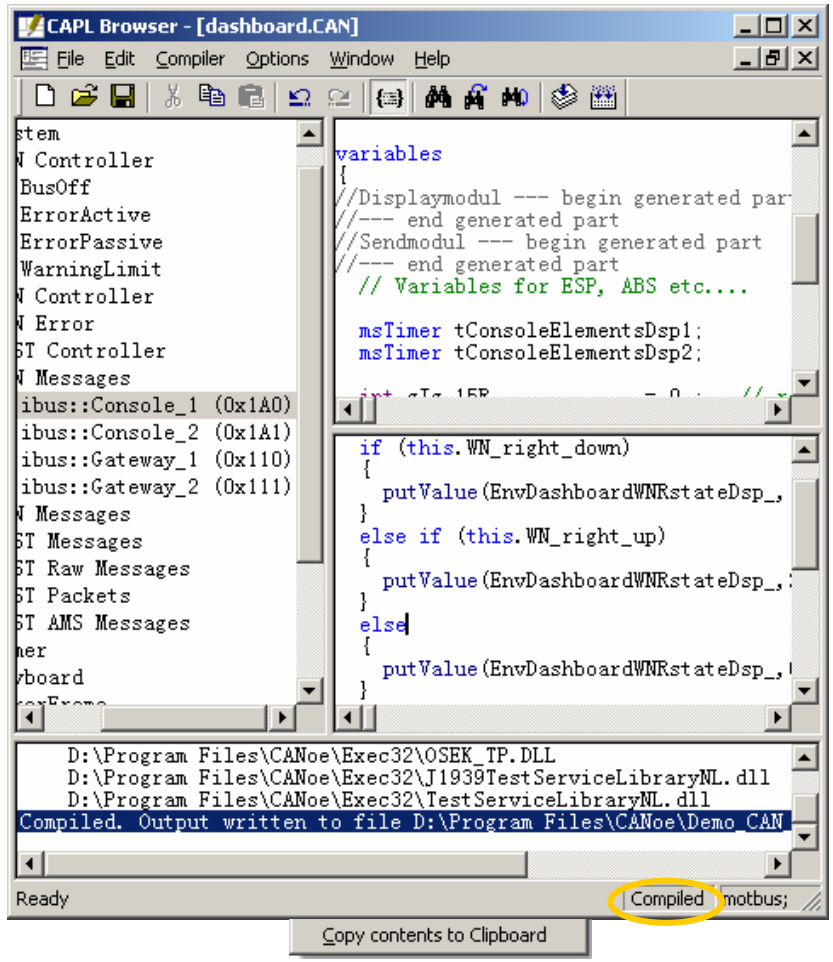
通过CANoe工具条按钮 打开CAPL浏览器。鼠标双击CAPL节点也可打开。

CAPL浏览器的窗口


标题栏
菜单条
工具条
浏览树窗格
编译信息窗格
状态条
全局变量编辑窗格
CAPL程序编辑窗格
CAPL Browser - [dashboard.CAN]
File Edit Compiler Options Window Help
System
CAN Controller
BusOff
ErrorActive
ErrorPassive
WarningLimit
LIN Controller
LIN Error
MOST Controller
CAN Messages
ibus::Console_1 (0x1A0)
ibus::Console_2 (0x1A1)
ibus::Gateway_1 (0x110)
ibus::Gateway_2 (0x111)
LIN Messages
MOST Messages
MOST Raw Messages
MOST Packets
MOST AMS Messages
Timer
Keyboard
ErrorFrame
variables
{
//Displaymodul --- begin generated part
//--- end generated part
//Sendmodul --- begin generated part
//--- end generated part
// Variables for ESP, ABS etc....
msTimer tConsoleElementsDsp1;
msTimer tConsoleElementsDsp2;
on message Console_1
{
if (this.WN_right_down)
{
putValue(EnvDashboardWNRstateDsp_,1)
}
else if (this.WN_right_up)
{
putValue(EnvDashboardWNRstateDsp_,2)
}
else
{
D:\Program Files\CANoe\Exec32\OSEK_TP.DLL
D:\Program Files\CANoe\Exec32\J1939TestServiceLibraryNL.dll
D:\Program Files\CANoe\Exec32\TestServiceLibraryNL.dll
Compiled. Output written to file D:\Program Files\CANoe\Demo_CAN_CN\AUTOMOT
Ready
Compiled
motbus; ibus

CAPL程序的编译

- CAPL程序必须通过编译才可执行
- CAPL的可执行文件扩展名为*.cbf
- 编译可通过菜单命令、工具条按钮、或快捷键来激活
- 信息窗格显示编译信息
- 通过错误信息定位错误



搜索运行时错误

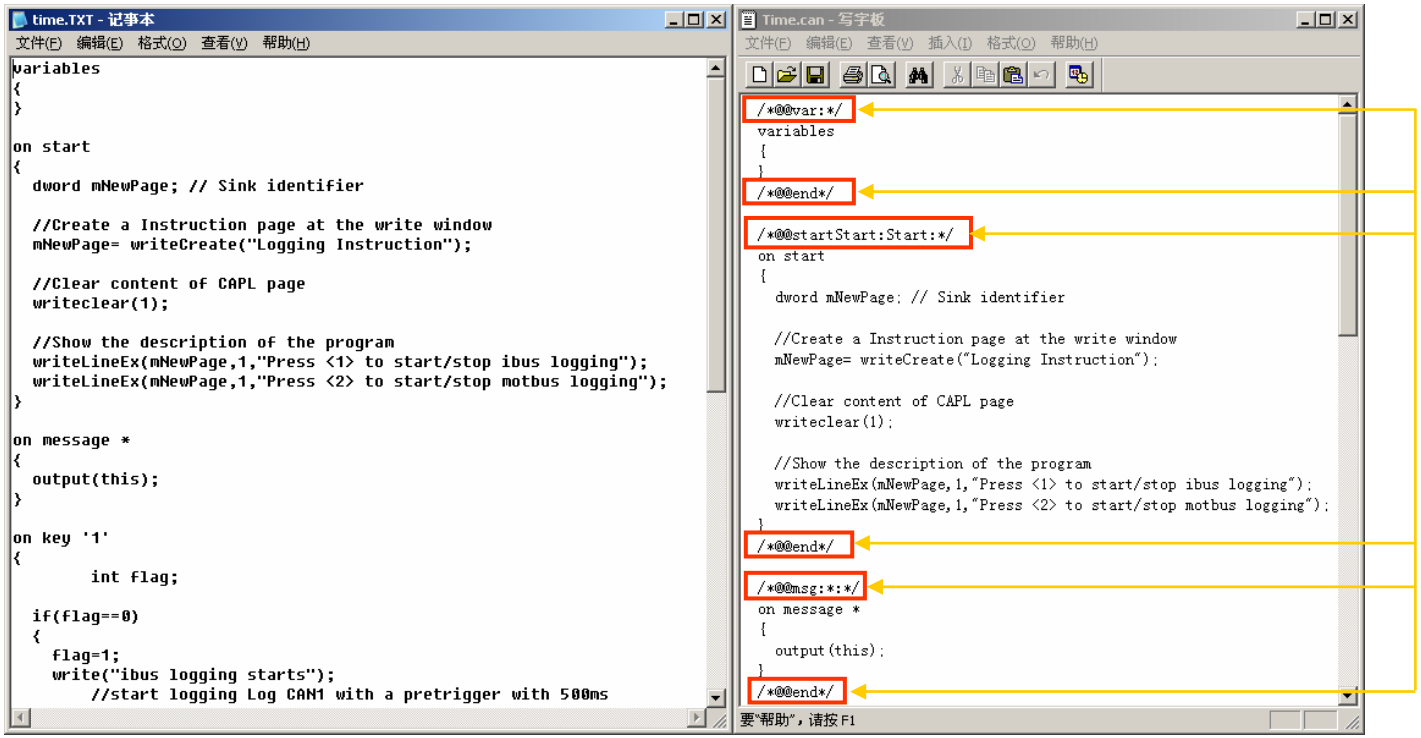
- 测量过程中自动检测CAPL程序：
  - 被0除
  - 超出数组的上限或下限
  - 超出消息数据域的上偏移或下偏移
  - 在CAPL子程序调用时堆栈溢出
- 使用浏览器菜单命令**Compiler｜Find run-time errors**通过错误索引号查找
- 使用**runError()**函数自定义运行时错误

数据库的访问

- 直接从CANoe的测量设置或仿真设置窗口打开CAPL浏览器，自动关联数据库
- 通过浏览器菜单命令关联数据库
- 在浏览器的编辑窗格中，通过交互菜单选项插入信号或消息名到CAPL程序中

导入/导出ASCII文件

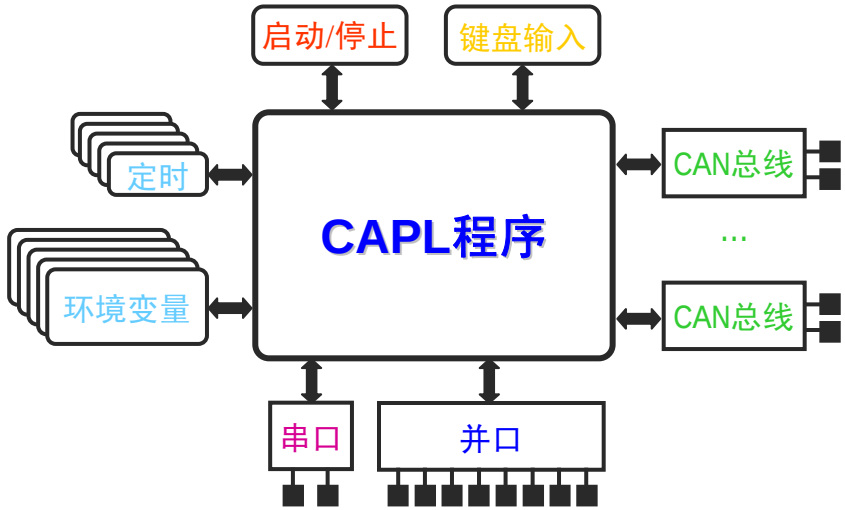
- **File｜Import...**导入纯ASCII文件到CAPL浏览器中
- **File｜Export...**导出CAPL程序为ASCII文件



导出的 ASCII格式的CAPL程序　　**浏览器格式的CAPL程序**

CAPL编程概述

- CAPL是CAN总线访问编程语言（ CAN Access Programming Language ）
- 类C语言
- 应用于Vector CAN工具节点编程
- 基于事件建模的语言
  - 总线事件
  - 属性事件
  - 时间事件



CAPL程序的应用

- 节点仿真
- 网络仿真
- 仿真控制系统的环境
- 节点测试
- 网关

CAPL程序对于事件的响应

- CAPL程序能够检测事件，并执行和事件相关的程序。检测的事件类型包括：
  - 程序开始执行事件
  - 程序停止执行事件
  - 键盘输入事件
  - CAN消息的接收事件
  - 定时器超时事件
  - 图形面板输入事件（该项只在CANoe中应用）
- CAPL程序是基于事件程序的组合

CAPL的事件类型

事件类型	事件名	程序执行条件	事件过程语法结构 *
系统事件	*PreStart*	CANoe初始化时执行	***on preStart*** { … }
	Start	测量开始时执行	***on start*** { … }
	StopMeasuremet	测量结束时执行	***on stopMeasurement*** { … }
CAN控制器事件	*BusOff*	硬件检测到BusOff时执行	***on busOff*** { … }
	ErrorActive	硬件检测到ErrorActive时执行	***on errorActive*** { … }
	ErrorPassive	硬件检测到ErrorPassive时执行	***on errorPassive*** { … }
	WarningLimit	硬件检测到WarningLimit时执行	***on warningLimit*** { … }
CAN消息事件	*自定义*	接收到指定的消息时执行	***on message*** *Message* { … }
时间事件	*自定义*	定时时间朝过时执行	***on timer*** *Timer* { … }
键盘事件	*自定义键值*	指定的键被下时执行	***on key*** *Key* { … }
错误帧事件	*ErrorFrame*	硬件每次检测到错误帧时执行	***on errorFrame*** { … }
环境变量事件	*自定义*	指定的环境变量值改变时执行	***on envVar*** *EnvVar* { … }

* “事件过程语法结构”列中兰色字体表示该程序的关键字；深红色字体表示用户自定义的名称；“***{ … }***”内是CAPL程序体，用户可根据需要使用CAPL语言编写。

CAPL 基本语法

- CAPL语言的语法与C语言基本相同：
- 注释
  - //　　放置在需要注释的语句之前，注释单行
  - /*　　注释起始符，其后的内容被注释
  - */　　注释结束符，结束由‘/*’开始的注释
- 事件过程
- 分号　　语句结束符
- 大括号

消息过程

- on message 123 //对消息123(dec)反应
- on message 0x123 //对消息123(hex)反应
- on message MotorData //对消息MotorData(符号
  //名字)反应
- on message CAN1.123 /*对CAN通道1收到消息
  123反应*/
- on message * //对所有消息反应
- on message 100-200 //对100-200间消息反应

键盘过程

- on key ‘a’ //按‘a’键反应
- on key ‘ ’ //按空格键反应
- on key 0x20 //按空格键反应
- on key F1 //按F1键反应
- on key Ctrl-F12 //按Ctrl + F12键反应
- on key PageUP //按PageUp键反应
- on key Home //按Home键反应
- on key * //按所有键反应

时间过程

- 时间过程表示法：
  - on timer myTimer　　//对myTimer 设定的时间到反应
- 定时器的申明
  - msTimer myTimer;　　//将myTimer 申明ms为单位的变量
  - timer　　myTimer;　　//将myTimer 申明s为单位的变量
- 定时器的设置
  - setTimer(myTimer,20);　//将定时值设定为20ms，并启动
  - cancelTimer(myTimer);　//停止定时器myTimer

每次使用setTimer的设置，只能触发一次时间过程

环境变量过程

- 环境变量过程on envVar 对环境变量值的改变产生反应
- 测量设置中的CAPL节点不会阻止环境变量在数据流图中的传输
- 环境变量过程常用的函数：
  - getValue()　　//获取环境变量的值
  - putValue()　　//设置环境变量的值
- 可使用this在过程内部访问环境变量的值

CAPL中的数据类型

- 无符号整数
  - byte （1字节）
  - word （2字节）
  - dword （4字节）
- 有符号整数
  - int （2字节）
  - long （4字节）
- 浮点数
  - float （8字节）
  - double （8字节）
- CAN消息类型
  - message
- 定时器类型
  - timer （秒为单位）
  - msTimer（毫秒为单位）
- 单个字符
  - char （1字节）

消息的申明

- 消息申明的格式
  - Message 0xA *my_msg1*;
  - Message 100 *my_msg2*;
  - Message EngineData *my_msg3*;
- 消息数据的索引
  - *my_msg1.byte(0)* //数据字节0
  - *my_msg2.word(2)* //从第2字节开始的一个字
  - *my_msg3.EngSpeed* /*如果使用了符号数据库，可使用信号符号名来索引消息中的信号*/

CAPL程序的组成

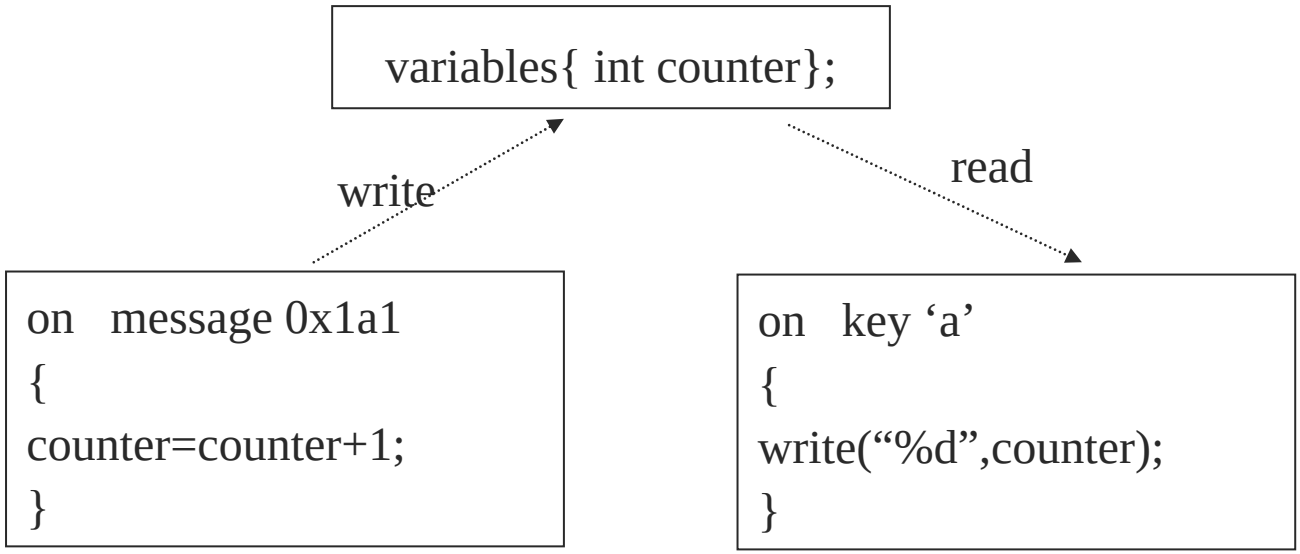
- 一个完整的CAPL程序由三个部分组成：
  - 申明与定义全局变量
  - 各种事件过程
  - 申明与定义自己的函数

```
variables
{
   …          //申明全局变量
}
```

```
on start
{
   …          //过程指令块
}

on message xxx
{
   …          //过程指令块
}

on key ‘1’
{
   …          //过程指令块
}
…
```

```
My_function_1(Para_1, Para_2, …)
{
   …          //函数体
}

…

My_function_n(Para_1, Para_2, …)
{
   …          //函数体
}
```

CAPL 程序执行顺序



- 事件过程之间无关联，执行顺序由运行时间事件决定
- 事件过程通过全局变量和子程序决定
- 事件过程为一整体，不能被其它事件中断

针对消息的一些常用语句

- 常进行读写

```
if (this.id==100) {...}          //消息ID
```

- 常写的

```
msg.can=2;                       //消息所使用的CAN控制器编号
msg.dlc=8;                       //消息中包含的数据字节长度
```

- 常读的

```
dword t ; t=this.time;           //消息的时标，单位是10us
if(this.dir!=RX) {return;}       //消息的收发特性
```

注意： this 是关键字，在事件过程中代表所定义的触发事件名

关键字 —— this

- 在事件过程中，关键字this指定事件对象的数据结构

```
on message 100 {
        byte byte_0;
        byte_0 = this.byte(0);
        ...
}
```

```
on envVar Switch {
        int val;
        val = getvalue(this);
        ...
}
```

- this可作为参数使用
- 对于this值的改变仅在过程内部有效

CAPL 指令块

```
Counter = counter+1;
if (counter==256)
 {
     counter=0;
    stop();
 }
```

CAPL 中输出文本

```
int h=100;
char ch='a';
char s100[8]="hundred";
write("Hundred as a  number:%d,%x",h,h);
write("Hundred as a  string:%s",s100);
write("The square root of two is %6.4g",sqrt(2.0));
```

处理信号

```
on message 0x64
{
    if(this.byte(2)==0xFF)
        write("Third byte of the message is invalid");
}
on message MotorData
{
    if(this.temperature.phys>=150)
        write("Warning: critical temperature");
}
```

传输信号

```
on key ‘a’ {
    message MotorData mMoDa;
    mMoDa.temperature.phys=60;
    mMoDa.speed.phys=4300;
    output(mMoDa);
}
on key ‘b’ {
    message 100 m100= {dlc=1};
    m100.byte(0)=0x0B;
    output(m100);
}
```

周期性消息发送的CAPL示例

```
Variables                          // 定义全局变量
{
    message 0x555 msg1 = {dlc=1}; // 定义消息变量 msg1，并初始化数据字节代码为1
    msTimer timer1;                // 定义定时器变量 timer1
}

on start                           // 系统过程
{
    setTimer(timer1,100);          // 初始化定时器变量timer1的值为 100 msec，并启动
}

on timer timer1                    // 时间过程（对于定时器变量timer1 ）
{
    setTimer(timer1,100);          // 重新设置timer1 ，并启动
    msg1.byte(0)=msg1.byte(0)+1; // 改变消息数据字节
    output(msg1);                  // 输出消息
}
```

环境变量过程的示例

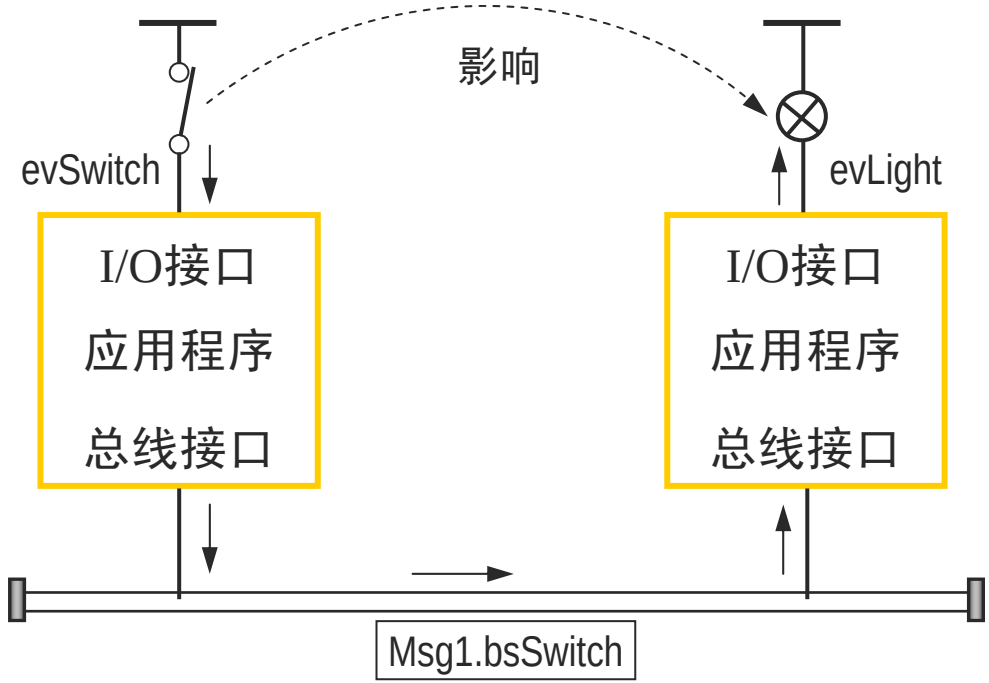
```
Variables
{
}

// Reaction to change of environment var. evSwitch
on envVar evSwitch
{
    // Declare a CAN message to be transmitted
    message Msg1 msg;
    // Read out the value of the light switch, Assign to the bus signal bsSwitch
    msg.bsSwitch = getValue(this);
    // Output message on bus (spontaneous transmission)
    output(msg);
}
```

小结

- 本章主要介绍了三个独立应用的编辑工具：
  - CANdb++编辑器
    - 按照创建CAN数据库的步骤介绍了CANdb++编辑器的使用方法，从而学习其最主要的一些功能
  - 面板编辑器
    - 主要介绍了编辑器的启动、控件的布置与配置、位图元件的处理、以及面板的测试与控制
  - CAPL浏览器及CAPL编程基础
    - 介绍了CAPL浏览器的窗口、编译及编译错误处理、数据库访问、以及CAPL程序ASCII文件的导入/导出
    - 同时还介绍CAPL语言的特点与应用、CAPL事件过程、数据类型、程序结构、以及实例演示了常用函数的使用

练习1：

- 创建一个完整的CANoe配置：
  - 两个节点
  - 与外部交互



创建分布式系统模型的步骤

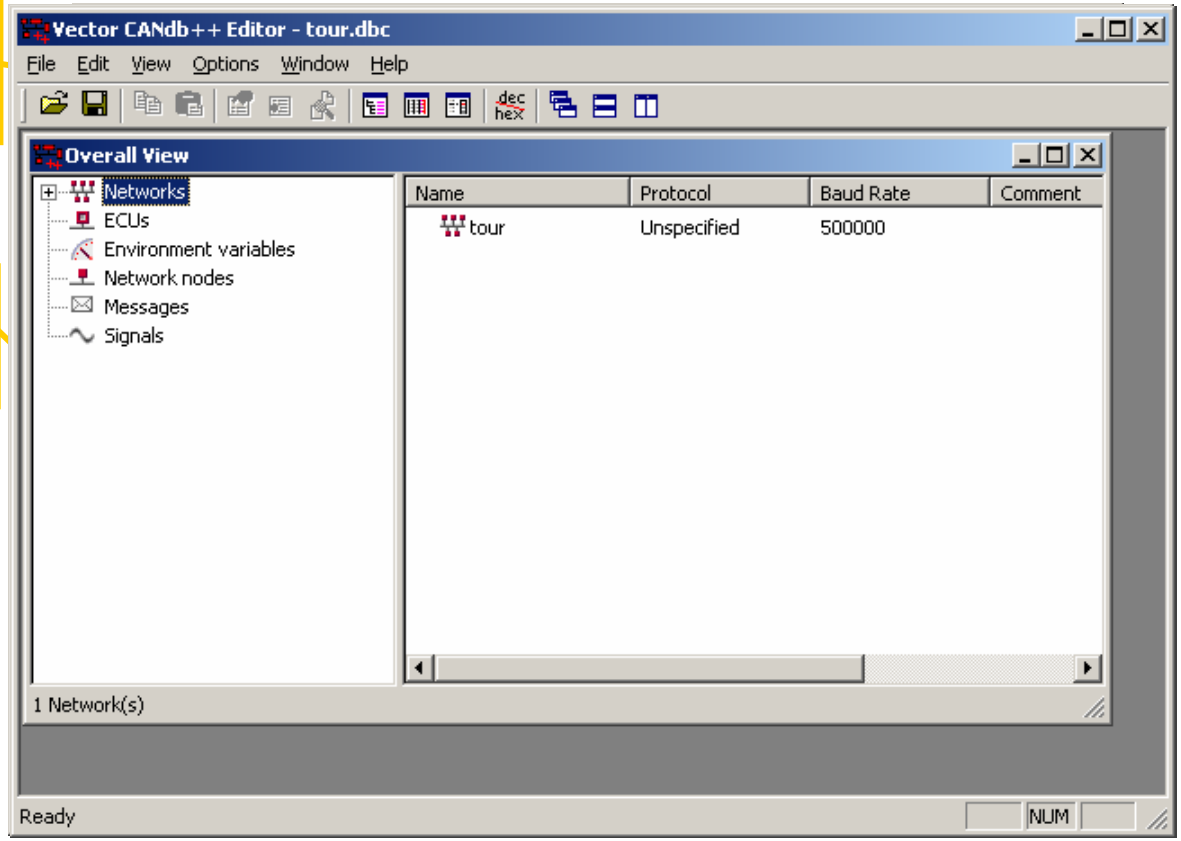
- 在CANoe中创建一个分布式系统模型，分为三个步骤：
  - 创建一个具有消息、信号和环境变量的数据库
  - 创建网络节点的外部界面。比如，控制面板
  - 使用CAPL程序创建网络节点模型

创建数据库

- 打开CANdb++编辑器

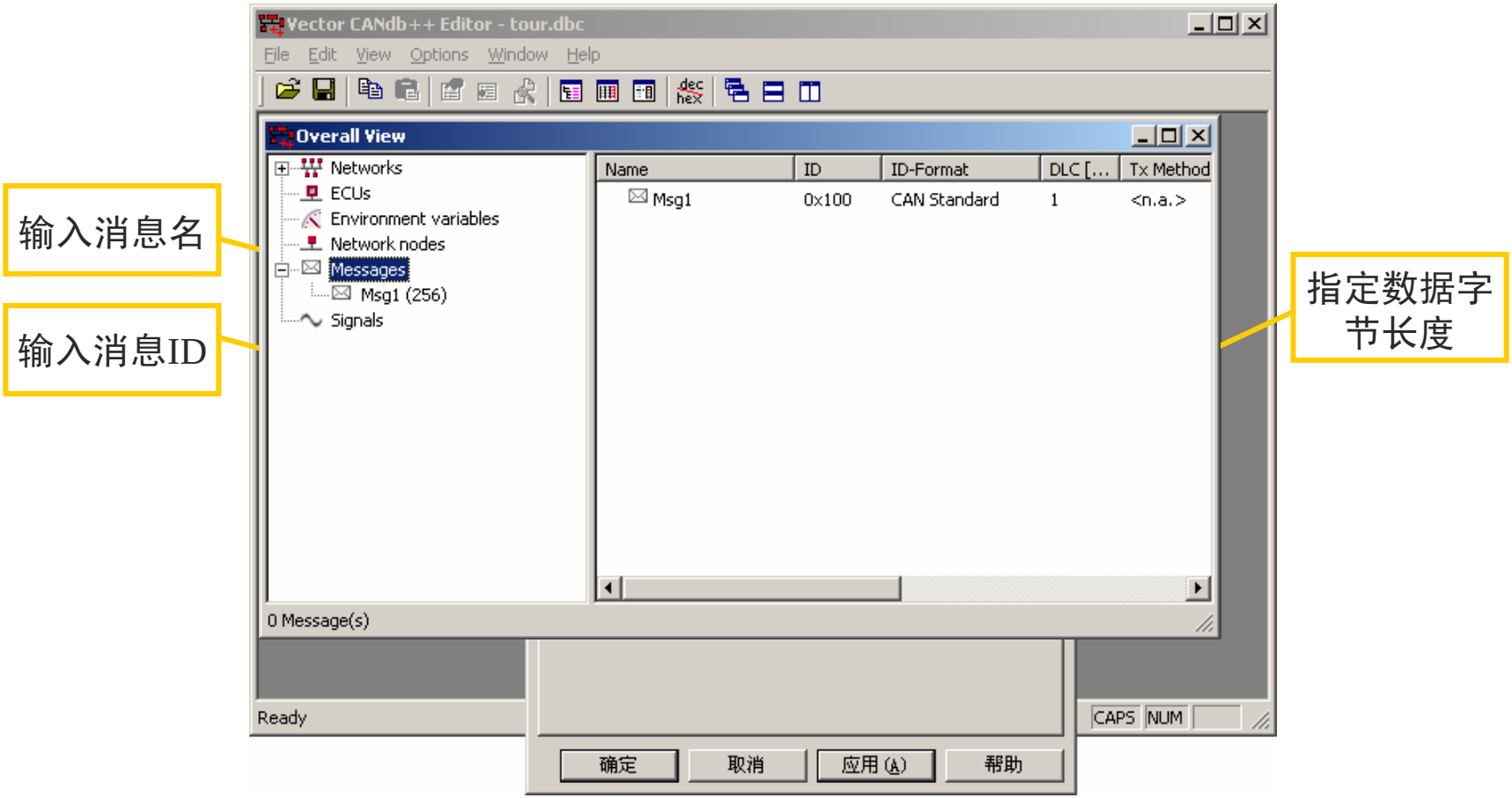
点击按钮
打开编辑器

用菜单命令
打开编辑器



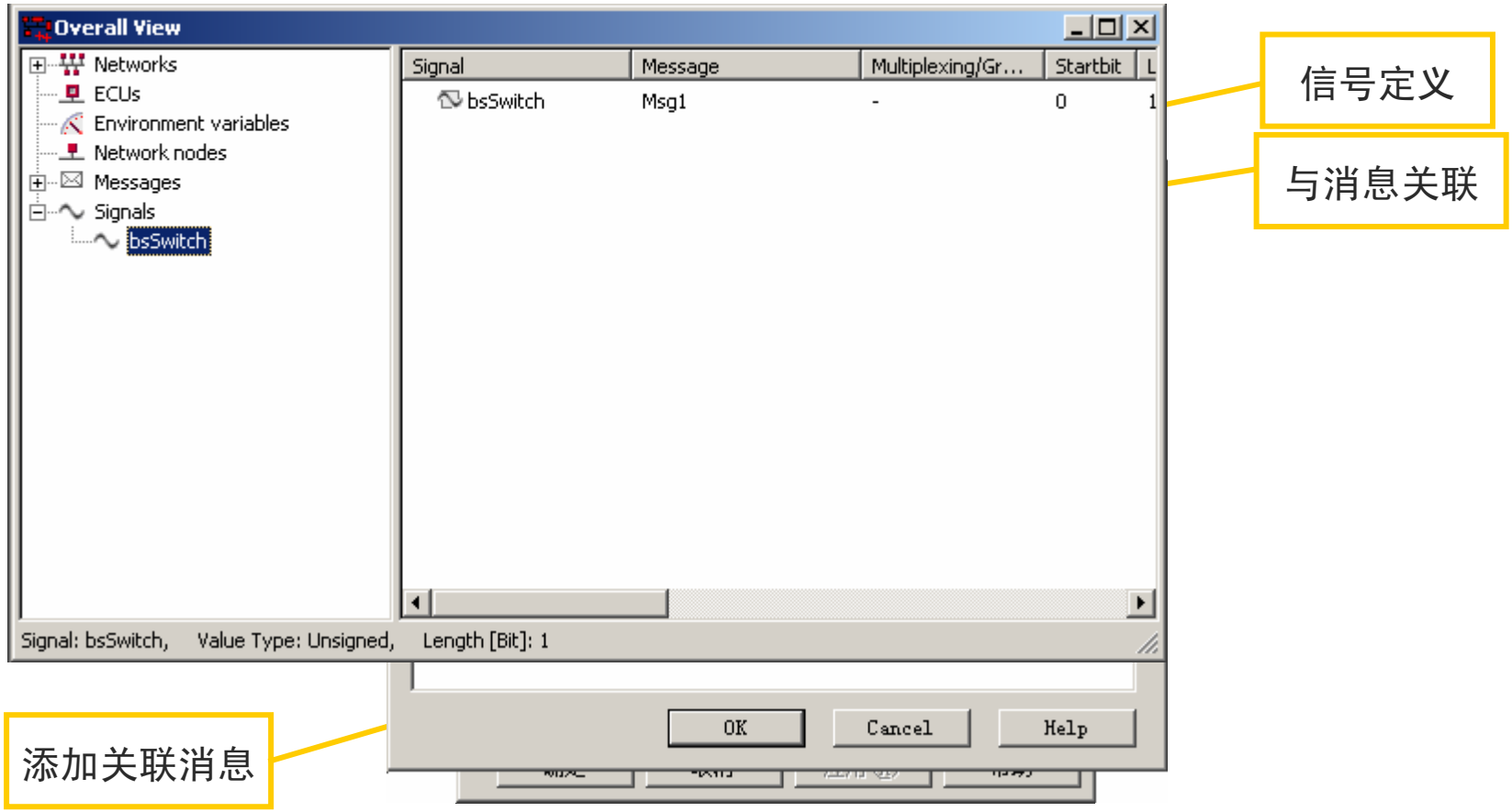
创建数据库（续）

- 创建消息Msg1



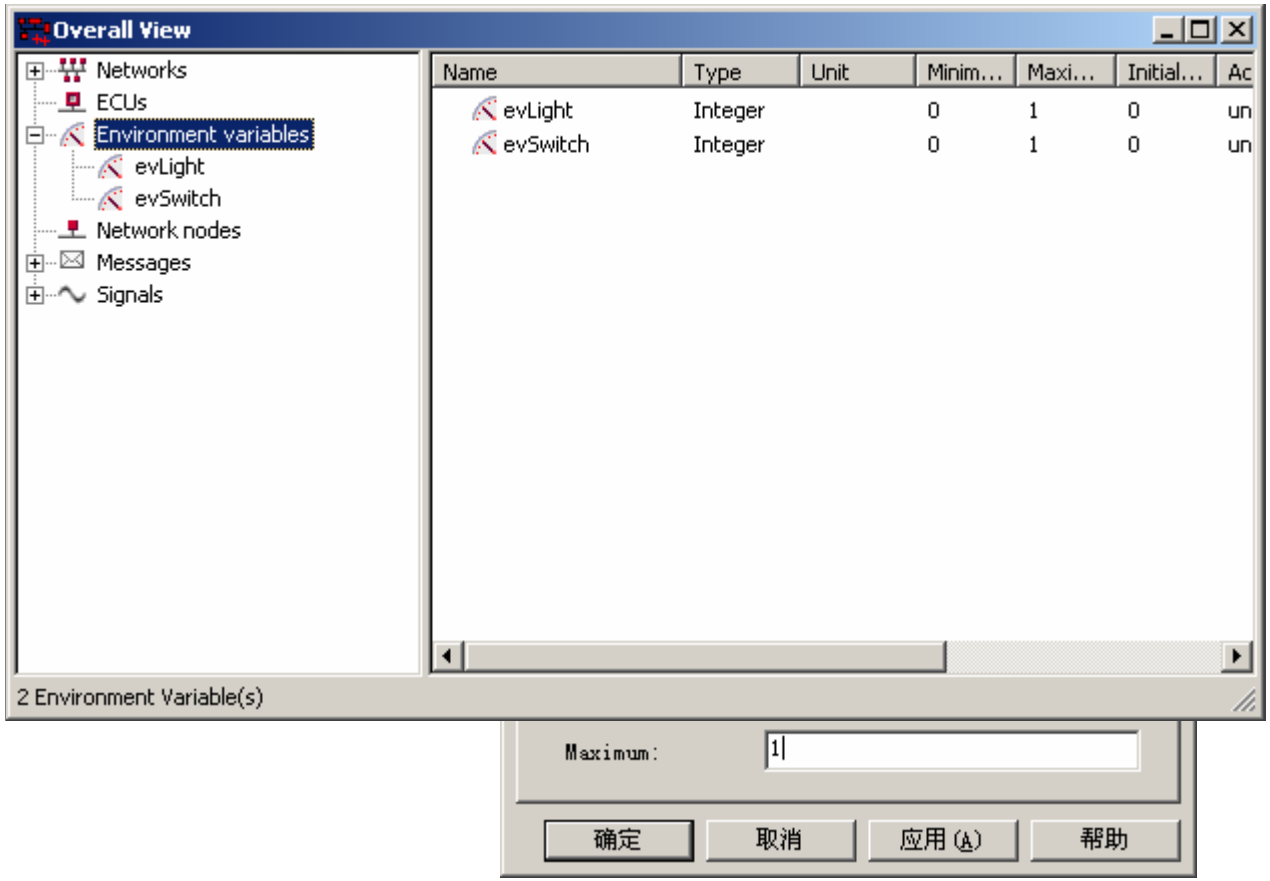
创建数据库（续）

- 创建信号bsSwitch（开关状态）



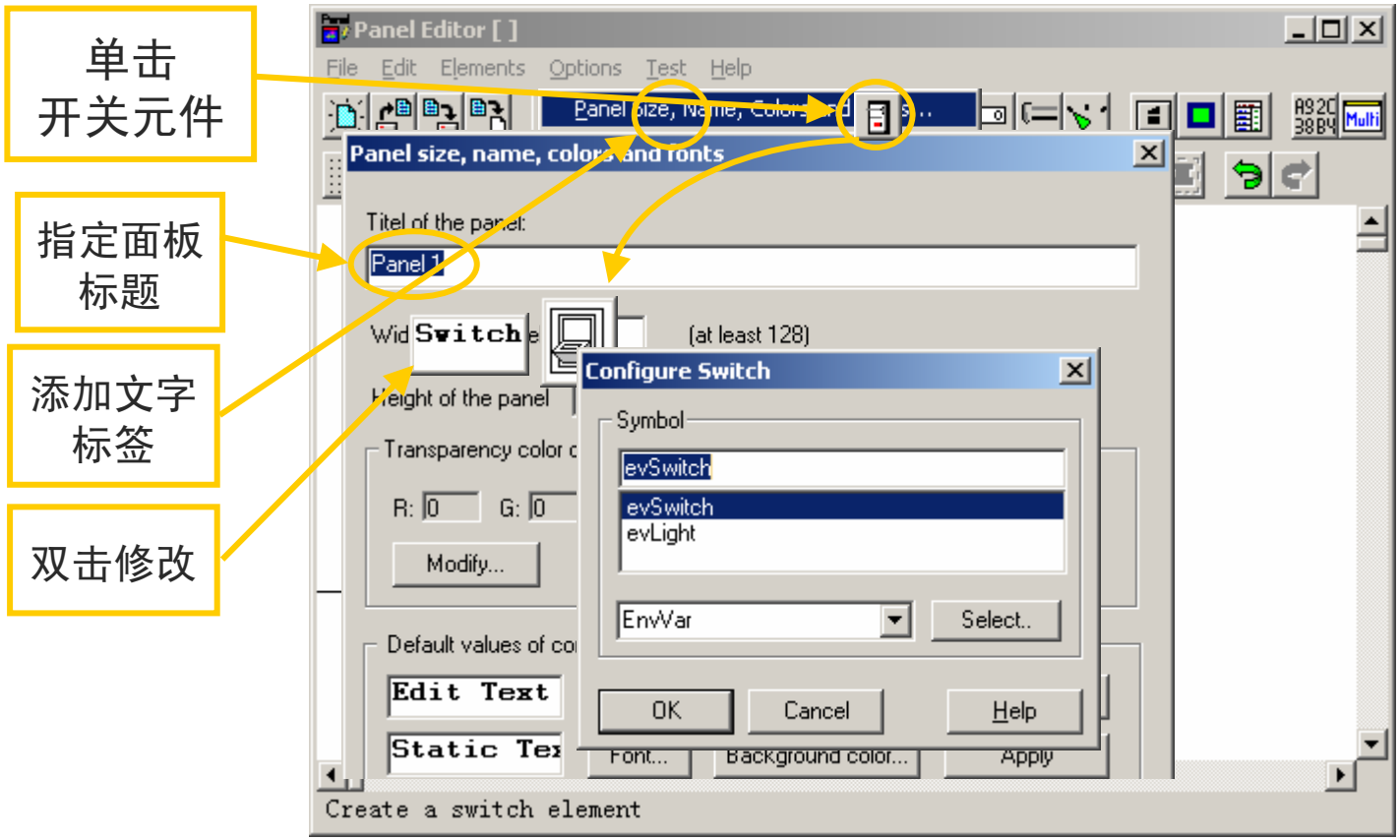
创建数据库（续）

- 创建环境变量 *evLight* 与 *evSwitch*



创建面板

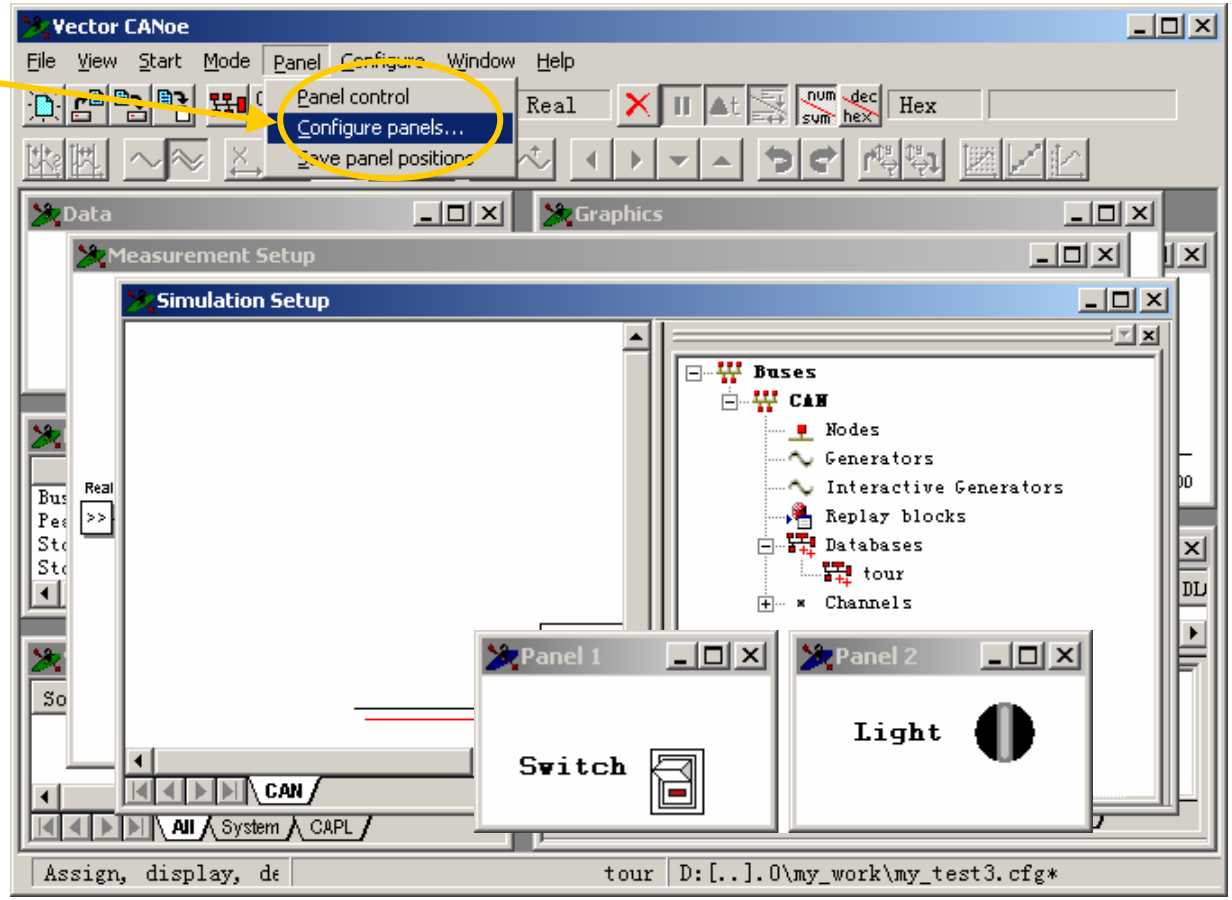
- 创建面板作为节点的外部设备



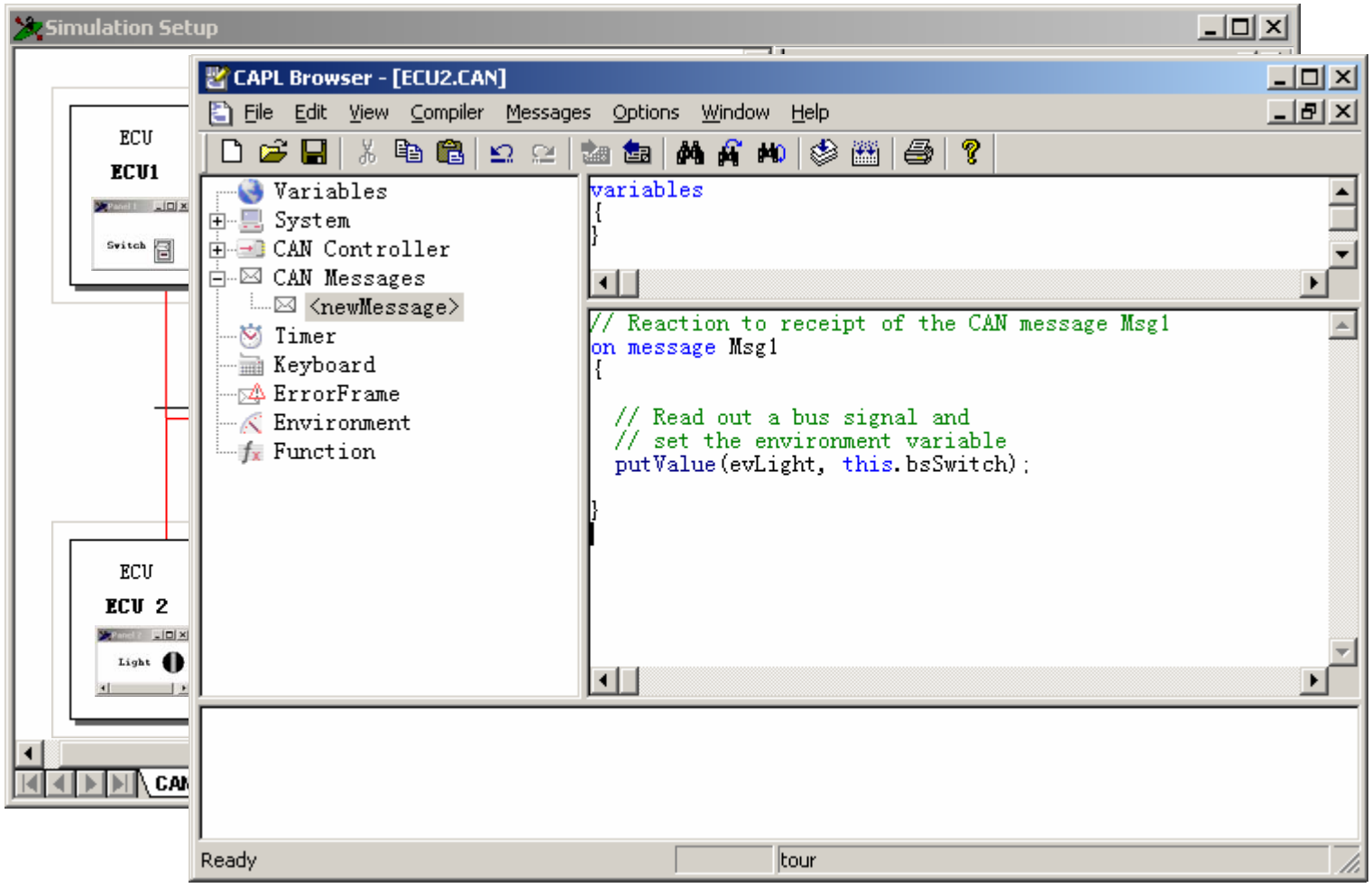
创建面板（续）

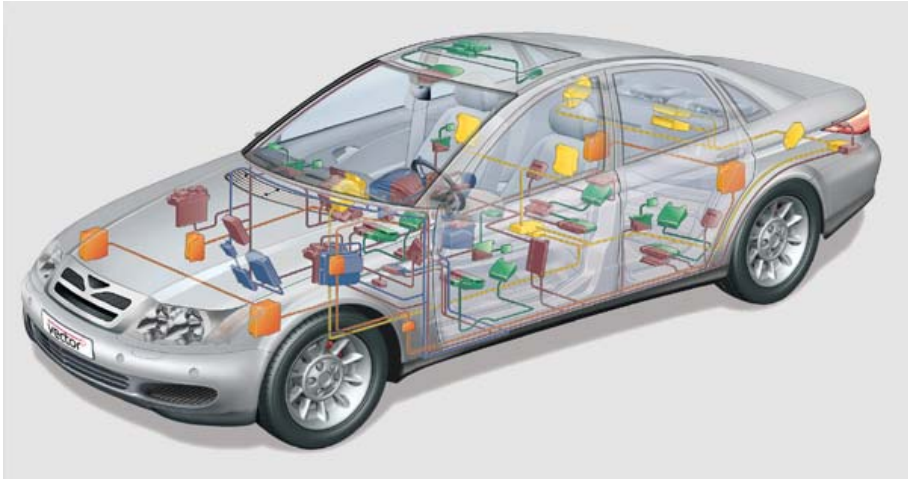
- 将面板文件集成到CANoe配置中

面板集成菜单命令



创建网络节点模型


Simulation Setup
ECU
ECU1
Switch
ECU
ECU 2
Light
CAPL Browser - [ECU2.CAN]
File Edit View Compiler Messages Options Window Help
Variables
System
CAN Controller
CAN Messages
<newMessage>
Timer
Keyboard
ErrorFrame
Environment
Function
variables
{
}
// Reaction to receipt of the CAN message Msg1
on message Msg1
{
// Read out a bus signal and
// set the environment variable
putValue(evLight, this.bsSwitch);
}
Ready
tour



CANoe快速建模CAPL Generator

恒润科技 HIRAIN TECHNOLOGIES

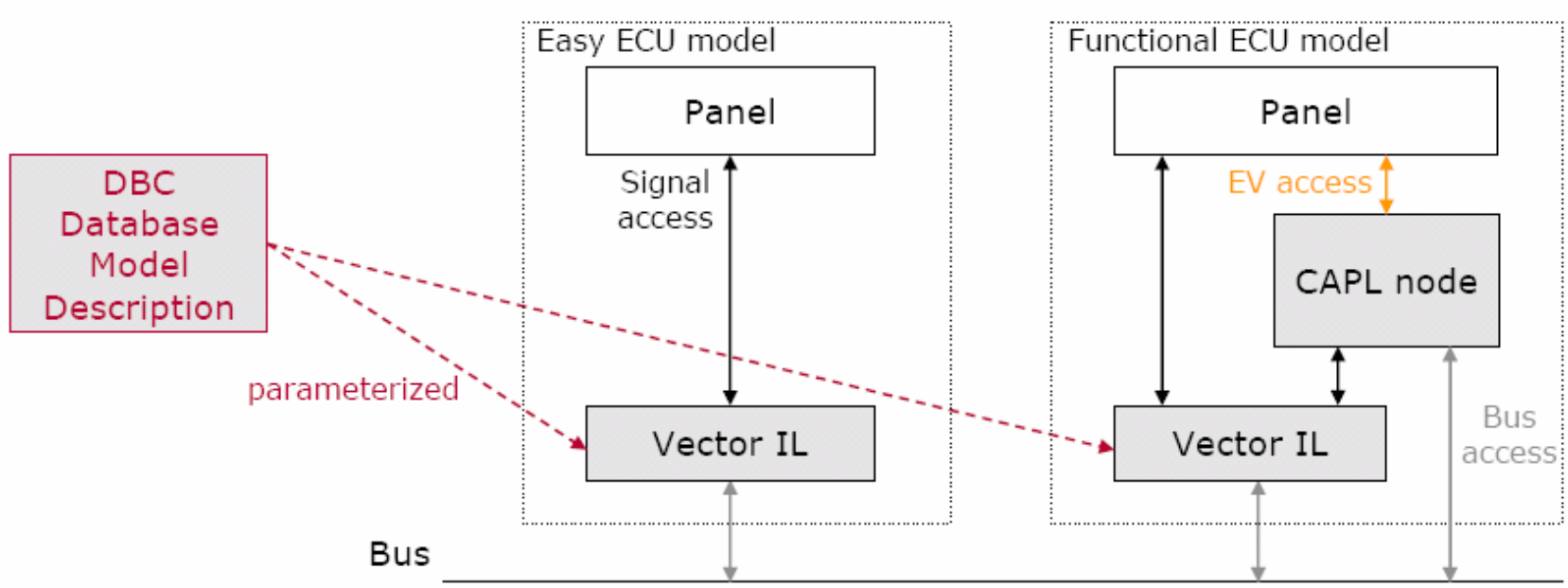
vector

目录

- 概述
- 数据库准备
- 生成CAPL模型
- 建立CANoe仿真
- 集成应用

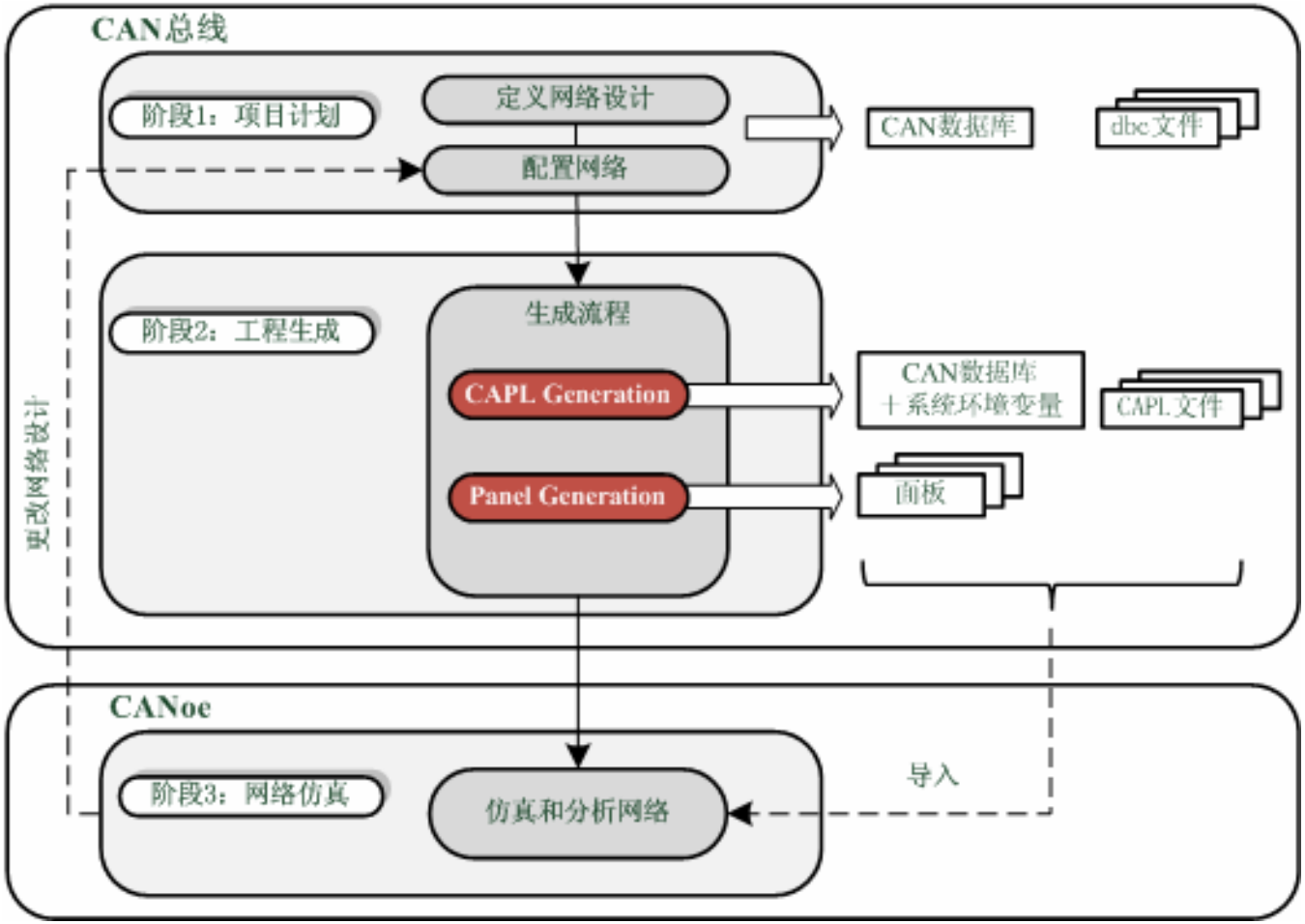
概述

- 两种CANoe模型



概述

- 阶段、流程、文件

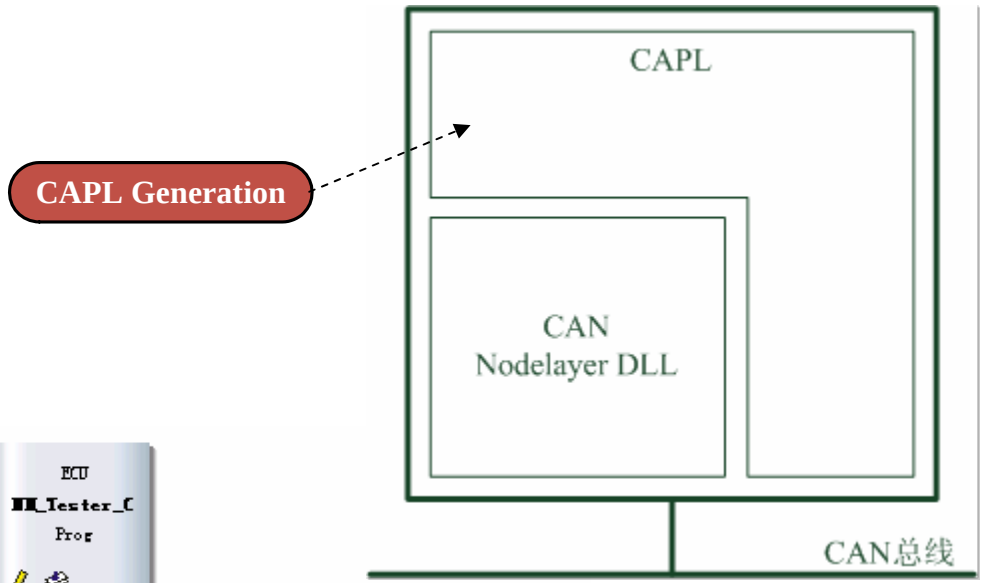


恒润科技 HIRAIN TECHNOLOGIES

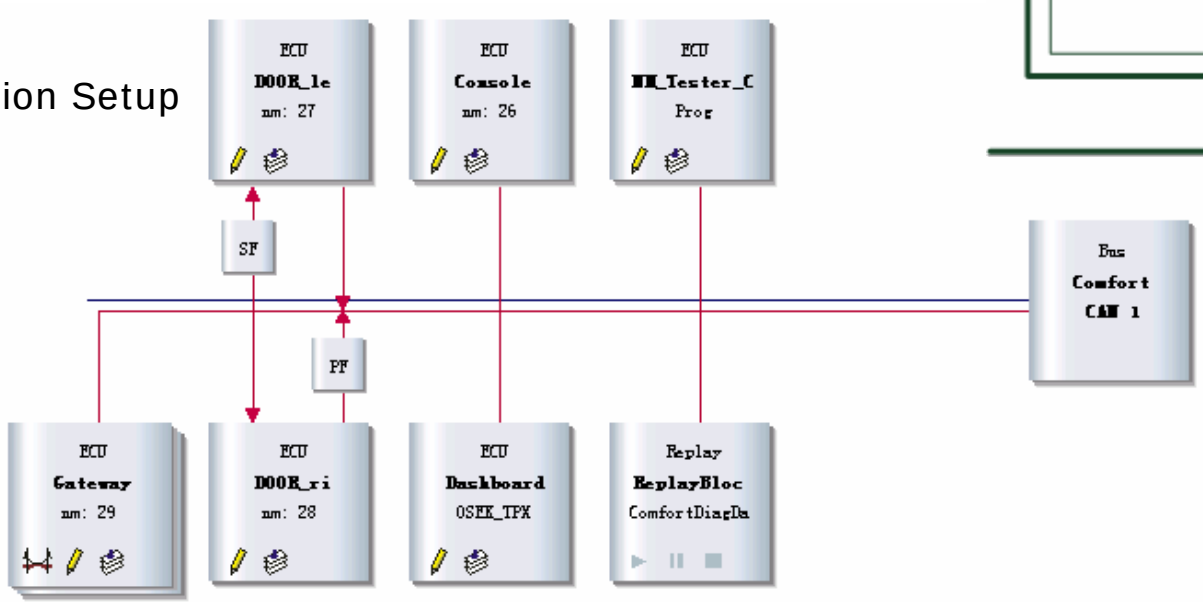
vector

概述

- 工程框架
  - 根据数据库生成
  - 与各个节点关联



CANoe
Simulation Setup

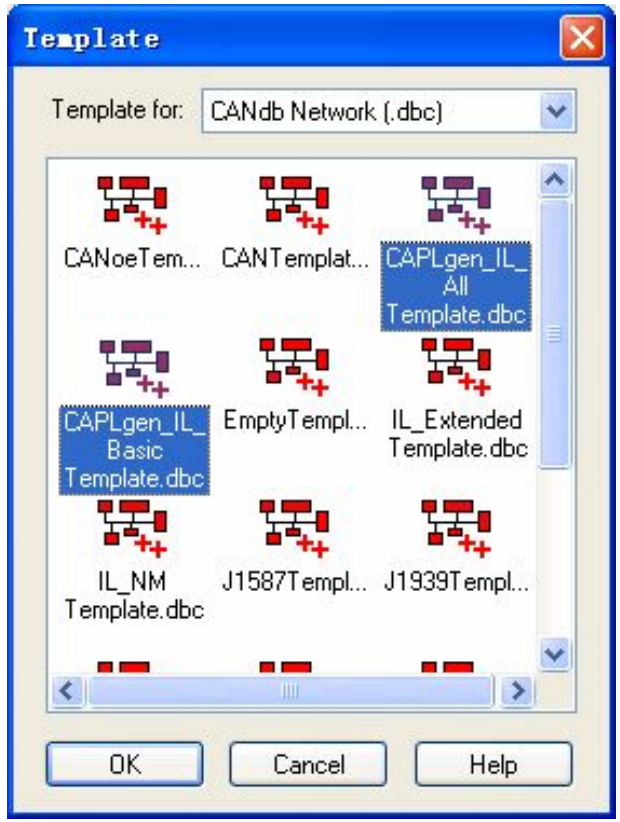


恒润科技 HIRAIN TECHNOLOGIES

vector

数据库准备

- 模板
  - CAPL Generator交互层
    - CAPLgen_IL_ALL_Template.dbc
      - 所有CAPL Generator交互层属性
    - CAPLgen_IL_Basic_Template.dbc
      - 发送报文的基本属性
  - Vector交互层
  - 扩展层

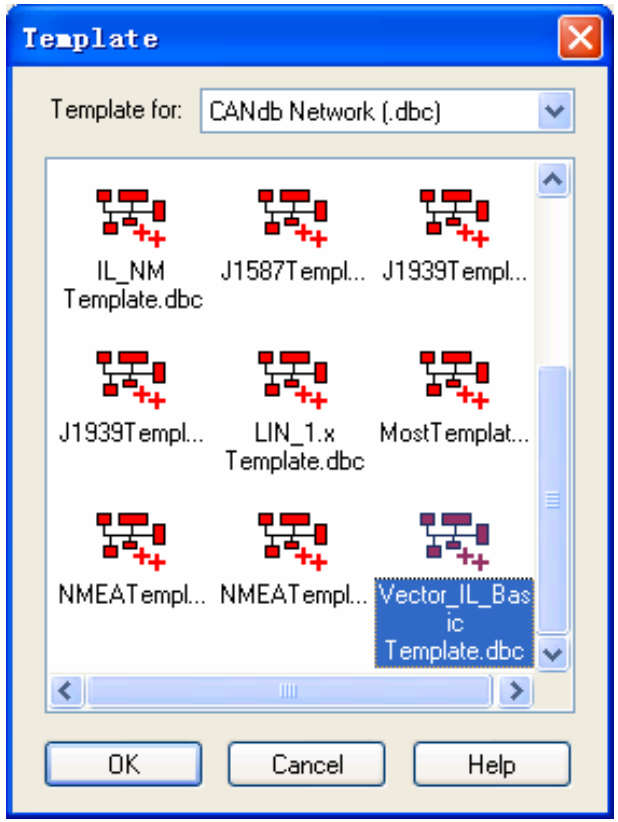


恒润科技 HIRAIN TECHNOLOGIES

vector

数据库准备

- 模板
  - CAPL Generator交互层
  - Vector交互层
    - Vector_IL_Basic_Template.dbc
      - 发送报文的基本属性
  - 扩展层

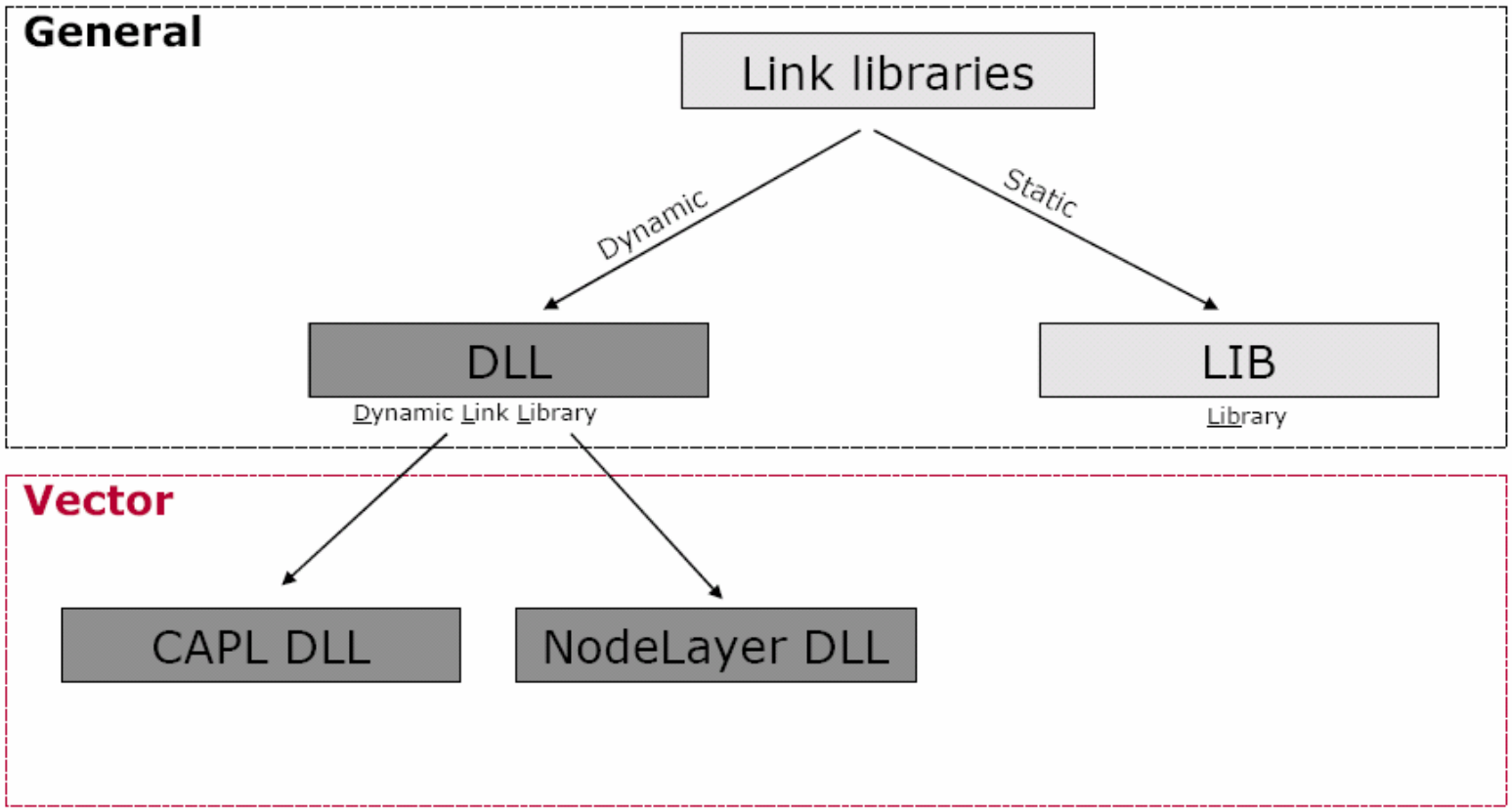


恒润科技 HIRAIN TECHNOLOGIES

vector

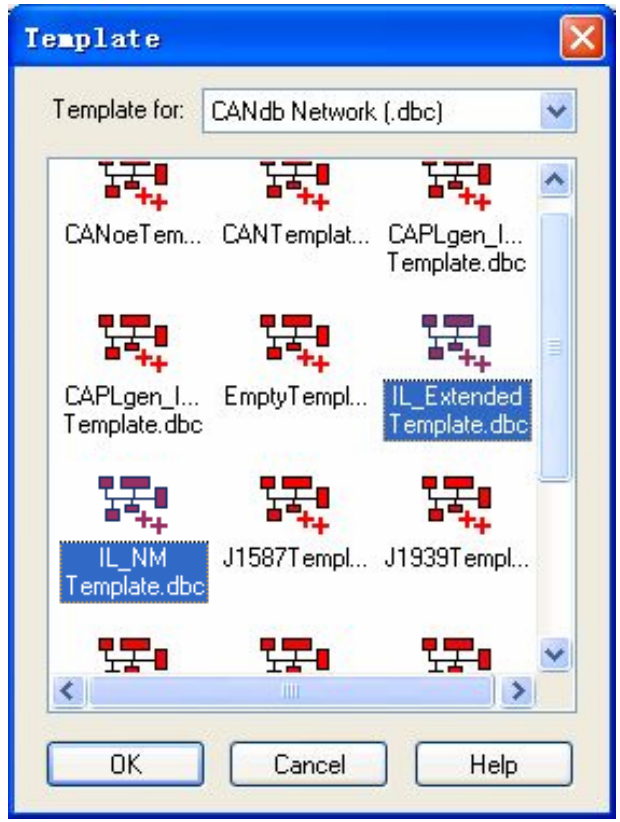
数据库准备

- 两个交互层接口区别



数据库准备

- 模板
  - CAPL Generator交互层
  - Vector交互层
  - 扩展层
    - IL_Extended_Template.dbc
      - 附加属性：条件发送和调用用户自定义函数
    - IL_NM_Template.dbc
      - 网络管理属性

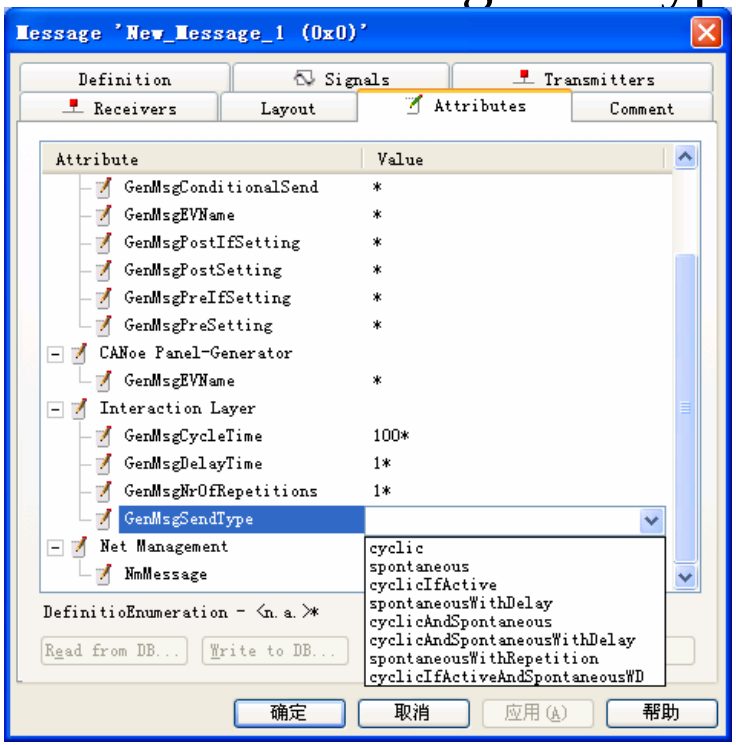


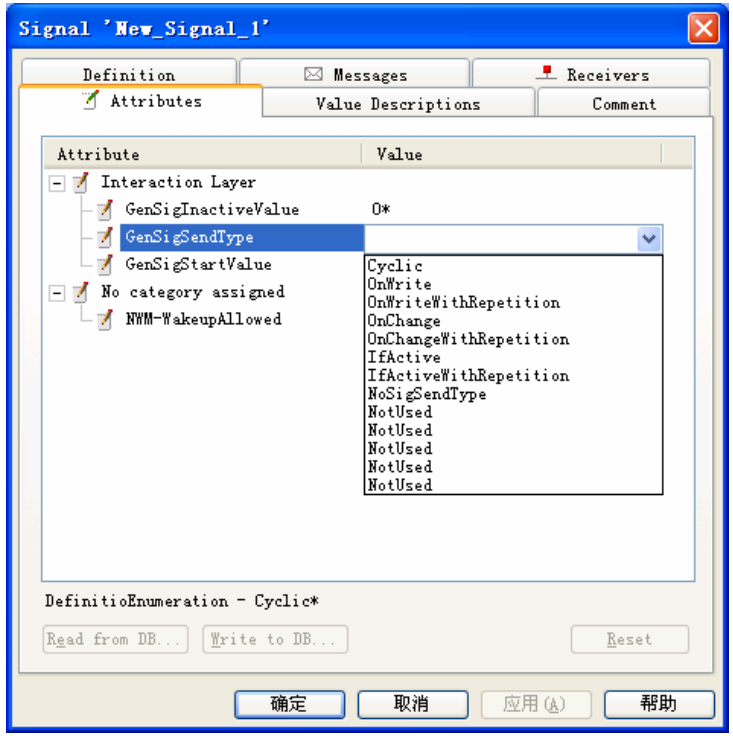
恒润科技 HIRAIN TECHNOLOGIES

vector

数据库准备

- 发送类型
  - CAPL Generator IL：GenMsgSendType
  - Vector IL：GenSigSendType





恒润科技 HIRAIN TECHNOLOGIES

vector

数据库准备

GenMsgSendType

报文发送类型	相关属性	行为描述
cyclic	GenMsgCycleTime	周期发送
spontaneous	-	信号值改变时发送
cyclickIfActive	GenMsgCycleTime; GenSigInactiveValue	信号激活发送
spontaneousWithDelay	GenMsgDelayTime	带延迟的值改变发送，指定该报文的最小发送间隔
cyclicAndSpontaneous	GenMsgCycleTiem	周期和值改变混合发送方式。要求GenMsgDelayTime=0
cyclicAndSpontaneousWD	GenMsgCycleTiem; GenMsgDelayTime	在周期点发送值改变信号。
spontaneousWithRepetition	GenMsgNrOfRepetitions; GenMsgCycleTime	信号值改变后，立即周期发送若干次数
cyclicIfActiveAndSpontaneousWD	GenMsgCycleTime; GenMsgDelayTime	值激活后周期发送，并在值改变时也发送，且报文满足最小延迟时间要求
cyclicIfActiveFast	GenMsgCycleTime; GenMsgCycleTimeActive	无激活信号时以基本周期发送，有信号激活时快速周期发送
cyclicWithRepeatOnDemand	GenMsgNrOfRepetition; GenMsgCycleTime	信号值改变后，在周期点发送若干次数

恒润科技 HIRAIN TECHNOLOGIES

vector

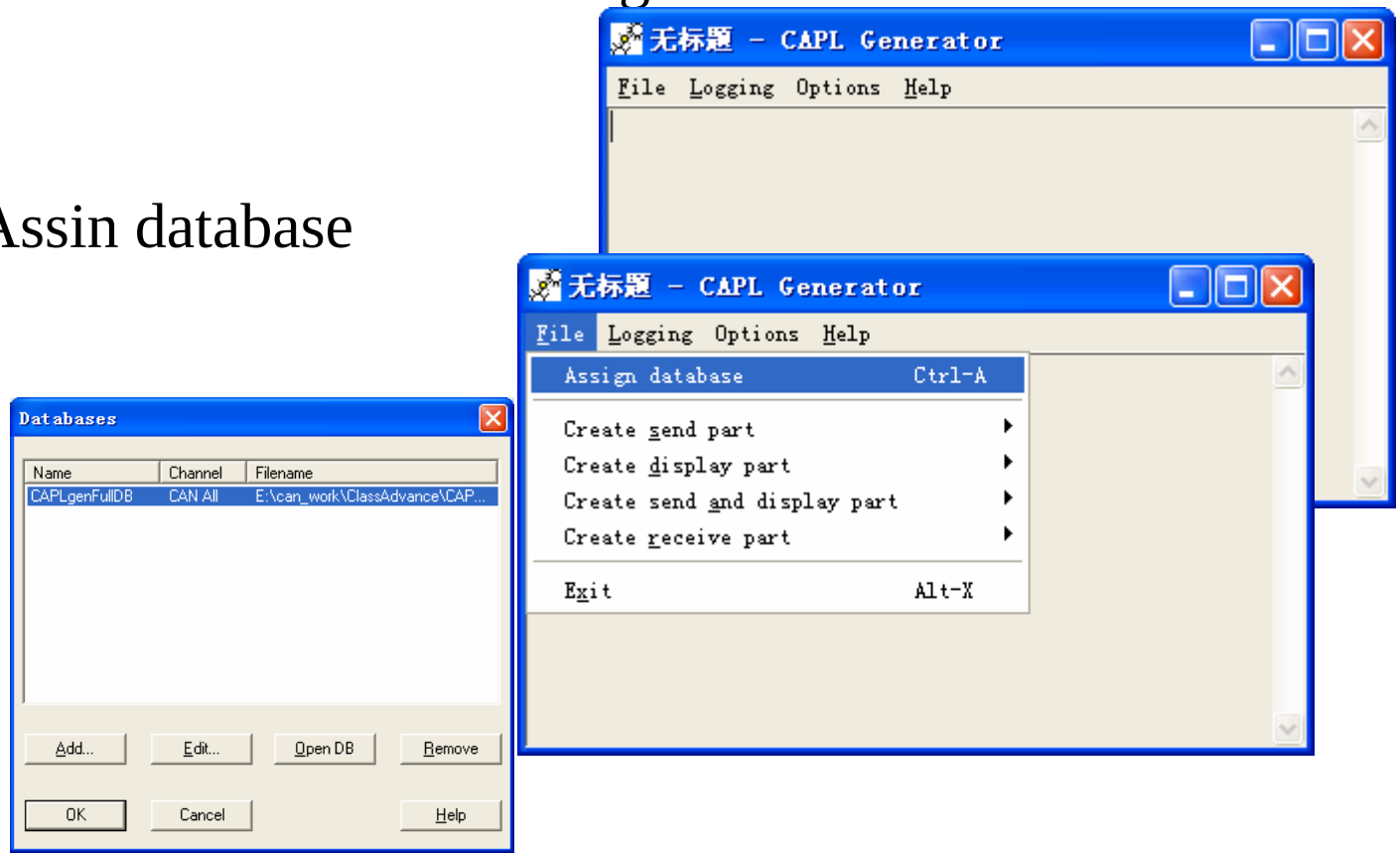
数据库准备

GenSigSendType

信号发送类型		相关属性	行为描述
cyclic	cyclic	GenMsgCycleTime	周期发送
OnEvent	OnWrite	-	写信号后发送
	OnChange	-	信号值改变后发送
	OnWriteWithRepetition	GenMsgNrOfRepetition	写信号后重复发送
	OnChangeWithRepetition	GenMsgNrOfRepetition	信号值改变后重复发送
IfActive	IfActive	GenMsgCycleTimeFast; GenSigInactiveValue	信号激活后，以快速周期发送报文，否则不发送。
	IfActiveWithRepetition		

工程生成

- 启动
  - <CANoe Install>\Exe32\CAPLgenerator.exe
- 导入
  - File->Assin database
  - Ctrl-A

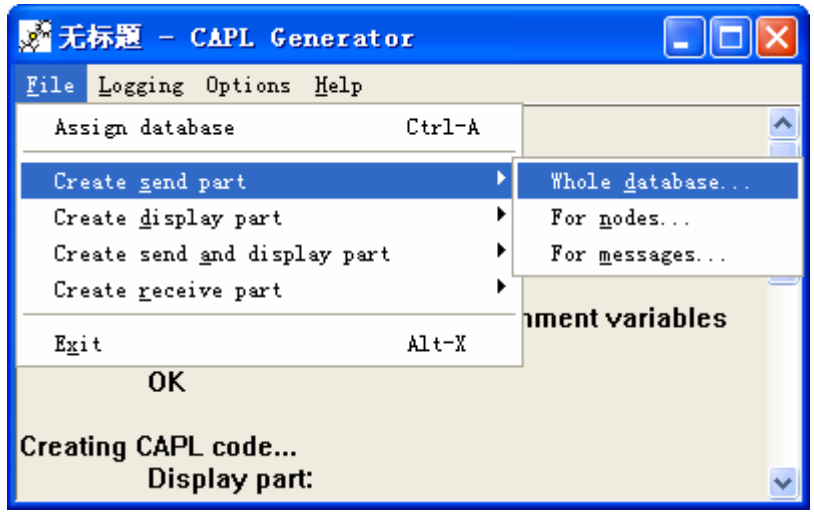


恒润科技 HIRAIN TECHNOLOGIES

vector

工程生成

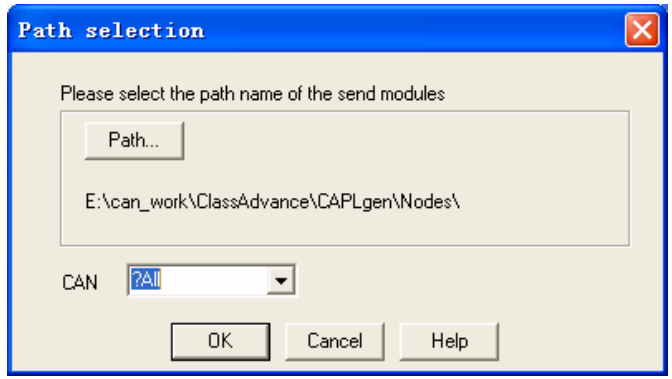
- 生成
  - Part
    - Send：信号发送
    - Display：报文、信号显示
    - Receive：报文接收
  - Range
    - Database：整个数据库（所有节点）
    - Node：部分节点
    - Message：部分

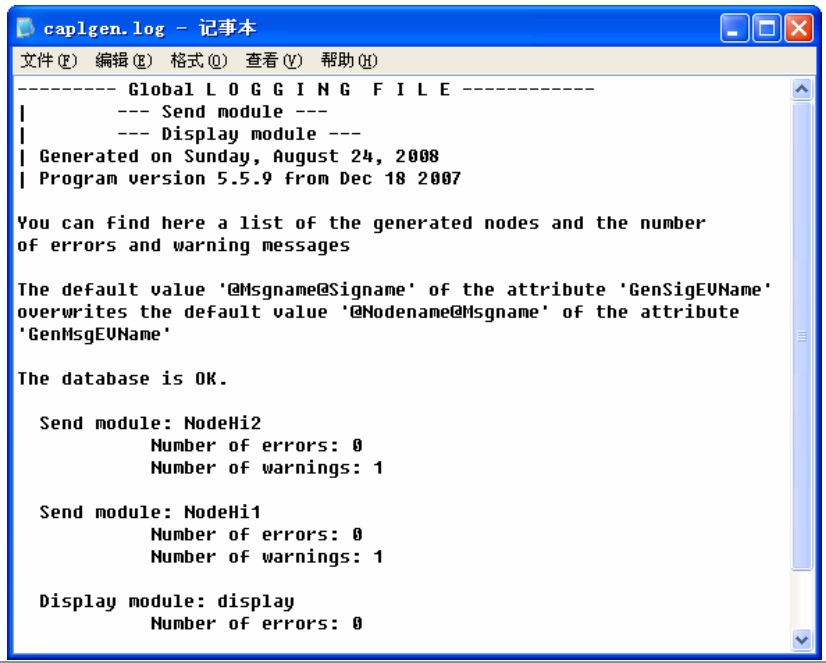


恒润科技 HIRAIN TECHNOLOGIES

vector

- 路径
  - 生成文件存放路径
- 文件
  - .can：CAPL脚本
  - .log：记录文件
- 记录
  - caplgen.log
    - 生成过程中错误及提示信息



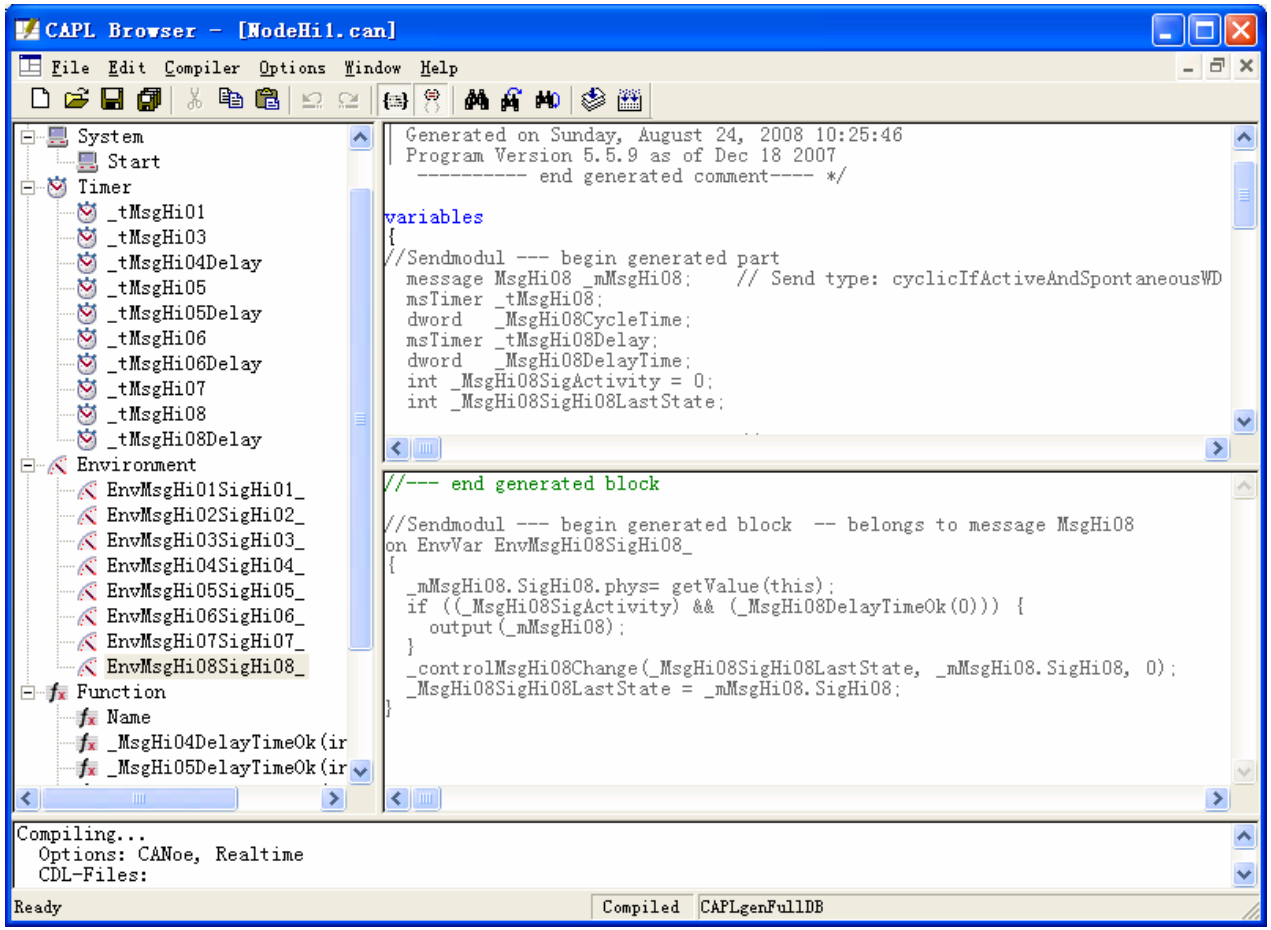


恒润科技 HIRAIN TECHNOLOGIES

vector

工程生成

- CAPL示例
  - Send:

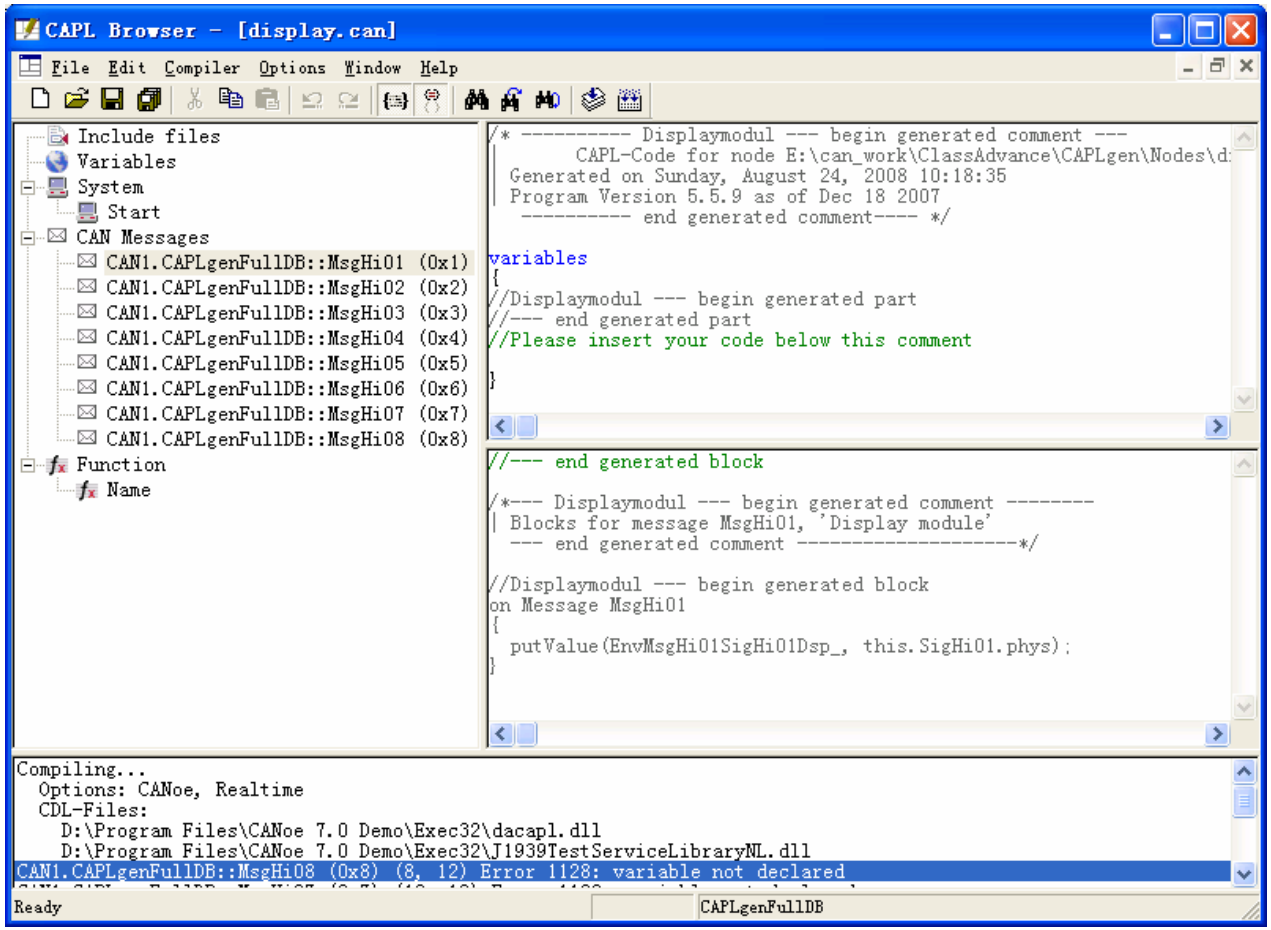


恒润科技 HIRAIN TECHNOLOGIES

vector

工程生成

- CAPL示例
  - Display:



恒润科技 HIRAIN TECHNOLOGIES

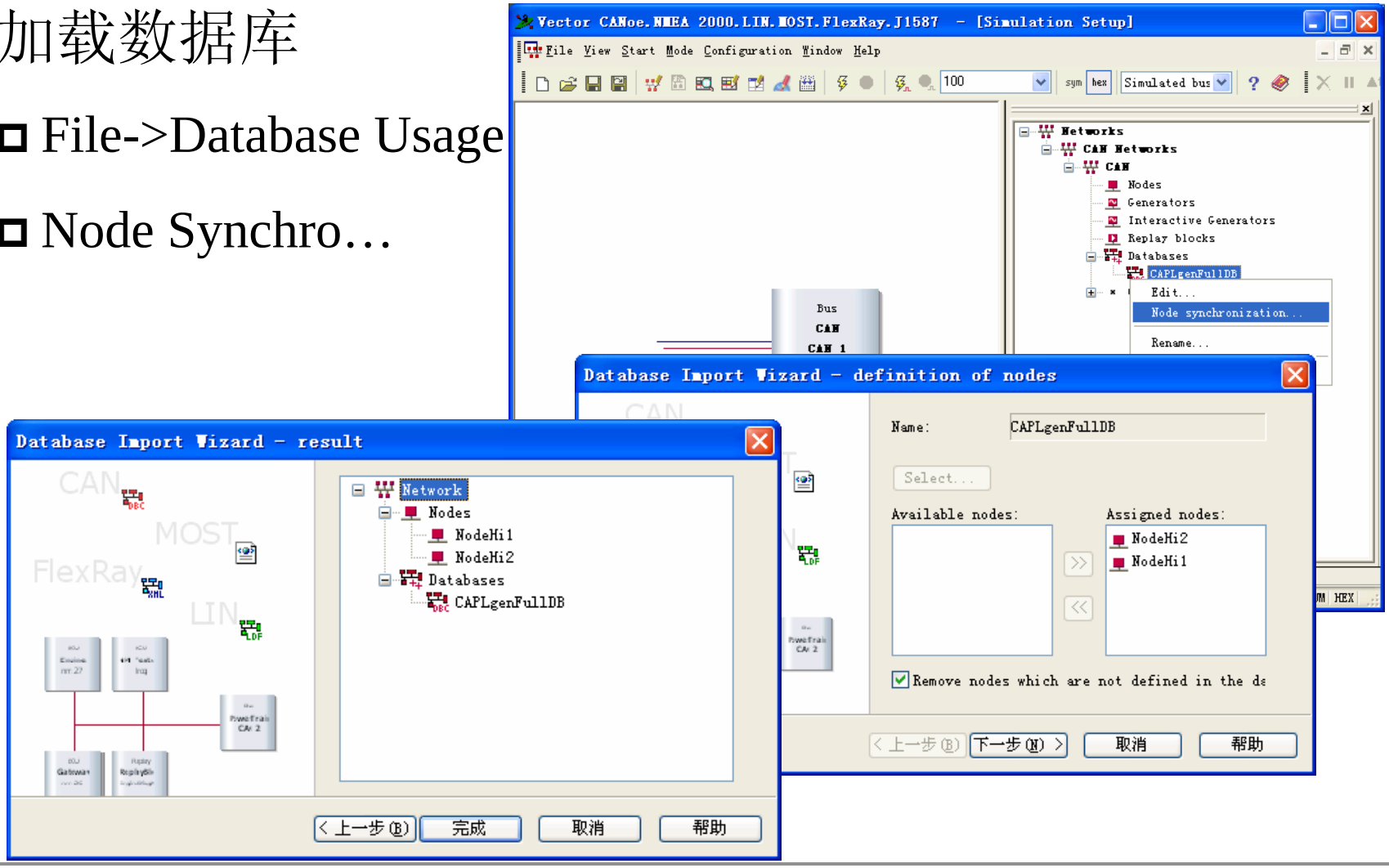
vector

网络仿真

- 加载数据库
  - File->Database Usage
- 加载CAPL脚本
  - 节点右键->Configuration-> Program Files
- 编译仿真运行
  - F9

网络仿真

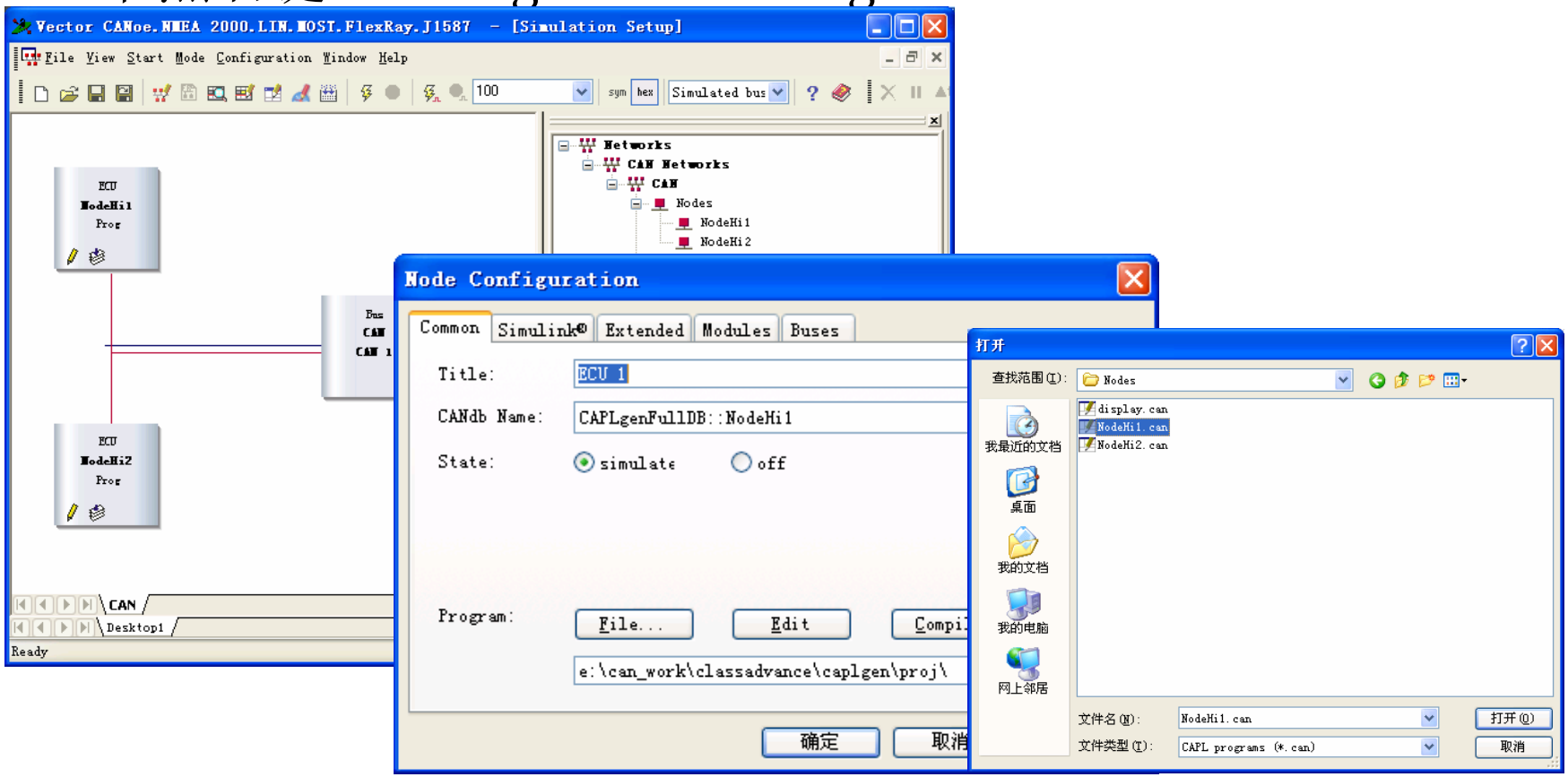
- 加载数据库
  - File->Database Usage
  - Node Synchro…



恒润科技 HIRAIN TECHNOLOGIES

vector

- 加载CAPL脚本
  - 节点右键->Configuration-> Program Files

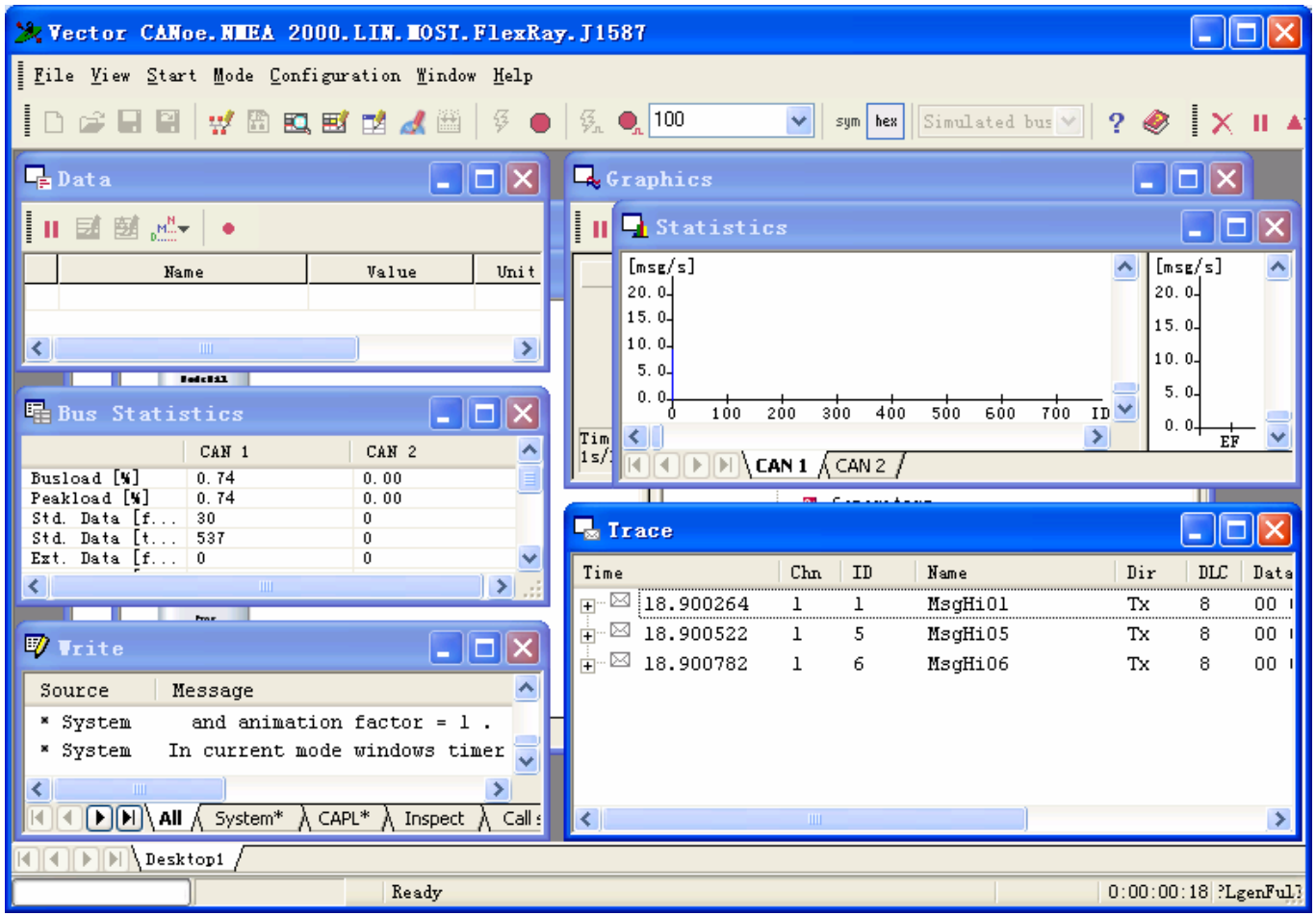


恒润科技 HIRAIN TECHNOLOGIES

vector

网络仿真

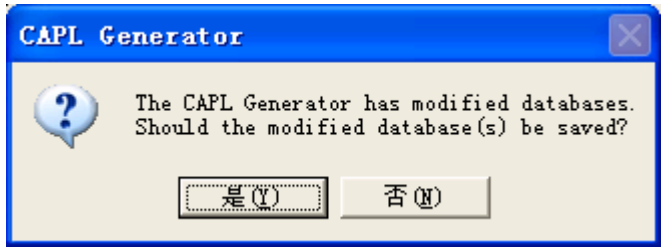
- 编译仿真运行
  - F9



恒润科技 HIRAIN TECHNOLOGIES

vector

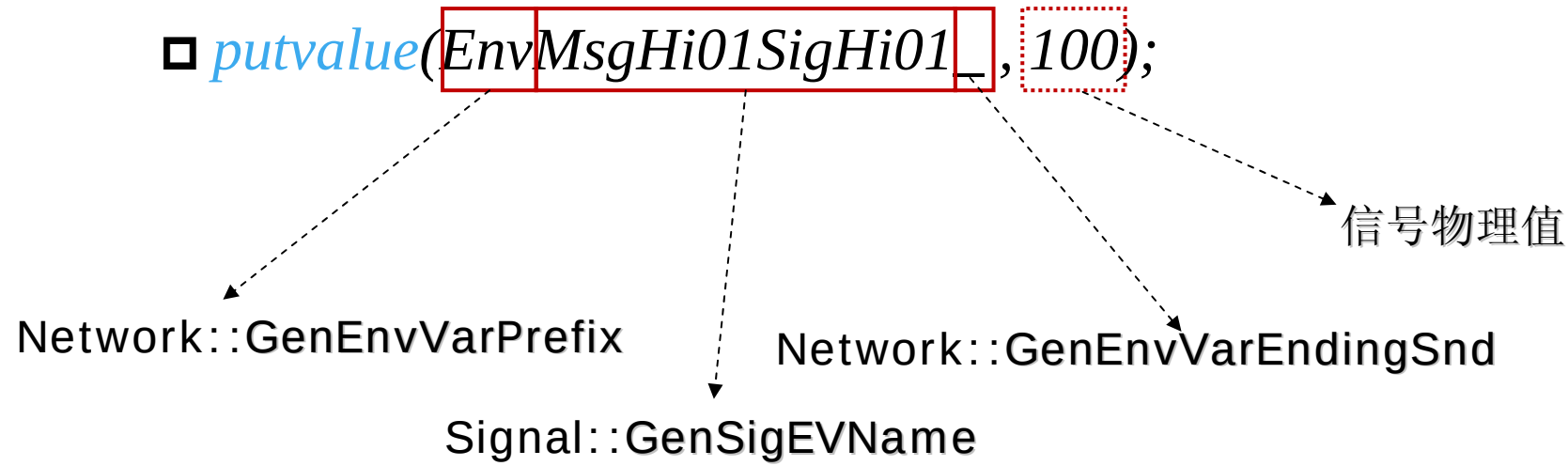
- 代码详解
  - IL接口
    - 信号环境变量：报文和信号发送
    - 该环境变量根据数据库属性在生成过程中自动添加



  - 发送行为
    - 脚本自动完成
    - 发送周期、重复次数、最小发送间隔、激活发送
  - 使用示例
    - 发送信号：
      - *putvalue(EnvMsgHi01SigHi01_, 100);*

集成应用

- 命名规则
  - *putvalue(EnvMsgHi01SigHi01_,100);*



  - 宏命名
    - @Nodename：取节点名字符串
    - @Msgname：取报文名字符串
    - @Signname：取信号名字符串
    - 例如：@Msgname@Signname

Q&A