Quick Start

--V20231003

目录

1. TSMaster

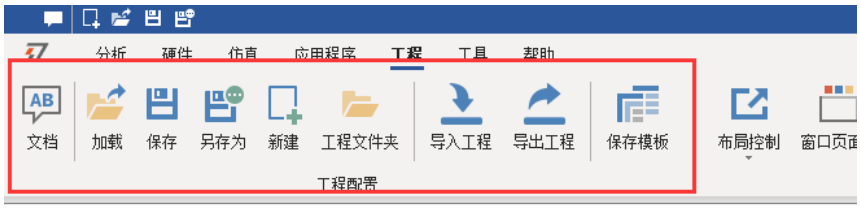
1.1. 工程管理

TSMaster 的工程配置文件后缀是.T7z,也就是带压缩加密功能的配置文件。T7z 文件包含了当前配置的所有内容：窗体，窗体上的信息，数据库，Panel，C 脚本。TSMaster 的工程配置文件具有如下优势：

- 工程配置文件里面包含了所有信息，发布文件非常简洁，一个 T7z 文件就可以。目前市面上大部分工程的工程配置文件里面保存的是文件链接（比如数据库，Panel 等链接），因此发布的时候工程配置目录下面除了配置文件还会有其他各种支持文件。
- 工程文件本身是带加密的。

1.1.1. 工程文件管理

工程文件主要包含如下操作：加载，保存，另存为，新建。



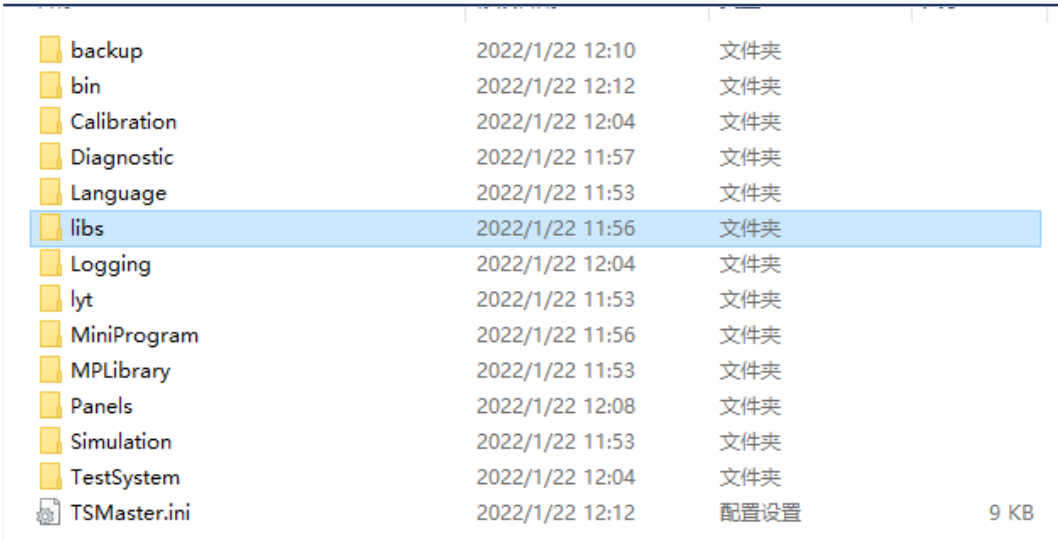
TSMaster 的工程项目管理有两种形式：工程目录和 T7Z 文件。其区别如下：

- 工程目录：所有工程相关的原始配置文件存放在单独的目录里面，方便基于 Git，SVN 等管理工具。
- T7Z 文件：把工程相关的原始配置文件打包压缩到一个独立的 T7Z 工程中。优点是：整个工程目录压缩为一个独立的文件，会带上所有的 DBC，A2L，Panel 等配置，方便工程发布，交互。

工程目录和 T7Z 文件是同时支持的，用户可以根据需求灵活使用。

1.1.2. TSMaster 工程目录

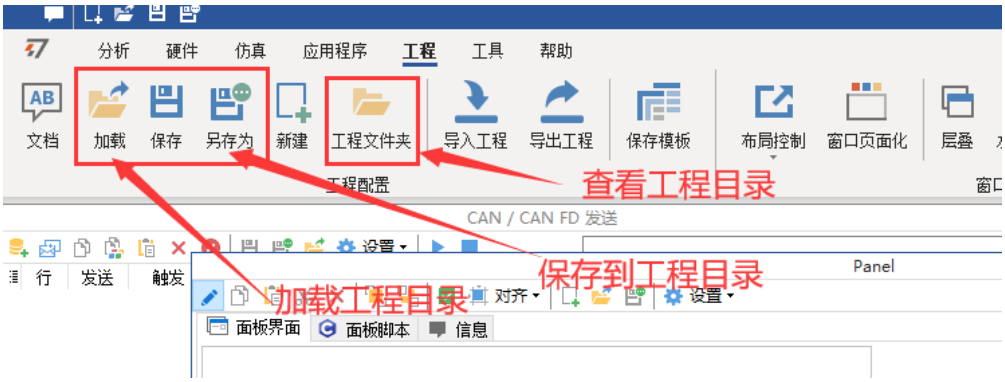
从 V2022.1.21.693 版本开始，TSMaster 引入了工程目录。就跟常规软件一样，把所有项目相关的文件存放在一个工程目录中。一个典型的 TSMaster 工程目录如下图所示：



如上图所示，各个目录存放的基本功能模块的配置如下所示：

- Calibration: 存放标定相关的配置
- Diagnostic: 存放诊断相关的配置
- Language: 存放多语言相关的配置，除非特殊情况，此文件夹不要改动。
- libs: 默认的 C 脚本工程的存放目录
- Logging： 默认的 blf 等报文数据存放目录
- MiniProgram：存放小程序的目录
- MPLibrary：存放小程序库的目录
- Simulation：存放 RBS 仿真相关的配置
- Panels：存放面板相关的配置
- TestSystem：存放测试系统相关的配置

1.1.2.1. 工程目录操作面板



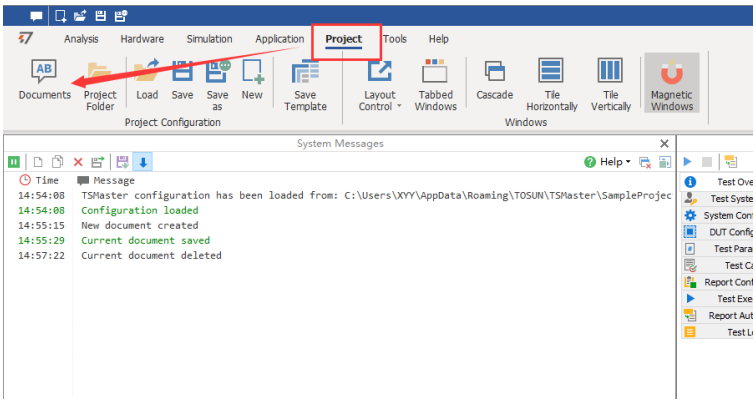
1.1.3. T7Z 工程文件

1.1.3.1. T7Z 操作面板

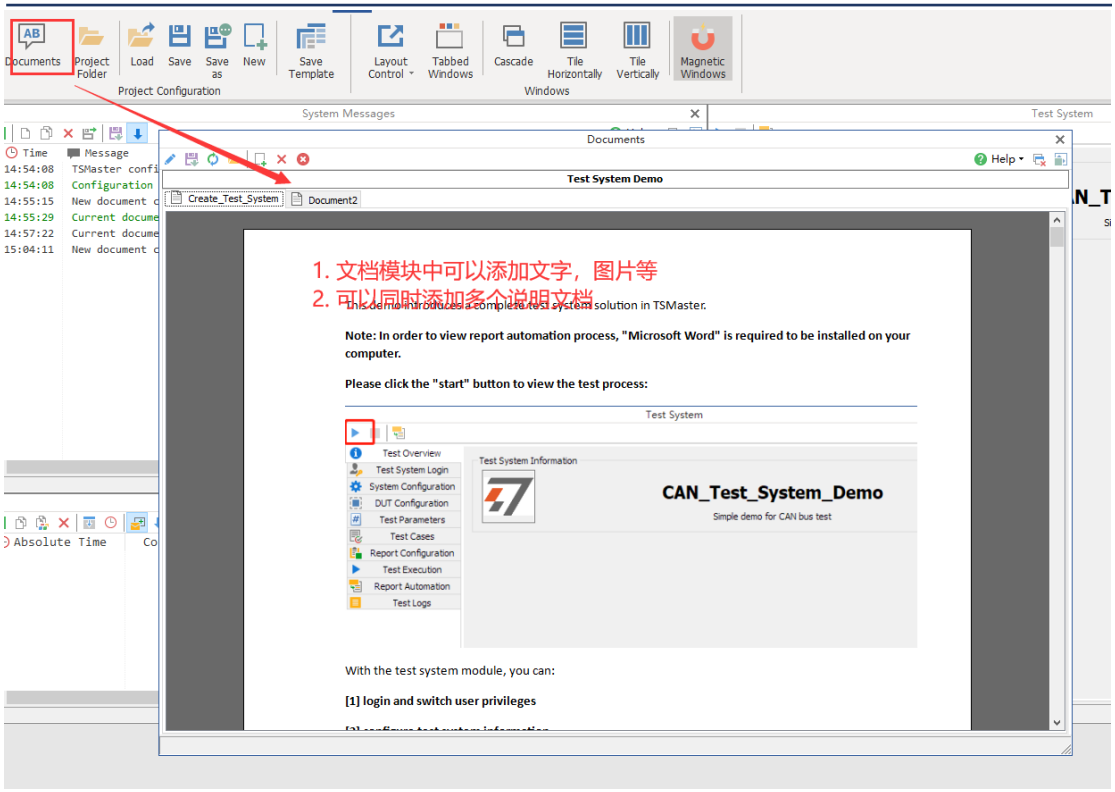
T7Z 操作面板如下图所示：

1.1.4. 为工程（Project）添加说明文档

用户完成工程开发过后，如果把工程发布给客户使用，需要编写针对该工程的使用说明书。TSMaster 内置了工程说明文档管理模块。

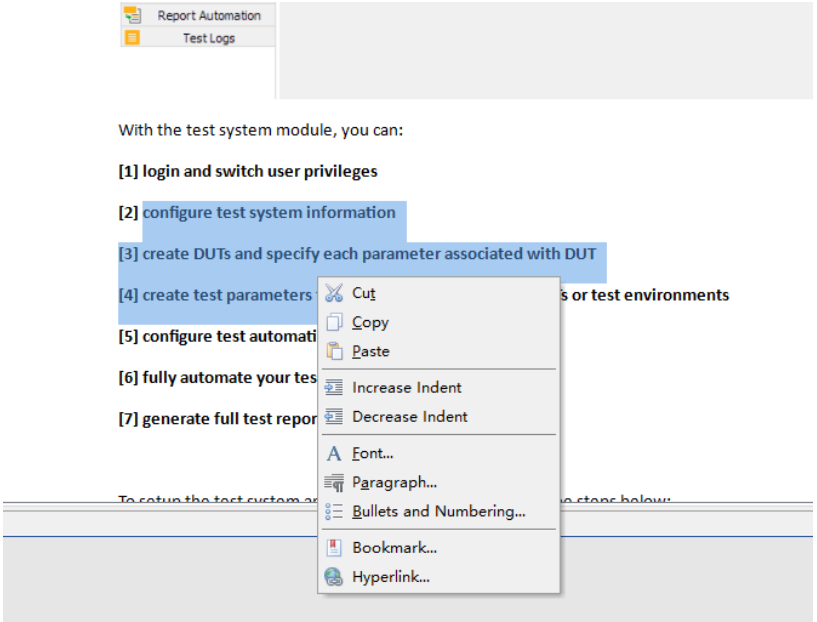


通过该模块，用户可以添加针对该配置工程的使用说明。保存工程的时候，该工程说明文档会一并打包到工程文件（t7z 文件）中。用户载入 t7z 文件过后，如果想获得该工程的使用说明，只需要到这里打开文档模块即可。

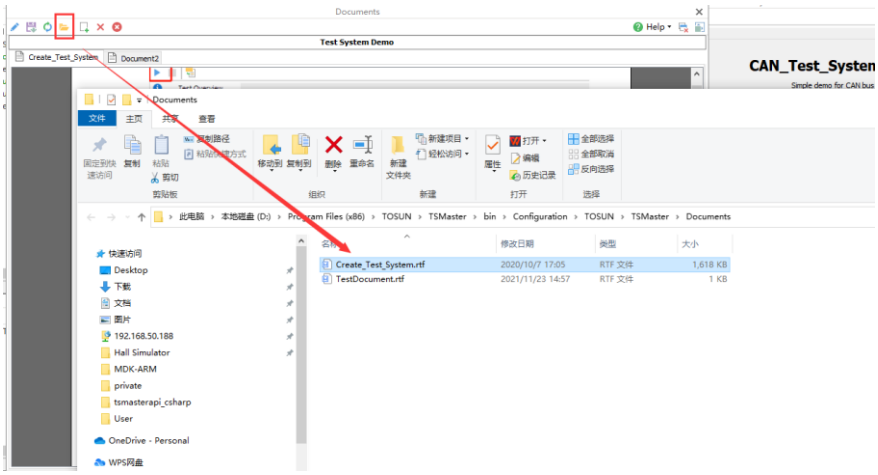


在文档中，如果要修改文字显示格式等，有两种方式：

1. 直接在 TSMaster 内置的编辑器中编辑：选中对应的文字，右键，就可以看到属性设置窗体，进入设置属性即可。



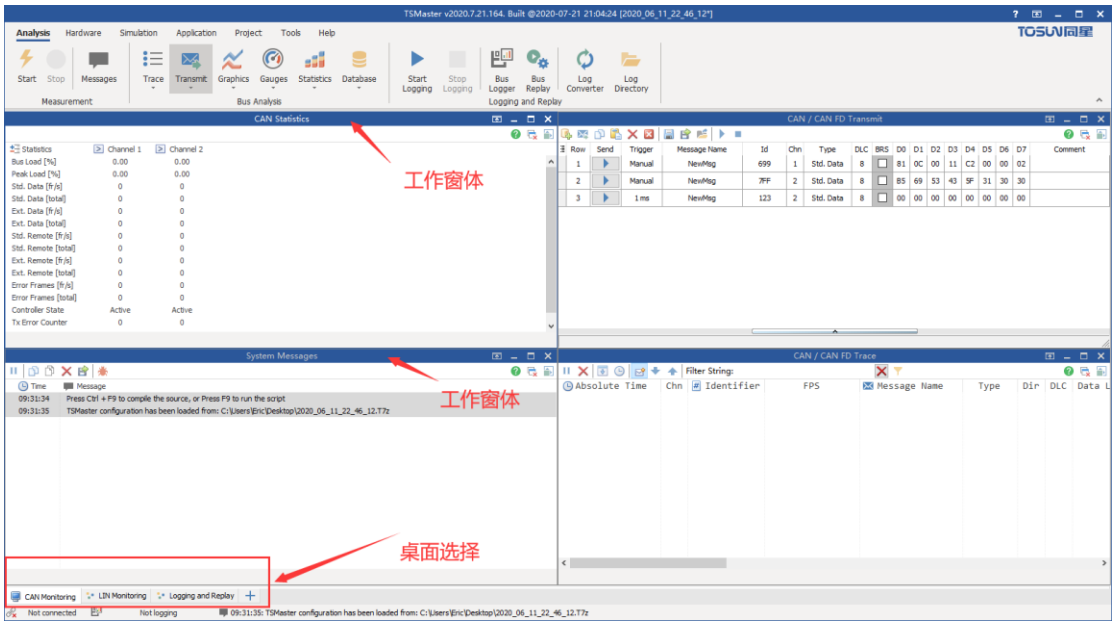
2. 从编辑器打开说明文档目录，直接找到原始 rft 文件，通过 office 软件进行编辑。如下所示：



1.2. 增加管控权限

公司都会有内部权限管理系统。用户在安装完 TSMaster 后，需要联系公司 IT，把 TSMaster 内部相关的模块添加到白名单中，才能够正常运行。主要有以下模块需要添加到白名单中：

1.3. 工作区和窗体

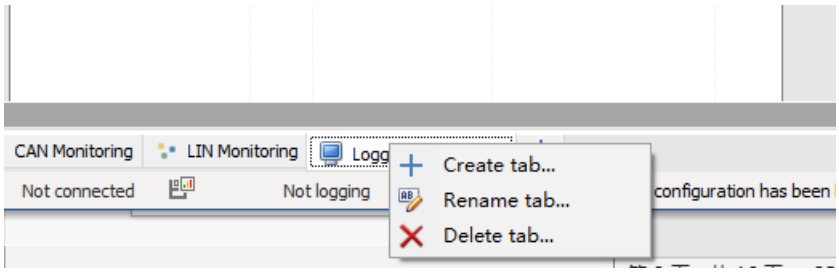


TSMaster 设计的一个重要的点就是尽可能给用户提供最大的灵活性，因此提供了工作区和窗体机制。

1.3.1. 工作区：

可以理解为窗体容器，便于用户把功能相关的窗体放到同一个容器里面。软件安装完

成过后，默认包含：CAN Monitor，LIN Monitor 以及 Logging And Replay 三个工作区。工作区支持增加，删除，重命名等操作，用户在完成工作区的配置过后，可以保存为模板供其他人使用。



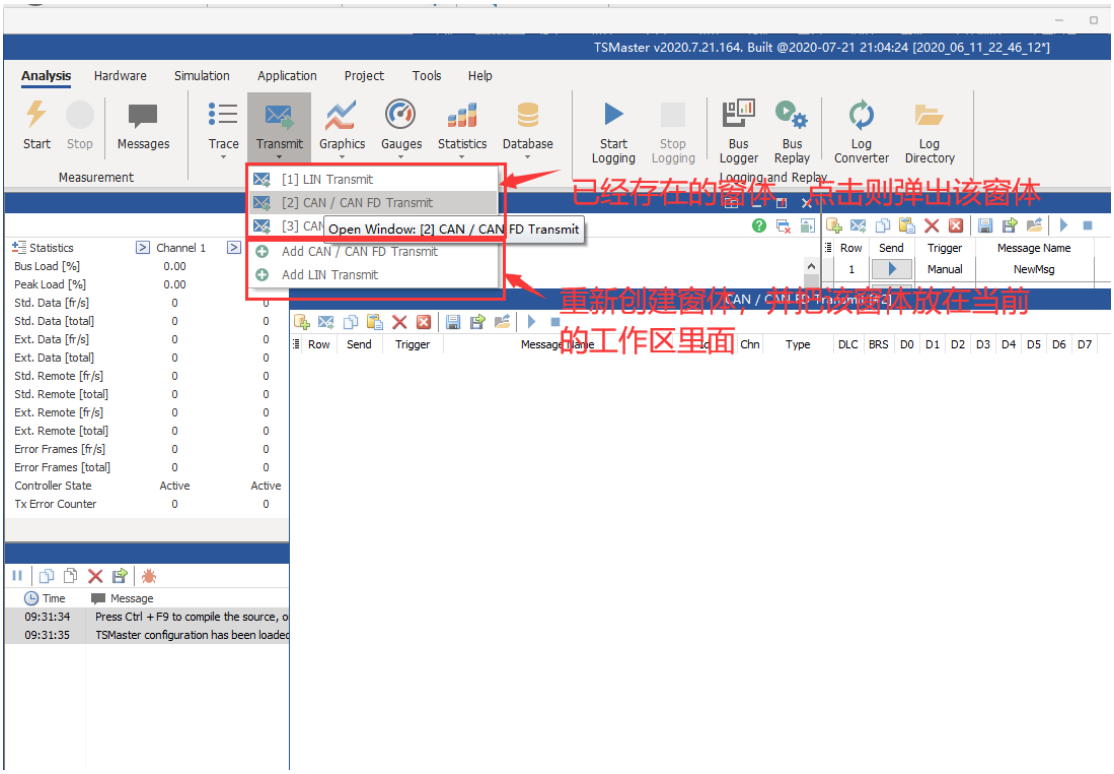
支持增，删，编辑等操作

创建工作区过后，用户就可以在其中添加窗体。

1.3.2. 窗体：

每一个窗体代表一个功能模块。如 CAN Trace, LIN Trace, CAN Transmit, LIN Transit，CCode 等窗体。

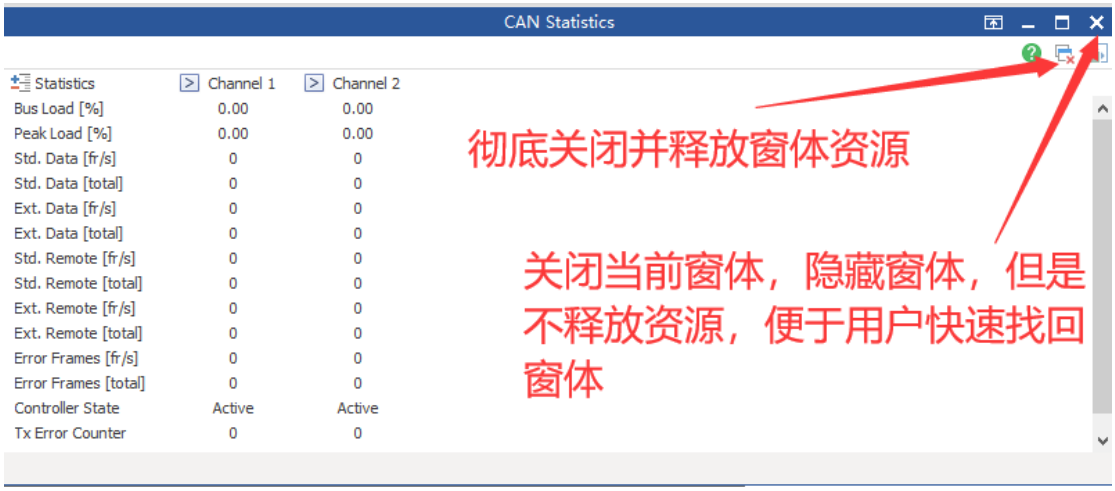
1.3.2.1. 添加窗体



用户可以同时创建多个同类型窗体，互相之间是独立的存在。以 CAN Trace 窗体为例，用户可以在 CAN Monitor 工作区添加一个 CAN Trace 窗体，同时也可以在 Logging And Repay 窗体中添加一个 CAN Trace 窗体，程序运行过程中，两个窗体并行独立工作。

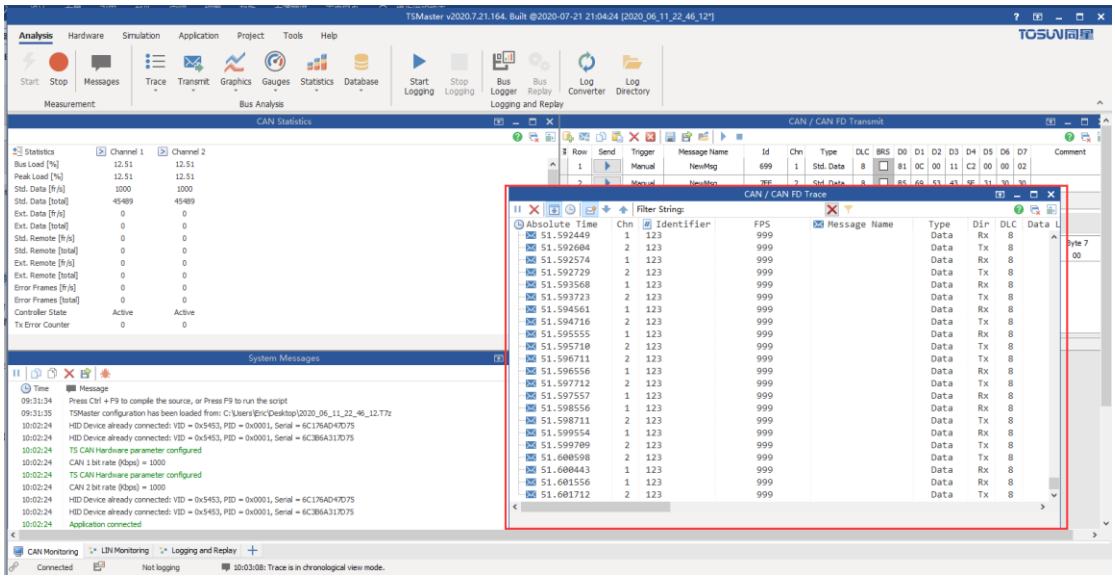
1.3.2.2. 关闭窗体

TSMaster 关闭窗体提供了两个层级的关闭。点击最右上角的  图标，只是隐藏窗体，但是不释放资源，便于用户快速找回窗体。如果要想彻底关闭本窗体，点击  图标即可。

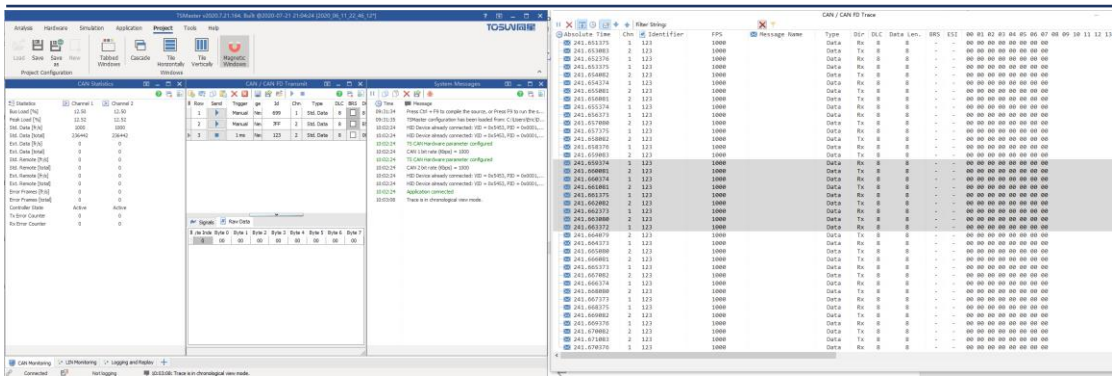


1.3.2.3. 窗体弹出/弹入

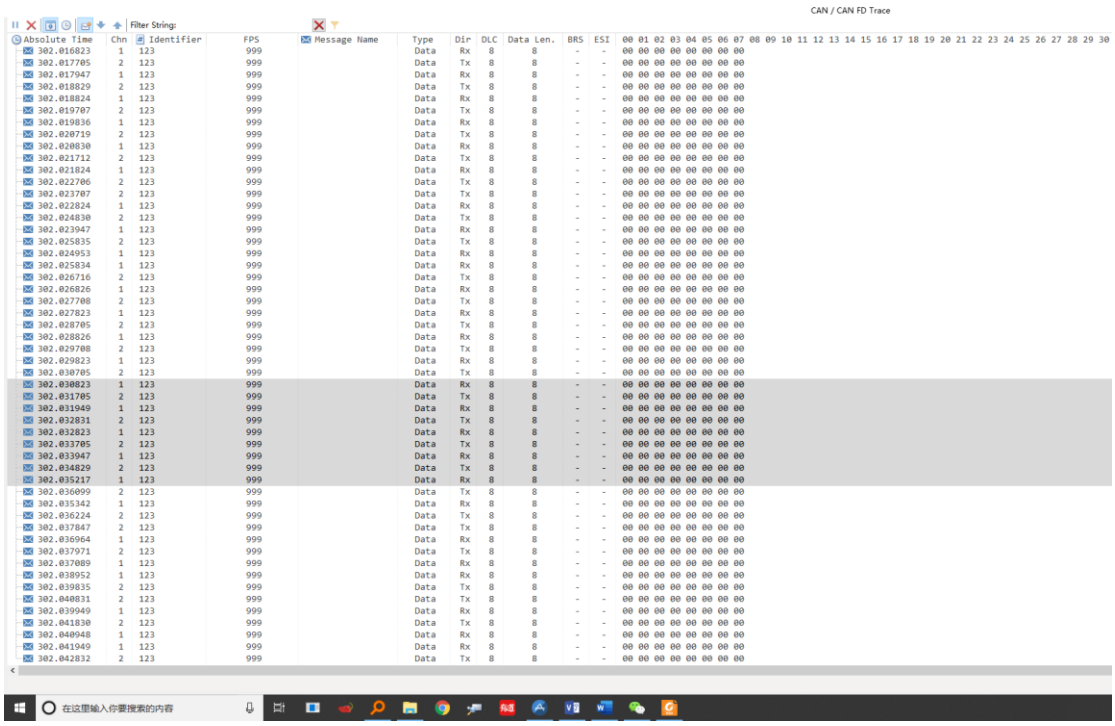
TSMaster 采用的是 MDI 窗体模式，各个子窗口运行在整个父窗体提供的工作区里面。如果用户想重点监测某些窗口，比如，想查看 Trace 窗口的详细内容，子窗体的形式就局限了观察的范围,如下所示：



因此,TSMaster 还提供了窗体弹出机制。用户点击子窗体右上角的  按钮，子窗体就跳出父窗体的限制，成为一个跟父窗体并行的窗体，如下图所示：



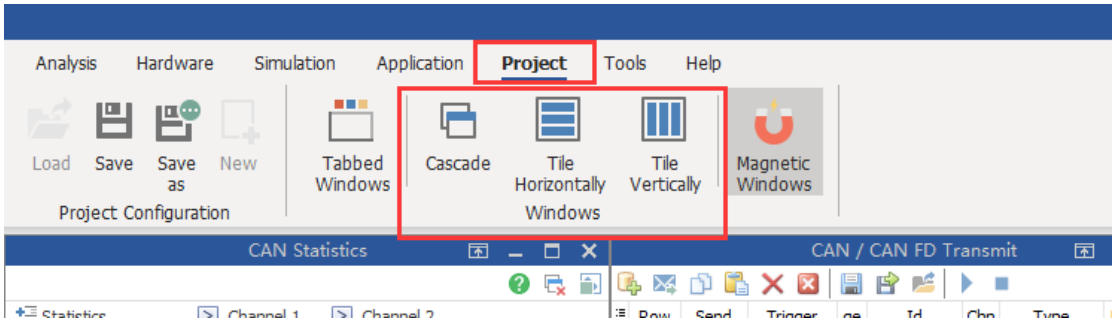
这种模式下，该子窗体成为一个完全独立的窗体，不再受到父窗体的约束，占满用户的整个屏幕都可以。



要回到 MDI 模式，同样点击该窗体的 图标，该窗体就弹入到之前的工作区中。

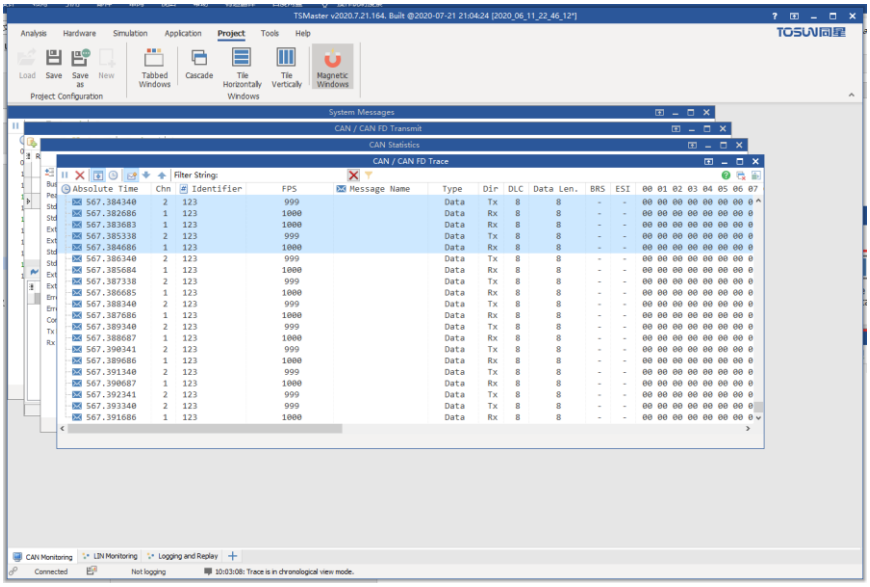
1.3.2.4. 窗体对齐

一个工作区中，经常存在多个子窗体同时工作的状况。窗体经过多次拖拽，相互间顺序会变得比较凌乱，TSMaster 提供了对齐快捷键，路径如下：Project->Window

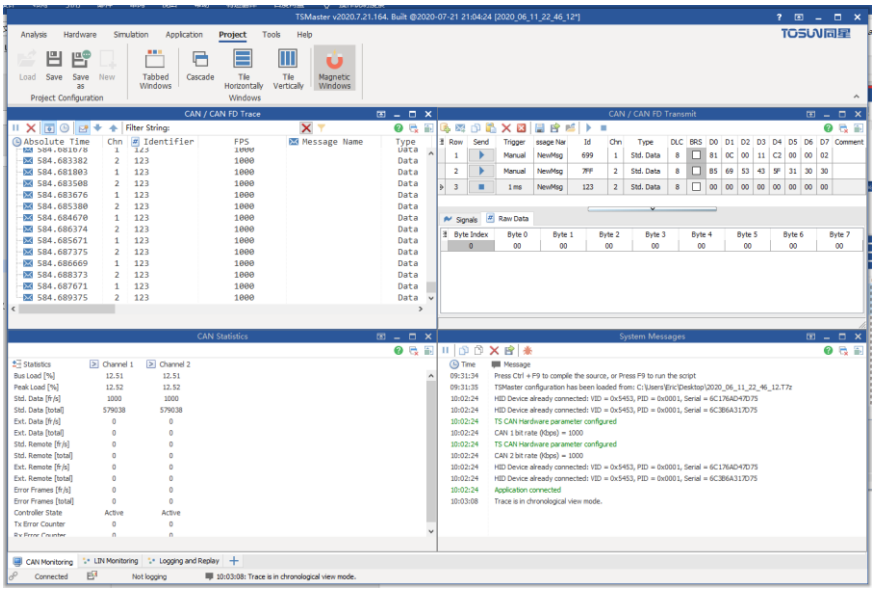


三种对齐方式的效果如下：

- Cascade：层叠排列



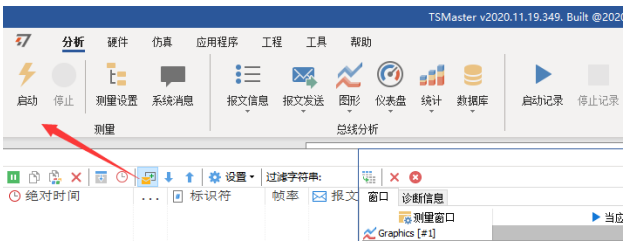
- Tile Horizontally：横向排列



- Tile Vertically：纵向排列

1.3.3. 启动程序（Start）

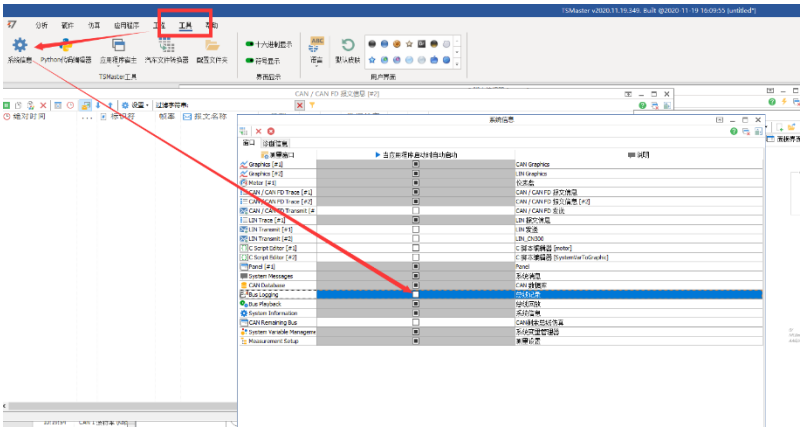
点击启动（Start）按钮，启动 TSMaster 运行环境，如下所示：



启动过程需要注意的是：有的需求场合，一些功能模块需要在启动瞬间同时启动，有

的功能模块无此需求。因此 TSMaster 提供了：启动（Start）和功能模块关联/解除关联的模块。主要在两个地方可以进行设置：

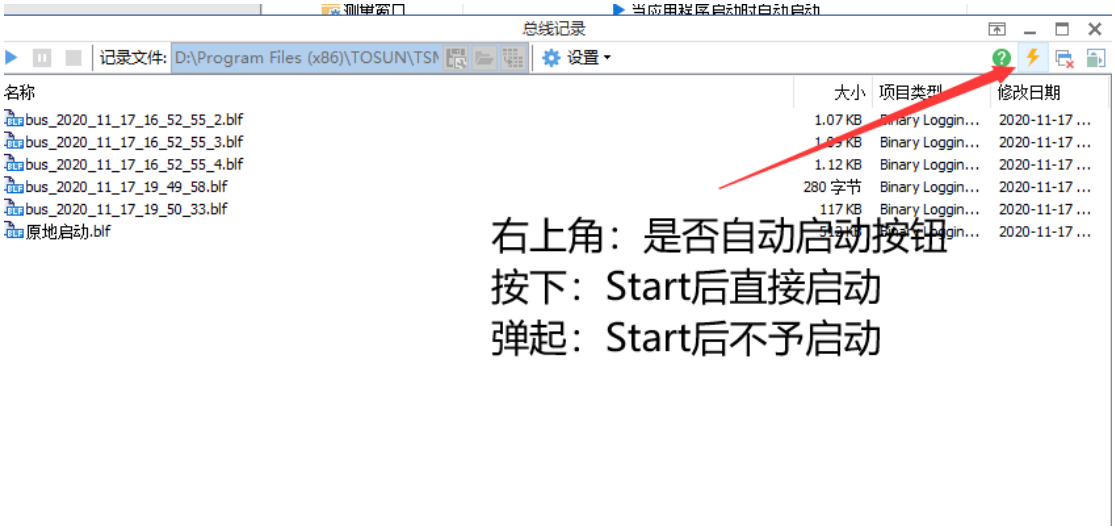
1.3.3.1. 在系统信息中进行关联



通过路径：工具->系统信息进入。在此面板上勾选相关的功能模块。 这种状态的表示启动瞬间默认必须启动的模块，不可配置。其他节点勾上，则启动瞬间响应的模块开始工作；不勾选，则启动瞬间响应模块保持静默。

1.3.3.2. 直接在功能模块上关联

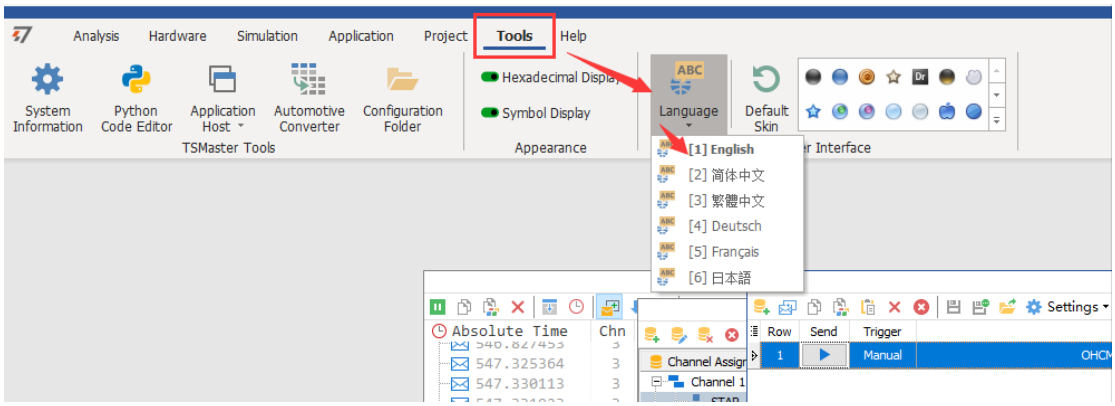
在每一个功能模块的右上角，有一个选择是否自动启动的按钮，其配置原理如下所示：



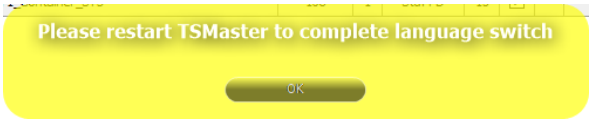
上述操作，本质上跟系统信息中设置是一样的。

1.3.4. 多语言支持

TSMaster 支持多种语言。可通过窗体按键切换，如下图所示：



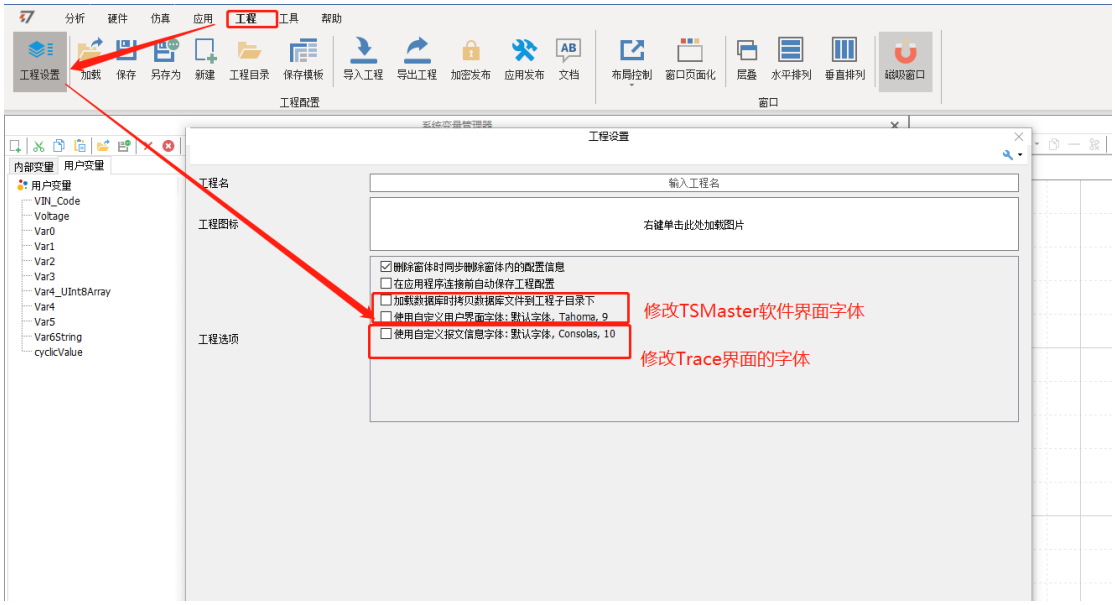
点击切换语言按钮后，系统会进行如下提示:



用户重启软件，软件即完全切换到新的语言环境下。同时，TSMaster 会记住此选项，下一次启动的时候默认选择该语言项。

1.3.5. 修改窗体字体

如果用户屏幕过大后者过小，软件默认字体无法满足用户的需求的时候，TSMaster 允许用户根据需求调整字体大小。主要包含软件通用字体和报文 Trace 界面的字体，调整的路径如下图所示：工程->工程设置->工程选项->使用自定义的界面字体/使用自定义报文信息字体（Trace 字体）。

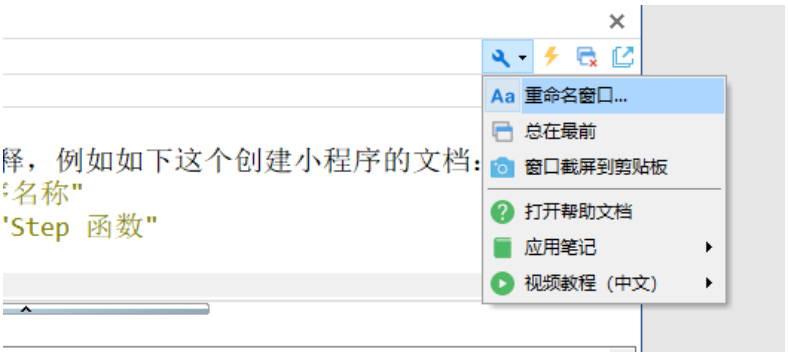


1.3.6. 修改窗体名称

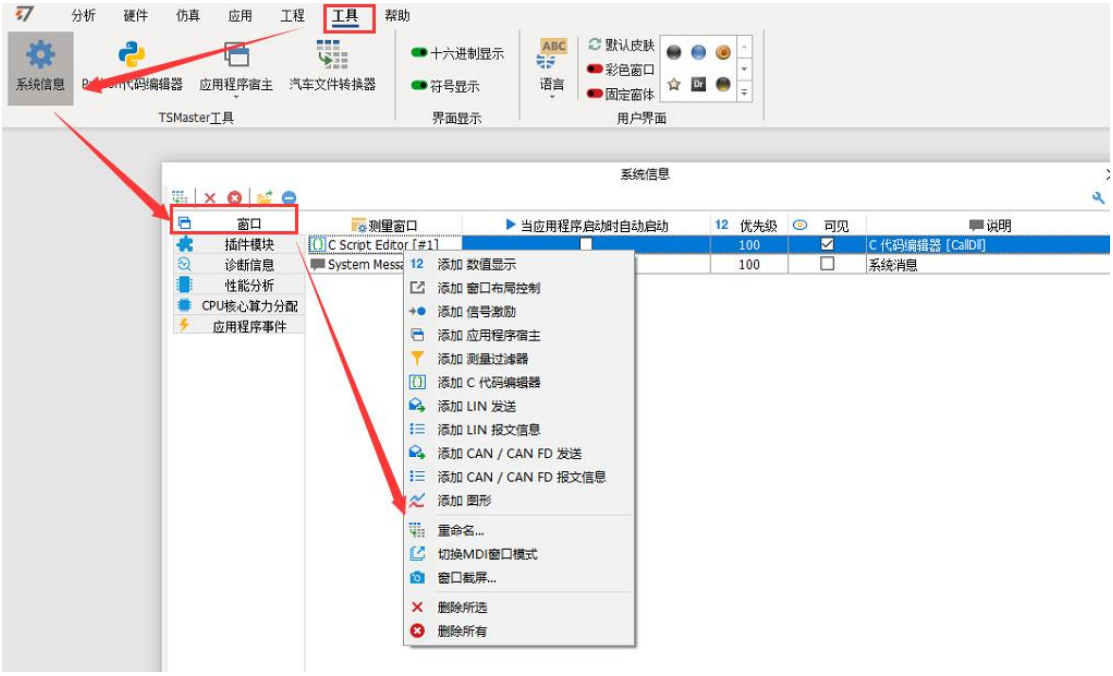
TSMaster 支持对窗口重命名，主要有如下方法：

- 方法 1：窗体右上角，点开配置按钮，在展开的下拉框中选择重命名窗口，如下

所示：

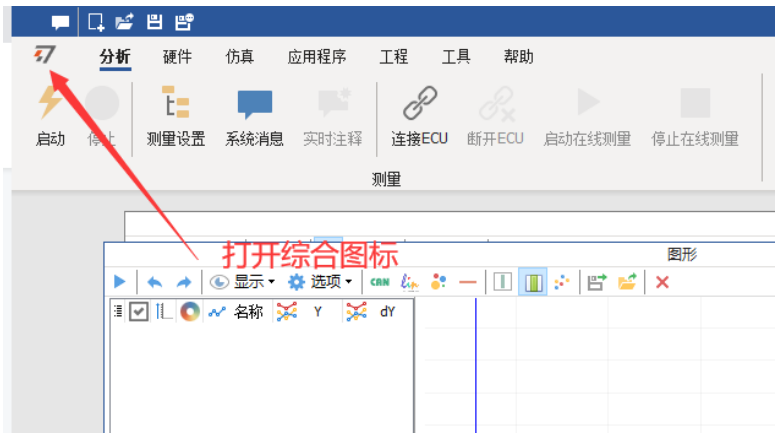


- 方法 2：通过路径：工具->系统信息->窗口->选中窗体，右键->重命名



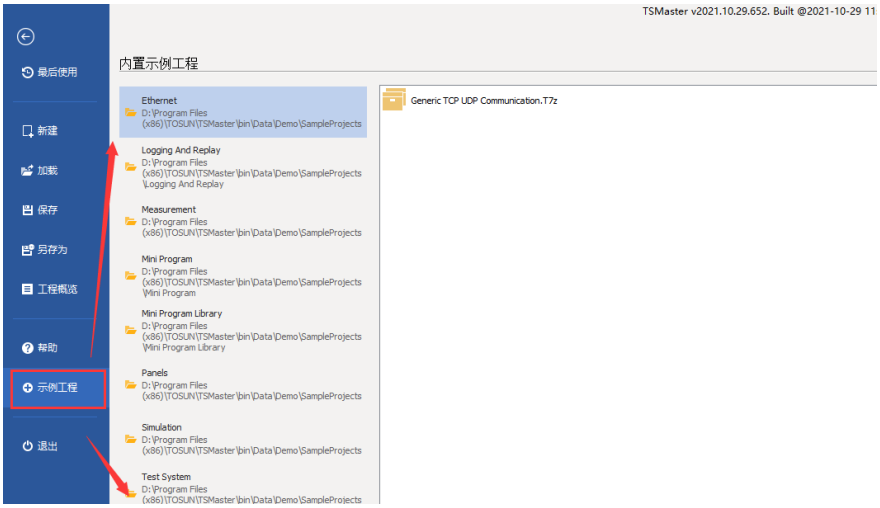
1.3.7. 打开示例工程

- 首先打开综合管理界面：



打开综合管理窗口

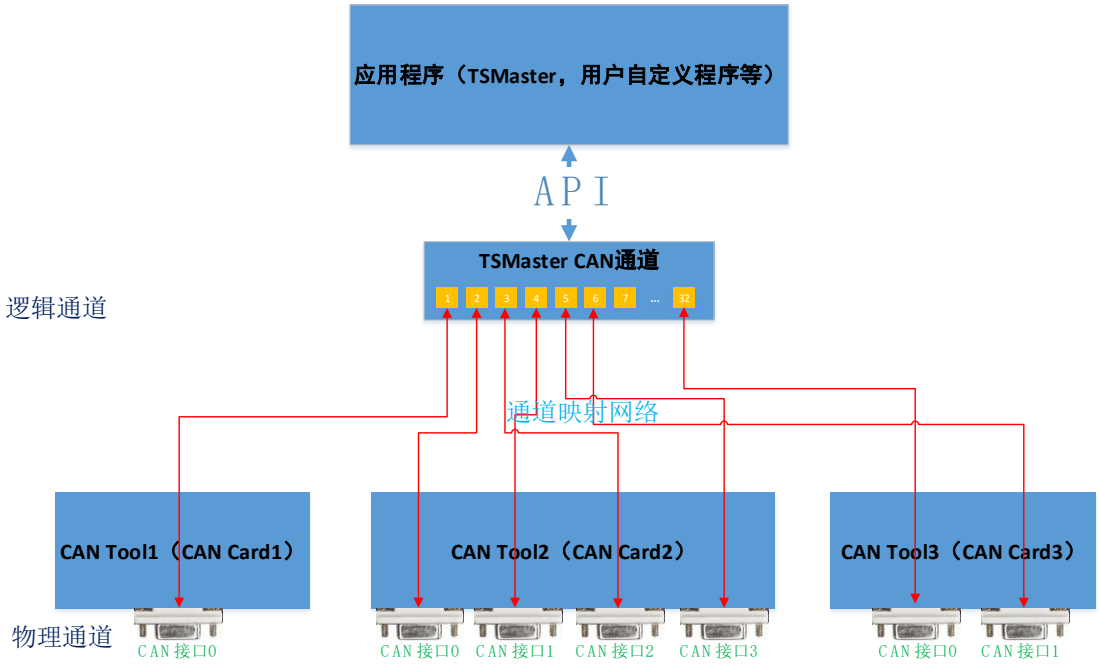
- 选择示例工程：



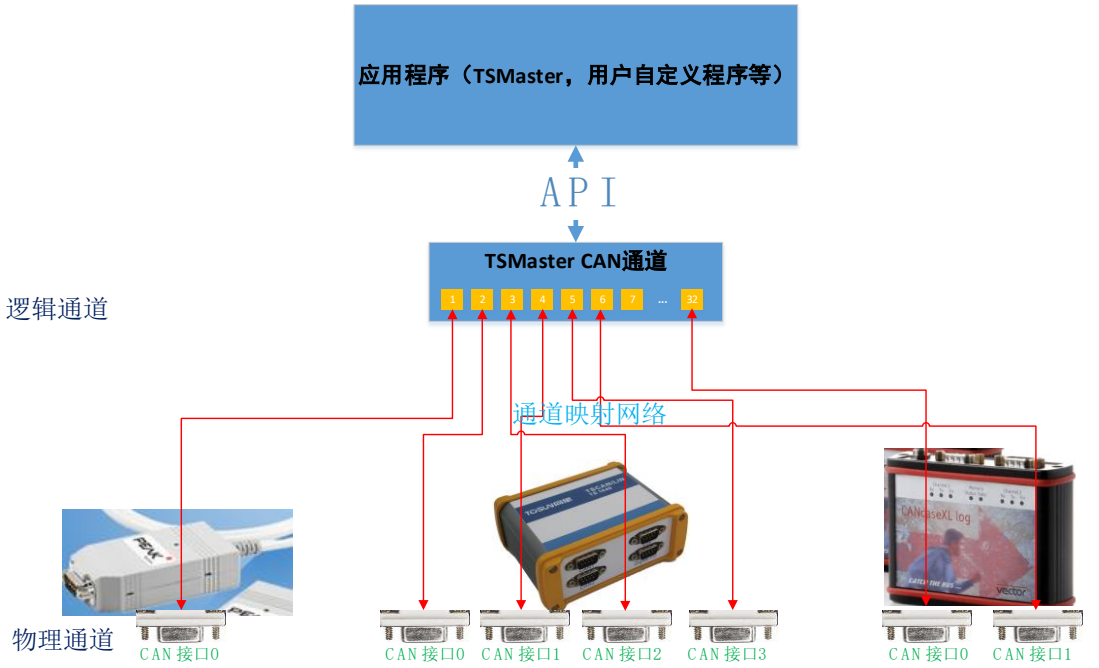
1.4. 硬件配置

1.4.1. 硬件通道映射

应用程序直接调用 TSMaster 驱动中的逻辑通道，逻辑通道通过映射的方式关联到实际的 CAN 工具（CAN 卡）的物理通道上。映射机制如下图所示：



更直观的展示映射的效果：



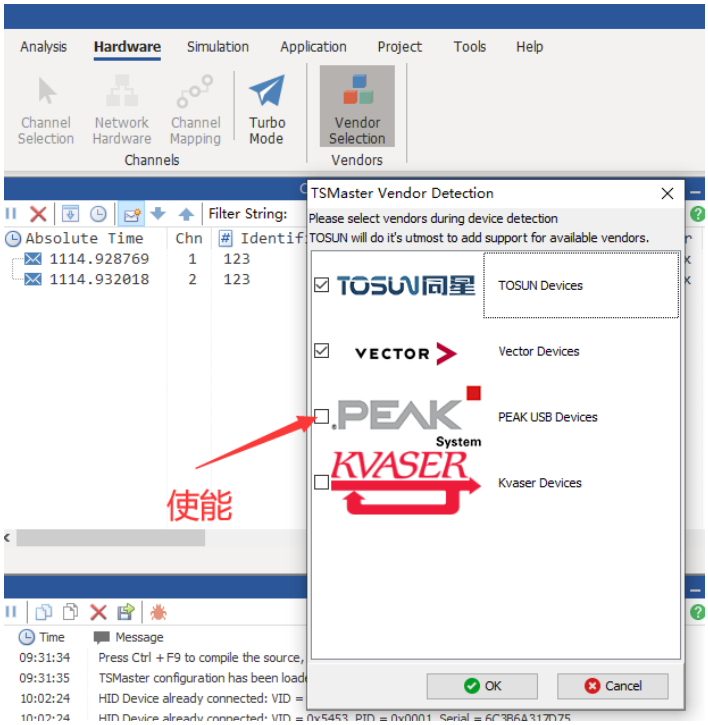
1.4.2. TSMaster 映射的意义

TSMaster 采用了映射的机制，主要处于目的：

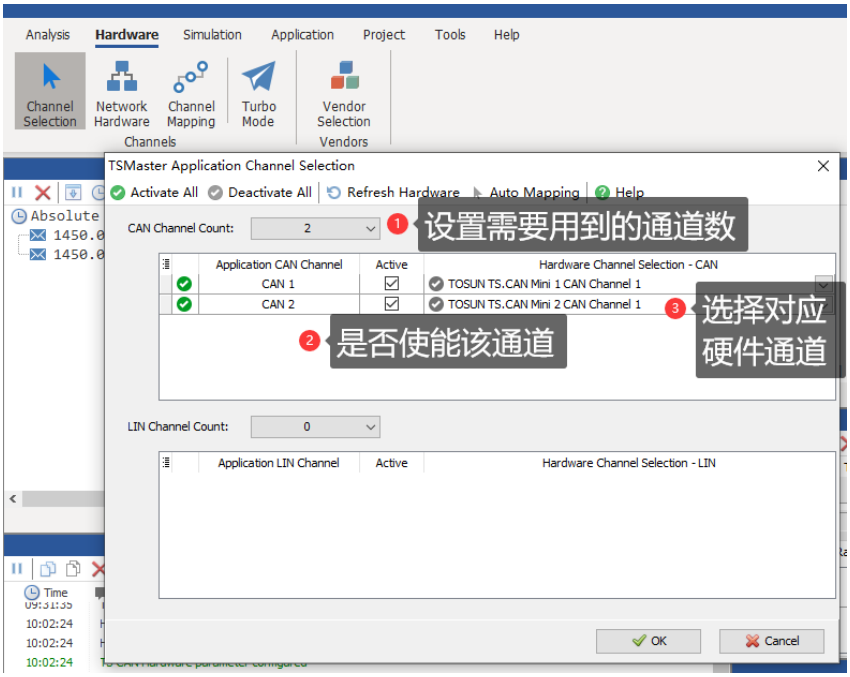
1. 尽可能兼容客户现有的硬件，客户手上无论是 Vector，Peak，Kvaser 还是同星公司自家的 TSCAN 系列工具，都可以通过映射的方式直接在 TSMaster 中使用。
2. 逻辑通道对于上层应用层是统一的接口，用户在使用 TSMaster 脚本开发测试程序的时候，不用担心切换工具过后，之前开发的工作被浪费。
3. 逻辑通道扩展了物理通道的通道数量限制。因为接口物理空间的限制，常见 CAN 工具通道有 1,2,4 路通道，如果在某些应用场合，需要超过 4 路，6 路甚至更多路的 CAN 通道协同工作，直接靠硬件设备无法满足要求，那就可以通过逻辑通道的方式，把多个硬件里面的通道映射到 TSMaster 中协同进行操作。关于通道之间的精确同步，后续会详细讲解。

1.4.3. 加载工具驱动

TSMaster 兼容的工具种类众多，默认只加载 TOSUN 和 Vector 的硬件驱动。如果想使用其他厂商的 CAN 驱动，需要到硬件工具提供商页面使能对应的硬件驱动，如下所示：

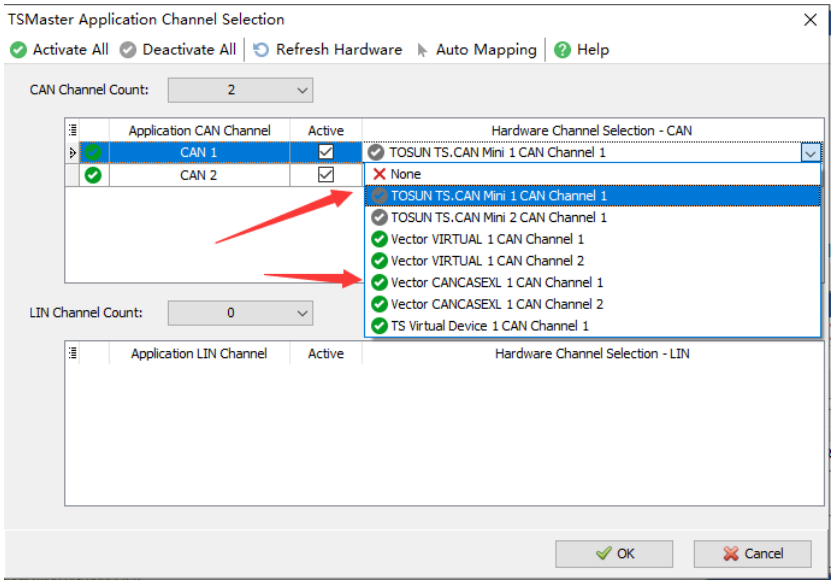


1.4.4. 选择硬件通道：



1.4.4.1. 选择硬件通道：

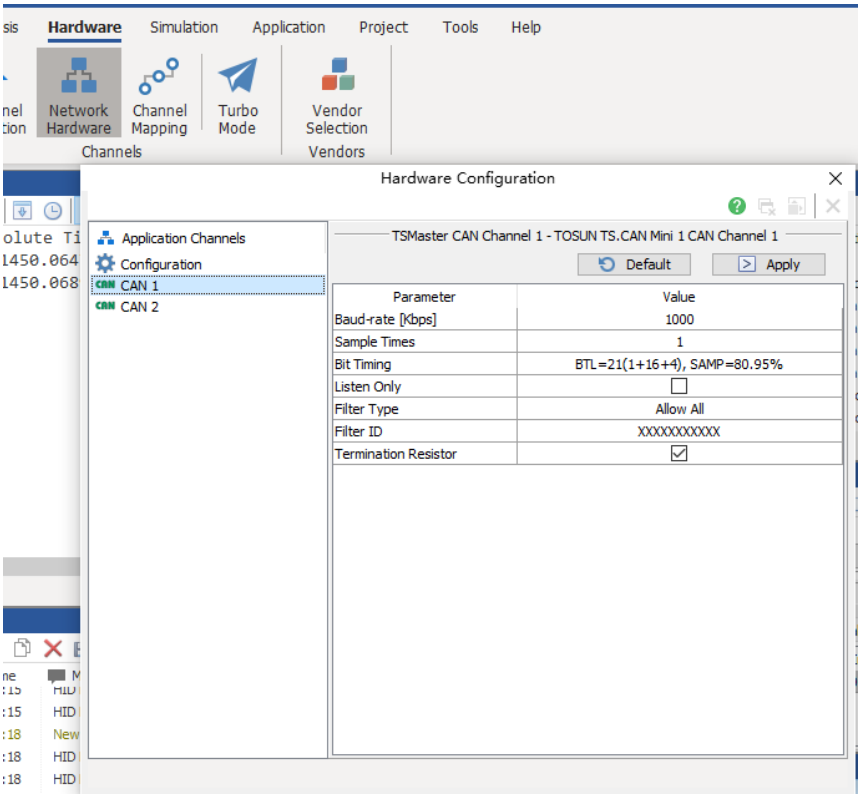
在选择硬件通道界面，用户可以选择插在电脑上的来自不同厂家的硬件的通道，选择过后，即完成了在 TSMaster 上的通道映射。



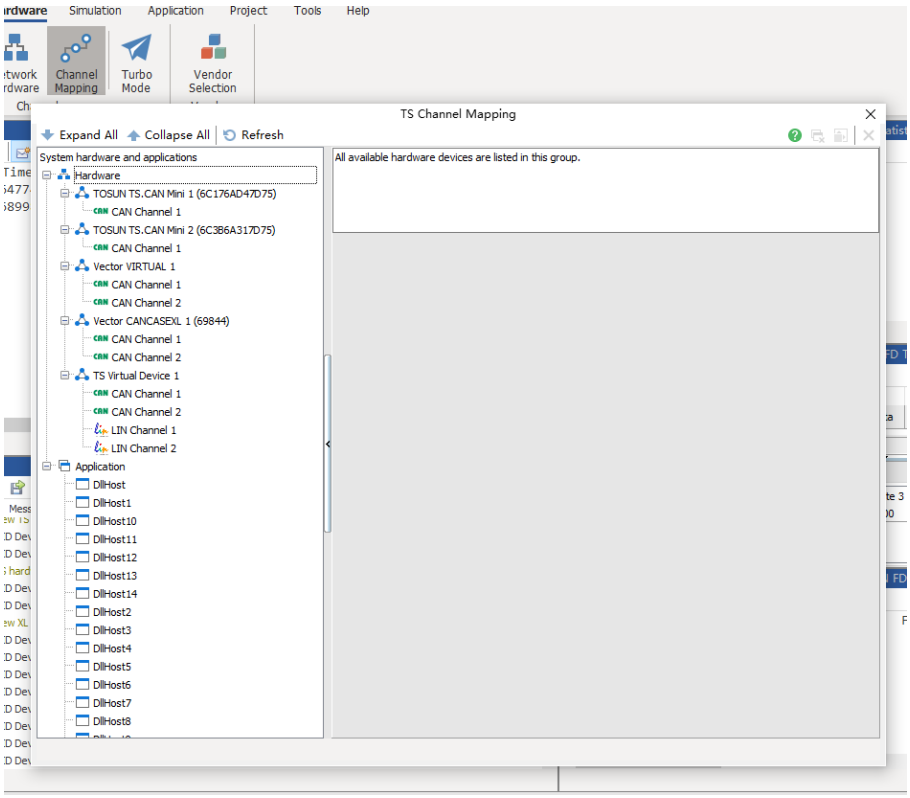
如上图所示：逻辑通道 CAN1，可以选择来自 TOSUN 的硬件，也可以选择来自 Vector CANCase 得到硬件。

1.4.4.2.配置参数

常规设置硬件通道参数：波特率，采样次数，位时间等属性值。



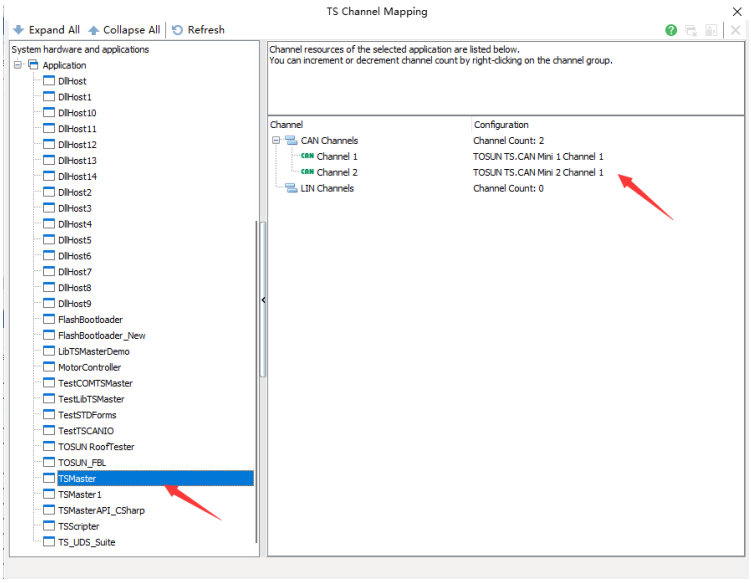
1.4.4.3.通道映射

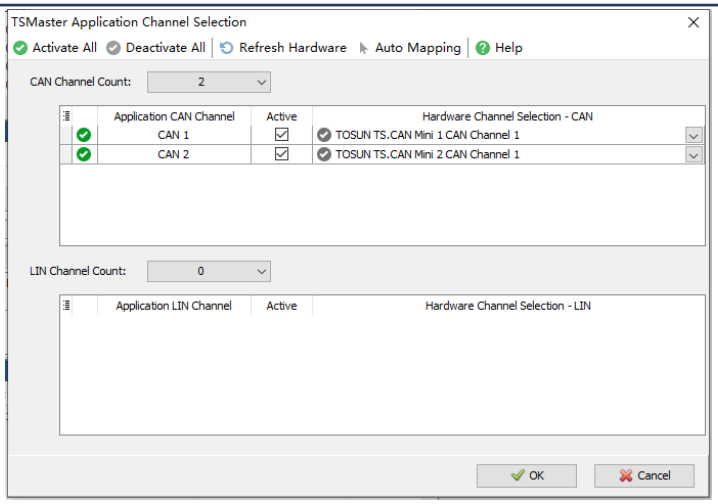


通道映射提供了一个图形化的映射管理器。主要包含两项：

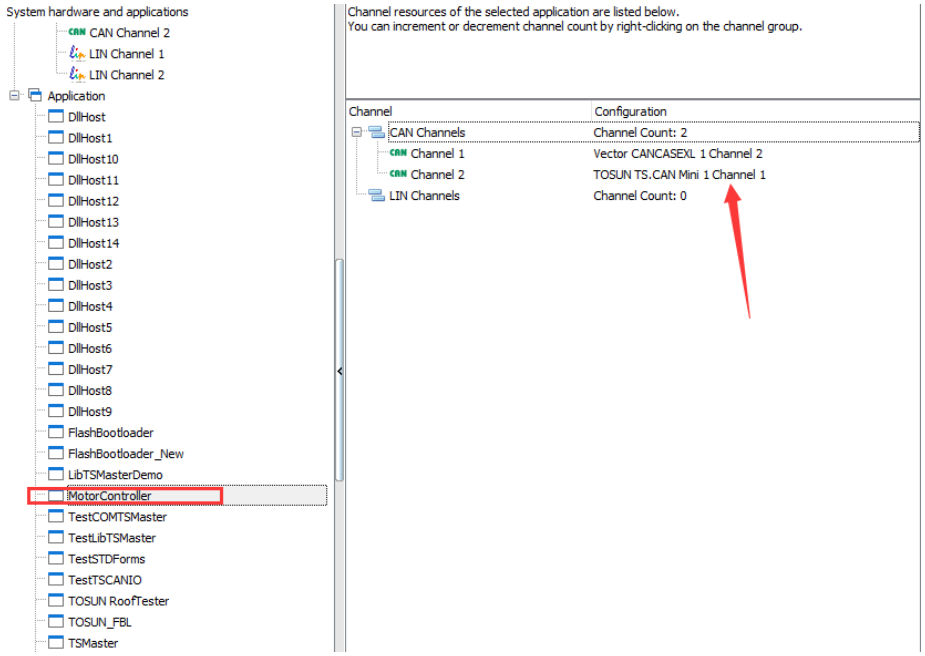
1. Hardware：呈现了当前插在用户电脑上的所有可用的硬件设备信息。如下所示：包含了 TOSUN 的 CANMini 硬件以及 Vector 的 CANCASEXL 硬件。
2. Application：呈现了所有基于 TSMasterAPI 底层的应用程序（当然也包含 TSMaster 自身）。

以 TSMaster 程序自身为例：





在映射管理器里面看到的参数设置跟 TSMaster 的通道设置里面是完全对应的。以 MotorController 应用程序为例：



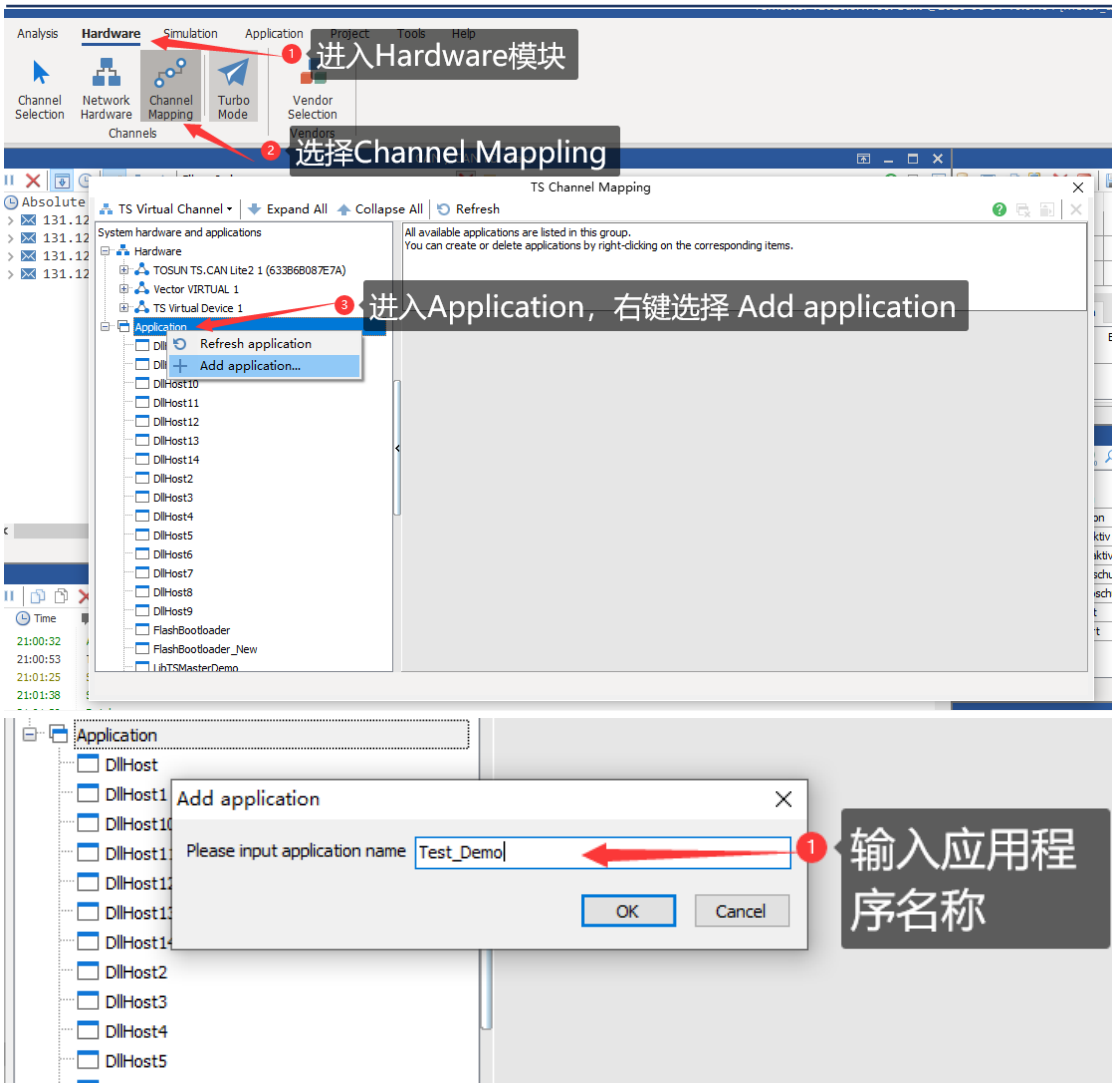
名称为 MotorController 的应用程序，使用了两路 CAN 通道，0 路 LIN 通道。CAN1 通道映射到了 Vector CANCASE XL 的通道 2 上面，CAN2 通道映射到了 TOSUN CAN Mini 通道上面。

1.4.4.4. 设置硬件通道

跟任何基于 TSMaster API 的应用程序一样，通道的映射可以采用图形化配置，也可以在连接应用程序之前，采用代码映射硬件。下面分别讲解：

1.4.4.4.1. 采用图形界面配置：

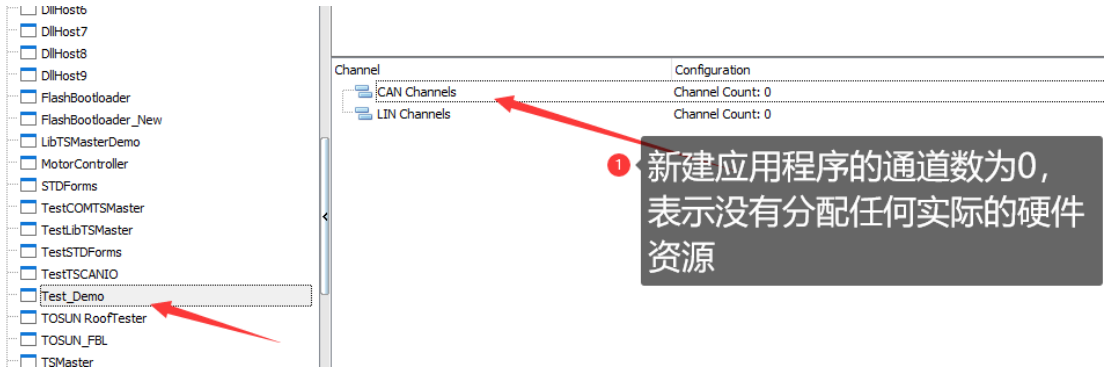
1. 首先添加应用程序：



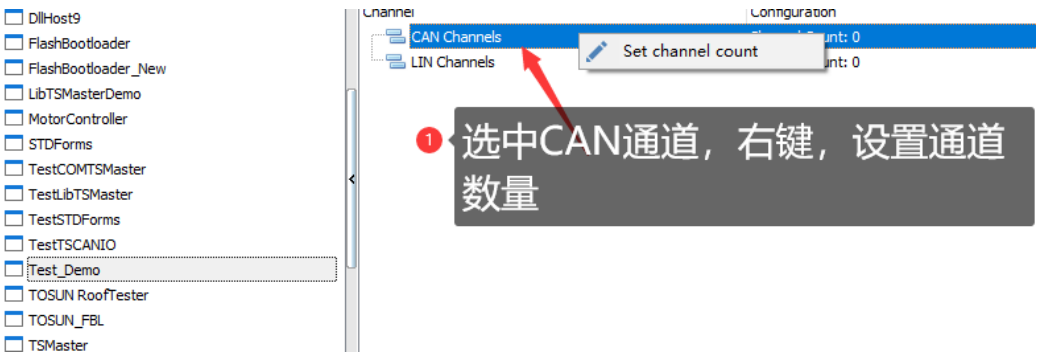
应用程序添加过后，就会一直留存在系统中，下一次使用的时候不需要再次创建。

2. 为该应用程序映射硬件

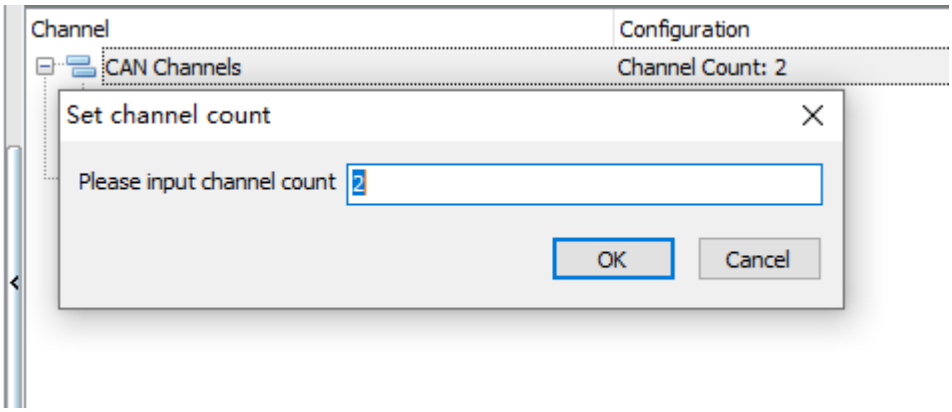
选中上一步添加的应用程序。如果是新建的程序，该程序对应的硬件通道 CAN Channels 和 LIN Channels 应该都等于 0.如下图所示：



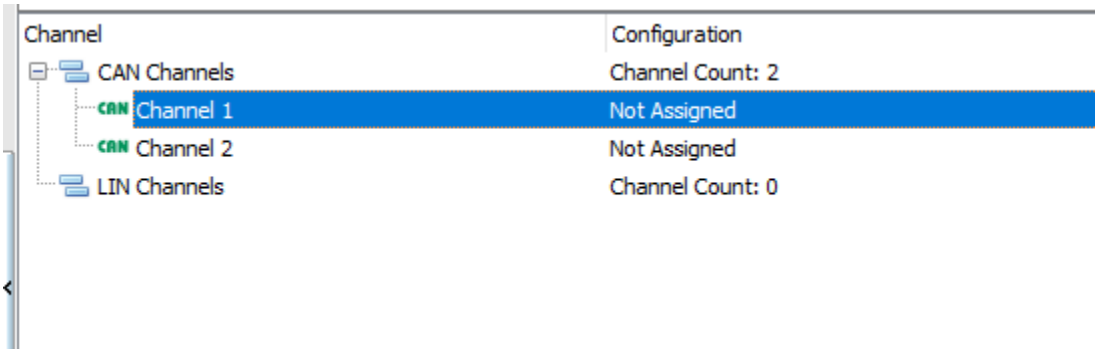
此时，选中该应用程序的 CAN Channels，右键设置通道数，如下所示：



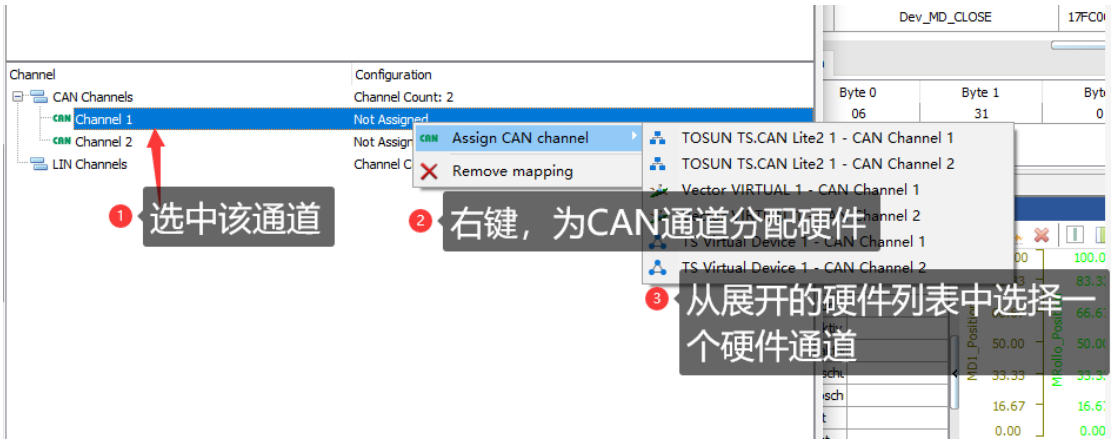
出现设置通道数界面，对于一个应用程序，最多同时支持 32 个通道，这里设置为 2 个通道，如下所示：



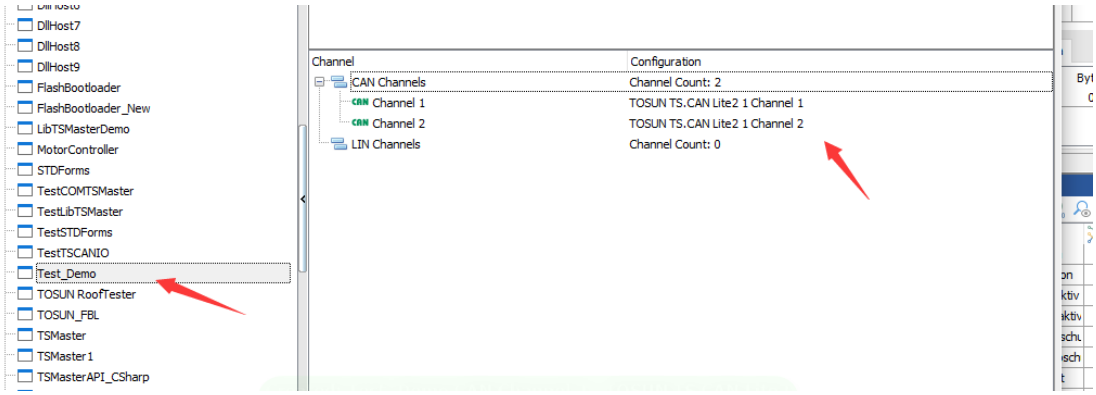
为应用程序创建两个 CAN 通道，但是没有指向任何实际的硬件。因此，通道都显示为没有分配，如下所示：



为通道映射具体的硬件通道，如下图所示：选中该通道->右键：Assign CAN Channel->从右边展开的硬件资源中选择一个通道，就完成了 CAN 通道的映射。



映射完成，如下图所示，该设备表示：名称为 Test_Demo 的应用程序，包含两个 CAN 通道，CAN 通道 1 使用 TOSUN TS.CAN Lite2 硬件的通道 1，CAN 通道 2 使用 TOSUN TS.CAN Lite2 硬件的通道 2.



1.4.4.4.2. 采用代码配置

```
void on_start_startMeasurement(void) { // on start event
//first: mapping channel
TLIBTSMapping m;
m.init();
sprintf(m.FAppName,"DatabaseOperationDemo");
m.FAppChannelIndex = 0;
m.FAppChannelType = APP_CAN;
m.FHWDeviceType = TS_USB_DEVICE;
m.FHWDeviceSubType = 3;  //3 means TS.CAN.Mini
m.FHWIndex = 0;
m.FHWChannelIndex = 0;
if(0==app.set_mapping(&m))
{
   log("Mapping channel success");
}
if(0 == app.configure_can_baudrate(0,500,false,true))
{
  log("CAN Baudrate on channel 1 hase been set to 500k bps");
}
if (0 == app.connect()){
  log("Application has stated, two CAN messages (5ms and 10ms) are being sent by timers");
```

如上图所示红框部分代码，完成的就是把 Application 名称为“DatabaseOperation”的应用的通道映射到 TSCANMini 的物理通道上的操作。

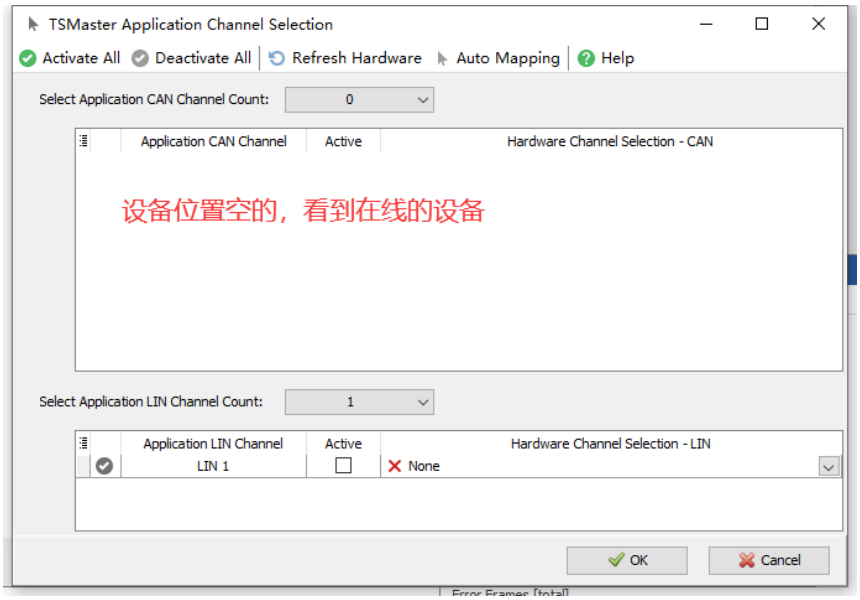
在映射完成后，配置了该逻辑通道的波特率，是否只听模式，是否使能内置终端电阻。

在完成所有设置后，调用 connect 连接 Application。在 connect 函数中，才实际初始化 CAN 工具硬件参数。在应用程序没有连接之前，CAN 工具必须保持静默状态，避免对 CAN 总线造成干扰。

1.4.5. 释疑

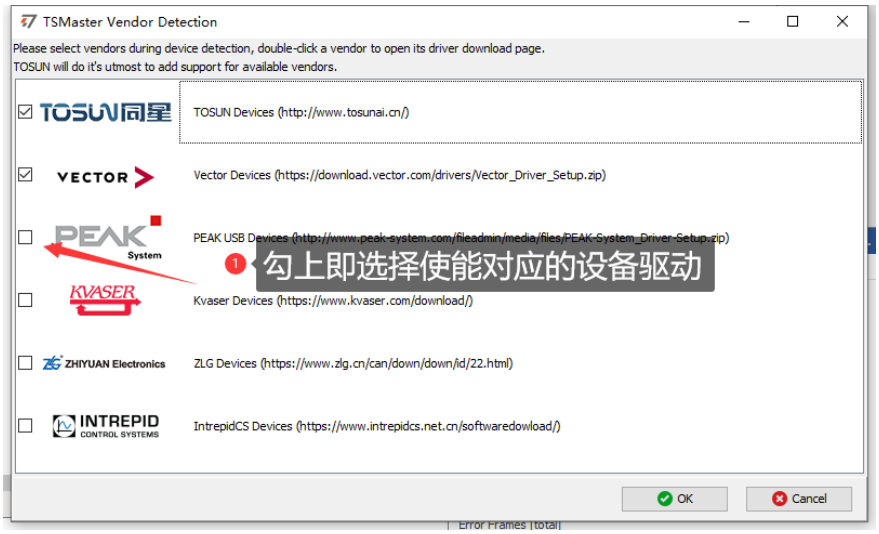
1.4.5.1. 找不到硬件设备

开发人员正确安装了设备驱动，也插上了硬件设备，但是 TSMaster“没有识别”该硬件设备，如下图所示，这种情况一般是下面两种原因：



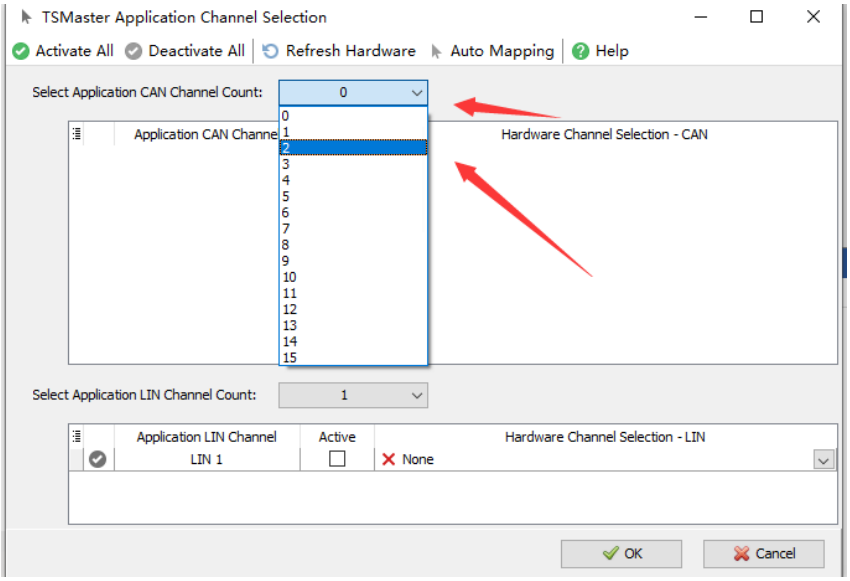
1. 没有选择设备驱动。

TSMaster 是一款开放的软件平台，兼容市面上主流的比如 TOSUN，Vector，VehicleSpy，Kvaser，Peak 以及 ZLG 等 CAN 工具。但是为了减少程序启动依赖项，默认只加载 TSOSUN 和 Vector 的 CAN 驱动。如果第一次使用，需要到硬件配置中使能加载对应厂商的驱动，如下图所示：

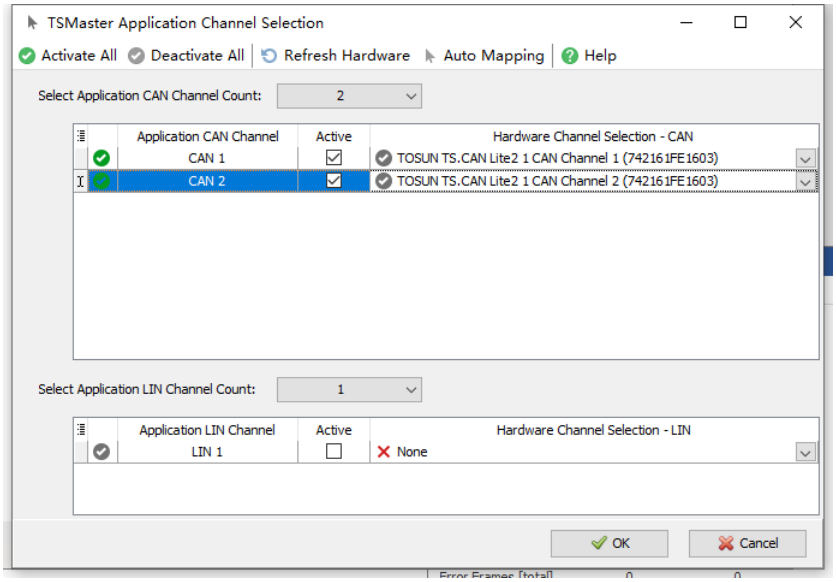


2. 没有设置通道数

如果既正确安装了驱动，又使能了该硬件驱动，还是无法在硬件面板中看到设备。请检查一下是否设置了通道数。如果 CAN Channel Count = 0，当然无法显示在线硬件。



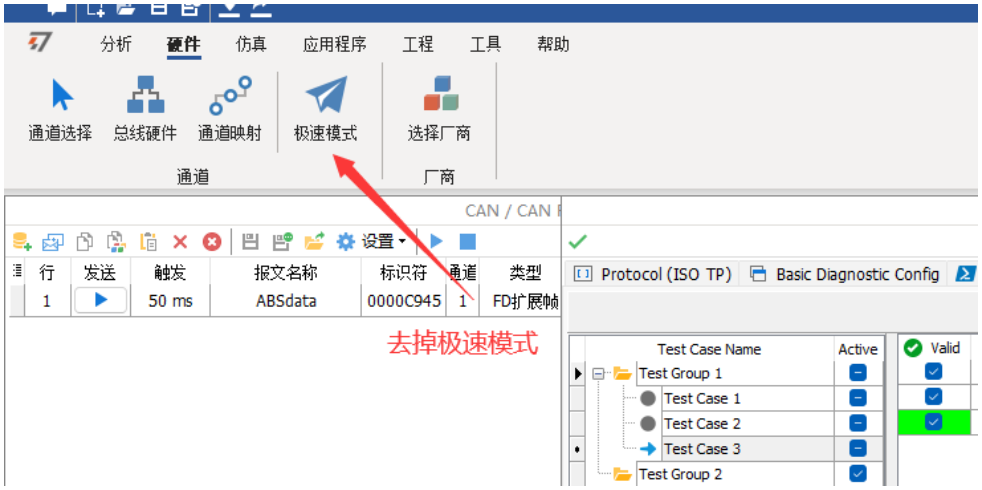
解决办法，设置通道数，并把通道映射到实际的 CAN 板卡物理通道上。原理见“硬件通道映射”章节，如下所示：



1.4.5.2. CPU 占用率过高

TSMaster 中设计了一个加速模式。在加速模式下，TSMaster 会占据一个 CPU 核心进行运算，因此会造成较高的 CPU 占用率。尤其是对于一些 CPU 核心比较少的处理器，可以看到的 CPU 占用率一下子就上去了。解决办法是：

在硬件配置界面，去掉加速模式（按钮按压下去，则是开启该模式），如下图所示：


分析
硬件
仿真
应用程序
工程
工具
帮助
通道选择
总线硬件
通道映射
极速模式
选择厂商
通道
厂商
设置
行
发送
触发
报文名称
标识符
通道
类型
1
50 ms
ABSdata
0000C945
1
FD扩展帧
去掉极速模式
Protocol (ISO TP)
Basic Diagnostic Config
Test Case Name
Active
Valid
Test Group 1
Test Case 1
Test Case 2
Test Case 3
Test Group 2

1.5. Measurement SetUp

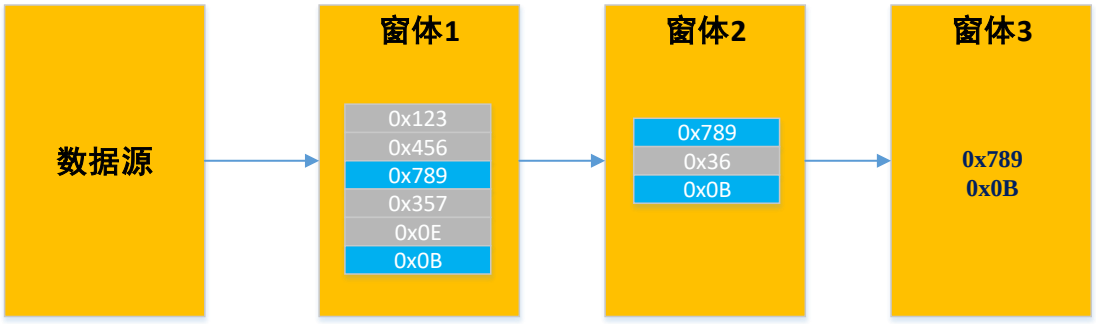
1.5.1. 功能介绍

Measurementsetup 窗体主要包含三个功能：

- 提供一个面板，用户能够快速创建需要的功能模块窗体。
- Measurement setup 模块汇总了整个工程内所有窗体。用户可以通过 measurement setup 快速访问目标窗体。
- 通过 measurement setup 创建数据流组合，实现数据流过滤。

1.5.2. 数据流过滤

TSMaster 提供了 Measurement SetUp 窗口。在这个窗口里面可以通过组合数据流方向，实现过滤的效果。其基本思路如下：数据流流过一个窗体，这个窗体内包含的数据才允许通过，其他数据不允许通道，如下图所示：



1. 窗体 1 中包含的数据有：0x123 等，只有 ID 包含在其中的数据才允许通过进入到窗体 2 中；
2. 窗体 2 中包含的数据有：0x789 等，只有 ID 包含在其中的数据才运行通过进入下一个窗体 3 中；
3. 经过前面两个窗体的过滤，最后到达窗体 3 的报文 ID 只能是 0x789 和 0x0B；

在 Measurement Setup 面板中，窗体过滤能力分为三类，通过颜色进行标识：

1. 白色窗体：允许所有数据通过。
2. 绿色窗体：允许满足条件的数据通过。
3. 红色窗体：禁止所有的数据通过。

如下图所以：

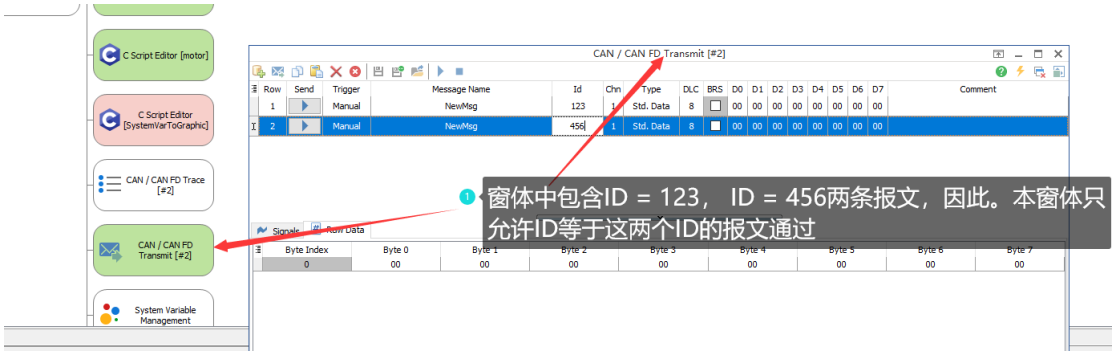
允许所有数据通过/禁止的窗体很好理解，下面详细讲解绿色窗体的实现原理。

1.5.2.1. 允许部分数据通过：

本质：提取该窗体中包含的报文的信息（如包含的报文数量，分别的 ID 以及通道编号），然后基于**通过+报文 ID** 进行过滤。

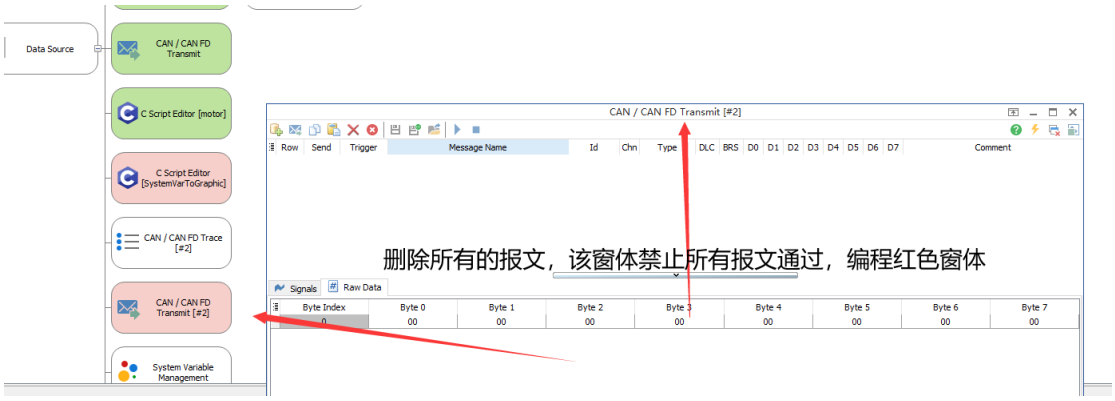
1.5.2.1.1. 以 CAN/CAN FD Transmit 窗体为例：

- 当窗体中包含报文的时候，如下所示：

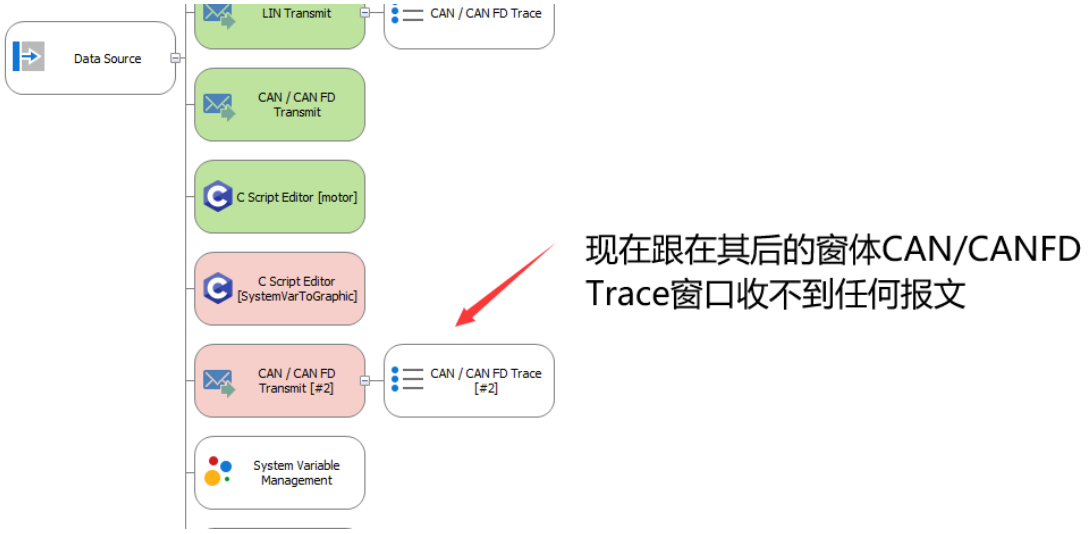


此时：跟着后面的窗体能够收到：来自通道 1 的报文 ID 等于 0x123 或者等于 0x456 的报文。

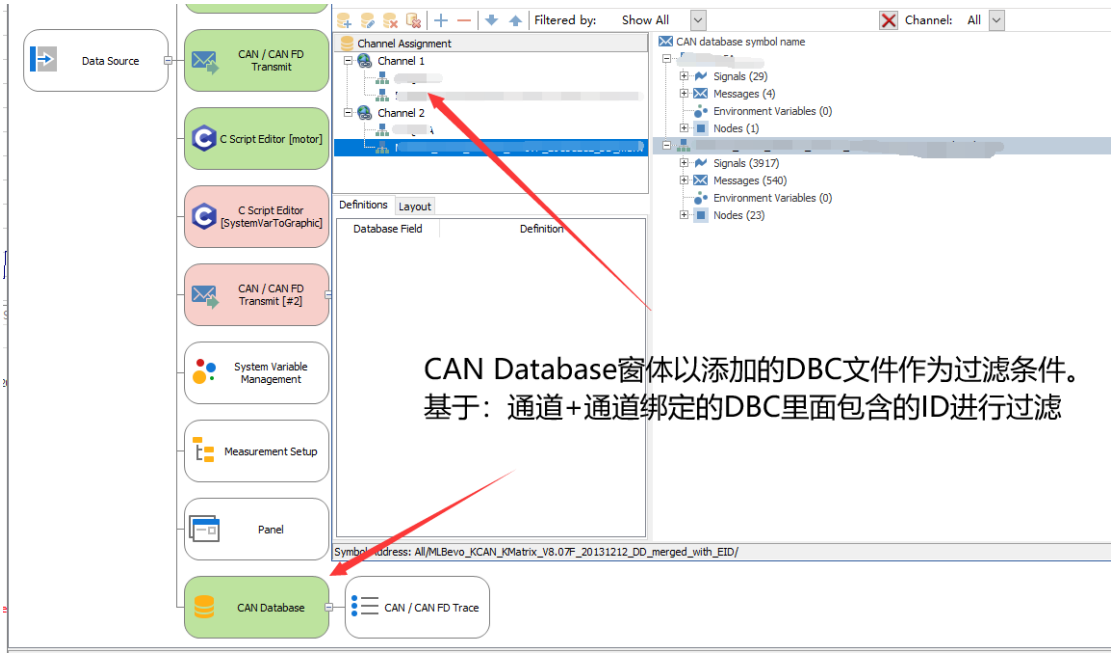
- 把发送窗体中的报文全部删除过后，如下所示：



➢ 跟在该窗体后面的模块，将收不到任何的报文，如下所示：



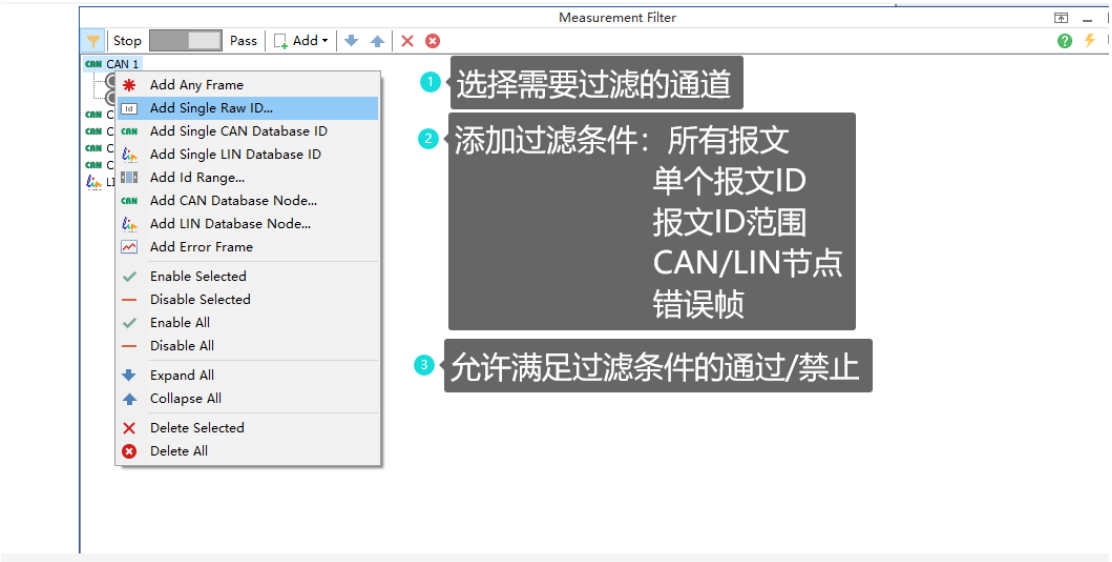
1.5.2.1.2. 以 DBC 窗体为例：



如上所示：报文的通道和 ID 包含在 CAN DataBase 窗体添加的数据库中，那么该报文就会被允许通过该窗体，后面的窗体才能够收到该报文。没有包含在数据库中的报文，则被禁止通过此窗体。

1.5.2.2. Measurement Filter 模块

Measurement Filter 作为专用于过滤的模块，提供了更加灵活的窗体过滤能力。如下所示：



1.5.3. 过滤条件的使能/失效

在 Measurement Setup 窗体中配置了过滤条件，TSMaster 也提供了灵活的机制使能/失效这些过滤条件。也就是通过拖拽的方式，给窗体之间添加关联，就能够使能/失效这些过滤条件。

1.5.3.1. 使能过滤条件：

1.5.3.2. 失效过滤条件：

不用修改过滤条件窗体，而是把这些窗体从过滤窗体后面拖拽开即可实现失效过滤条件。

1.5.4. 窗体缩放

Measurement Setup 窗体支持自动缩放和普通显示功能，如下所示：

- 自动缩放模式：模块根据父窗体尺寸自动调节自己的大小。优点：所有的模块都能显示在面板上；缺点：当模块添加太多的时候，模块尺寸会偏小。
- 普通显示模式：模块始终保持设计时的尺寸大小。优点：模块的尺寸保持不变，不会因为添加过多模块造成尺寸变小，不方便操作；缺点：添加过多模块的时候，一部分模块会被窗体边缘掩盖，需要通过拖拽滑块才能够显示。

1.5.5. 应用实例：

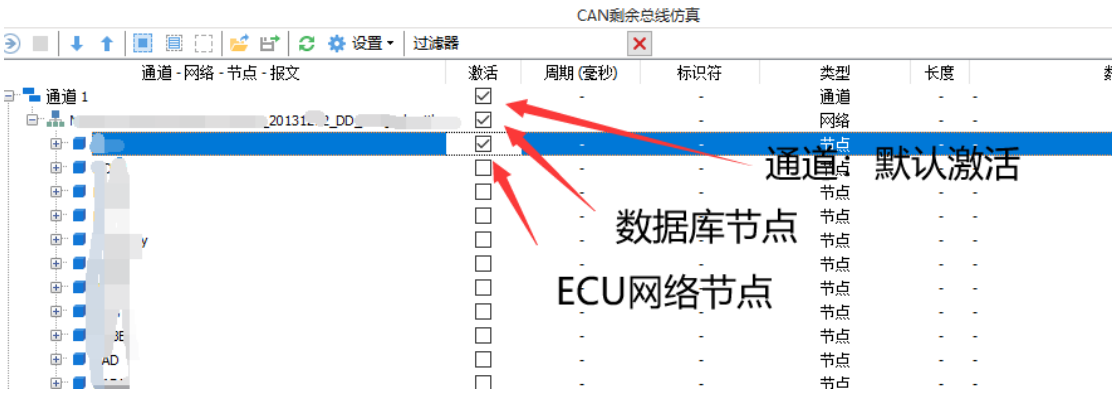
只需要接收通道 1 的

1.6. RBS

1.6.1. RBS 仿真

RBS 全程是：residual bus simulation，也就是所谓的剩余总线仿真。主要是基于车载网络数据库，如 CAN/LIN/Flexray/以太网数据库，仿真该网络内部各个节点的通讯行为。

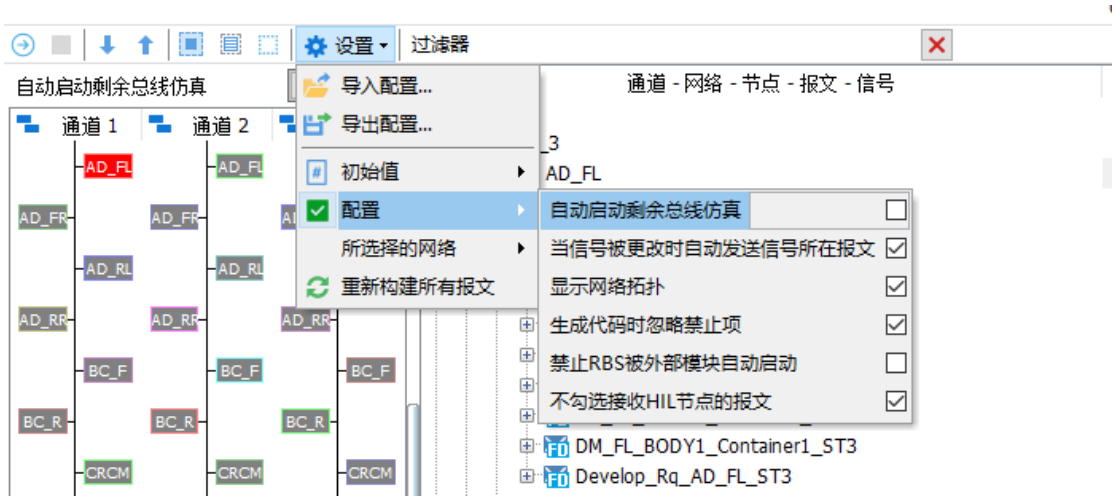
在 TSMaster 中，用户想启动 RBS 的话，通过路径：仿真->RBS，选中对应的 RBS 模块，就进入对应的 RBS 参数配置界面。如下图所示：



RBS 仿真引擎是独立模块。如果用户勾选了 RBS 通道->数据库节点->控制器节点。那么启动瞬间，TSMaster 会启动该节点仿真。如果上述几个勾全部都勾上了，表示要选择通道 x 中的数据库节点进行仿真，所以启动瞬间就会开启 CAN RBS 仿真，总线上也会出现报文数据。

1.6.2. 配置项：

TSMaster 的 RBS 模块，主要包含如下的配置项：



- 自动启动剩余总线仿真：如果使能，则在连接应用程序的时候，自动启动剩余总线仿真模块。
- 当信号被更改时自动发送信号所在报文：如果使能，则信号修改的时候立即发送该报文。
- 显示网络拓扑：在界面上显示网络的网络拓扑示意图。
- 生成代码时候忽略禁止项
- 禁止 RBS 被外部模块自动启动：当 RBS 模块被 Panel 等调用的时候，如果不禁止此选项，则即使没有主动打开 RBS 模块，RBS 模块也会被 Panel 等使用到 RBS 的外部模块启动起来。如果勾选了此选项，则 RBS 模块即使被外部模块调用了，也必须要用户主动来启动才可以。
- 不勾选接收 HIL 节点的报文

1.6.3. 释疑

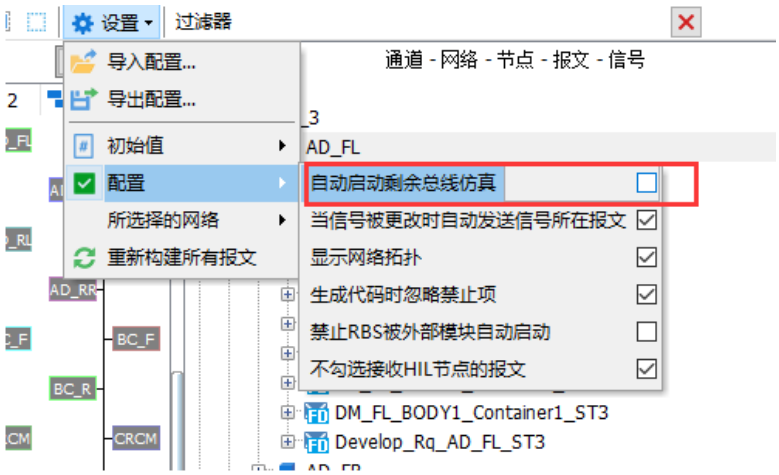
1.6.3.1. 为什么 RBS 模块会被自动激活？

1.6.3.1.1. 用户配置了自动启动

用户在设置选项中选择了自动启动仿真节点，并且选择了需要仿真的节点。则程序在连接的时候就会自动启动对应的网络节点仿真。用户就能在 Trace 中观测到仿真的报文。

解决办法：

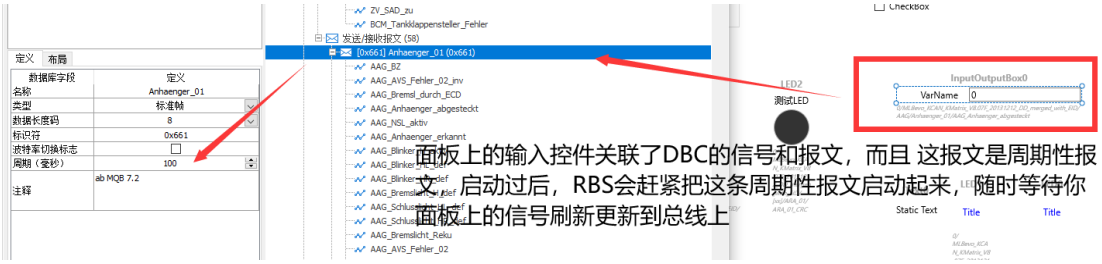
在配置中选择不予自动启动 RBS 仿真，如下图所示：



1.6.3.1.2. 通过 Panel 激活了 CANRBS 仿真

这种情况是最隐蔽，用户比较难找到为什么启动瞬间就发送报文的原因的。首先有几个前提：

1. 面板关联到 CAN 报文的时候，是基于 CAN RBS 模块运行的。这样真实仿真总线环境。
2. 面板上添加了输入控件，如果输入控件跟 DBC 里面的周期性报文相关联，其启动 Panel 的时候，CANRBS 需要赶紧把这条周期报文准备好。因为用户在 Panel 上对该周期信号的更新，CANRBS 需要及时的更新和处理这条报文值。



为什么及时用户没有设置自动启动 RBS 模块，也没有手动启动该模块的情况下，RBS 模块会被外部模块调用起来呢？

因为外部模块的运行要基于 RBS 模块，如果 RBS 模块没有跟着被自动启动起来，用户

不知情的情况下会觉得是否软件哪里有异常，因此默认情况下，TSMaster 中如果外部模块比如 Panel 等使用了 RBS 模块的话，Panel 运行的时候会自动启动 RBS 模块。

解决办法：

设置 RBS 模块不允许被外部模块自动启动。如下所示：

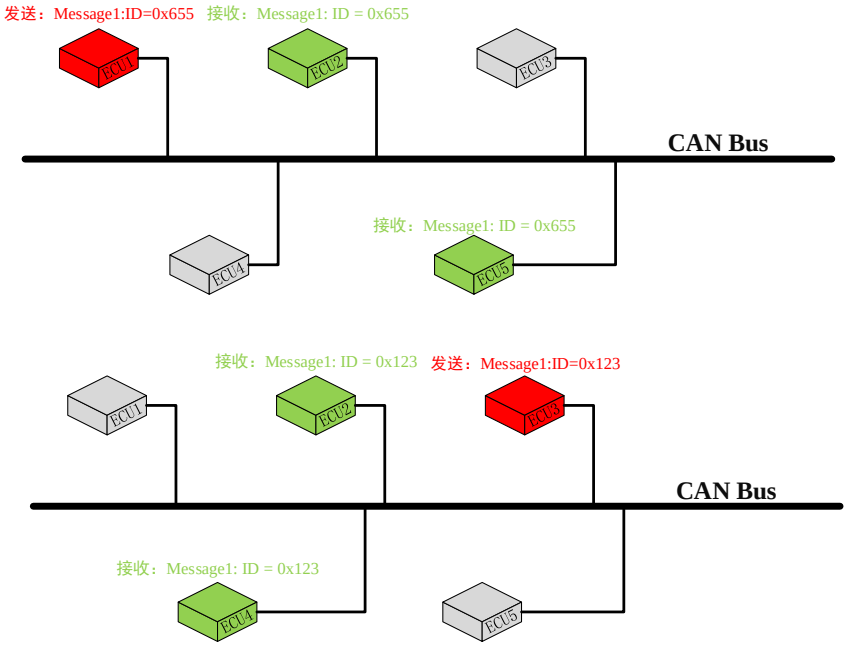
用户谨慎关掉此选项。否则 Panel 打开的时候，对应的 RBS 模块没有打开的话就比较奇怪。

1.6.3.2. 为什么读取/写入 RBS 节点信号无效

CAN 的 RBS 仿真根据网络数据库中的网络配置进行仿真，其仿真逻辑是非常严谨的。如下图所示，该 CAN 网络里面定义了 5 个 ECU 节点，从 ECU1 到 ECU5。每一个节点有自己发送和接收的 CAN 报文。因此，对于同一条报文来说，其有对应的发送节点，也就有对应的接收节点。正常来说，在 CAN 网络中，报文只有一个发送节点，有一个或者多个接收节点。如下图所示：

- 对于 ID = 0x655 的报文。其他发送节点为 ECU1，接收节点为 ECU2 和 ECU5，对于其他两个节点，这条报文对他们是无意义的，直接过滤掉。
- 对于 ID = 0x123 的报文。其发送节点为 ECU3，接收节点为 ECU2 和 ECU4，对于其他两个节点，这条报文对他们是无意义的，直接过滤掉。

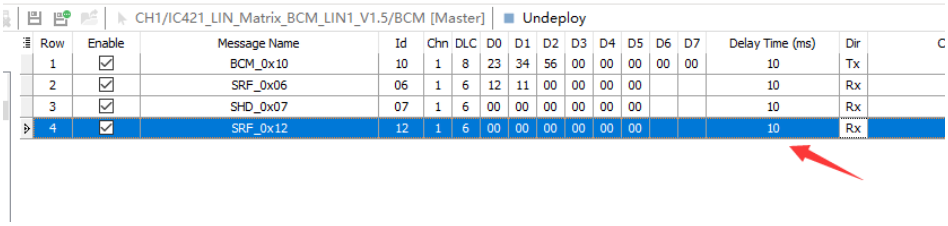
基于如上所示的网络规划逻辑。比如对于报文 ID=0x655 报文里面的信号，在 RBS 仿真的过程中，如果用户修改 ECU2 或者 ECU5 里面 ID = 0x655 报文里面的信号，他们作为接收节点，即使修改了该信号，该信号也没有机会被发送到总线上，因此就会出现写入信号过后无效的情况。



当 RBS 仿真的时候，如果某一个 ECU 节点没有激活，则不会更新该 ECU 节点内部的总线数据。因此，当读取该节点内部的信号的时候，就不会是当前总线的实际有效数据，就造成读取的信号值无效。因此读取 RBS 节点信号的时候，要确定该节点是否在 RBS 总线上面激活了。

1.7. 发送窗口

1.7.1. LIN 发送窗口



DelayTime：是 LIN 报文之间的时间间隔，是从报文开始发送的瞬间开始计时，不是该报文发送结束过后等待的时间间隔。因此，该时间不能设置过小。如果设置太小，会造成前面一帧数据还未发送成功，后面一帧数据就开始发送，前面的数据就被覆盖掉。如果全部设置过小，就造成所有的报文还未发送成功就被覆盖掉，一方面总线上时钟没有完整的报文发出来；另一方面上位机 LIN Trace 窗口将看不到报文。

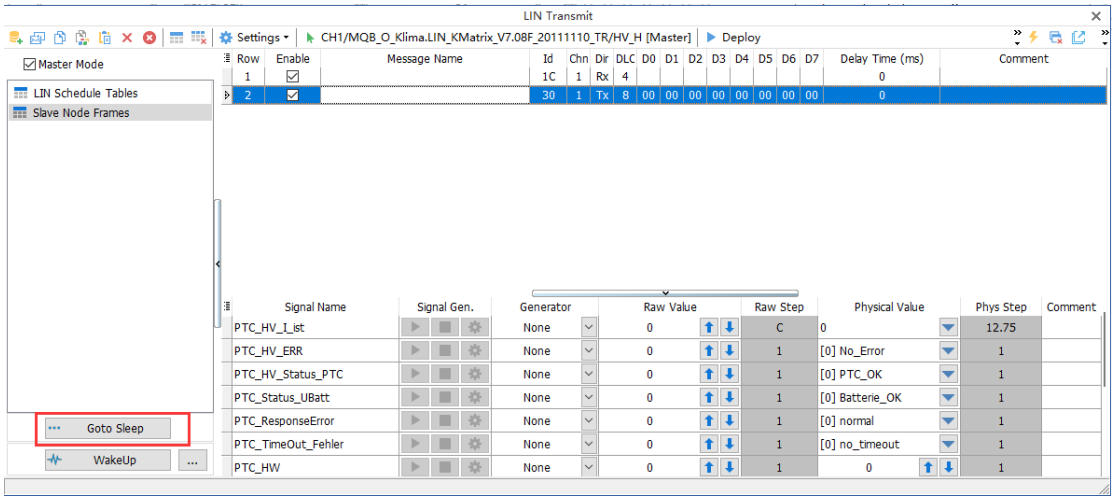
1.7.1.1. 发送睡眠帧

根据 LIN2.1 协议规范，睡眠帧必须由主节点发起，睡眠帧内容如下：

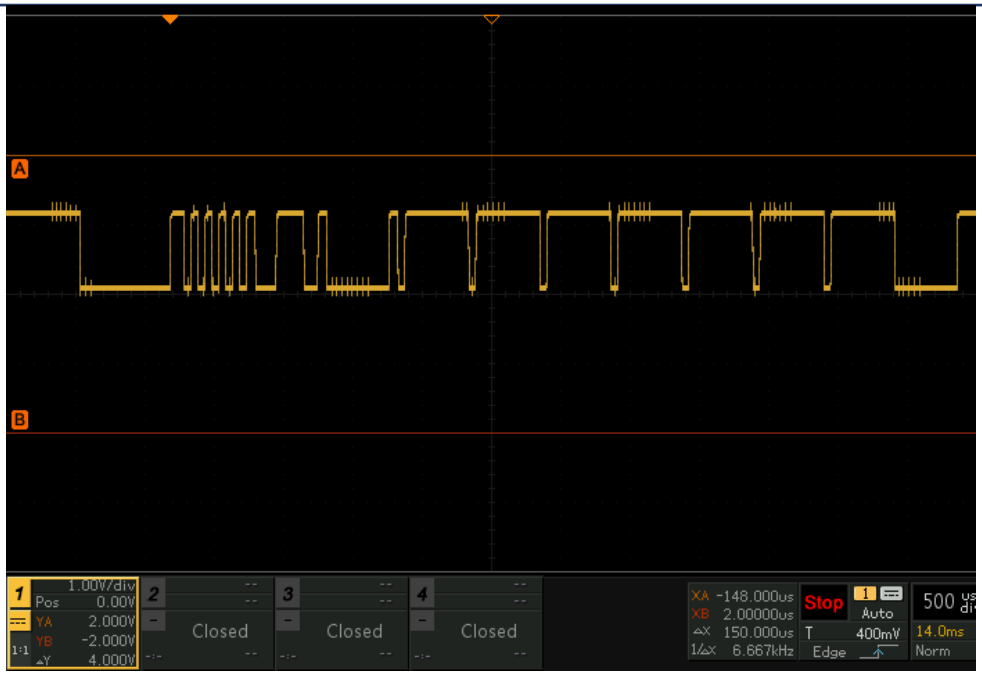
data1	data2	data3	data4	data5	data6	data7	data8
0	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF

Table 2.1: Go to sleep command

在 TSMaster 的 LIN 发送界面中，进入 LIN Transmit 发送界面，如下所示：



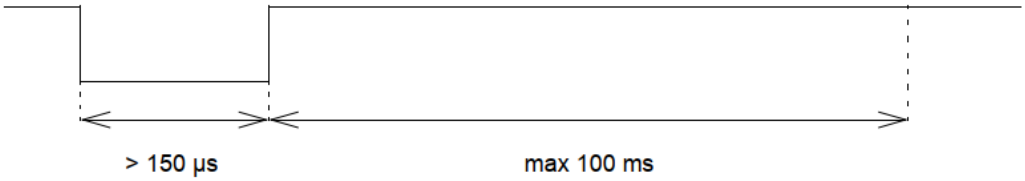
选择节点和通道，然后点击 GotoSleep 按钮，通过示波器查看报文如下所示：



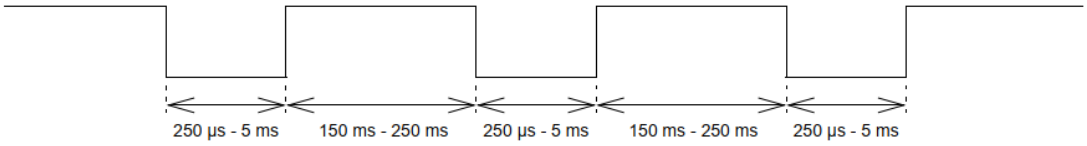
从节点收到上述报文后，进入睡眠状态。

1.7.1.2. 发送唤醒帧

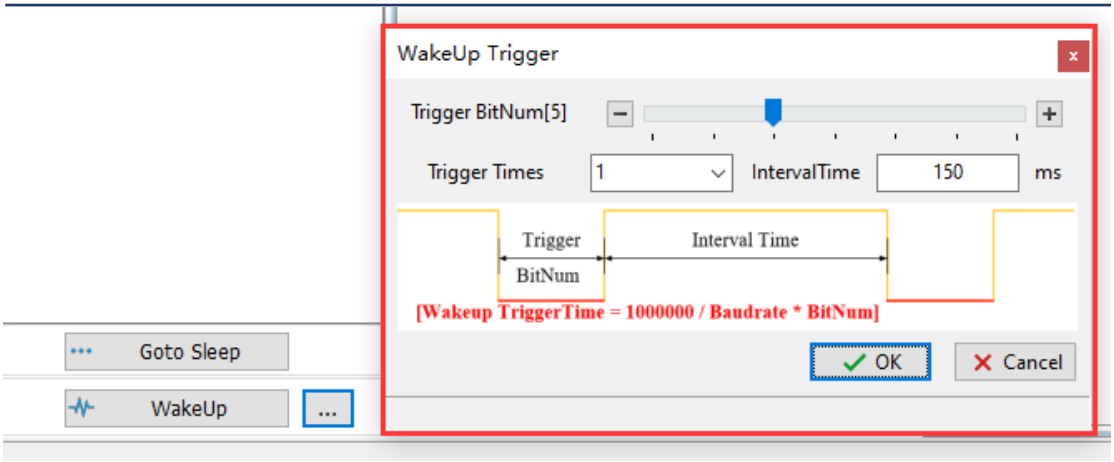
根据 LIN 2.1 规范定义，当节点处于睡眠状态的时候，收到超过 150us 长度的波形，则被唤醒，然后进入初始化状态，并重新进入工作状态。以波特率 20k 的节点为例，当他收到超过 150us 的波形的时候，则被唤醒，并进入工作状态。



在实际进行唤醒测试的时候，要实现唤醒报文长度可调，而且有的节点需要间隔一段时间后多次发送唤醒报文才能够唤醒，如下图所示：

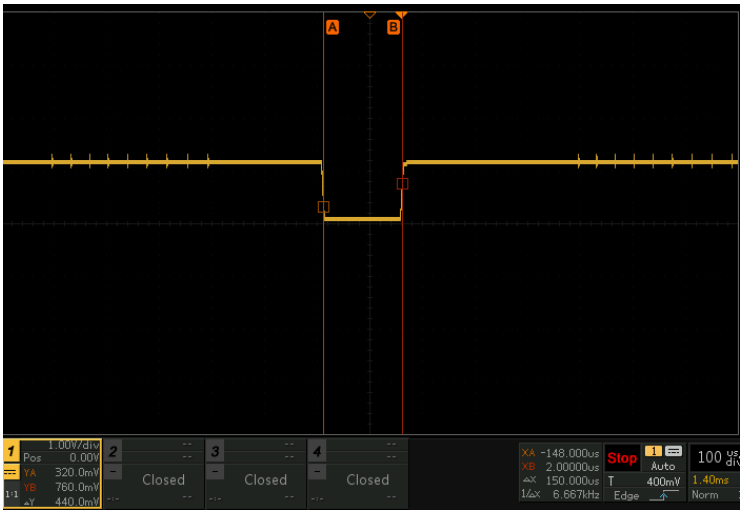


为了满足上述要求，TSMaster 提供了 WakeUp 信号配置参数配置模块，如下图所示：

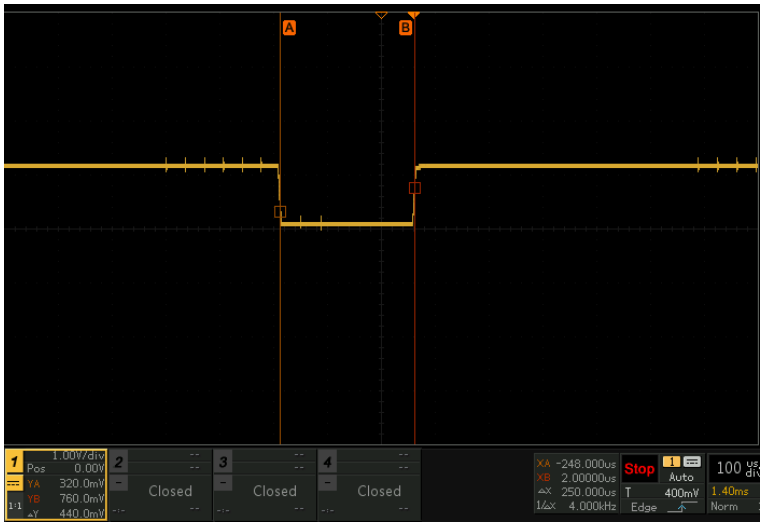


主要包含以下配置参数：

- Trigger BitNum: 唤醒报文触发电平长度 = 1000000/Baudrate *(BitNum). 比如波特率为 20k，设置 BitNum = 3 的话，则触发电平宽度为 150us。如下图所示：

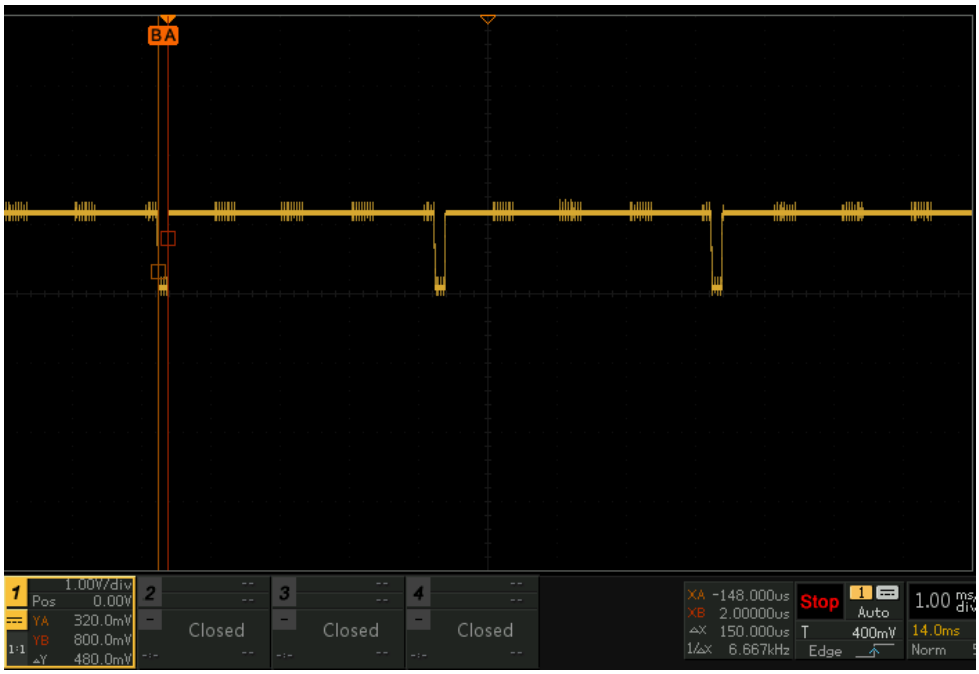


当设置 BitNum = 5 的时候，则触发电平宽度为 250us，如下图所示：



当设置 Trigger Times 等于 3，触发间隔时间（Interval Time）= 3ms，则 TSMaster 会发出多帧唤醒报文，报文间隔等于 3ms，如下图所示，先发送唤醒报文（250us 电平），然

后等待 3ms 过后，再次发送唤醒报文，循环发送了三次：

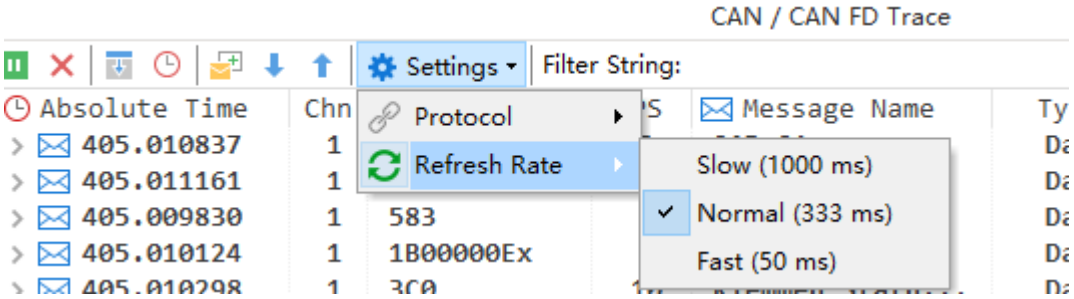


连续发送 3 帧唤醒信号

1.8. Trace 窗口

1.8.1. 设置显示刷新率

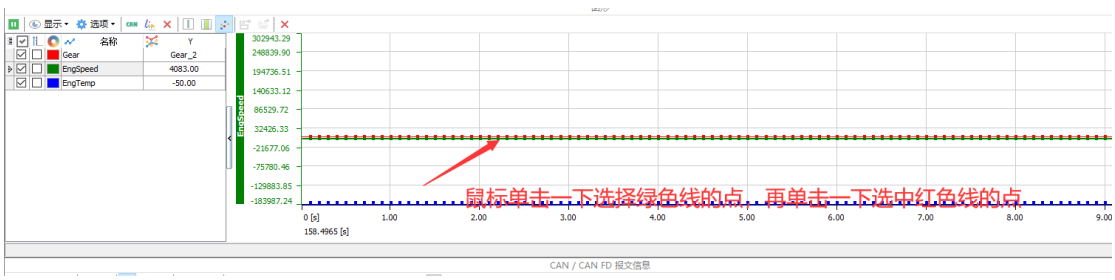
为了降低软件 CPU 占用率，Trace 窗口提供了几种显示刷新率让用户选择。如下所示：



对于一些老式电脑，可以选择较低刷新率，降低电脑负担。

1.8.2. 靠近的曲线，选点功能

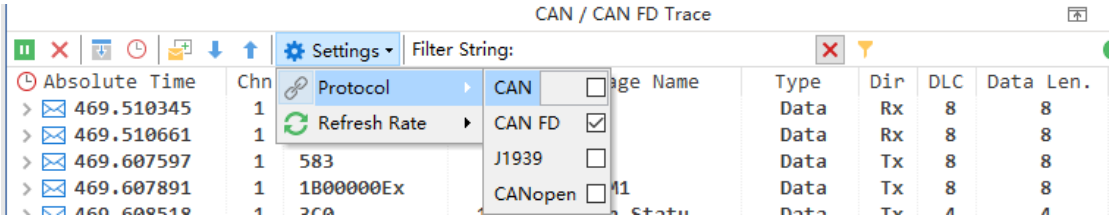
TSMaster 的 Trace 窗口，如果几条曲线非常靠近，鼠标不易选中的情况下，TSMaster 提供了单击鼠标切换曲线的功能。



1.8.3. 设置显示报文格式

CANTrace 支持按照 CAN，CANFD，J1939，CANOpen，15765-2，15765-3，CCP，XCP 等格式进行显示，用户只需要切换到对应的显示格式即可，如下所示：

对于模块的设置，路径都是：Setting->子选项->子选项的模式。



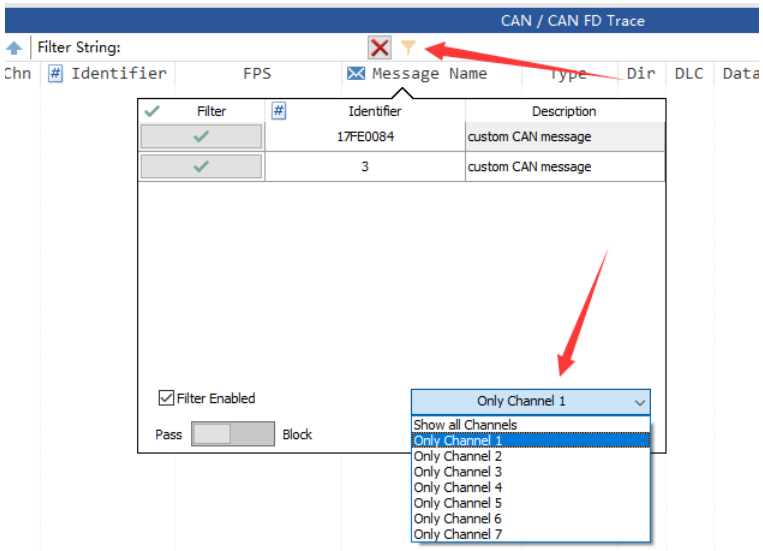
1.8.4. 报文过滤

在 Trace 窗口，TSMaster 包含三种类型的过滤:基于数据通道；基于报文 ID；基于信号值范围。

1.8.4.1.基于报文通道：

使用多路 CAN 通道设备的用户，可以选择只查看自己关心的通道的报文。设置方式如下：

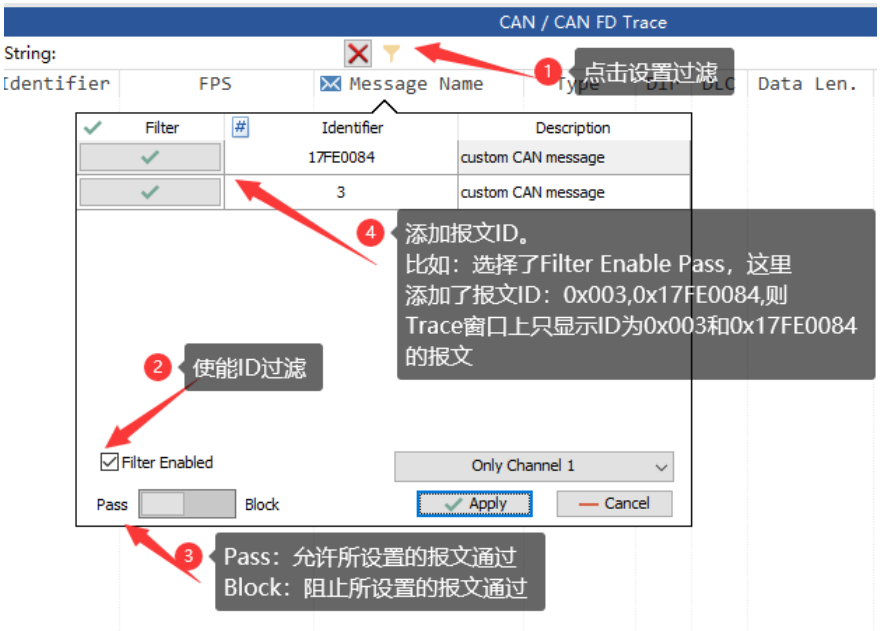
过滤->选择通道，可选通道类型如下所示：



如果选择 OnlyChannel1，则只有来自通道 1 的数据会显示到 Trace 窗口上。

1.8.4.2.基于报文 ID 过滤：

配置步骤如下所示：

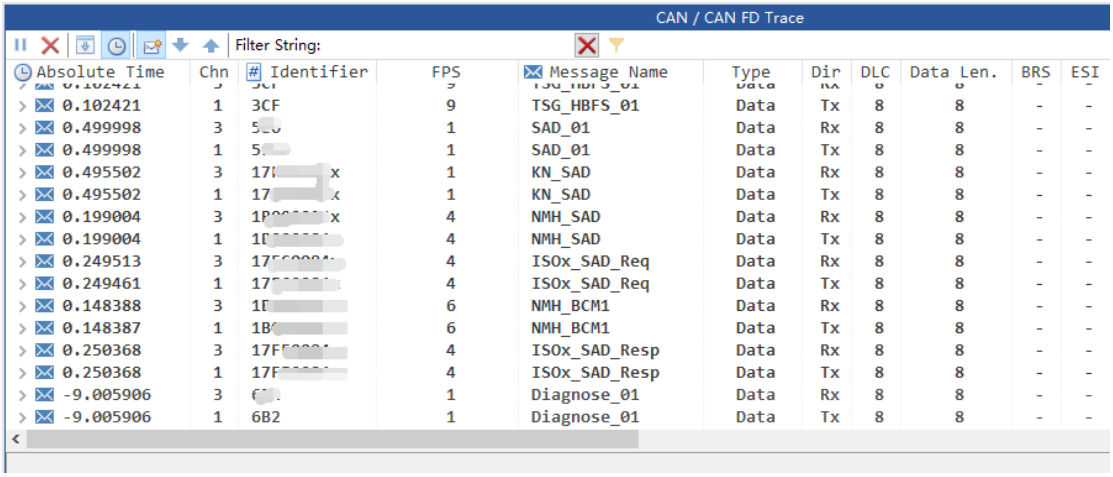


1.8.4.3.基于过滤字符串（FilterString）过滤

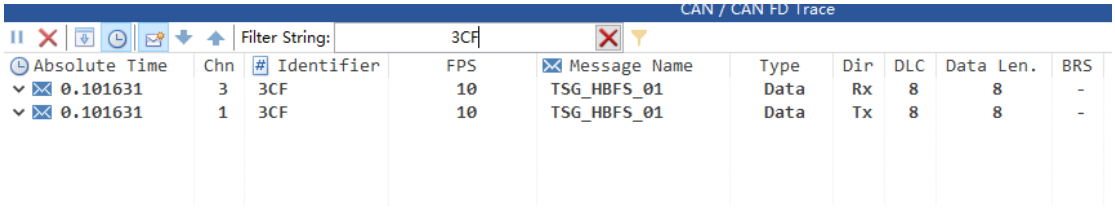
在过滤字符串中可以直接输入 ID 值，则 Trace 窗口只留下跟该 ID 对应的报文；或者输入信号名称，则 Trace 窗口只留下跟该信号相关联的报文。详细如下：

- FilterString 输入 ID 值过滤：

  原始报文窗口如下，里面有多条 ECU 通讯的报文，下图所示：



在 Filter String 窗口中，输入 3CF，则 Trace 窗口只剩下 3CF 报文，如下所示：

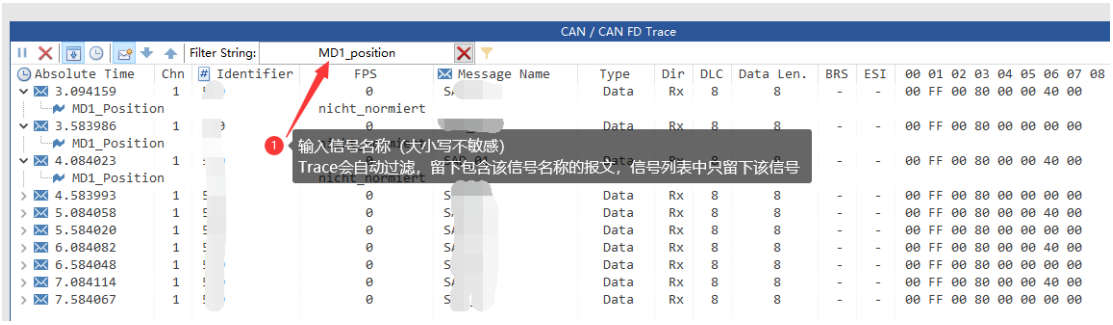


FilterString 窗口支持模糊查询，如果单独输入数值 3，则 Trace 窗口只剩下 3 开头的报文。

注意：这里是直接输入报文 ID 值，不需要其他额外的描述。

- FilterString 输入信号名称过滤：

基于信号的前提是加载了数据库，因为有 DBC 数据库，才有信号的概念。这种过滤方式方便研发公司更加细致的查看自己关心的通讯信号和报文。过滤方式非常简单，如下所示：



注意：

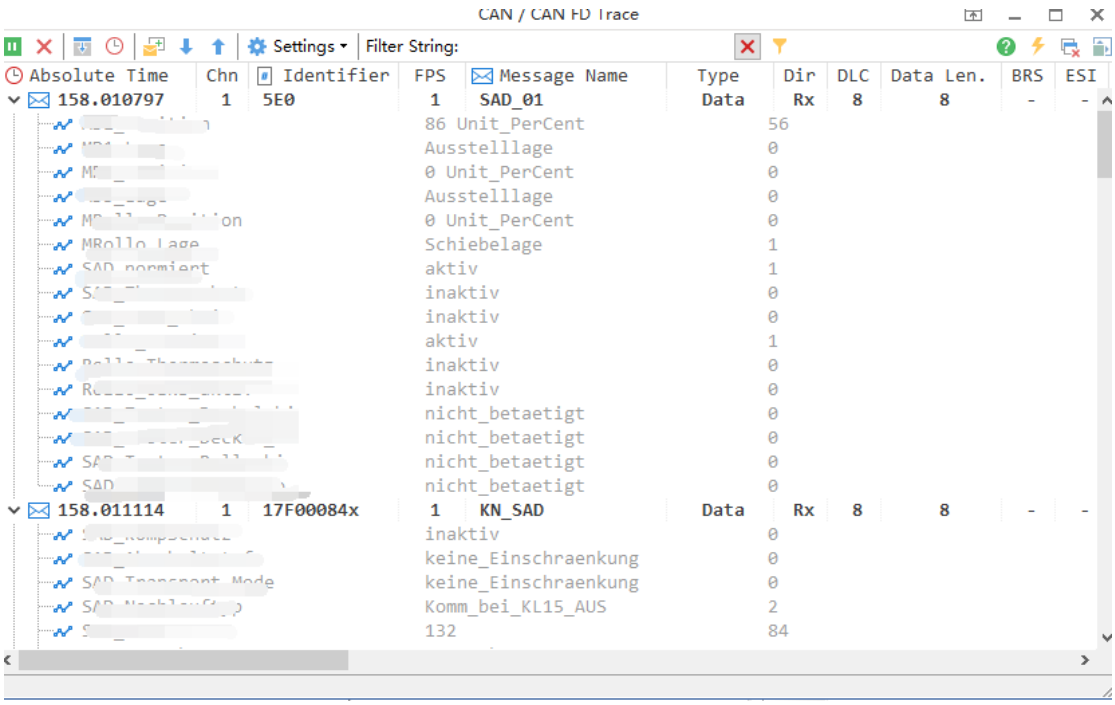
1. 每一个 Trace 窗口是独立的存在，可以针对不同的 Trace 窗口设置对应的过滤条件。互相之间不影响。
2. 总线系统已经工作，Trace 窗口无显示或者显示异常，可能是之前设置的过滤条件屏蔽了相关的显示，请检查设置。
3. Trace 窗口过滤在查看记录文件过程中依然有效。

1.8.4.4. 基于 Measurement Setup 窗口过滤

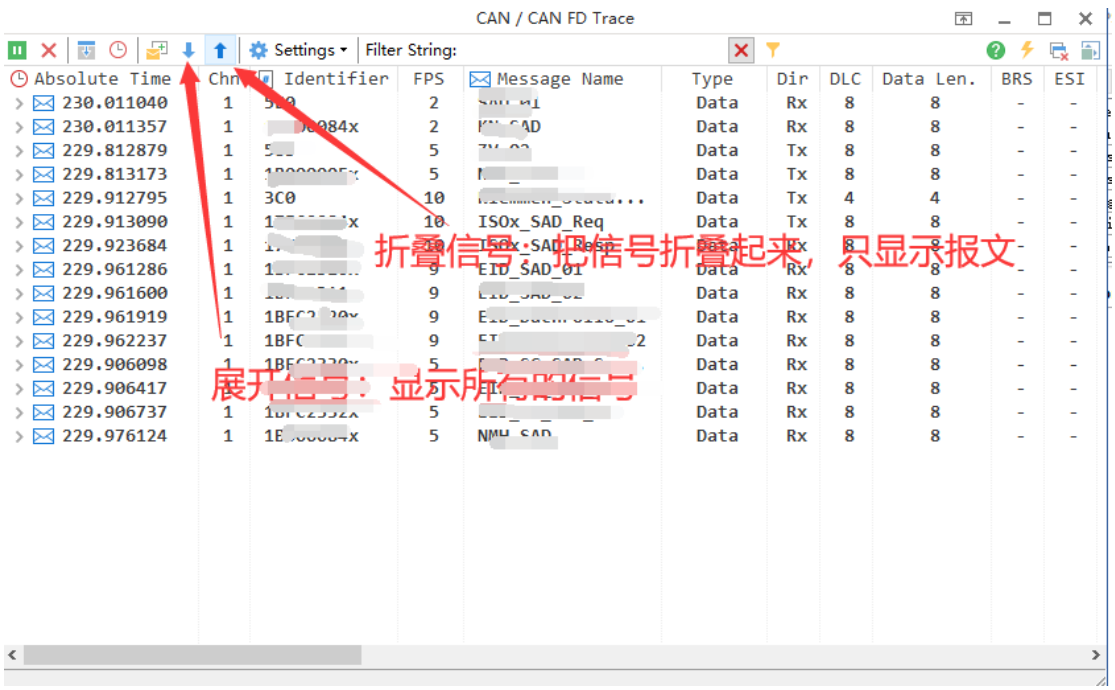
请查看章节：Measurement Setup。

1.8.5. 释疑：

1.8.5.1. 报文按照信号全部展开，不方便观察，如下所示：



结局办法：展开和折叠信号显示即可，如下操作：



1.8.5.2. 为什么连接 Application 瞬间，会监测到报文

问题描述：在点击 Connect Application 瞬间，总是在 Trace 窗口监测到报文，而自己并没有想要发送报文。有以下几个可能的原因：

1.8.5.2.1. Start 关联了报文发送模块

TSMaster 程序支持启动（Start）过程跟功能模块关联起来。此详细操作见章节：启动程序（Start）。比如报文发送（Transmit）窗体被勾选了，则启动瞬间，Transmit 上面的报文就会开启发送；总线回放（Bus Replay）窗体被勾选了，则启动瞬间，自动总线报文；以此类推。

解决办法：在工具->系统信息窗体中接触 Start 和该功能模块的关联；或者直接在该功能模块上点击右上角⚡按钮接触关联。

1.8.5.2.2. RBS 模块被自动激活

关于为什么会自动激活 RBS 模块，请查看 RBS 章节中关于“为什么 RBS 会被自动激活”子章节。

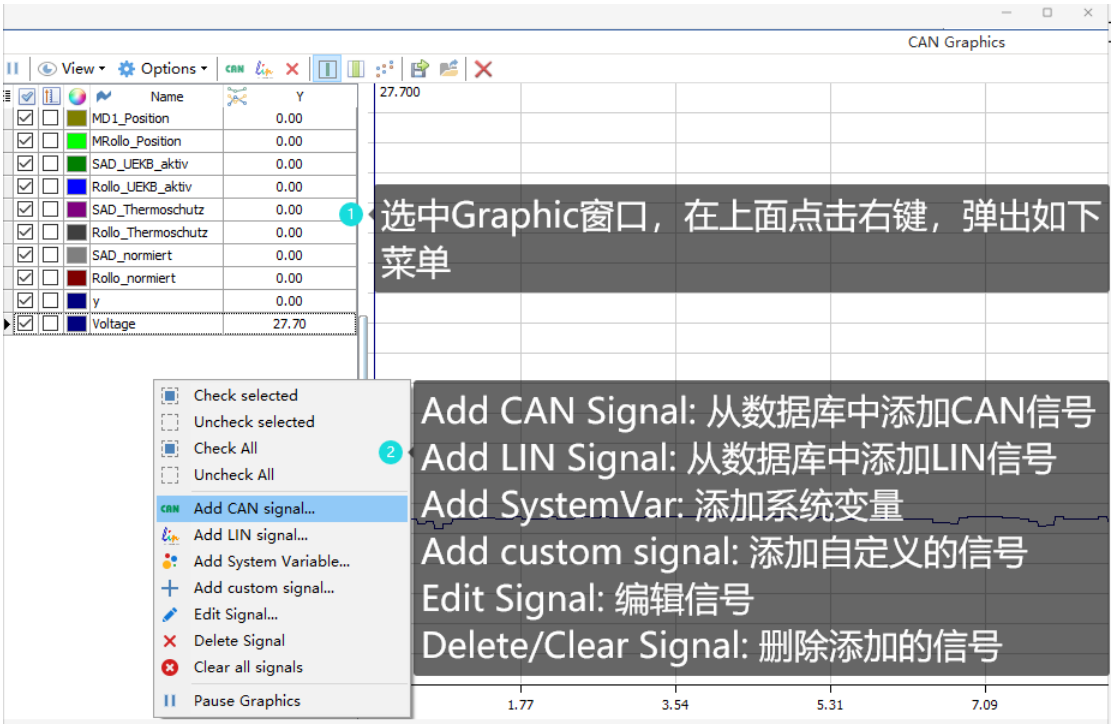
1.9. 曲线窗口（Graphic）

1.9.1. 基本操作

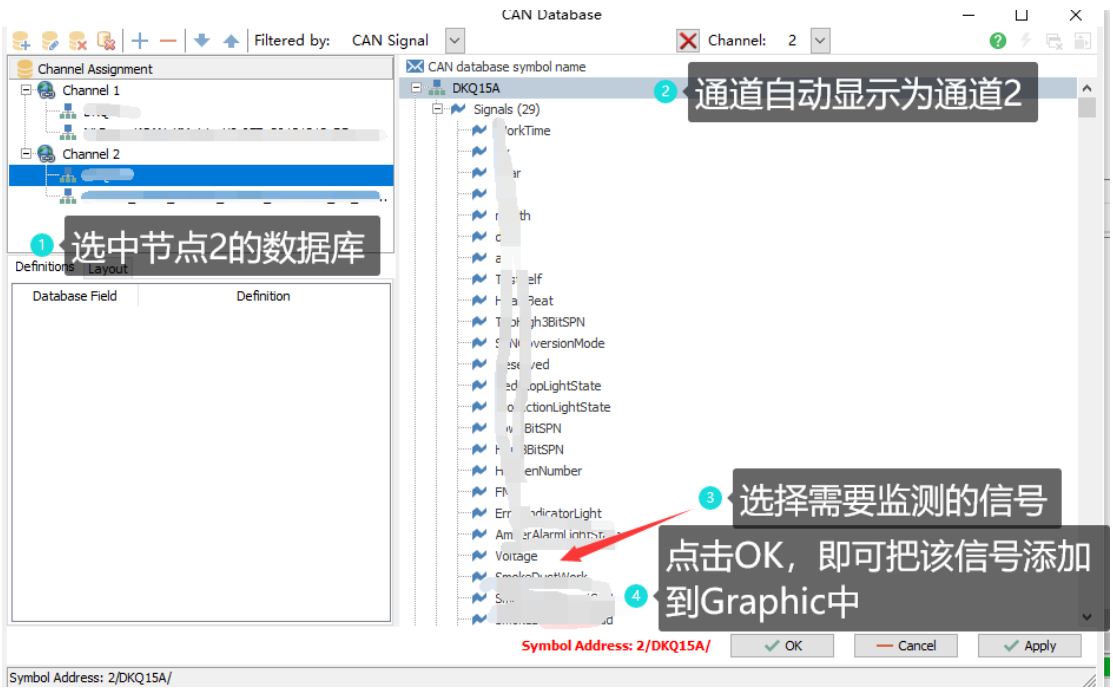
1.9.1.1. 添加监测变量：

往 Graphic 窗口中添加监测变量，主要有两种方式：1. 直接在曲线窗口中添加；2. 从 Trace 窗口中添加。

- 直接在 Graphic 窗口添加：

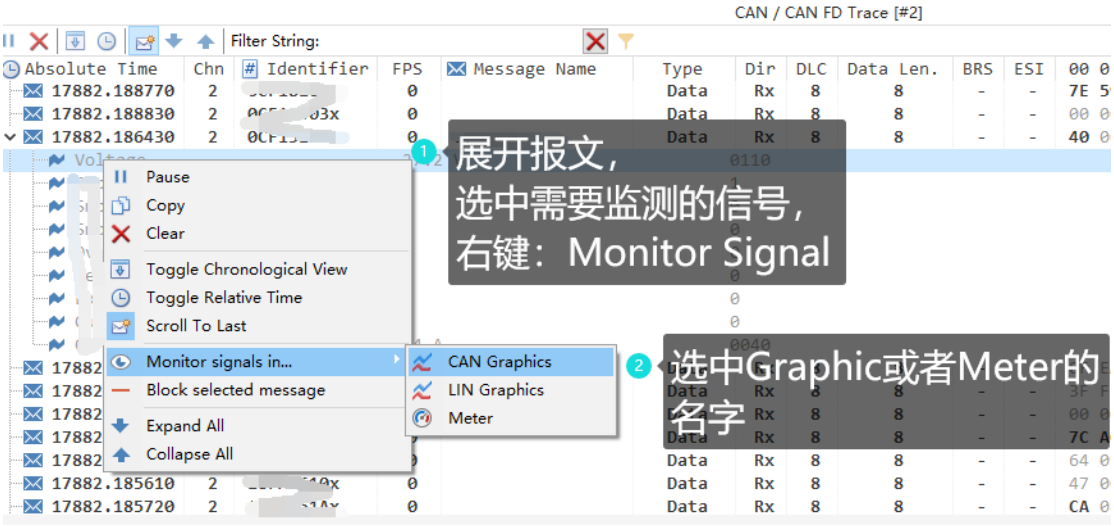


比如：选中 Add CAN Signal，则进入数据库界面，选择观测的信号。**这里需要强调的是**：数据库跟通道是绑定的，如果数据来源是通道 1，但是选择的是通道 2 的信号，那肯定是无法解析出对应的信号值的。如下所示：



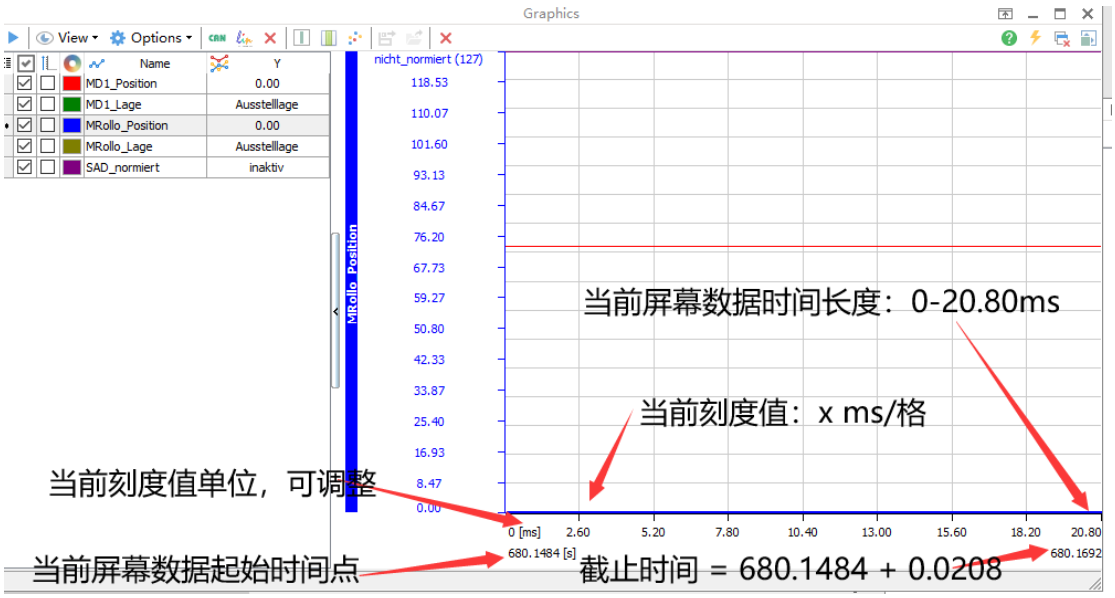
➢ 从 Trace 窗口中添加：

选中 Trace 窗口->展开报文->选中需要监测的信号->右键展开菜单->移动鼠标到 Monitor Signal In->把信号添加到对应的监测窗口即可。



1.9.1.2.X 轴调整

X 轴是时间轴，显示信号随着时间变化的规律。X 轴示意图如下图所示：



上图所示 X 轴解释如下：

当前屏幕上数据的起始点为 680.1484s，当前刻度大小为：2.6ms/格。总共包含 8 格，因此当前屏幕的时间宽度为：2.6*8 = 20.80ms，屏幕数据截止时间为：680.1484 + 0.0208 = 680.1692s。分析数据过程中，用户往往会涉及到调整观察细腻度（X 轴刻度），观察点位置（X 轴横向位置）等。下面讲解如何操作。

1.9.1.2.1. 调整 X 轴刻度

要调节 X 轴刻度，只需要把鼠标放着 X 轴上，**不用点击（单击和双击）**，直接通过滚轮调整 X 轴刻度值。

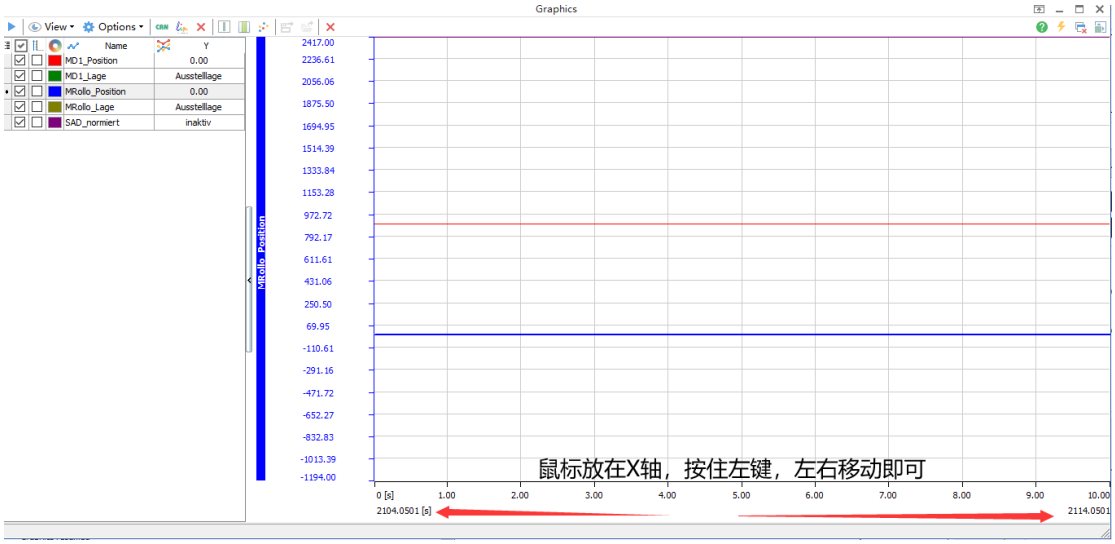
- 往上滑动滚轮，把刻度值往小了调，比如从一格表示 1s 调整为 1 格表示 1ms。

- 往下滑动滚轮，把刻度值往大了调，比如从一格表示 1ms 调整为 1 格表示 1s。

1.9.1.2.2. 移动 X 轴位置（X 轴显示范围）

方法 1：

鼠标放在 X 轴上，按住鼠标左键，左右移动，这种方式非常直观，推荐使用。



方法 2：

鼠标放在 X 轴上，按住 Ctrl 按键，往上滑动滚轮，实现往左移动的效果，往下滑动滚轮，实现往右移动的效果。

1.9.1.2.3. 使用建议：

移动 X 轴位置和调整刻度值配合使用效果会更好：需要比较大的位置移动的时候，先把刻度值调整为较大值，较快的速度移动到观测点附近，然后把刻度值调整到较小状态，精确移动到观测位置。

1.9.1.3.Y 轴调整

1.9.1.3.1. 调整 Y 轴刻度

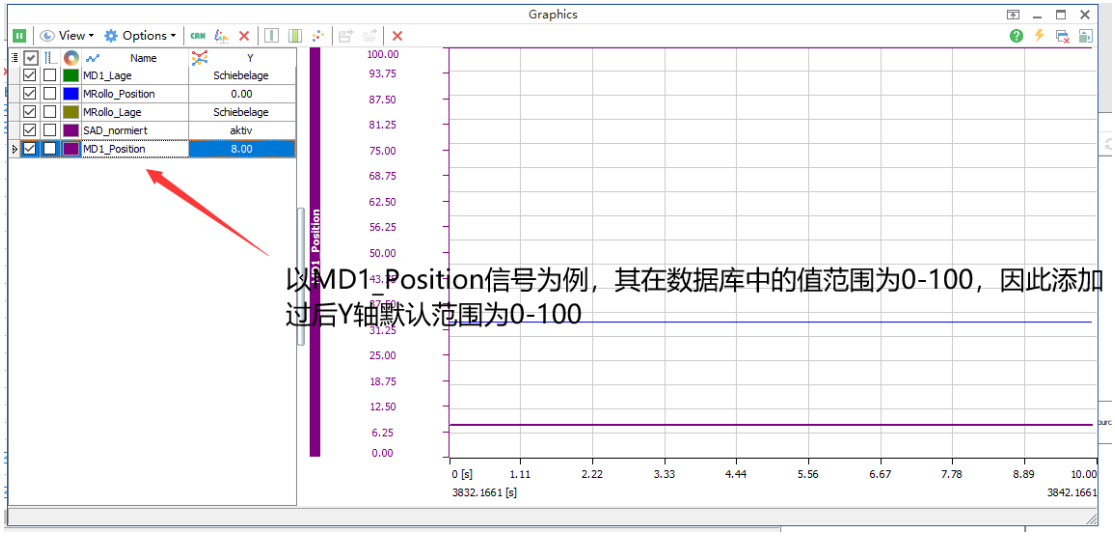
要调节 Y 轴刻度，只需要把鼠标放着 Y 轴上，**不用点击（单击和双击）**，直接通过滚轮调整 Y 轴刻度值（此操作跟 X 轴是一样的）。

- 往上滑动滚轮，把刻度值往小了调，比如从一格表示 1s 调整为 1 格表示 1ms。

➢ 往下滑动滚轮，把刻度值往大了调，比如从一格表示 1ms 调整为 1 格表示 1s。

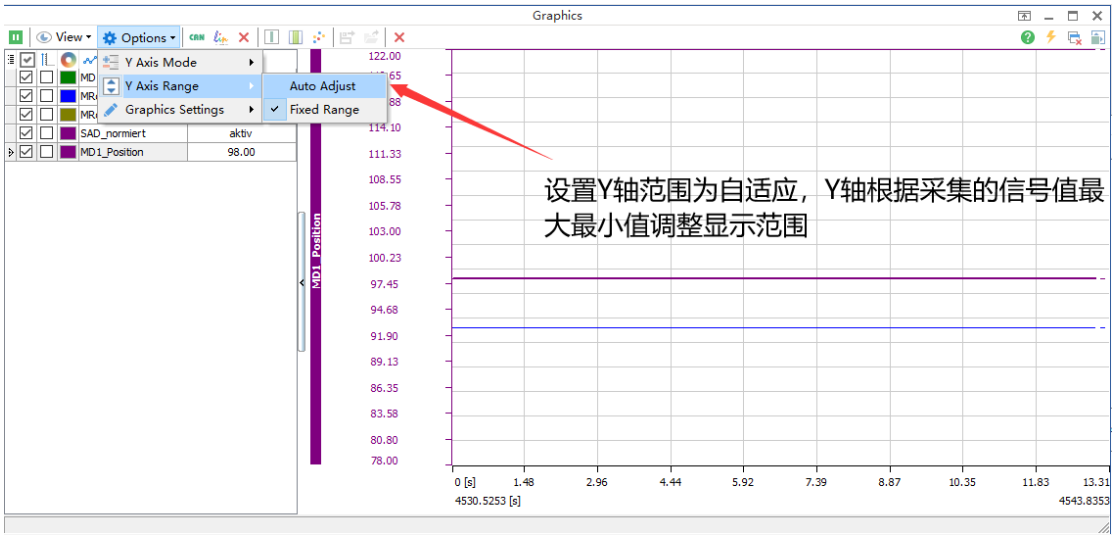
1.9.1.3.2. 移动 Y 轴（Y 轴显示范围）

Y 轴关联的是信号值，为了和数据库信号建立关联性，也为了防止用户直接拖拽把 Y 轴移动到了不合理的位置造成数据曲线看不到，Y 轴不支持用户直接通过鼠标拖拽来修改显示范围。Y 轴的范围默认取值是数据库中信号值的 Max 和 Min 值，如下所示：

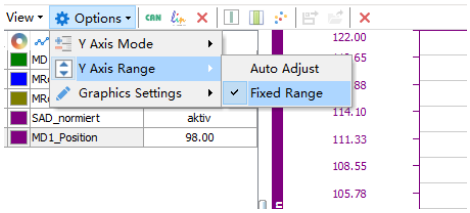


如果要调整 Y 中的显示范围，有如下三种方式：

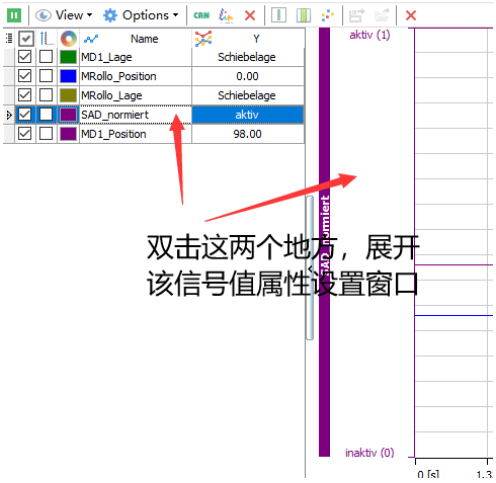
➢ 设置 Y 轴显示范围为自适应。Y 轴会根据数据值的大小范围，动态调整显示的范围，这种方式能够涵盖 Y 轴的所有显示值范围。设置方式如下图所示：



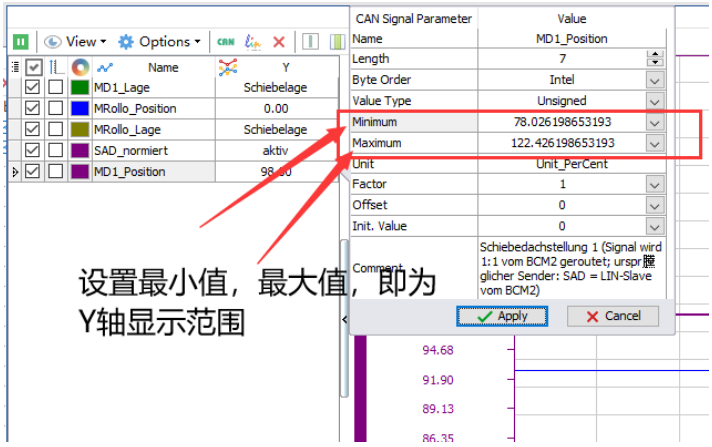
- 设置 Y 轴显示范围为固定值，如下：



双击信号值（或者直接双击该信号对应的 Y 轴），如下所示：



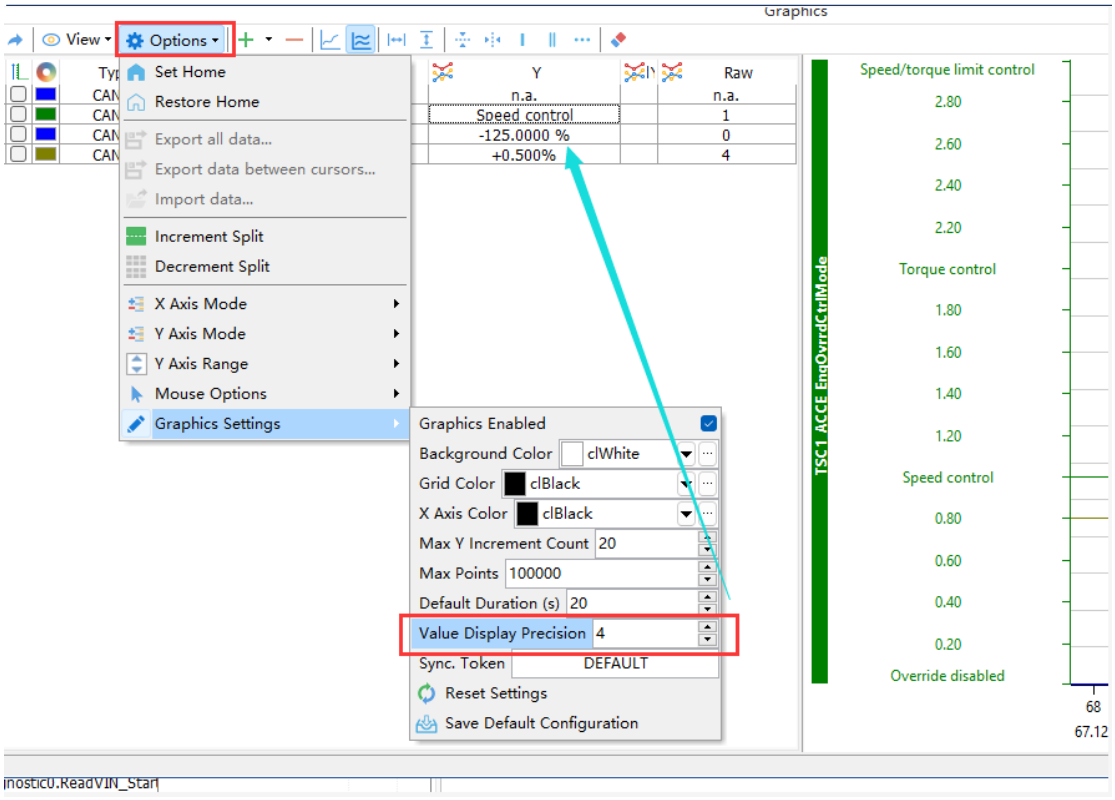
展开信号属性窗口如下所示，设置该信号值的最小值和最大值，即为 Y 轴显示范围：



- 通过鼠标滚轮调整

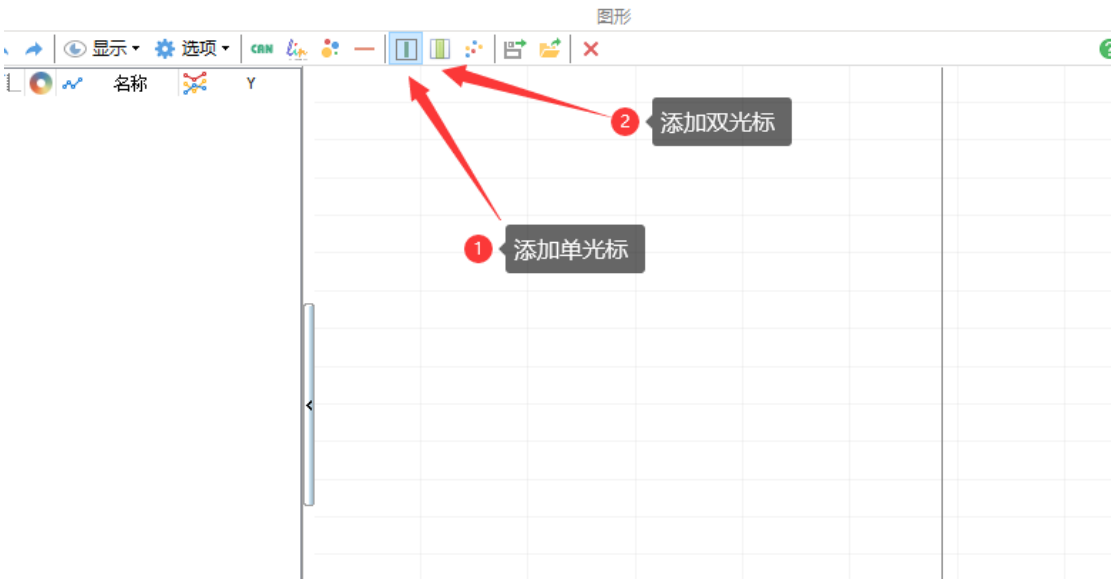
1.9.2. 浮点数精度

Graphic 中浮点数的显示精度，位置如下图所示：



1.9.3. 移动光标

1.9.3.1. 添加光标：



TSMaster 的 Graphic 窗口支持添加单个光标或者两个光标观察数据，添加方式如上图所示。取消光标显示，只需要再点击一下添加的按钮，光标则消失。

1.9.3.2.移动光标

1.9.3.2.1. 单光标移动

1) 右键点击，触发光标跟随，光标跟着鼠标移动。
2) 左键点击，把光标定在想要观察的位置处。

1.9.3.2.2. 双光标移动

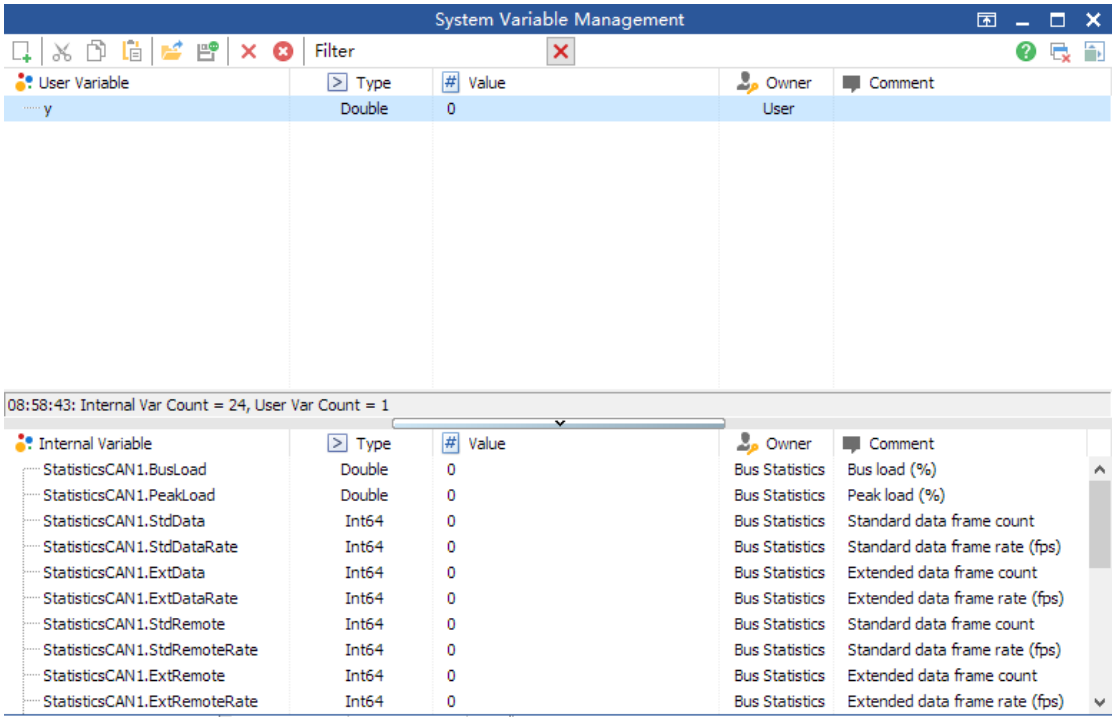
1) 左边点击，选择蓝色光标的观察位置
2) 右键点击，选择红色光标的观察位置

1.9.4. 应用案例介绍

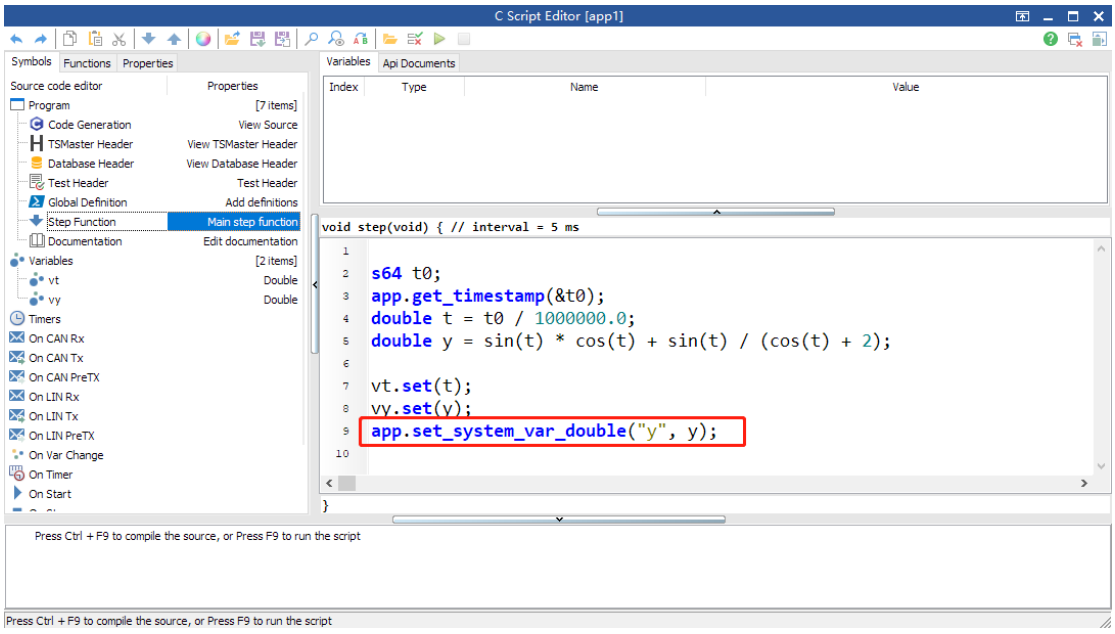
1.9.4.1.使用 TSMaster 绘制任意 ECU 变量曲线

【1】 定义系统变量 y

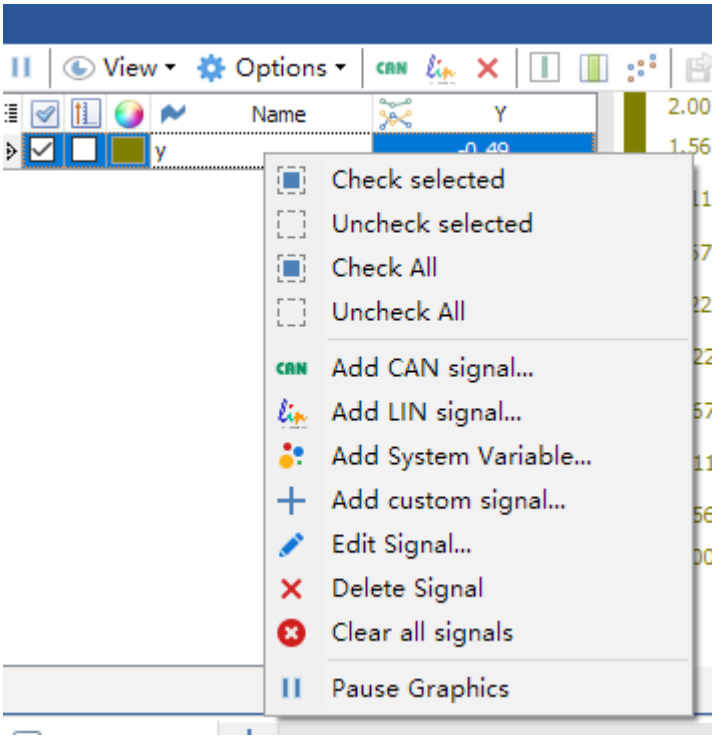
关于系统变量的详细说明，请查看章节：系统变量。



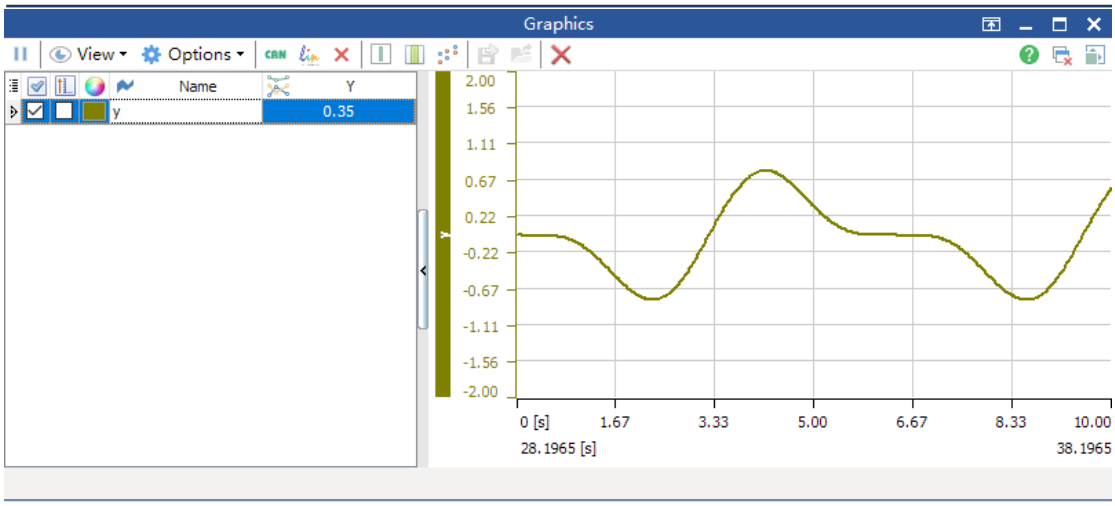
【2】 通过小程序将 ECU 内部变量赋值给这个系统变量 y



【3】 在 graphics 中添加变量 y



【4】 在 graphics 中查看这个变量 y



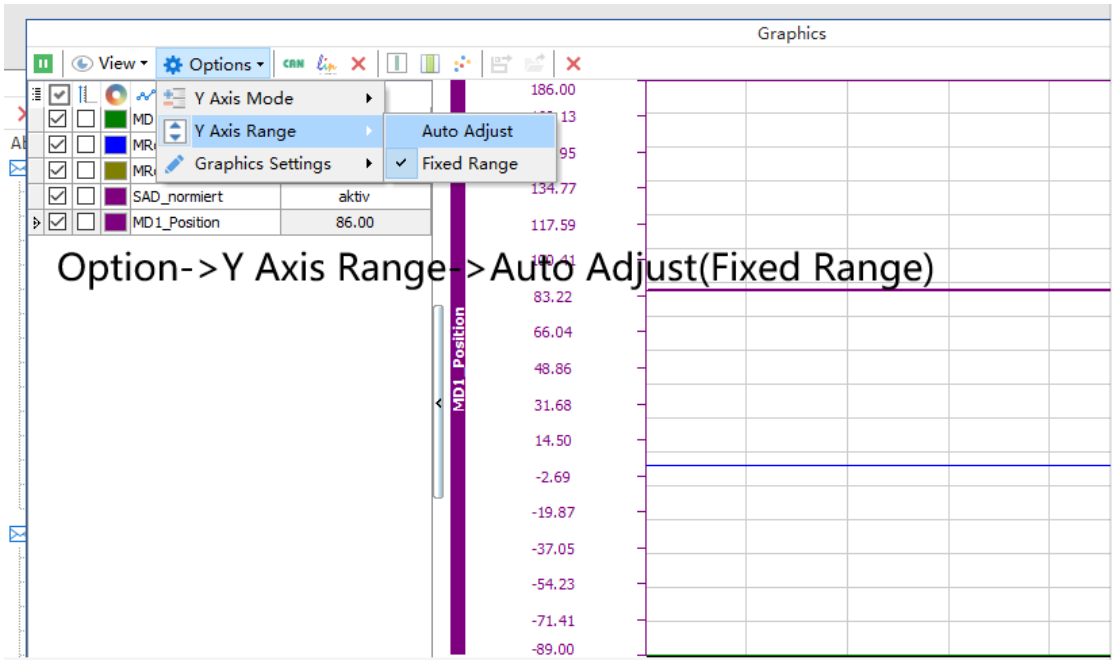
1.9.5. 释疑

1.9.5.1. 为什么数据库中设置了信号值范围，但是 Y 轴显示远远超过此范围？

问题描述：数据库中设置了信号值范围，比如信号值范围设置为 0--100,为什么 Y 轴的范围远远超过比如达到了 0---10000.

这是因为 Graphic 的 Y 轴的显示模式被设置为了自动模式，Y 轴的范围会根据信号值出现过的最大值最小值自动调整。这个信号值在显示的时候曾经打到过 10000 这个数字，Y 轴的范围就被调整到了 0—10000。

设置 Y 轴显示模式的操作如下：



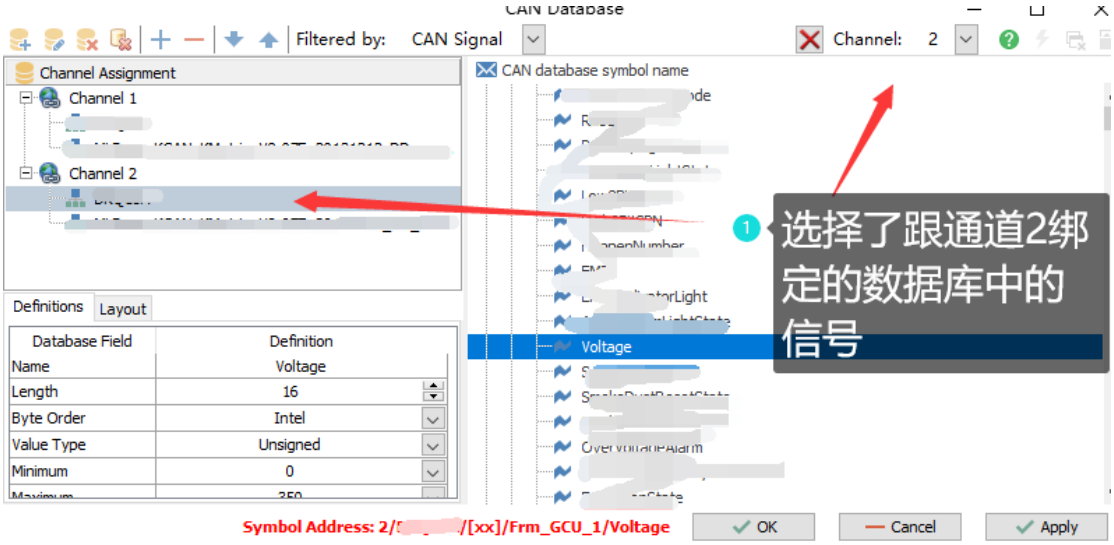
Auto Adjust：Y 轴根据实际值自动调整范围。

Fixed Range：Y 轴固定范围值，这种模式下，调整 Y 轴显示范围（Y 轴位置）的操作见 1.7.1.3.2 章节。

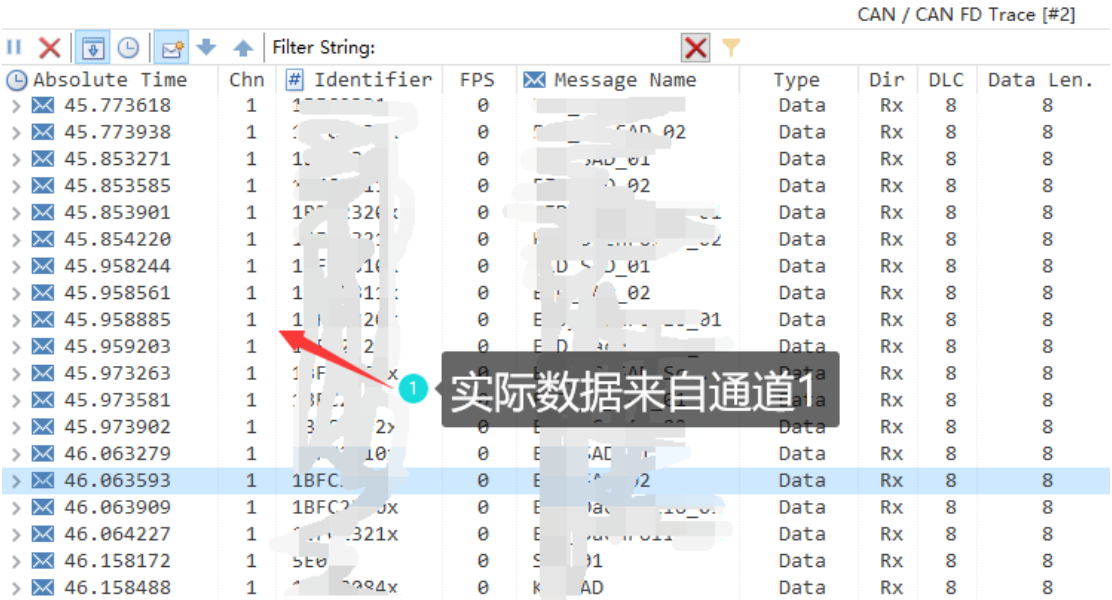
1.9.5.2. 为什么添加了信号，报文回放或者监测的时候在 Graphic 等窗口看不到信号变化？

可能原因：添加信号所在的通道跟当前报文（无论是回放还是实时监测）的通道不一致。比如：

- 添加信号选择的通道是 2，如下所示：添加的信号是来自通道 2 的数据库中的信号。



- 但是报文是来自通道 1 的报文，如下：



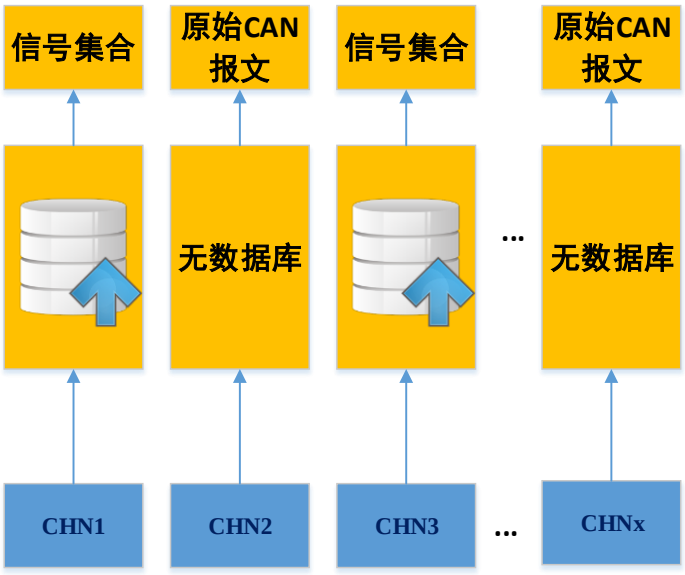
这种情况下，解析出来的信号值肯定是没有数值的。

1.10. CAN DBC/ LIN LDF

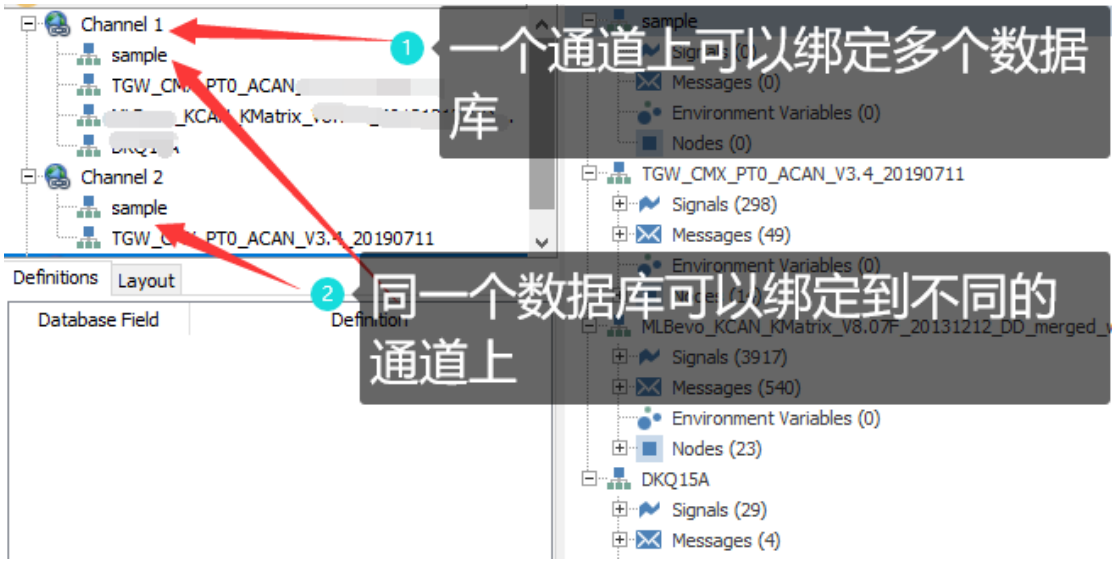
1.10.1. 基本概念

在使用 CAN/CANFD/LIN 总线数据库之前，需要明确几个基本概念。

- 数据库跟通道是绑定的。每个通道收到 CAN/CANFD/LIN 总线数据过后，根据该通道绑定的数据库分别解析数据，如下图所示：



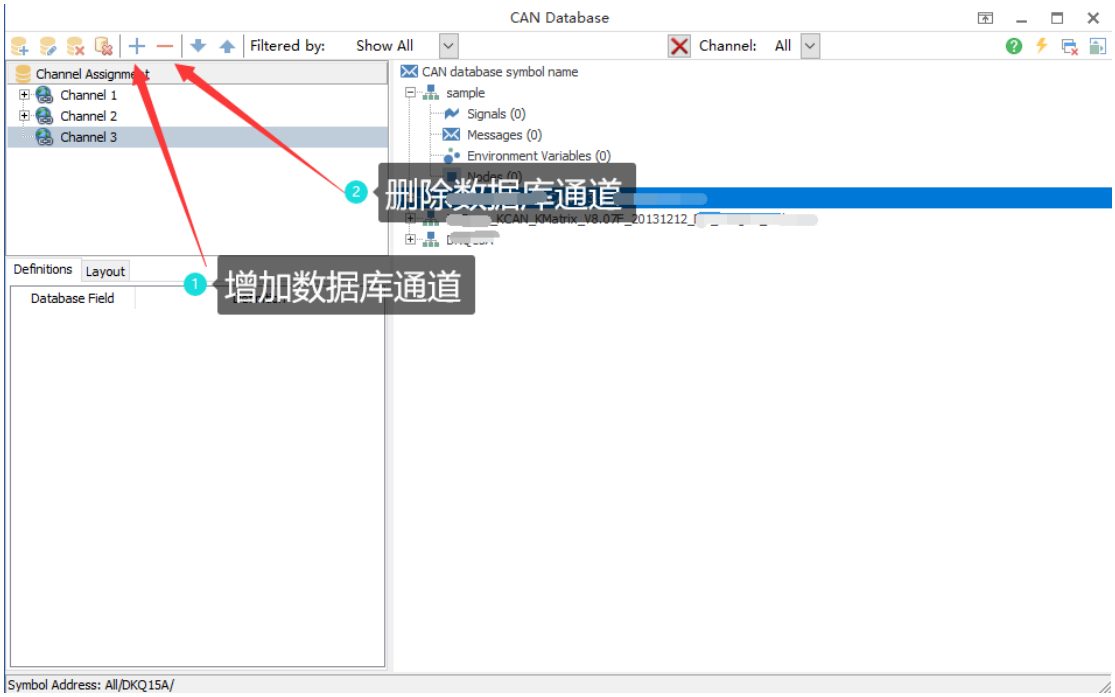
- 一个通道可以绑定多个数据库。一个数据库也可以绑定到不同的通道上。



同一个通道可以绑定了多个数据库，这些数据库是并列关系。该通道收到 CAN 报文数据过后，TSMaster 数据库引擎会轮流查询这几个数据库，在数据库中查询到该报文 ID 后，完成对报文数据的解析。

1.10.2. 数据库通道管理

数据库通道跟硬件通道是一一对应的。如下所示：Channel1 对应的就是 TSMaster 硬件配置界面中的应用程序通道 1.其他通道依次类推。在数据库管理界面中，用户可以根据需求增加和删除数据库通道。



1.10.2.1. 关于通道编号：

关于通道编号的对应到软件中的实际数值，如下所示：

```
#define CH1 0
#define CH2 1
#define CH3 2
#define CH4 3
#define CH5 4
#define CH6 5
#define CH7 6
#define CH8 7
#define CH9 8
#define CH10 9
#define CH11 10
#define CH12 11
#define CH13 12
#define CH14 13
#define CH15 14
#define CH16 15
```

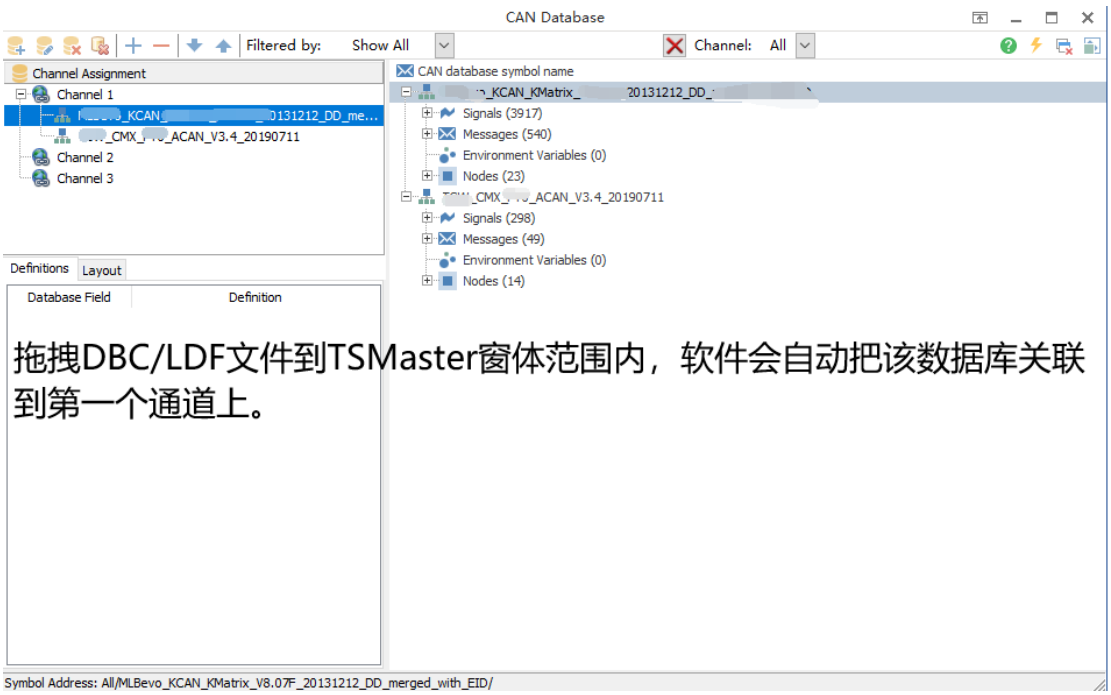
也就是应用程序中的 CHANNEL1 对应的编号是 0，CHANNEL2 对应的编号是 1，依次类推。因为对于程序开发来说，编号一遍从 0 开始（比如数组等变量）；对于现实生活中的描述习惯来说，一般从 1（比如通道 1 等）开始。

1.10.3. 添加 CAN/LIN 数据库文件

TSMaster 支持两种方式添加 CAN/LIN 数据库文件：

1.10.3.1. 直接拖拽加入

用户直接把 CAN/LIN 数据库拖拽到 TSMaster 窗体范围内，程序会自动把数据库关联到第一个通道。如下所示：



优点：

操作方便，用户甚至无需打开数据库管理窗口，直接把 DBC/LDF 文件拖拽到 TSMaster 窗体内，即可完成对该数据库的添加。

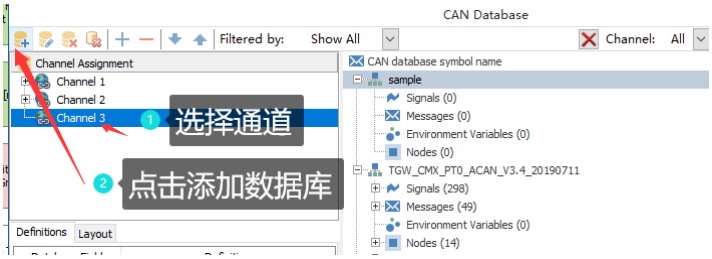
缺点：

这种添加方式，默认只能把该数据库关联到第一个通道上。如果需要关联到其他通道上，需要到数据库管理界面上手动添加。

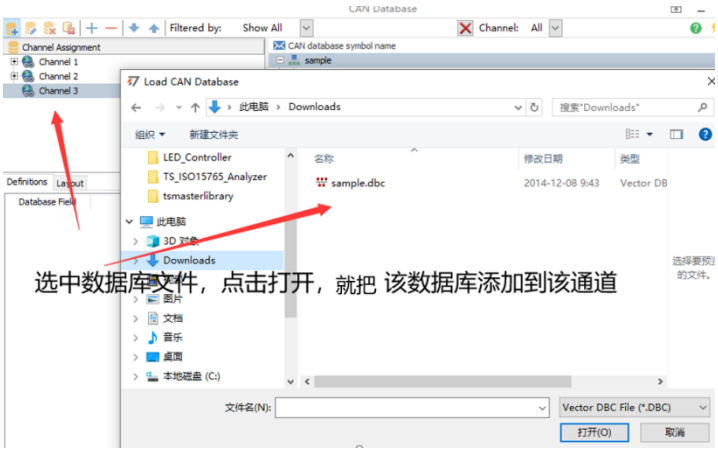
1.10.3.2. 在数据库窗体中加入

在数据库窗体中添加数据库管理通道步骤如下：

- 选中通道，点击添加数据库按钮，如下所示：

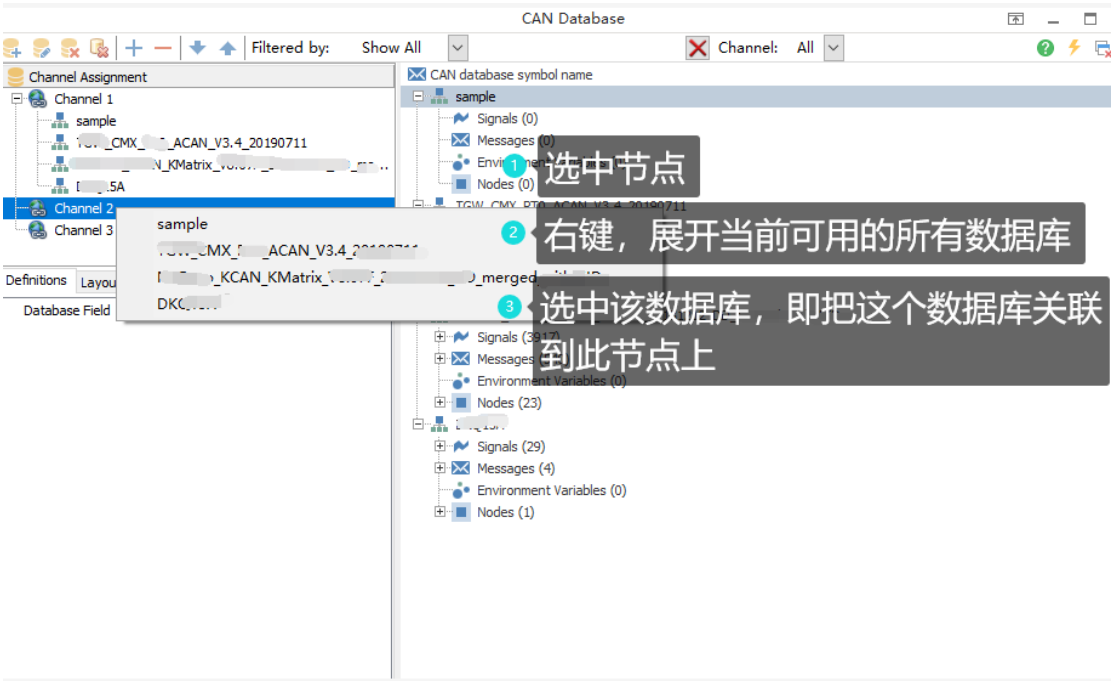


➢ 在路径中选中 dbc 文件，即可添加到数据库列表中，如下：

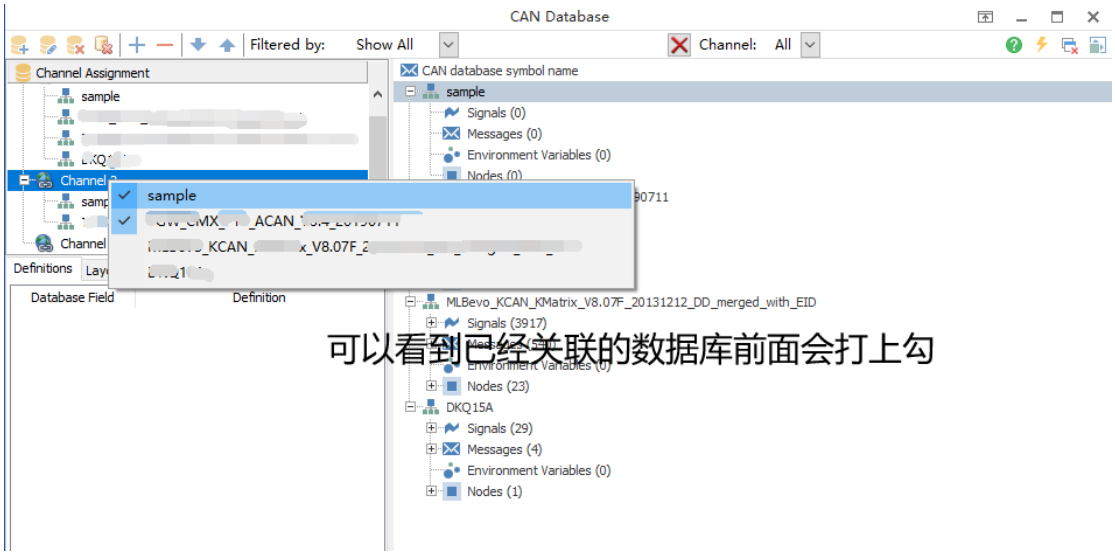


1.10.3.3. 关联数据库到 ECU 通道上

通过数据库管理器，TSMaster 提供了方便的操作方式允许用户在不同通道之间切换数据库文件，如下图所示：



数据库关联到该通道后，右键点击该通道，可以看到已经完成关联的数据库前面打上了勾，如下所示：

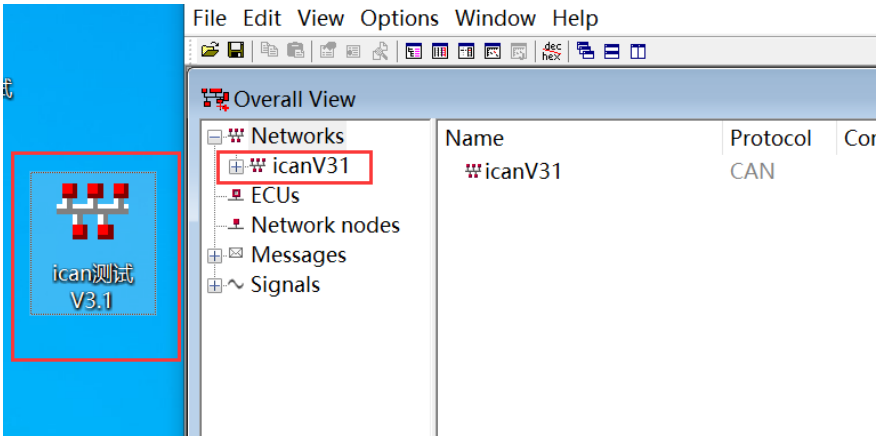


如前述章节所讲：在 TSMaster 中，直接拖拽数据库文件，会默认关联到第一个通道。在拖拽完成后，可以到数据库管理界面中重新关联数据库和 ECU 通道。因此，推荐采用如下步骤，可以以最简单的方式完成数据库的添加和绑定：

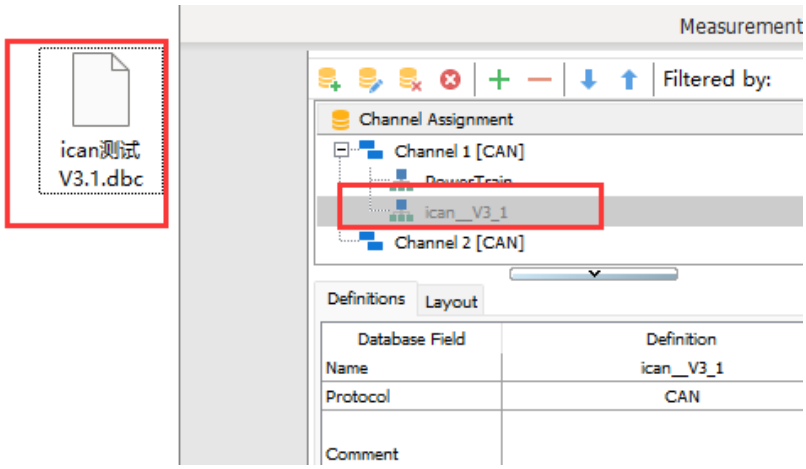
- 先采用拖拽的方式，一股脑的需要用到的数据库全部拖拽到 TSMaster 中，如 1.7.1.1 所讲。
- 进入 TSMaster 的数据库管理界面中，把刚才添加进来的所有数据库根据需求绑定到对应的通道上，如 1.7.1.3 所讲。

1.10.4. 数据库名称

根据 DBC 文件格式（file format）的要求，DBC 网络定义中禁止使用中文和点号等字符。比如名称位“ICAN 测试 V3.1.dbc”的文件，载入到 VECTOR 的 CANDB++中，其显示的名称为 ICANV31，直接把中文和点号给抛弃了，如下图所示：



TSMaster 中也严格遵循此原则，依然以上图所示的 DBC 文件为例，载入到 TSMaster 中，网络名称显示如下图所示：



所有中文字符，点号或者其他不符合规范的字符全部通过下划线替换掉。
用户使用过程中一定要注意 DBC 文件的符号定义，因为跟后面的信号关联等直接相关。

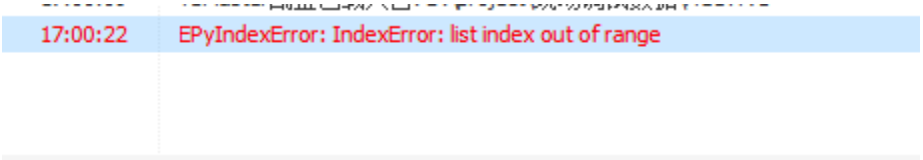
1.10.5. 查看报文定义

1.10.6. 查看信号定义

1.10.7. 释疑

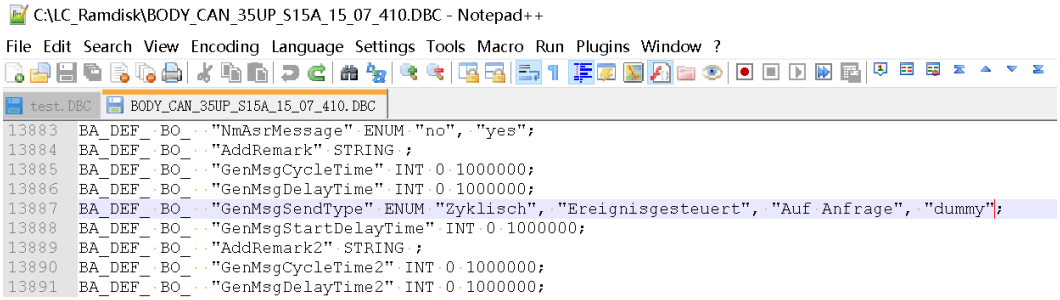
1.10.7.1. 加载 DBC 失败，提示 Index 超出

载入数据库的时候碰到如下错误：

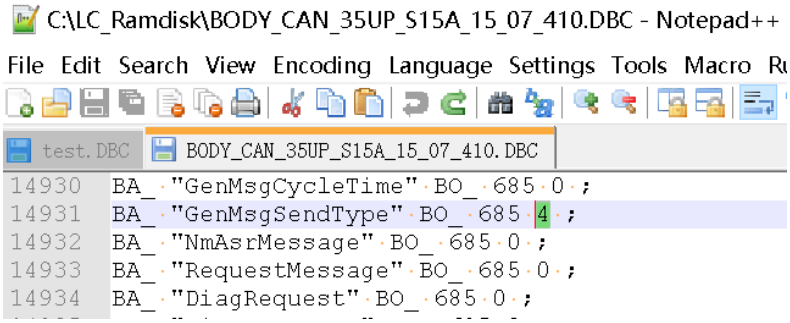


DBC 里面定义的属性值超出了实际的定义范围，举例如下所示：

dbc 用编辑的时候少定义了属性，导致文件内部信号下标越界，不能自圆其说，如下所示的 dbc 文件，属性 GenMsgSendType 只定义了 4 个元素：



下标却使用了第五个元素



解决办法：将所有相关属性字段增加一个垃圾字段，现在就可以正常载入到 DBC 中了。

1.10.7.2. 加载的 DBC 文件丢失

现象描述：加载的 DBC 文件，保存了工程，重新打开过后，DBC 文件丢失了，还得重新加载

这是因为，TSMaster 默认不支持中文路径。如果该 DBC 在中文路径下面，就会发生上面的问题。

解决办法：把 DBC 文件存储到纯英文路径中，即可。

1.10.7.2.1. 加载 DBC 文件失败，提示 CANDBParser.ini Not Existing

1.11. 报文格式转换

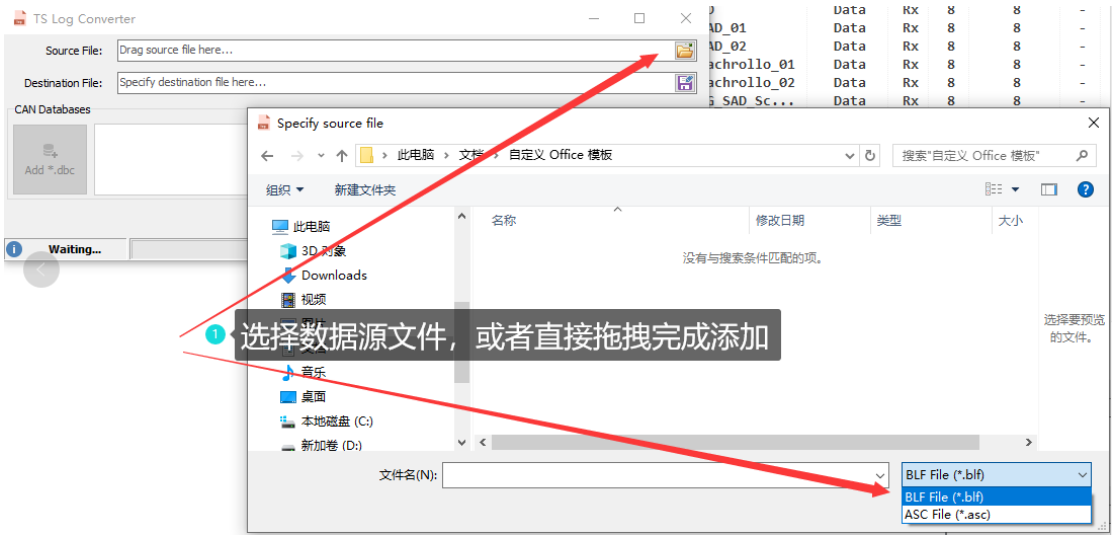
TSMaster 格式转换器支持：blf 和 asc 文件的互转，以及 blf 转 mat 文件，asc 转 mat 文件。注意：无论是报文格式转换还是报文回放，文件名中都不要有空格。

1.11.1. ASC 与 blf 互转

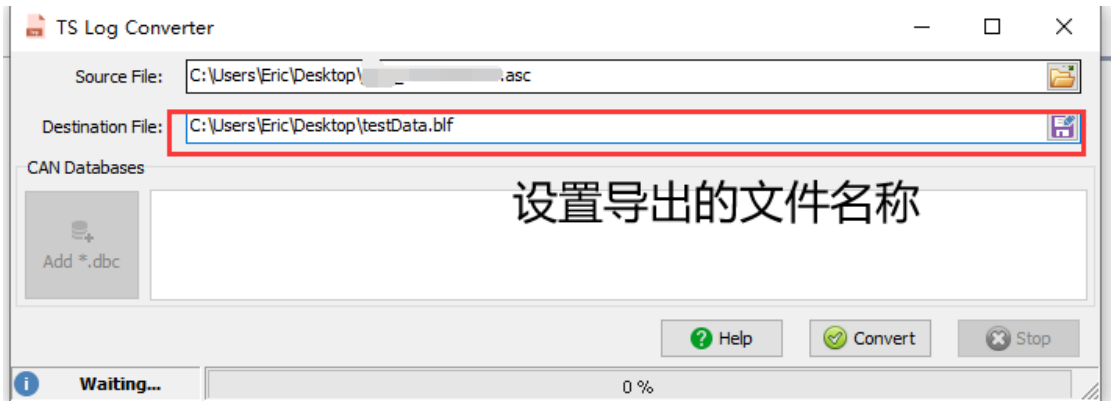
ASC 与 blf 文件格式转换过程如下所示：

- 打开格式转换器

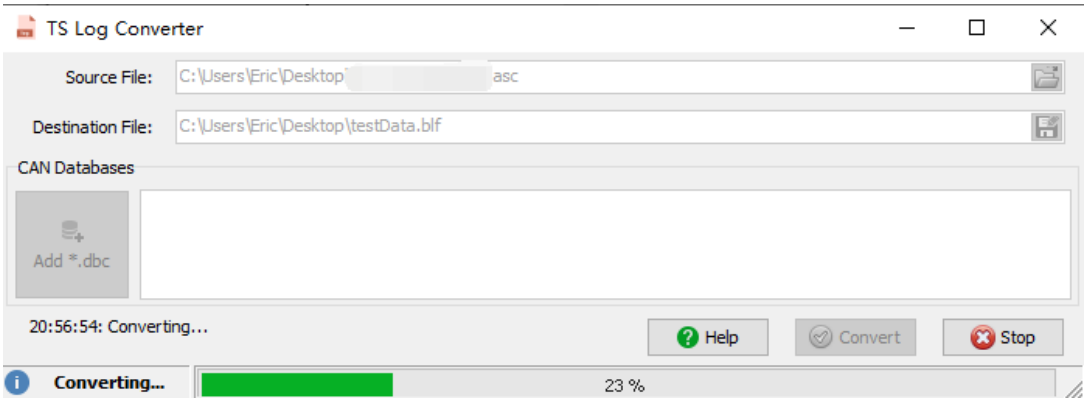
- 添加转换源文件



- 设置输出文件路径和名称



➢ 点击 Convert，输出目标数据文件，如下所示：



1.11.1.1. ASC 支持头部时间格式

包含下面 10 种格式：

date 21-11-2020 9:46:39.851

date 21-11-2020 9:46:39

date 21/11/2020 8:20:28.851

date 21/11/2020 8:20:28

date 2020-11-21 9:46:39.851

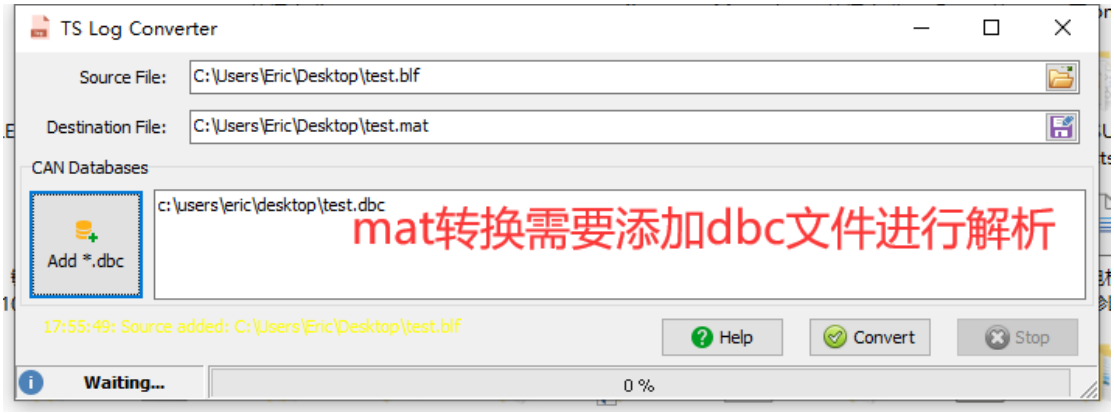
date 2020-11-21 9:46:39

date 2020/11/21 8:20:28.851

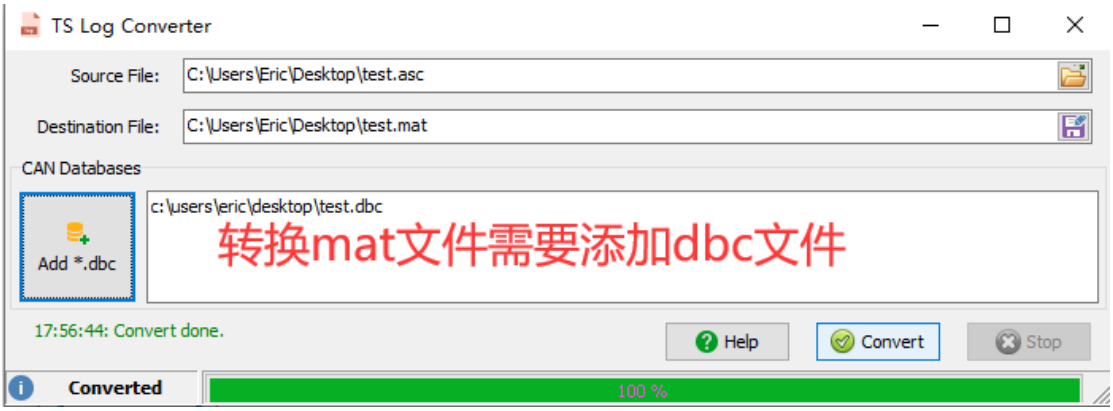
date 2020/11/21 8:20:28

date Sun Sep 27 02:03:58.450 pm 2020
date Wed Nov 25 10:36:29 2020

1.11.2. Blf 转 mat 文件



1.11.3. asc 转 Mat 文件



1.11.4. 释疑

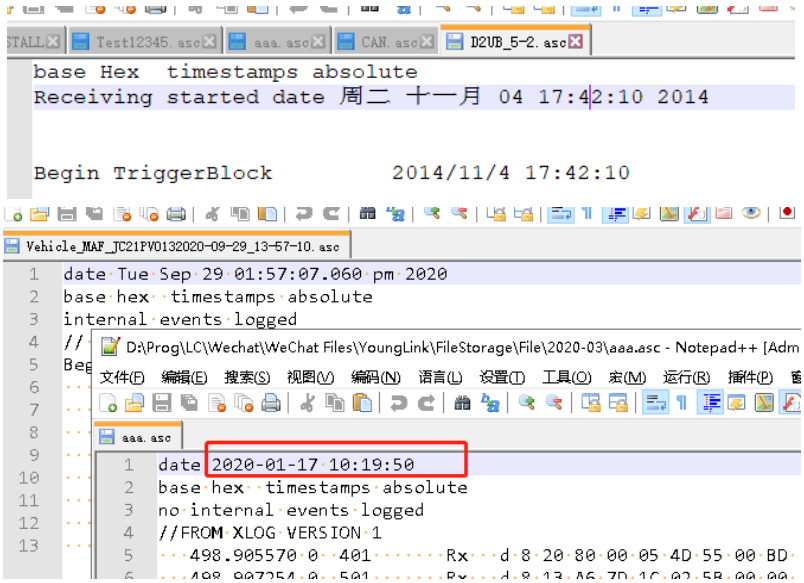
1.11.4.1. 文件转换失败

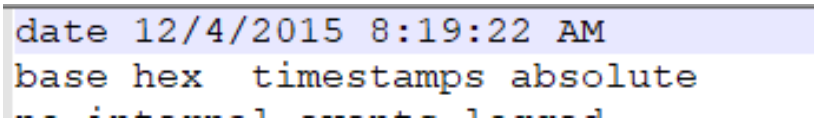
注意被转换文件和转换出来文件的文件名，中间不能有空格。如果有空格，则会造成文件的转换失败，错误类型为 6。

1.11.4.2. 为什么有的 asc 转换 blf，起始时间戳会对不上？

Asc 文件转 blf 文件的时候，会碰到这种情况：报文的相对时间戳是正确的，但是整段数据的起始时间点不对。这是因为，ASC 文件格式中起始时间点的描述没有按照 TSMaster 设定的格式来写。为什么会出现这种情况呢？

因为 ASC 文件是明码格式，对于 ASC 文件的文件头描述，**没有明确的格式规范来约束**，各个厂商可以按照自己能够解释的格式进行编写，直接写入中文注释也可以。如下所示：





可以看到图片 1 的格式描述在第一行，时间戳数据在第二行，数据排布完全根据开发者个人喜好；其他图片时间戳在第一行，格式描述又在第二行。各家定义的格式各异，大小写，间隔符，表述方式等的差异，表述方式可以达到几十上百种，不一而足，要全部支持完全不现实。因此 TSMaster 内置支持了多种时间戳格式，详细情况见章节：ASC 支持头部时间格式。

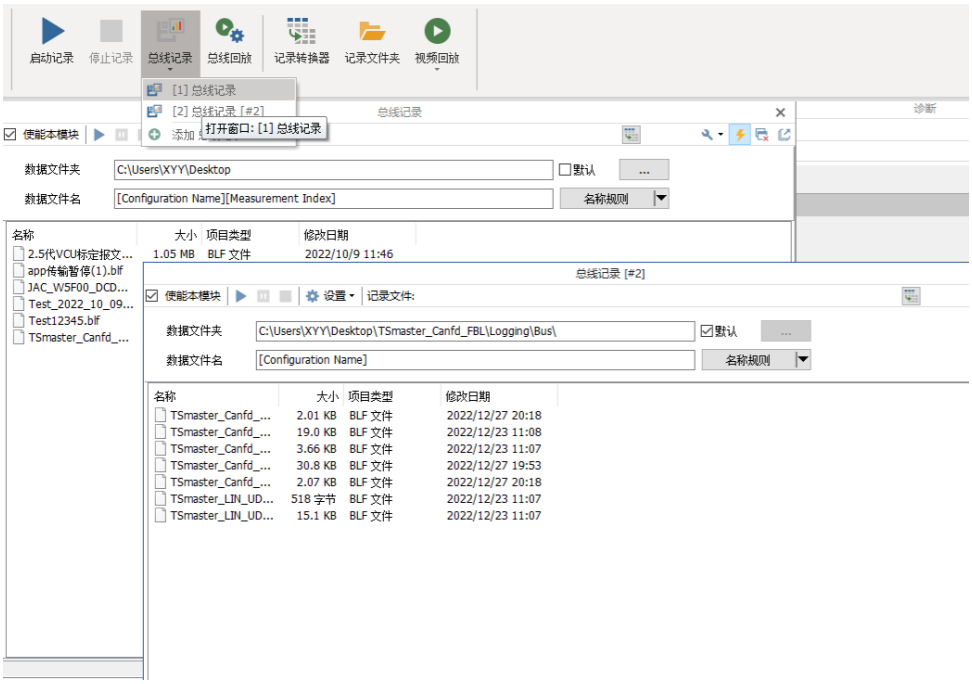
用户拿到其他 ASC 文件的时候，在转换之前，请首先确认一下起始时间的数据格式，如果不是这种数据格式的，请手动调整为上述格式，这样系统才能够正确识别数据块的起始时间点。

1.12. 报文记录

1.12.1. 总线记录模块

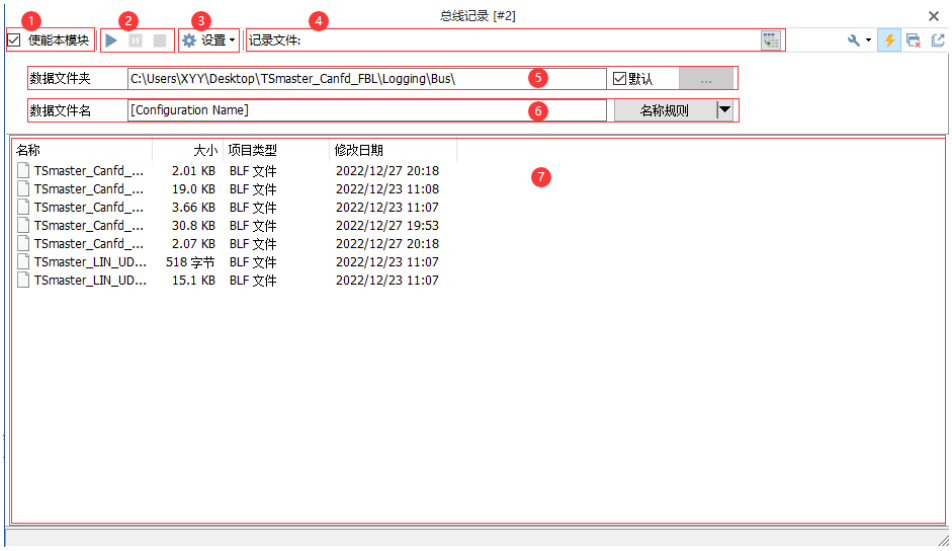
TSMaster 总线数据记录模块路径如下：分析->记录与回放。记录模块具有如下特点：

- 可以添加多个记录文件同时进行记录。
- 每个记录文件可以单独设置存储路径，存储文件大小，存储命名规则等参数。
- 存储文件名称灵活可配置，用户可以根据需求设定名称生成的规则。

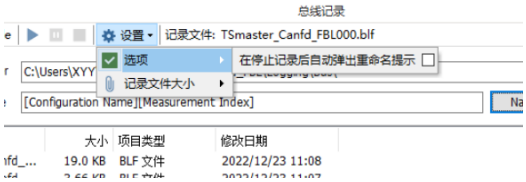


1.12.2. 记录文件参数

一个记录模块，主要包含如下可设置的参数：



- 区域 1：是否使能本记录模块。如果选择不是能，则启动记录的时候，不启动本模块。
- 区域 2：操作区，包含启动，暂停，停止记录功能按钮。
- 区域 3：设置。展开此按钮，如下所示，主要包含下面配置：
  1. 是否则停止记录的时候弹出提示重命名窗体；
  2. 记录文件的大小是无限还是指定数据大小。



- 区域 4：记录文件名。该区域是只读的，当区域 6 中的命名规则发生改变的时候，本区域同步更新当前的文件名。
- 区域 5： 数据文件路径名。用于设置存储数据文件的文件夹路径。如果选择默认配置，则选择工程默认存储数据的目录。
- 区域 6： 数据文件命名规则。用户可以通过组合命名规则，灵活设置数据文件的名称。主要包含如下的命名规则：
  1. UserName 为当前电脑的用户名称；
  2. Computer Name 为当前电脑的设备名称；
  3. System Time 为当前的系统时间；
  4. Measurement Index 为记录的索引，默认从 000 开始，当用户添加多条总线记录时，记录的索引自动递增；
  5. Measurement Start Time 为记录的开始时间；
  6. Configuration Name 为当前的配置名称

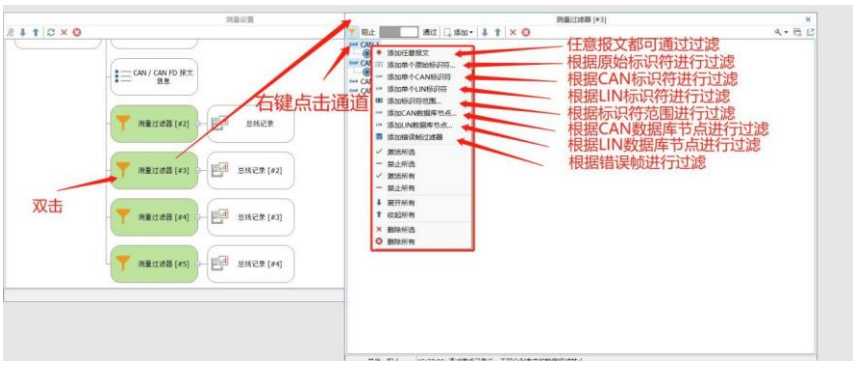
  比如，配置如下[UserName][Measurement Index],则对应的数据文件名为：XYY001.blf。修改规则为[UserName]_[Measurement Index],则对应的数据文件名为：XYY_001.blf。
- 区域 7：呈现当前选中数据文件夹中包含的记录文件。

注意：当多个模块使用同一个文件路径存储数据，如果使用相同的数据文件名会发生文件冲突，不能启动数据记录。

1.12.3. 添加记录过滤器

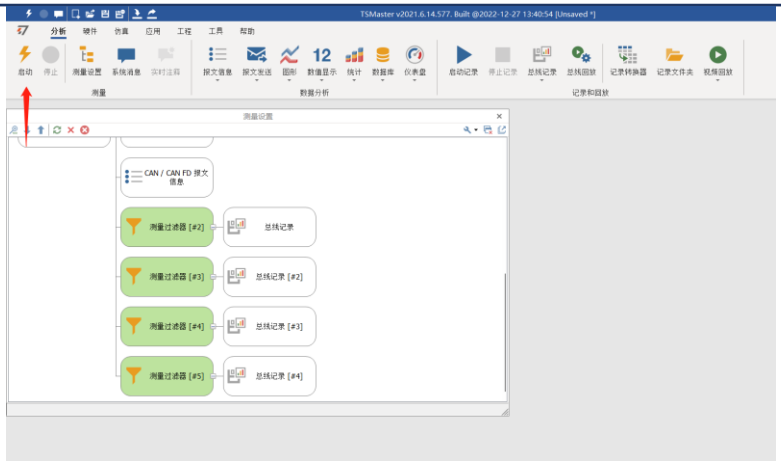
在测量设置模块中，用户鼠标右键点击选定的总线记录可以为该总线记录插入过滤器。

用户用鼠标左键双击测试过滤器可以打开测试过滤器的配置页面，在页面中显示了当前项目使用到的通道列表，用户选定某一通道，点击鼠标右键可以添加需要进行过滤的报文信息。

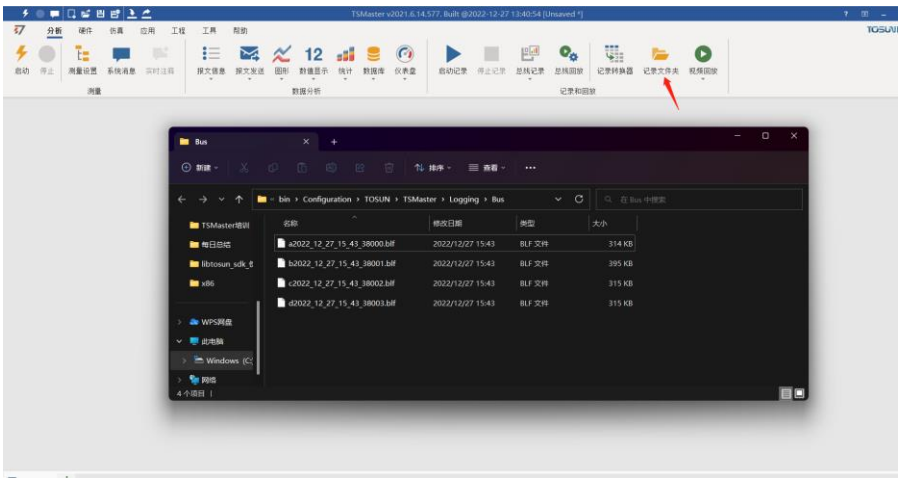


当用户希望在该总线记录中记录多个通道的报文信息时，只需为通道添加报文过滤器即可，启动程序后，添加了报文过滤器的通道都会被记录，没有添加报文过滤器的通道则不记录，如下图所示。

在过滤器配置完成后就可以启动程序，发送报文进行总线记录了

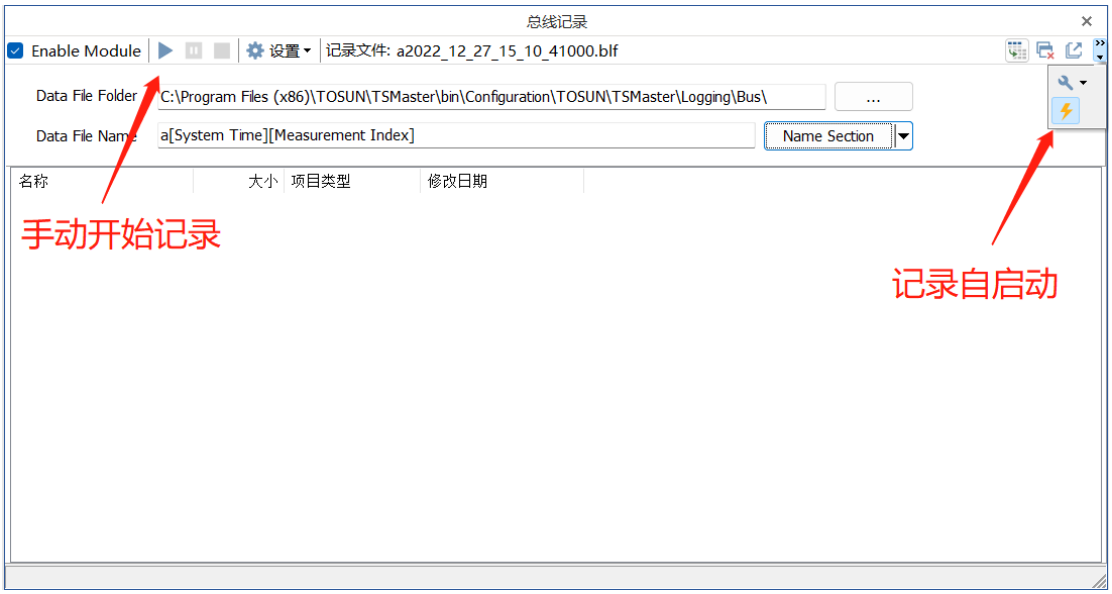


当程序停止后，记录文件会自动保存到预先设置的路径下，用户点击记录文件夹按钮，就能直接打开预先设置路径的文件夹，如果用户预先设置了多个存储路径，那么点击记录文件夹按钮后，所有的保存了记录的文件夹都会被打开。

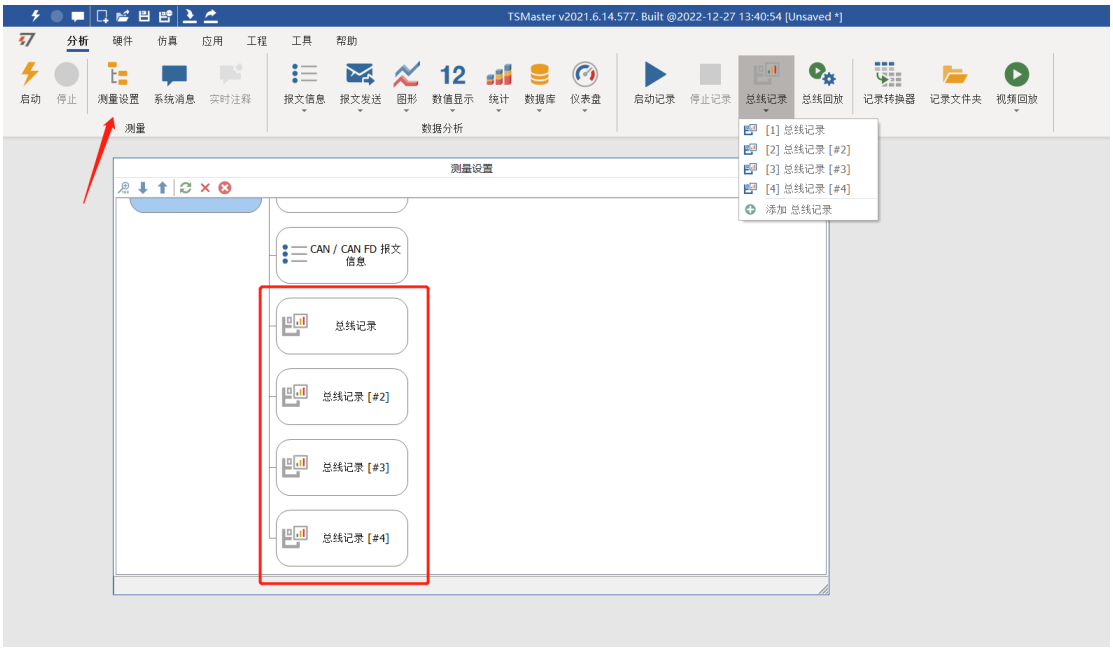


1.12.4. 开启自动记录

数据记录可以手动也可以自动启动，记录模块可以单独启动，也可以同时启动，点击 按钮可以手动开始记录，用户也可以在上图所示位置，点击 按钮设置记录自启动，即程序启动就自动开始记录，当用户需要同时启动多个记录时，只需将多个总线记录都开启记录自启动，在启动程序时就能同时开始多个记录了，如下所示：



当用户添加完成多条总线记录后，在测试设置中会显示用户添加的总线记录。如下图所示：

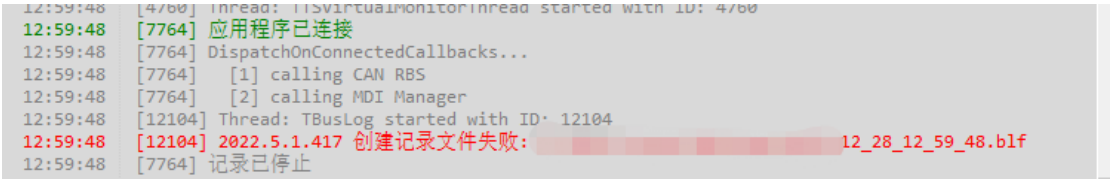


1.12.5. 常见错误

首先，如果发生错误情况，用户要养成习惯去查看系统信息窗口，大部分情况下都能在其中找到解答。

1.12.5.1. 路径权限管理，启动记录失败 1

用户选择了数据存储文件夹，如果该文件夹设备了管理权限，不允许在其中创建新的数据文件，用户启动数据记录的时候，会启动失败。提示如下所示：

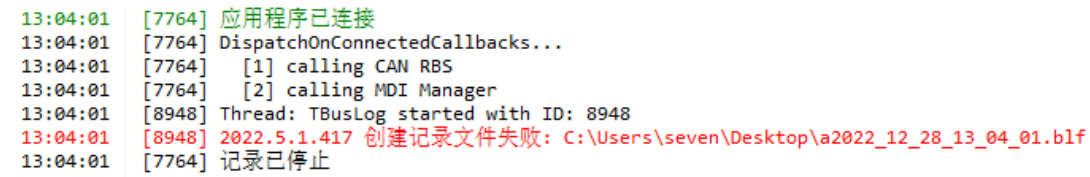


解决办法：

重新设置用户数据目录，选择允许用户修改的用户目录。

1.12.5.2. 从其他用户拷贝的工程，启动记录失败 2

当用户使用来自其他用户的配置工程，如果该工程配置的数据记录文件夹选择了一些特殊目录，比如用户桌面位置。拷贝到当前电脑的时候，因为用户名称等不一样，自然会造成创建数据目录和数据文件夹的失败，如下所示：



因为是从名称为 seven 的用户拷贝过来的工程，在当前用户下，就不存在 seven\Desktop 路径，而且因为在 C 盘目录下，不允许创建新的目录，造成创建数据文件失败。

解决办法：

重新设置用户数据目录。建议设置跟工程文件夹相关的目录，避免使用跟电脑和用户绑定的目录，以免分享给其他用户使用的时候出现路径失效的情况。

1.13. 报文回放

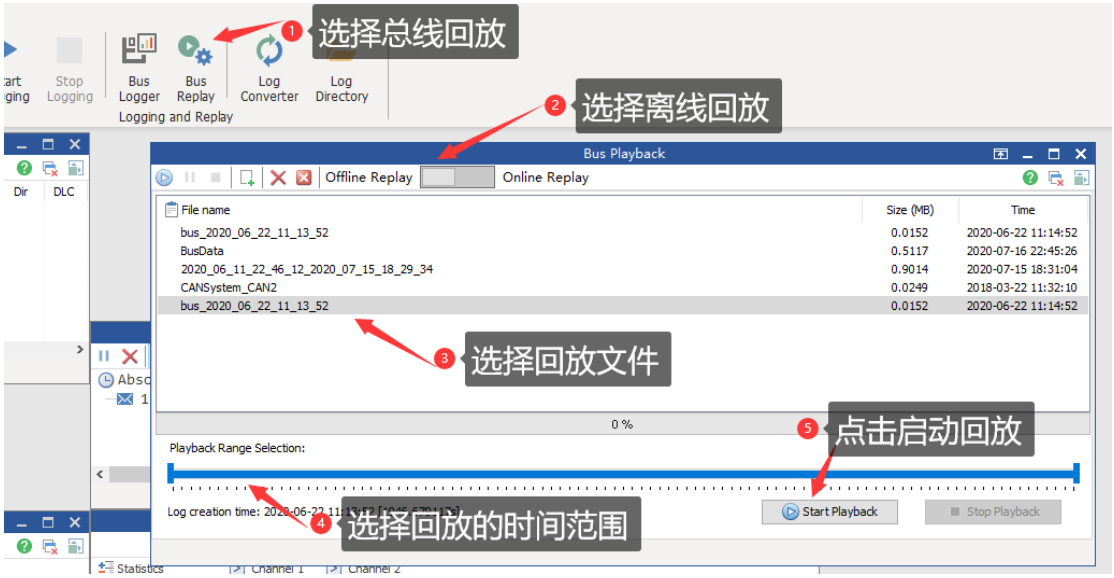
1.13.1. 支持格式

TSMaster 的数据回放默认支持 blf 格式（未来会增加对其他格式的支持）。如果需要分析其他数据格式的 log 文件，需要通过文件转换器从其他格式转成 blf 格式。格式转换步骤见前一个章节。

1.13.2. 离线回放

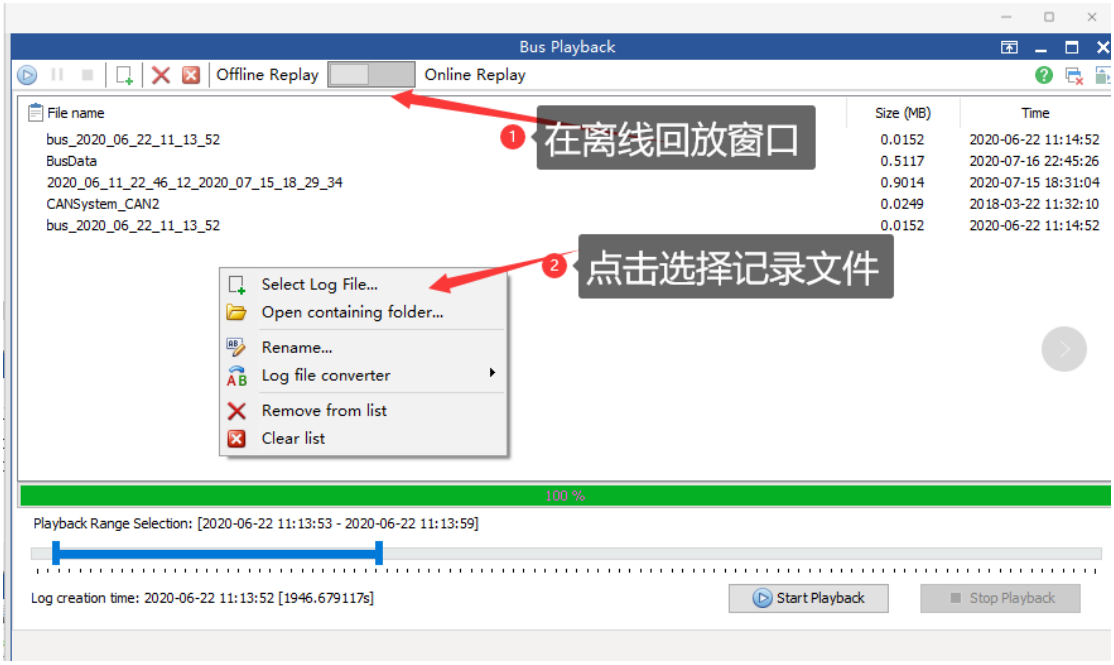
离线回放也就是通常所说的查看记录报文。离线回放完全模拟接收报文的过程，用户可以到 Trace 窗口中直接查看报文记录。同样的，Trace 窗口的所有属性，比如过滤等都是有效的。

1.13.2.1. 离线回放基本步骤：



1.13.2.2. 添加回放文件

1.13.2.2.1. 从回放窗口添加



1.13.2.2.2. 直接拖拽添加

在桌面上选择一个 Log 文件，拖拽进入 TSMaster 软件区域，放掉鼠标，TSMaster 即自动回放该报文，同时把该报文添加到报文回放管理窗口中。

1.13.2.3. 选择报文范围

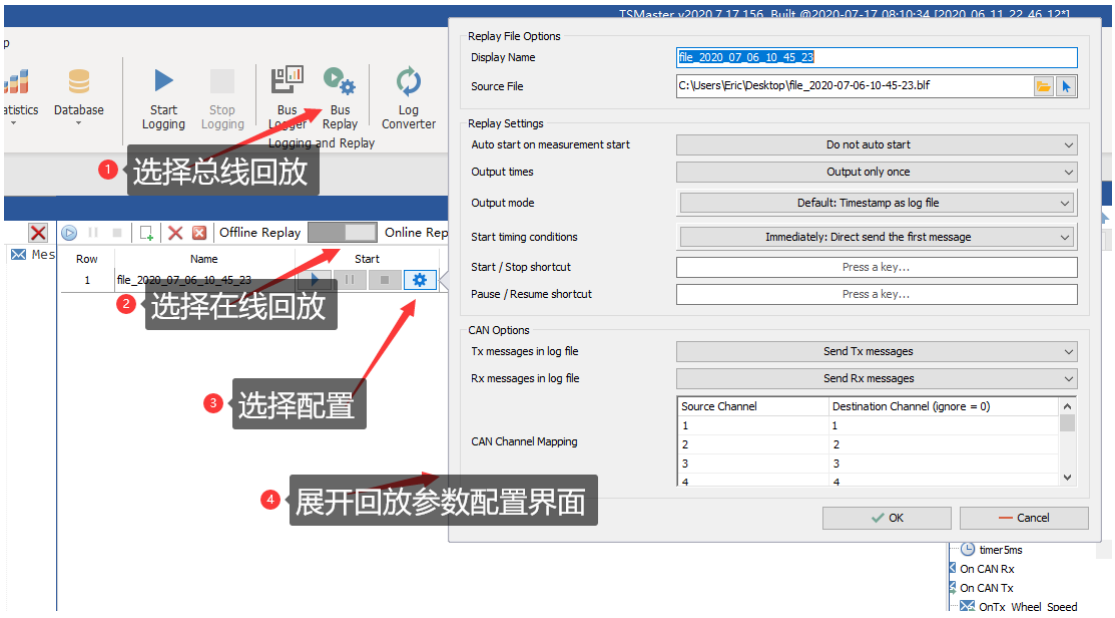
因为 Trace 窗口一个屏幕最多一次呈现 9999 帧报文，因此，在分析记录文件的时候，需要合理选择报文范围。

在新的版本中，将在报文回放模块中增加脚本模块，给报文回放分析提供更大的灵活性。

1.13.3. 在线回放

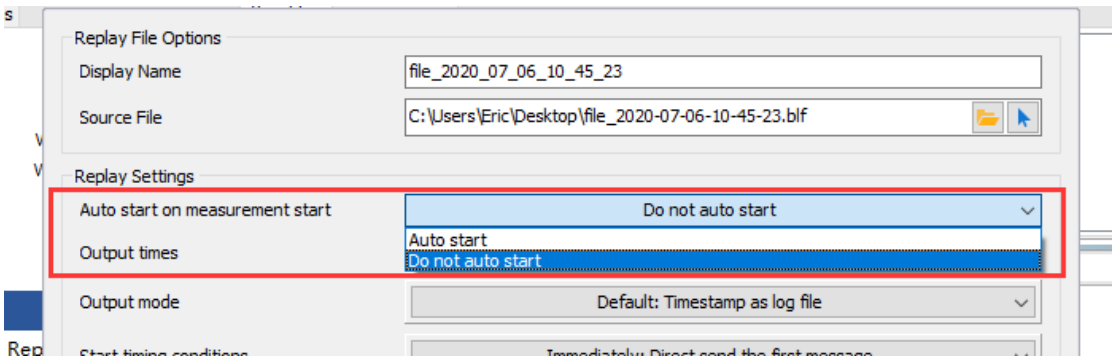
1.13.3.1. 在线回放配置

在线回放又被通俗的称为“数据回灌总线”，为了给用户提供尽可能大的灵活性，提供了如下的配置界面：



如上配置界面，主要包含如下的配置参数：

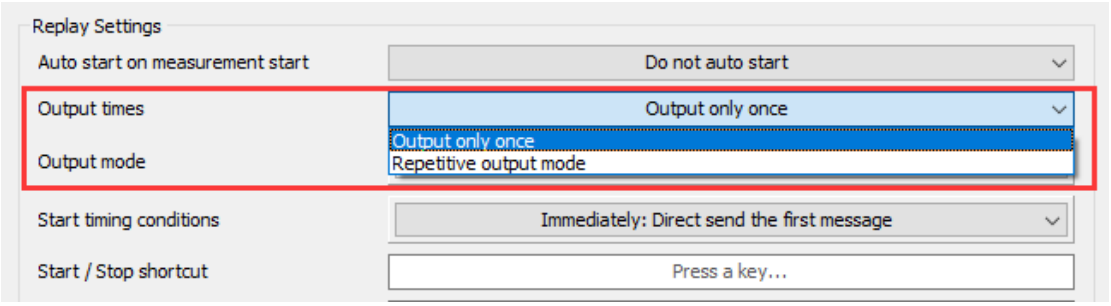
1.13.3.1.1. 是否自动启动在线回放



AutoStart:在设备连接之后就自动启动报文的回放。

Do not auto start:在设备连接之后并不立即启动报文回放，而由用户进入回放界面中自己启动。

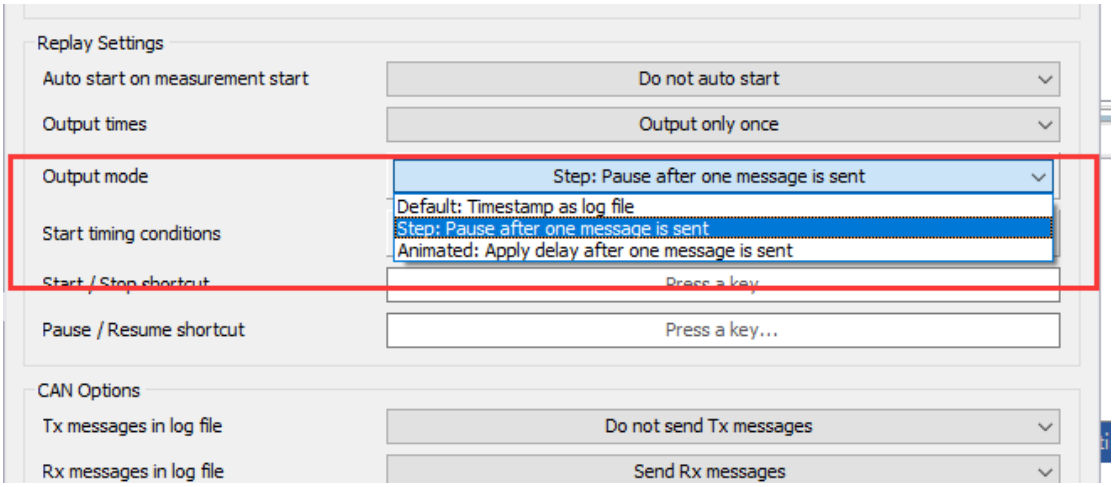
1.13.3.1.2. 输出次数选择



Output only once：只回放一次即可

Repetitive output mode: 循环回放报文记录

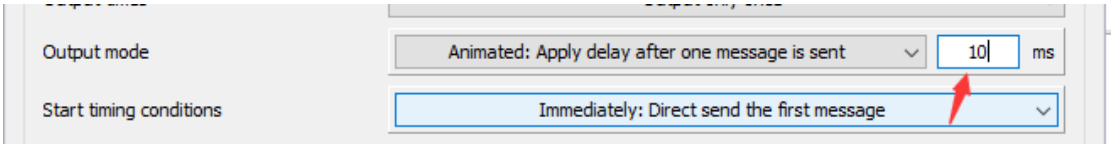
1.13.3.1.3. 输出模式选择



Default：TimeStamp As Log File：基于 Log 文件中的时间戳进行报文回放。

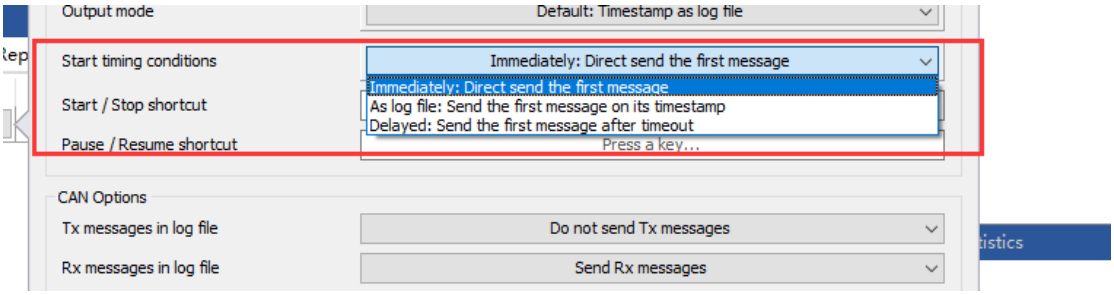
Step: Pause after one message is sent：需要用户点击，一次只发送 Log 文件中的一帧报文。

Animated：Apply delay after one message is sent：不急于 Log 本身的时间戳，而是用户设置一个报文事件间隔，按照这个时间间隔进行报文的回放。如下所示：



就表示报文之间按照 10ms 的时间间隔进行回放。

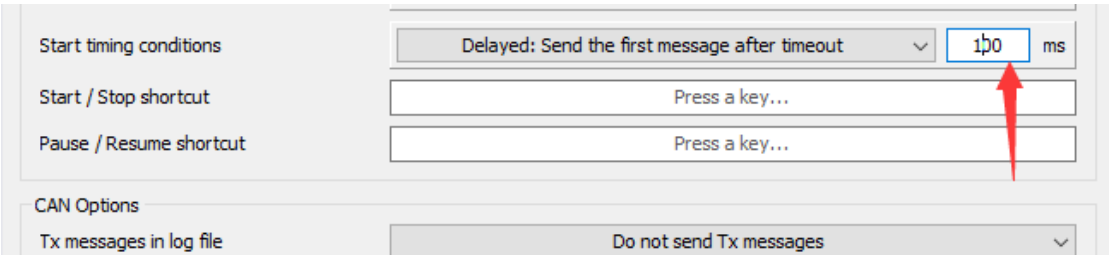
1.13.3.1.4. 启动时间模式选择



Immediately：Direct send the first message：用户选择回放报文过后，立即就开始回放报文。

As Log File: Send the first message on its timestamp: 根基 Log 文件的时间戳来确定从什么时间开始播放报文。比如如果 Log 文件中第一帧报文的时间戳是 15s，则连接设备过后，等到 15 过后才开始回放报文。

Delayed：send the first message after timeout：用户人为配置一个延迟时间。当连接设备过后，延迟设置的时间间隔后，开始回放报文。如下所示：

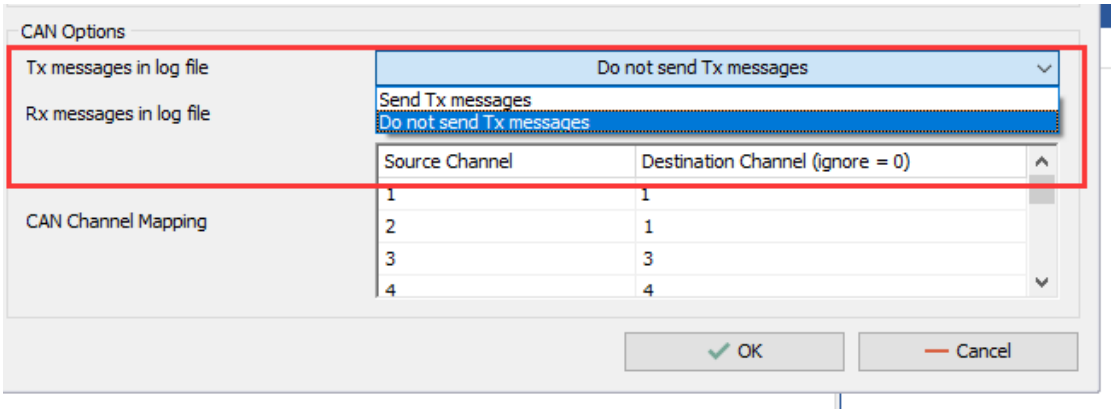


就表示用户启动报文播放过后，延迟 100ms 开始实际回放报文。

1.13.3.1.5. 启动/暂停快捷键

让用户设置启动/暂停回放的快捷键，如上所示：用户按下键盘上的 S 键时，启动回放；用户按下键盘的 P 键时，暂停回放。

1.13.3.1.6. 选择回放 TX/RX 报文

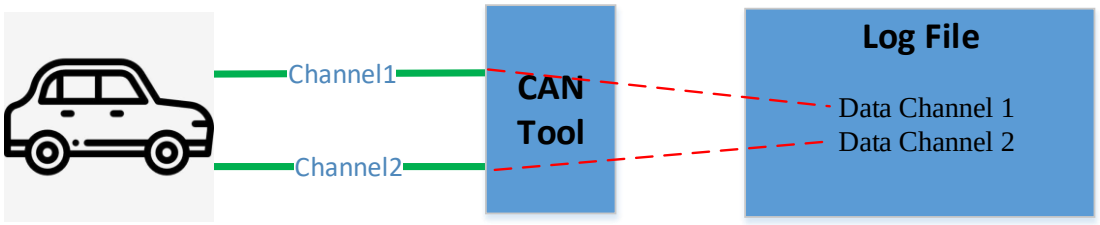


Send Tx messages: 回放中包含方向为发送的报文
Do not send Tx messages: 回访中不包含方向为发送的报文

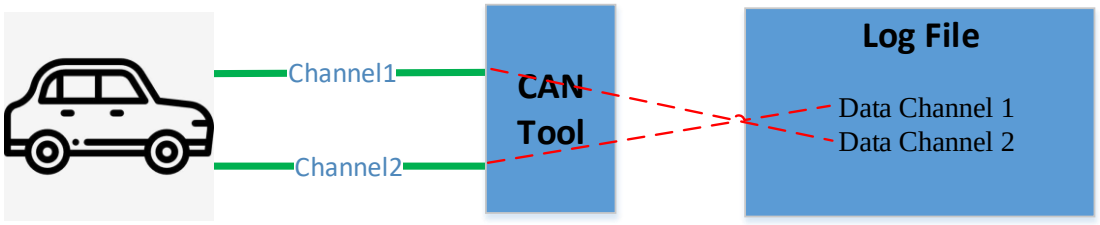
1.13.3.1.7. 回放通道选择

为了给用户提供尽可能大灵活性，TSMaster 的回放模块提供了回放通道的映射。主要为了解决以下一些应用场景的问题：

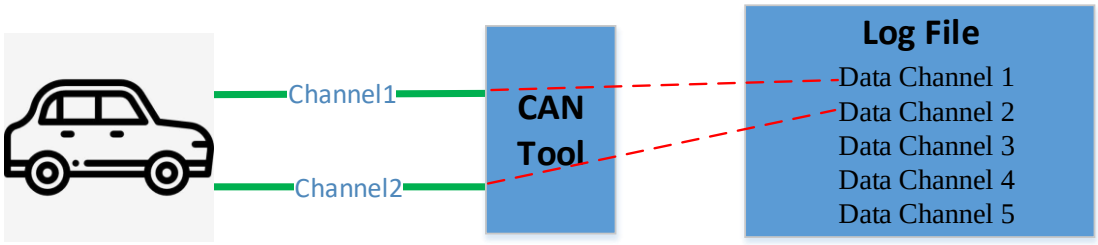
1. 物理通道已经连接好，但是想灵活切换通道，如下图所示：



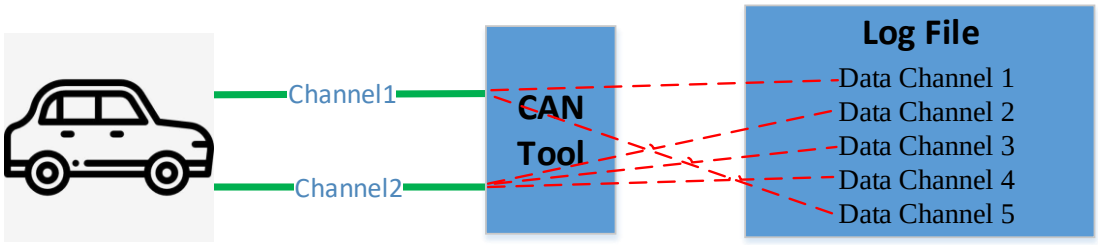
Log 报文中有通道 1 的数据，也有通道 2 的数据，最好理解的是通道 1 的数据在工具端对应的通道上播放。如果数据通道 2 的数据需要到 CAN 工具的通道 1 上面播放，数据通道 1 需要到 CAN 工具的通道 2 上面播放，则需要用到映射，如下所示：



2. 实际通道数不够：在原始的 Log 报文中用到了 1,2,3,4,5 五个通道，但是目前手上只有 2 个通道，如下所示：

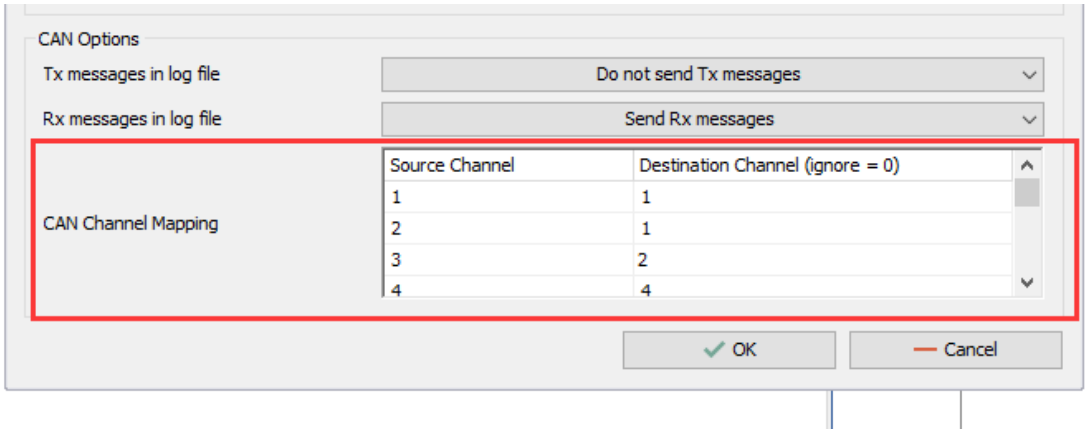


这种情况下，数据通道 3,4,5 的数据就没有通道播放了。如果基于通道映射的方式，用户可以选择数据通道到任意 CAN 工具的通道上播放，如下所示：



通道上述映射，就把数据通道 2,3,4 的数据映射到 CAN 工具的通道 2 上进行播放，把数据通道 1,5 的数据映射到 CAN 工具的通道 1 上进行播放。

回放通道映射配置界面如下所示：



Source Channel：Log 文件中的数据通道

Destination Channel：TSMaster 中 CAN 工具的数据通道

如上所示的配置就表示：Log 文件中数据通道为 1 和 2 的数据在 CAN 工具的通道 1 中回放；数据通道为 3 的数据在 CAN 工具的通道 2 中回放。

1.13.4. 应用案例介绍

1.13.4.1. 自动回放屏蔽报文

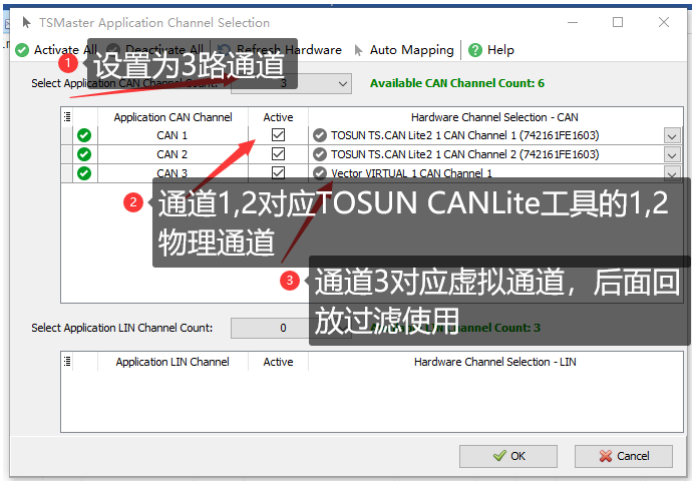
根据前面的介绍，自动回放报文提供了选择通道，选择回放 RX，TX 等机制。但是用户使用过程中，往往还需要选择性的回放一部分报文，或者选择性的屏蔽部分报文。因为回放报文数量可能会很多，因此，基于 ID 的回放过滤，目前没有做到在线回放模块里面。但是通过 TSMaster 的 C 脚本工具，用户一样可以基于 ID 屏蔽报文的在线回放的功能。

- **基本思路：**

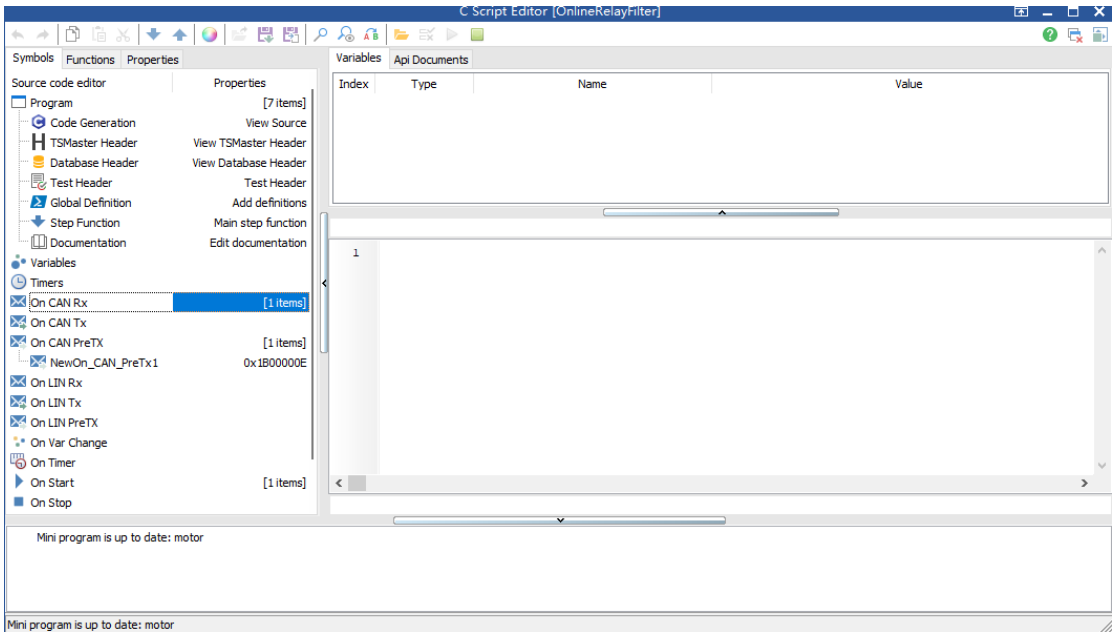
把需要屏蔽的 ID 的报文发送到虚拟通道上，这样这些报文就不会真实回放到物理通道上。

- **操作步骤详解：**

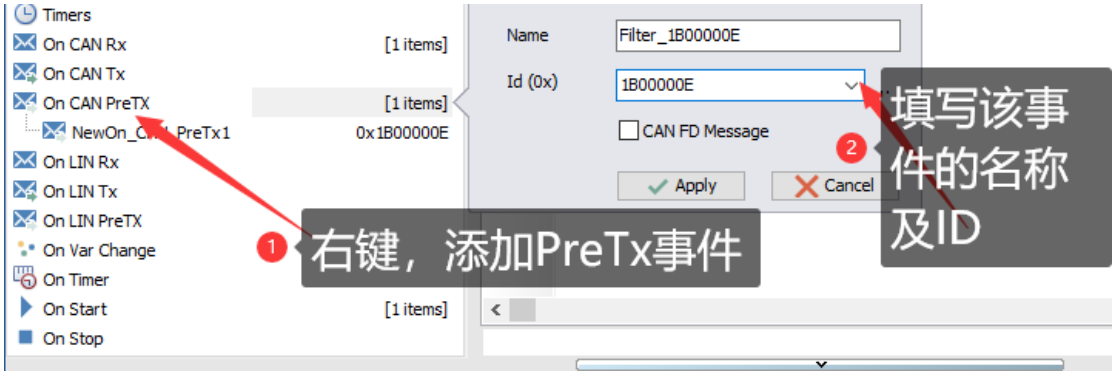
1. 在 TSMaster 硬件配置中，增加一路通道，并选择该通道为虚拟通道，如下所示：



2. 新建脚本模块，取名为 OnlineRelayFilter：创建脚本模块并命名等方式见“C 脚本”章节。创建后脚本如下：

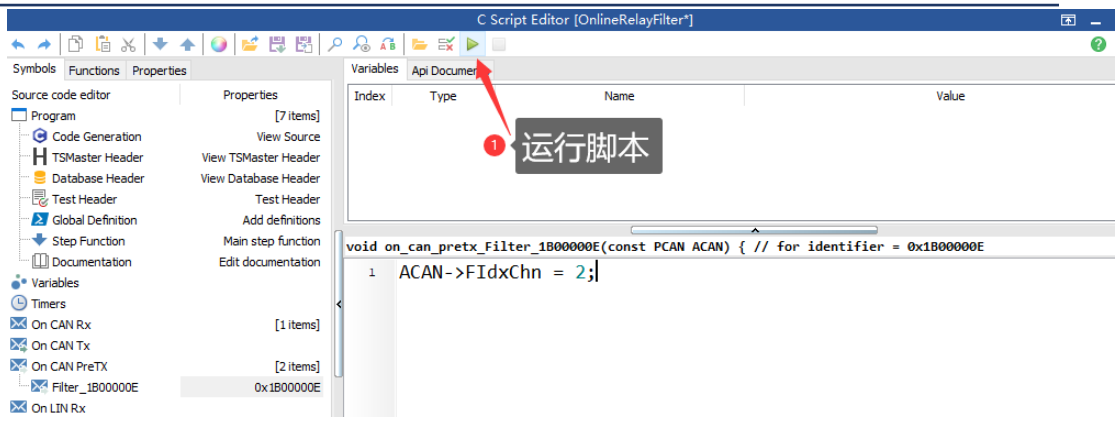


3. 新建 CAN 报文 PreTX 事件（该事件在报文发送到总线上之前被调用）。如果要屏蔽 ID = 0x1B00000E 的报文，则创建该报文的 PreTX 事件，如下所示：



4. 选中该事件，修改该报文数据的发送通道为通道编号 2（0,1,2），也就是通道 3.

5. 点击运行脚本，如下：

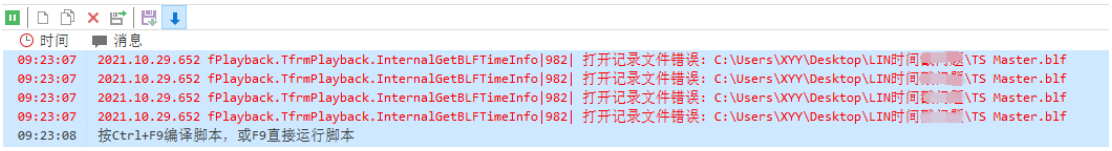


6. 此时再点击在线回放，就可以看到 0x1B00000E 报文被发送到了虚拟通道 3，其他报文发送到物理通道 1 和 2，通过这种方式实现了对在线报文的过滤。依次类推，对需要过滤的报文通过在脚本中添加 Pre_TX 事件就可以达到想要的效果。

1.13.5. 释疑

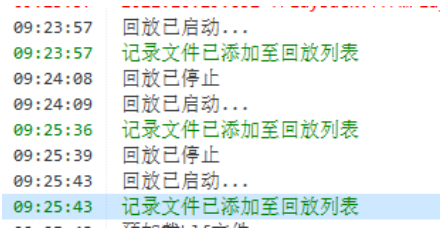
1.13.5.1. Blf 文件（文件名带空格）无法加载

当把 blf 加载到 TSMaster 回放（离线/在线）模块中，发生加载错误，系统消息如下所示：



```
09:23:07   2021.10.29.652 fPlayback.TfrmPlayback.InternalGetBLFTimeInfo|982| 打开记录文件错误: C:\Users\XYY\Desktop\LIN时间[redacted]\TS Master.blf
09:23:07   2021.10.29.652 fPlayback.TfrmPlayback.InternalGetBLFTimeInfo|982| 打开记录文件错误: C:\Users\XYY\Desktop\LIN时间[redacted]\TS Master.blf
09:23:07   2021.10.29.652 fPlayback.TfrmPlayback.InternalGetBLFTimeInfo|982| 打开记录文件错误: C:\Users\XYY\Desktop\LIN时间[redacted]\TS Master.blf
09:23:07   2021.10.29.652 fPlayback.TfrmPlayback.InternalGetBLFTimeInfo|982| 打开记录文件错误: C:\Users\XYY\Desktop\LIN时间[redacted]\TS Master.blf
09:23:08   按Ctrl+F9编译脚本，或F9直接运行脚本
```

原因：blf 文件名中间不能有空格。如上图所示，修改文件名称为 TSMaster.blf，则加载成功，如下所示：



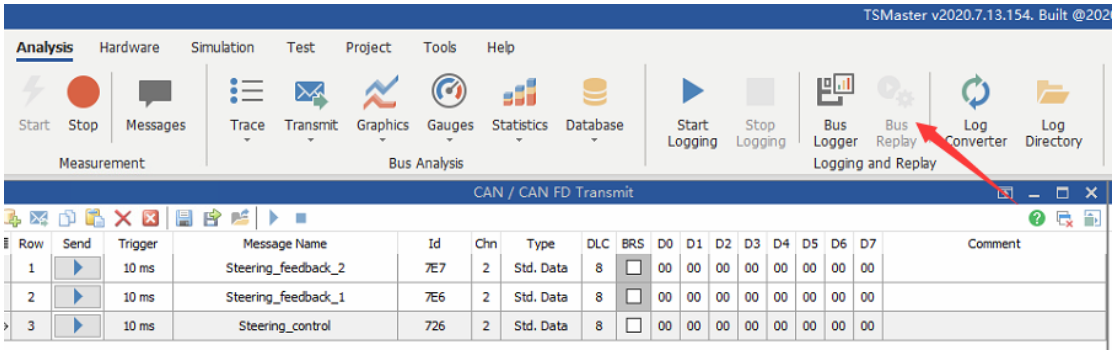
```
09:23:57   回放已启动...
09:23:57   记录文件已添加至回放列表
09:24:08   回放已停止
09:24:09   回放已启动...
09:25:36   记录文件已添加至回放列表
09:25:39   回放已停止
09:25:43   回放已启动...
09:25:43   记录文件已添加至回放列表
```

1.13.5.2. 想按照采集的时间回放报文

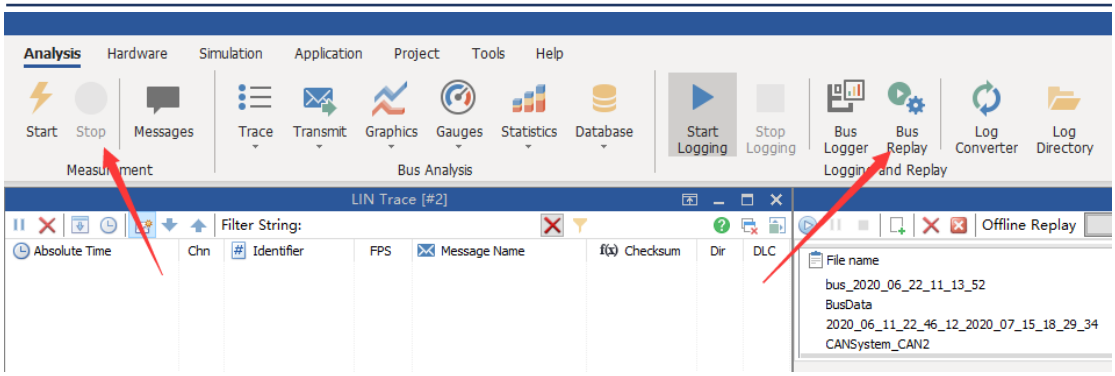
如果想按照采集时间戳回放报文，请选择在线回放。离线回放主要用于查看报文，要求就是回放速度越快越好。如果想按照采集的时序回放报文，则直接采用在线回放的方式，具体的设置见在线回放章节。

1.13.5.3. 总线回放按钮为什么是灰色（不使能状态）

在总线连接工作状态，不能进行报文记录的回放。需要点击 Stop 按钮，停止工作状态，BusReplay 才能变成使能状态，允许用户添加回放的报文。



Row	Send	Trigger	Message Name	Id	Chn	Type	DLC	BRS	D0	D1	D2	D3	D4	D5	D6	D7	Comment
1	▶	10 ms	Steering_feedback_2	7E7	2	Std. Data	8	☐	00	00	00	00	00	00	00	00	
2	▶	10 ms	Steering_feedback_1	7E6	2	Std. Data	8	☐	00	00	00	00	00	00	00	00	
3	▶	10 ms	Steering_control	726	2	Std. Data	8	☐	00	00	00	00	00	00	00	00	

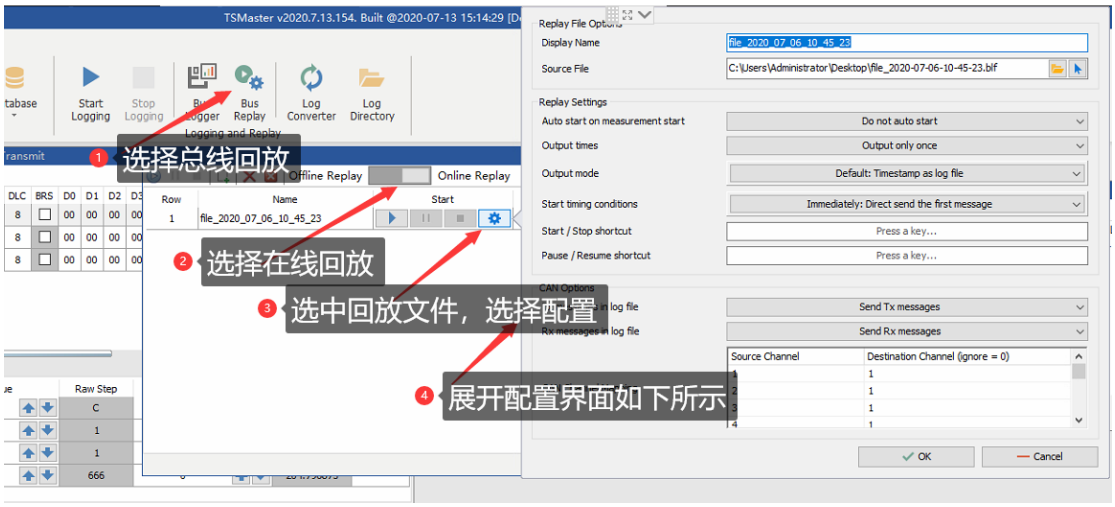


停止按钮->使能总线回放按钮

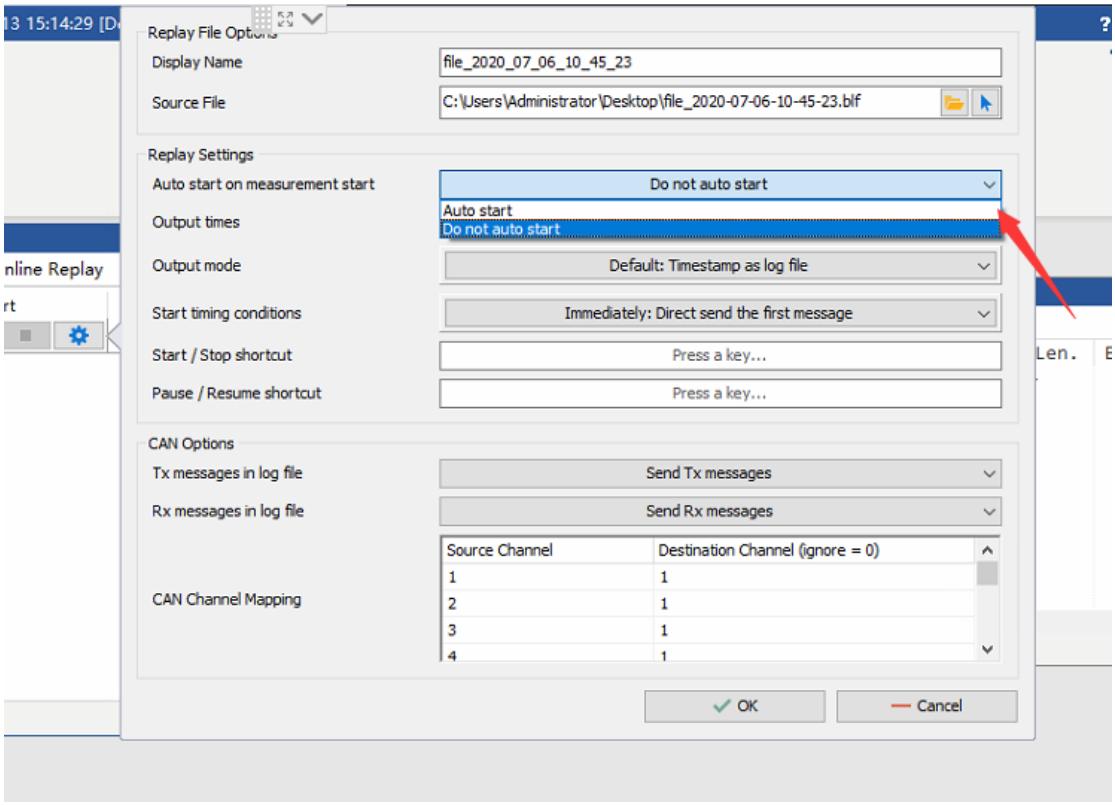
1.13.5.4. 为什么 TSMaster 连接设备后立即往总线发送报文？

TSMaster 提供了在线回放数据的功能，为了支持用户挂上总线就开始回放的需求，在回放设置中添加了一个连接总线过后自动回放的功能，如下图所示：

首先，通过如下步骤进入在线回放配置界面：



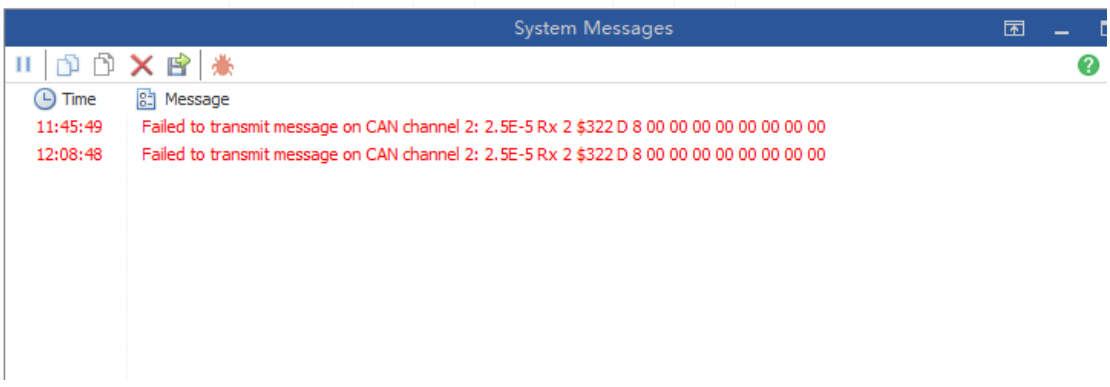
配置界面中，选择在启动设备瞬间是否自动启动报文回放，如下所示：



如果选择 do not auto start，则连接设备的时候不会自动播放报文。

1.13.5.5. 回放提示通道错误失败

在线回放中，报文播放了一段时间就提示播放失败，通道错误，错误情况如下所示：

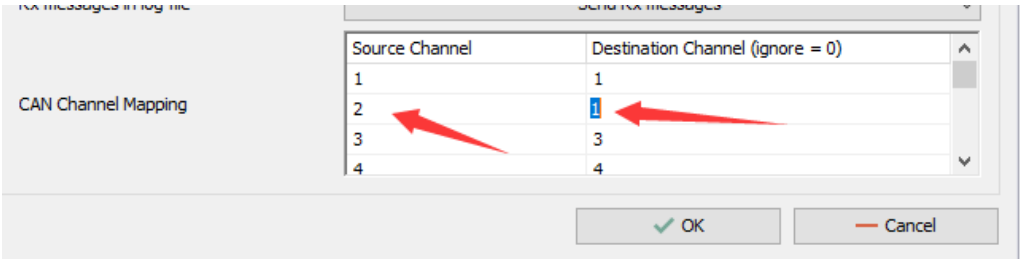


触发原因：通道映射错误。

Log 报文中的数据通道没有正确的映射到 CAN 工具的通道上。比如上面报的错误，Log 文件中使用了数据通道 2，数据通道 2 映射到了 TSMaster 中的 CAN 工具的通道 2，但是实际上 TSMaster 中没有配置 CAN 通道 2，就造成发送失败。具体通道映射，见前序章节：在线回放配置->回放通道选择

解决办法：

在进入在线回放配置界面，配置通道映射。如下所示：



如上所示，Desination Channel 中实际上不存在通道 2，因此，把 Source Channel =2 的通道也配置到 Destination Channel = 1 上。再次启动回放，不再出现此错误。

1.13.5.6. 在线回放直接错误帧

如果用户回放的时候，总线直接错误帧。很可能的情况是，在 Log 文件中，不同的数据通道中有同样 ID 的报文，结果播放的时候，这些报文在同一条 CAN 总线上面通过不同的通道播放出来，造成帧 ID 冲突，引起错误帧进而无法正常的回放数据。

1.13.5.7. 在线回放卡死

在线回放如果出现卡死情况，请检查是否使用了虚拟通道。有些电脑平台上虚拟通道支持有些问题，碰到这种情况，用户尽量避免使用虚拟通道。或者插上实物通道进行报文的在线回放。

1.13.5.8. 为什么出现回放数据跟三方工具对不上的情况

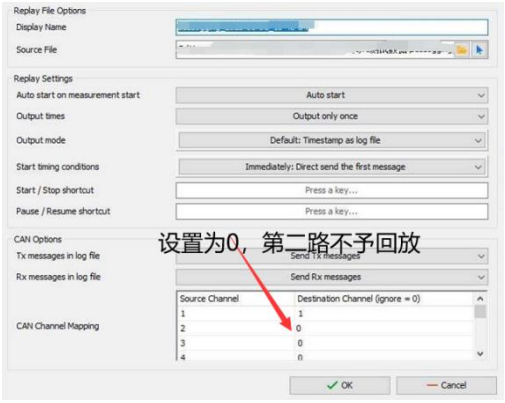
情况描述：

通过在线回放数据，监测信号值，发现总线报文上有信号值比如信号 A 值，跟第三方监测工具（如 OE）上解析的 A 值不一致。比如 TSMaster 上 A 值为 0.0，OE 上解析的 A 值为 0.3.

原因分析：

经过分析后，发现回放的 blf 文件中包含来自两个通道的数据（Channel1 和 Channel2），**两个 Channel 中有同样 ID 的报文（比如都包含信号 A 的报文）**，而且来自通道 2 中的报文值全部为 0，因此，回放的时候，从通道 2 中解析出来的信号 A 值就全部为 0。**造成用户错误的把通道 2 中的值跟第三方工具解析出来的值进行比较。**

解决办法：因为通道 2 中的信号值不是有效数据，回放过程中屏蔽掉通道 2 的回放即可。如下所示：



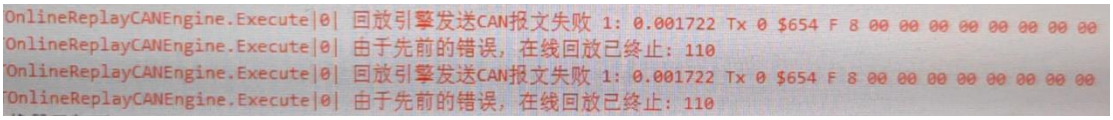
Tips：

用户如果分析数据信号如果发现疑问点，请注意报文的通道，时间戳，这样数据才有可比性。

1.13.5.9. CAN 通道已就绪，软件在线回放失败

情况描述：

载入 blf 文件，报文里面只有通道 1 的报文，但是回放还是失败，错误提示如下：



原因分析：

在线回放是把记录的报文反向回灌回物理总线上。本次案例中 Blf 文件记录的报文是 fd 报文，但是硬件是普通 CAN（classic CAN），因此造成在线回放失败。

解决办法：

更换支持 FDCAN 的硬件，如果硬件已经支持，则把该硬件工作模式设置为 FDCAN 模式。然后重新回放即可。

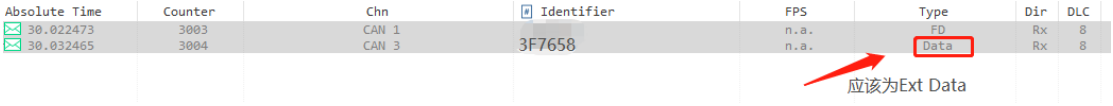
1.13.5.10. 进行数据分析的时候，其中一个 ID 离线回放的时候可以看到，在线回放的时候看不到了

错误情况描述：

载入 blf 文件进行分析，发现在离线回放的时候可以看到 ID = 0x3F2758 的报文，但是在线分析的时候，怎么也看不到这一帧报文。

原因：

该报文属性错误。该报文 ID = 0x3F7658，应该属于扩展帧报文，但是该报文自带的属性显示该报文为标准帧报文。在离线回放的时候如下图所示：



离线回放的时候，直接原原本本呈现 blf 的数据情况，不做矫正，便于用于发现问题。

在线回放的时候，TSMaster 会根据报文的属性矫正 ID，比如上面的情况。起 ID 属性属于标准 ID，则实际发送的 ID 就会被矫正为：

ID = SRCID & 0x7FF = 0x3F7658 & 0x7FF = 0x658.

所以实际在线回放的时候，用户就看不到这一帧报文 0x3F7658，而是多了一帧 0x658 的标准帧报文。

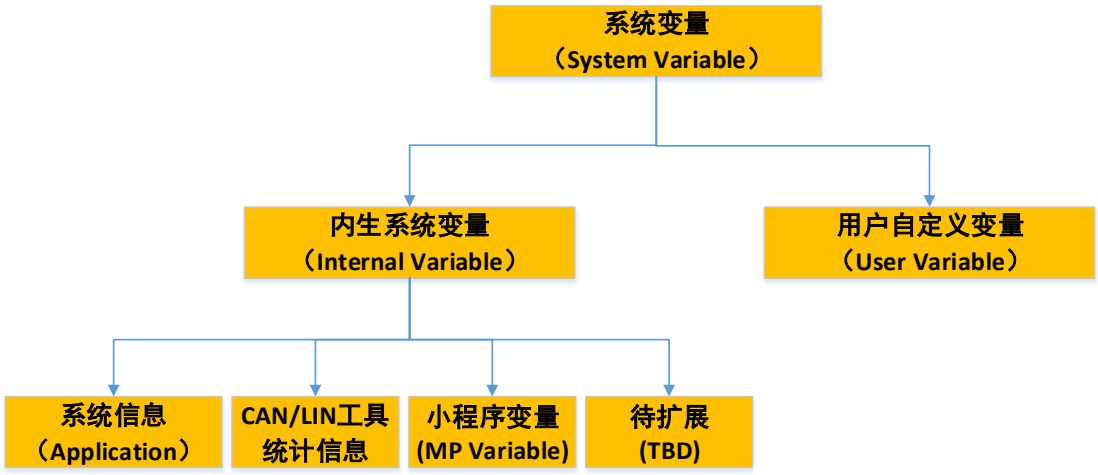
1.14. 系统变量

1.14.1. 综述：

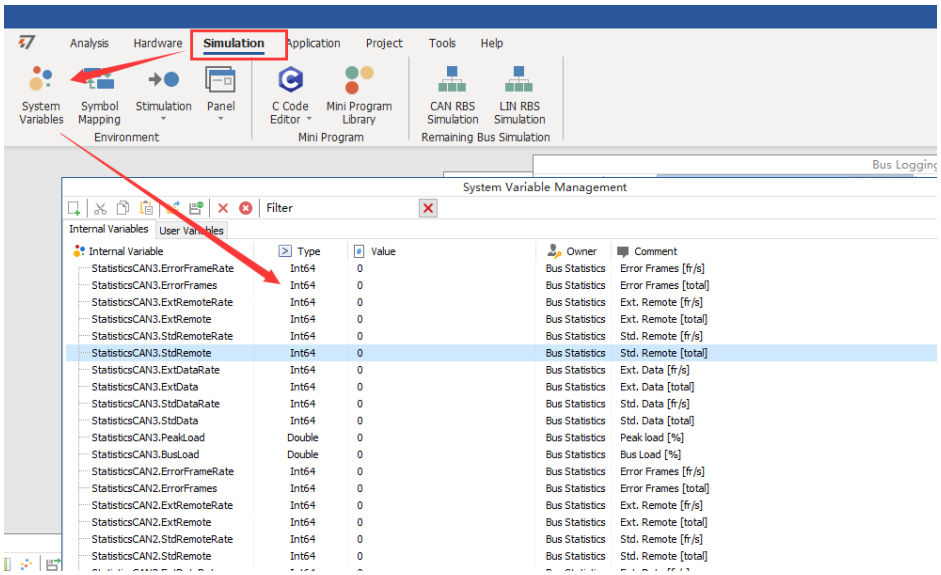
用户在使用 Panel，脚本，标定，诊断或者其他扩展功能的时候，能够直接访问的变量被定义为系统变量。根据系统变量生成的原理，主要分为两个类型：Internal Variable(内生系统变量)和 User Variable(用户自定义系统变量)。他们的主要区别是：

- 内生系统变量是系统自动生成，自动释放的，用户不能直接对其进行增删操作；
- 用户定义变量是用户自己创建和管理的

系统变量系统架构如下图所示：



用户要查看当前可用的系统变量，需要根据路径 Simulation->System Variable 打开系统变量管理界面，如下图所示：



1.14.2. Internal Variables（内生系统变量）

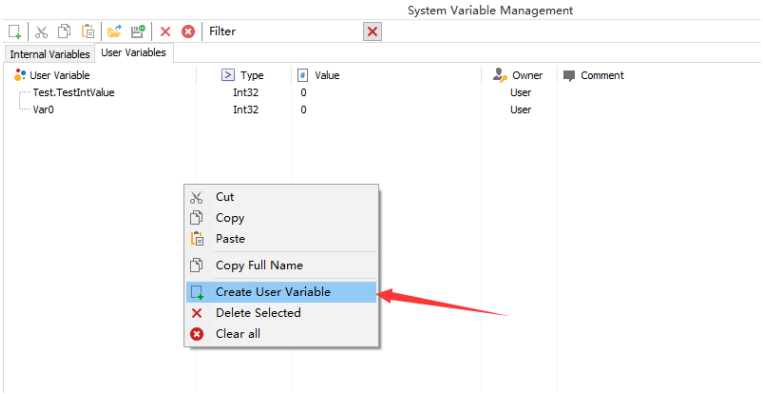
内生系统变量是跟随系统**自动生成**，自动释放的。**常见的**内生系统变量如下图所示：1. 系统信息；2.设备统计信息；3. 小程序变量。

以设备统计信息为例，如果添加了 CAN1 设备，才会动态生成 StaticsCAN1 相关的统计信息；如果删除 CAN1 设备，这些统计信息会消失。随着 TSMaster 软件系统的升级，后续会增加更多的这种动态生成和加载的数据类型。其中关于小程序变量的详细信息，请见章节：C 脚本->C 脚本运行机制->小程序变量。

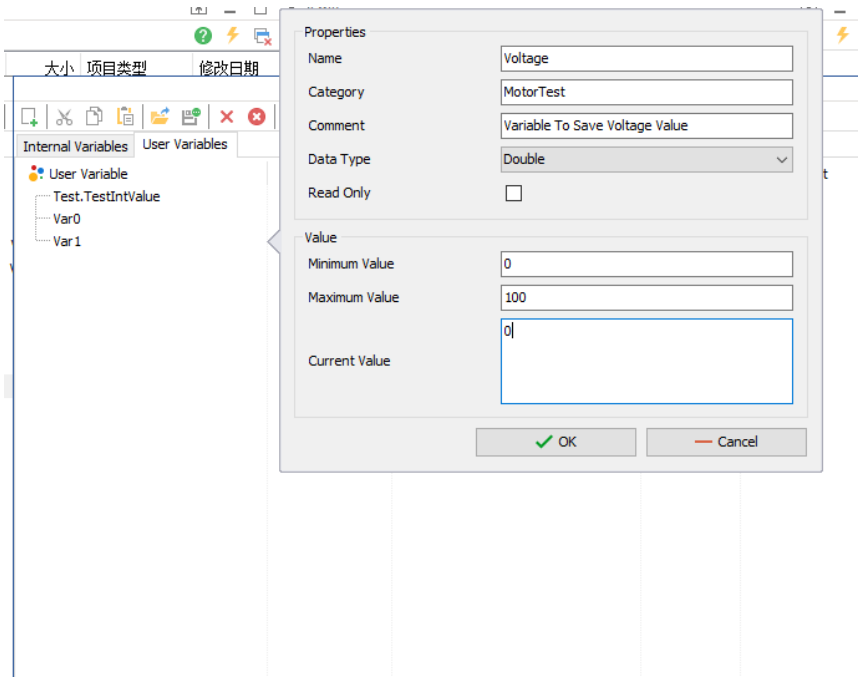
1.14.3. User Variables (用户定义系统变量)

这种类型的变量是用户自定义的，用户可以进行增删等操作。新增用户变量流程如下所示：

- 第一步：在系统变量管理界面，右键，点击：创建用户变量



- 第二步：设置用户变量的属性



Name: 变量名称。

Category: 变量所属分组，便于用户管理变量，同名变量可能用于不同的应用程序中。

Comment: 对变量增加一些评论和描述。

Data Type: 变量类型，包括整形和 Float 等类型。详细可展开查看。

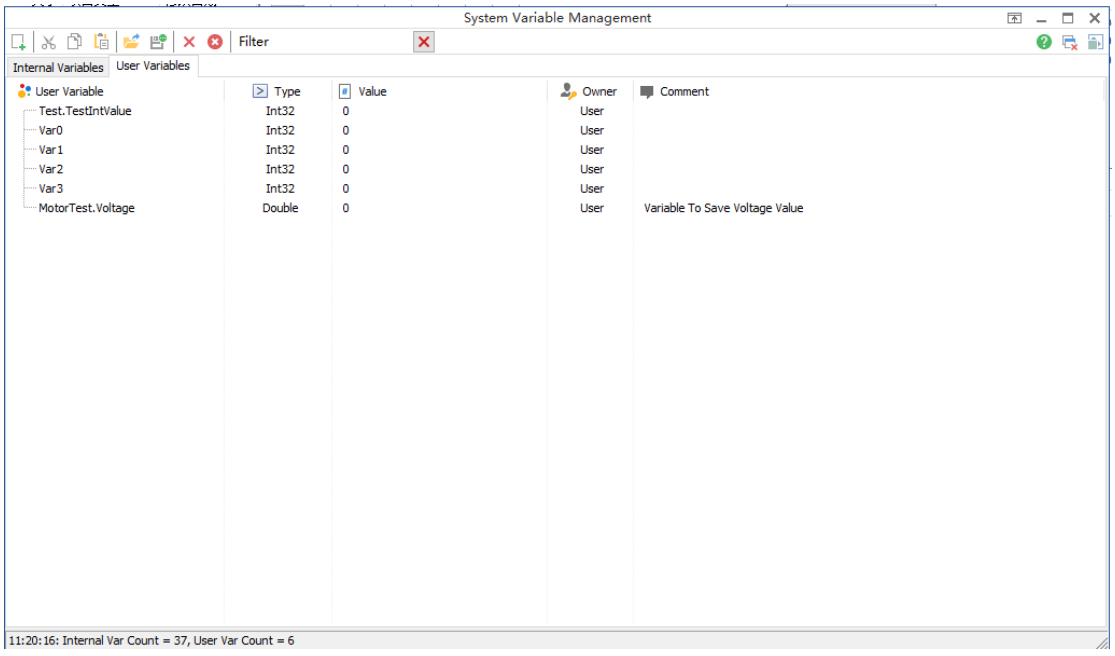
ReadOnly: 该变量是否只读，如果是只读的，用户无法对其进行修改。

Minimun Value: 该变量允许的最小值。

Maximum Value: 该变量允许的最大值。

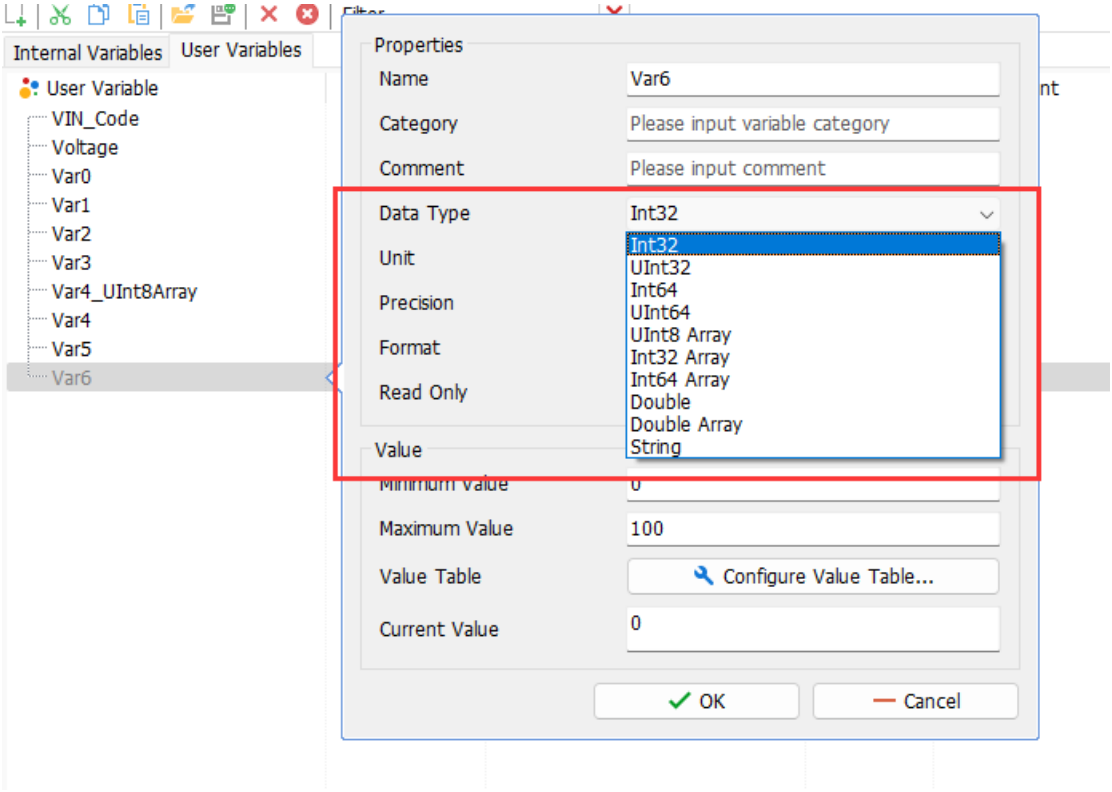
Current Value: 该变量当前值。

- 添加到系统后如下图所示：



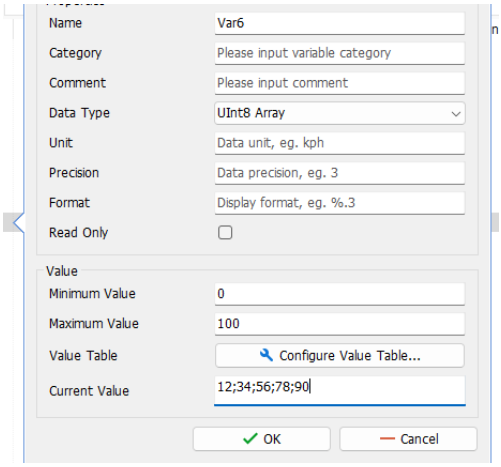
1.14.4. 系统变量数据类型

TSMaster 系统变量主要包含如下数据类型：Int32，UInt32，Int64，UInt64，UInt8 Array，Int32 Array, Int64 Array, double ,double Array, string，如下所示：



其分别代表的意义如下：

- Int32：有符号 32 位类型，可以涵盖住 Int8，Int16 等类型。
- UInt32：无符号 32 位类型，可以一并涵盖住 UInt8，UInt16 等数据类型。
- Int64：有符号 64 位数据类型。
- UInt64：无符号 64 位数据类型。
- UInt8 Array：无符号 8 位数组，也就是最常用的 Byte 数组。对于数据类型的数据，在赋值的时候，**数组元素之间通过';'隔开**，如下所示，其等同于数组定义为：UInt8 Var6[5] = {12,34,56,78,90};

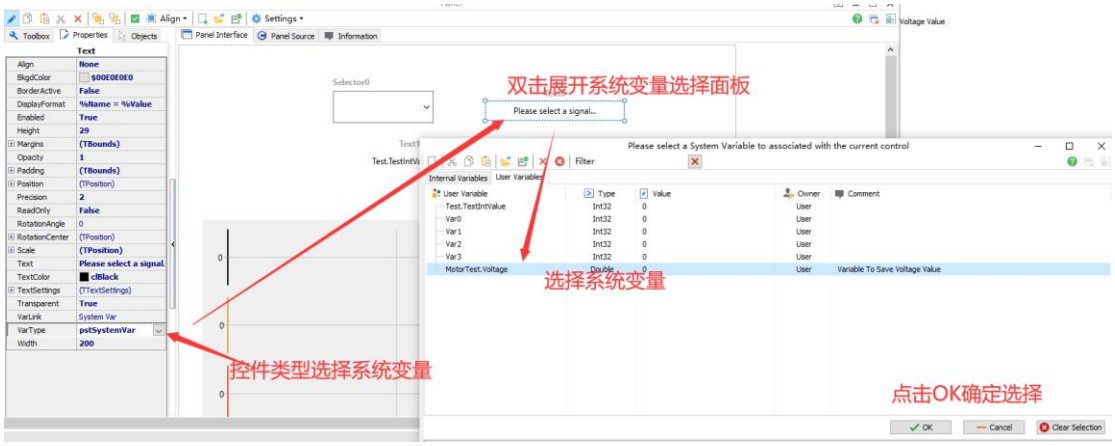


- Int32 Array：
- Int64 Array：
- Double：
- double Array：
- string：

1.14.5. 访问系统变量

无论是内生变量，还是用户定义变量，其访问方式是完全一样的。主要介绍两种应用场景：1. Panel 关联系统变量；2. 通过脚本读写系统变量。

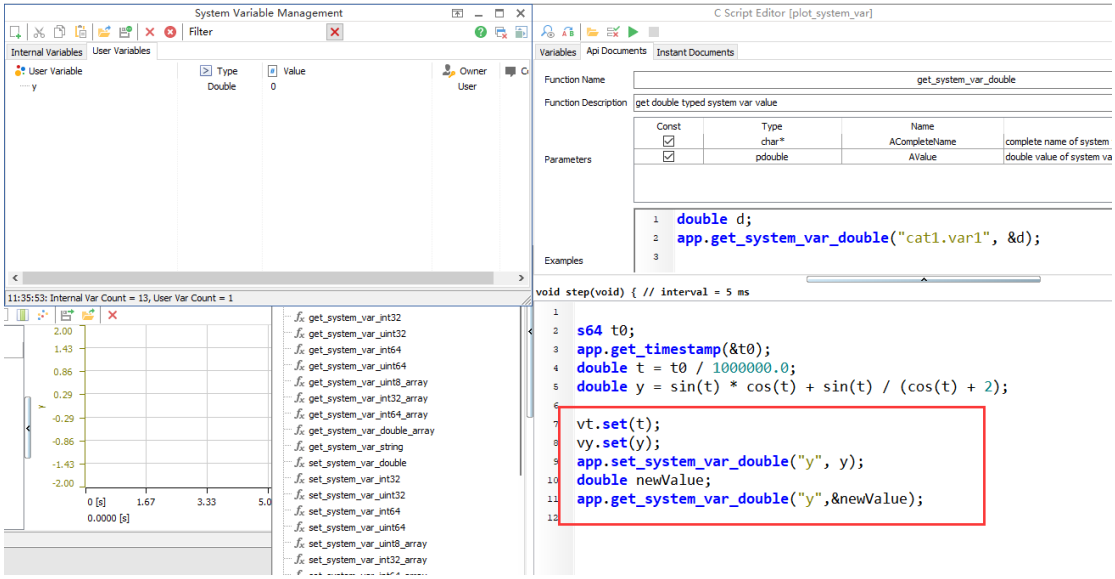
1.14.5.1. Panel 关联系统变量



Panel 关联系统变量流程如上图所示：

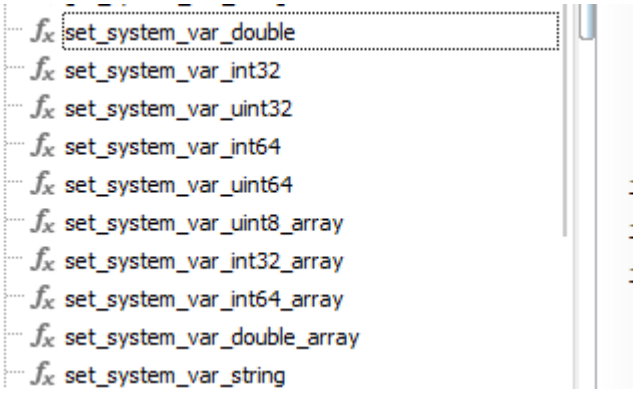
1. 控件关联变量类型选择为系统变量（SystemVar）。
2. 双击展开变量选择面板。
3. 在面板上选择内置系统变量和用户定义系统变量即可。

1.14.5.2. 脚本读写系统变量

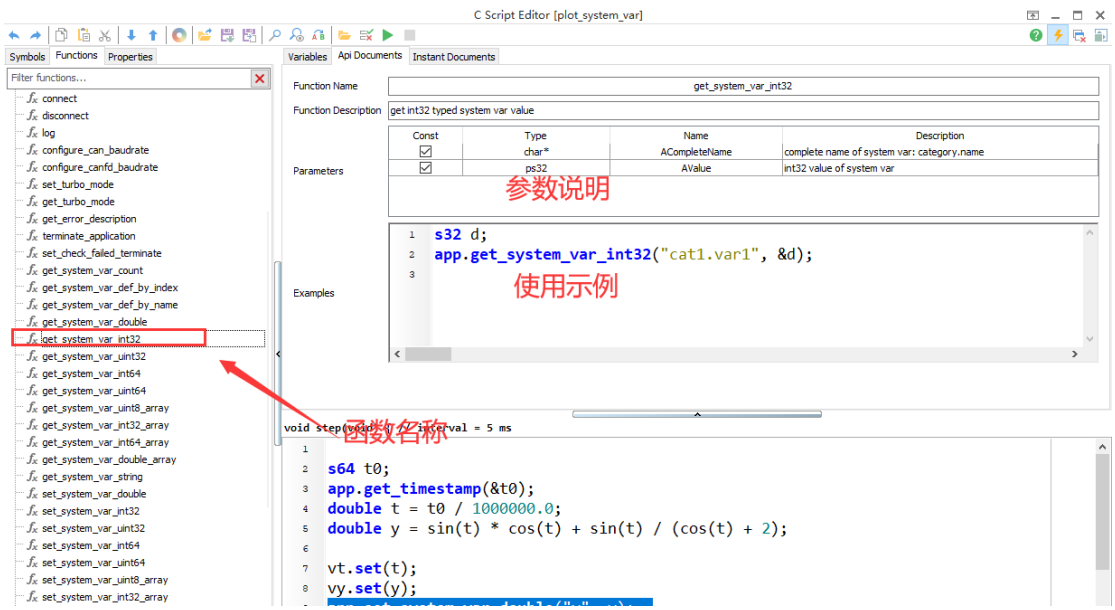


如上图所示，简单来说，系统变量的访问主要用到两类函数，这两类函数位于小程序 Function 的 APP 目录下面。

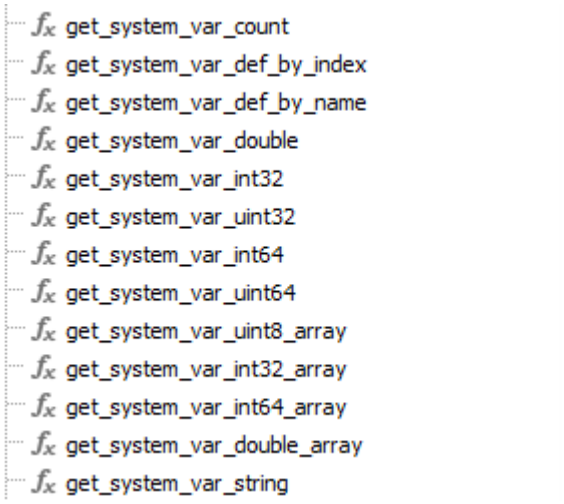
- **写入变量：**



关于函数参数等，单击选中该函数过后，右上角会呈现该函数的使用说明，如下图所示：



➢ **读取变量：**

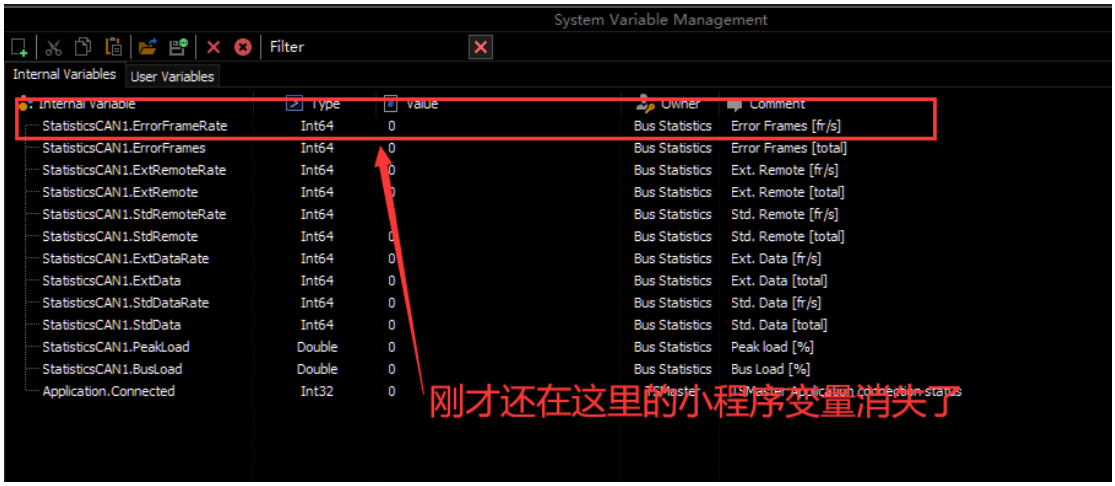


基于 set_system 和 get_system 函数，赋予了脚本系统跨脚本访问脚本内部变量的能力。

1.14.6. 释疑

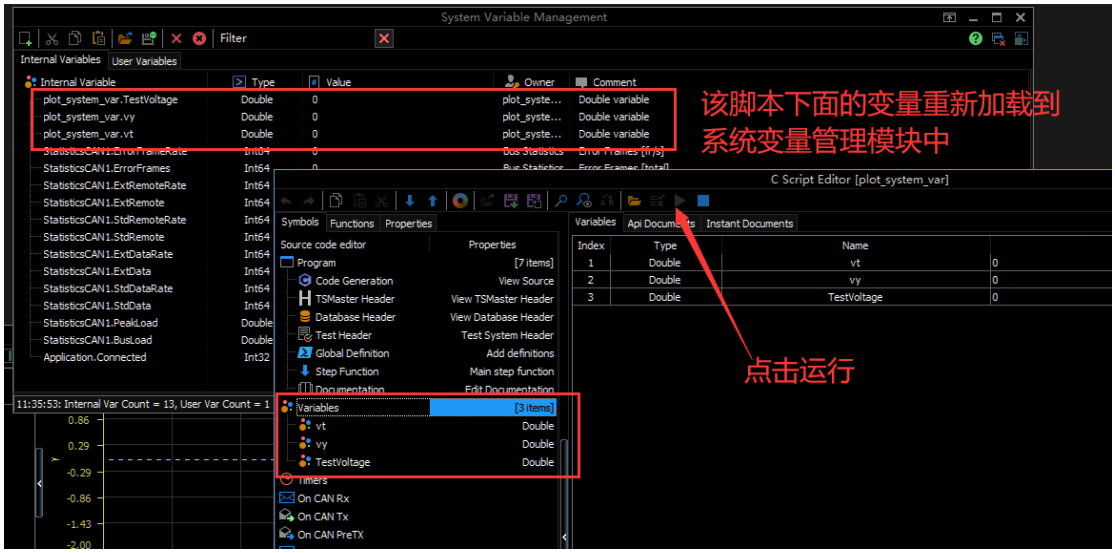
1.14.6.1. 看不到小程序变量

用户想访问小程序变量，比如在 Panel 中关联小程序变量的时候，发现面板上面没有这个变量。如下图所示：



这是因为，小程序变量是动态加载的。TSMaster 支持任意多个小程序运行，每个小程序里面都可能会有自己的变量，因此，只有点击运行脚本过后，才会把该小程序内置的系统变量加载到系统中，供其他模块访问。

因此，找到该脚本，点击运行，该小程序变量重新出现在系统变量中，如下图所示：

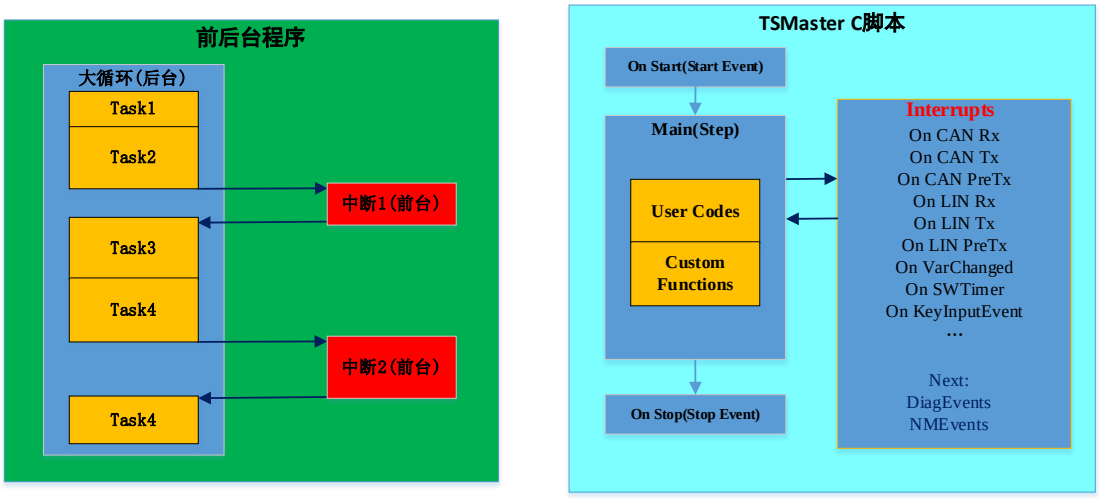


1.15. C 脚本

1.15.1. C 脚本运行机制

1.15.1.1. 软件架构：

TSMaster 中，**每一个测试脚本，可以把它形象的理解为一个 MCU**，其内部的代码框架采用的是前后台模式，如下图所示：



TSMaster C 脚本软件架构

前后台程序架构，主要包含一个大循环，也就是所谓的后台，该后台程序是默认不断运行的；然后就是基于事件驱动(中断机制)的前台。在 TSMaster 的 C 脚本中，主循环就是 step 函数。然后前台中断是各个事件，如 CAN 的接收完成事件 On CAN Rx 等。

在脚本编辑器中完成配置和代码编辑后，整个工程最终动态生成的完整代码可以在 Code Generation 中看到。因此，Code Generation 栏目是只读的。

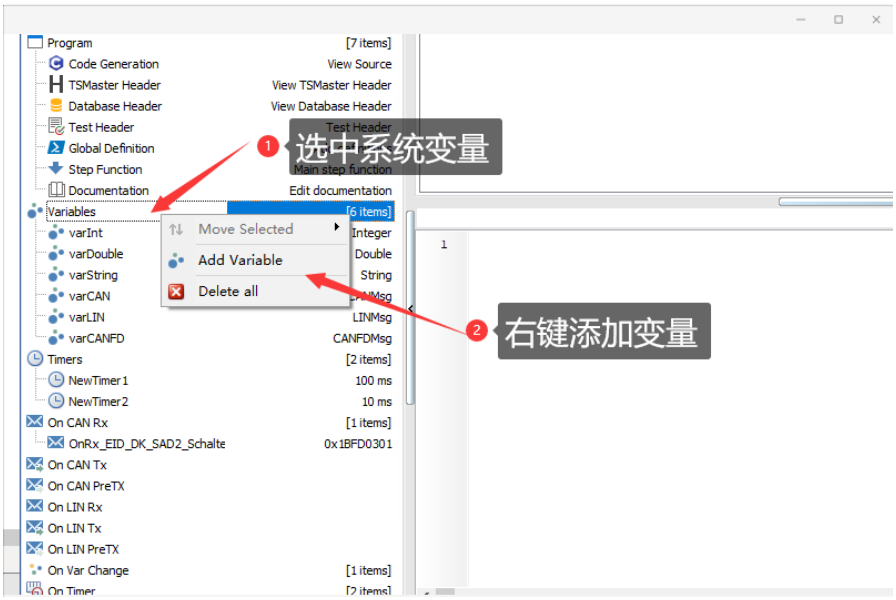
1.15.1.2. 小程序变量(Variable In Mini Program)

C 语言中，变量通常包括全局变量和局部变量。在 TSMaster 的 C 脚本中还提供了一个称为小程序变量的数据类型，其具有如下特点：

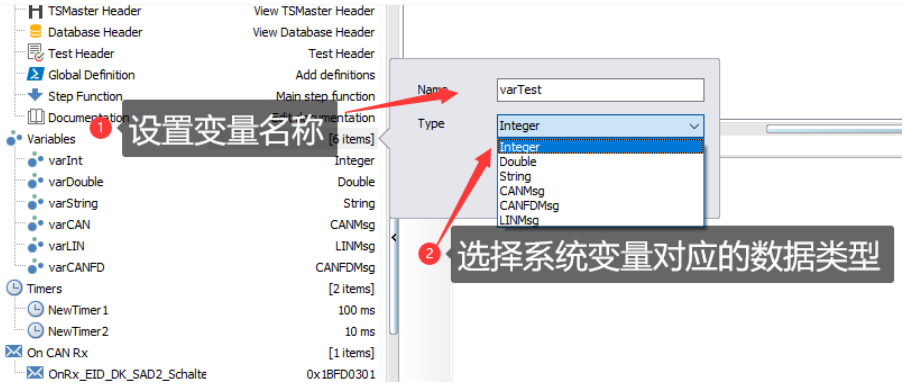
- 小程序变量本质上是对全局变量的一种封装，因此，其访问范围同全局变量。
- 在脚本程序运行过程中，用户能够直接查看小程序变量值；也能够直接修改小程序变量值；当小程序变量发生改变的时候，能够触发数值改变事件。这三个特性也是创建小程序变量的目的所在。
- 小程序变量不能直接读写，需要通过 set 函数写入，通过 get 函数获取变量值。

1.15.1.2.1. 创建小程序变量

1. 添加小程序变量



2. 设置小程序变量属性：名称+数据类型



1.15.1.2.2. 访问 MP Variable

Panel 关联 MP Variable

MP Variable 本质上是内生系统变量，因此，Panel 关联小程序变量的操作见前述章节：系统变量->访问系统变量->Panel 关联系统变量。

跨脚本读写 MP Variable

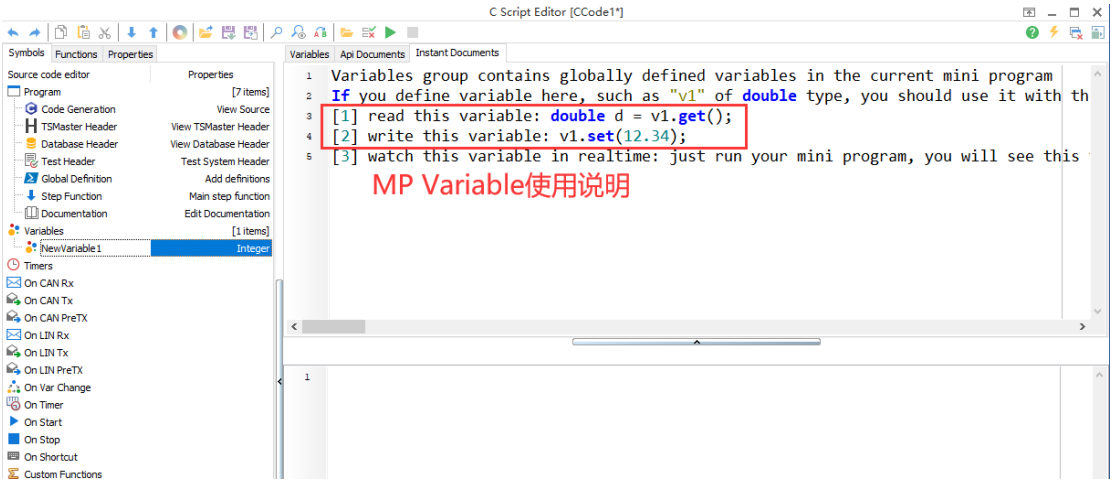
MP Variable 本质上是内生系统变量，因此，Panel 关联小程序变量的操作见前述章节：系统变量->访问系统变量->脚本读写系统变量。

脚本内读写 MP Variable

在本地脚本访问小程序变量不同于普通的变量，对其的读写简单来说就是：

- 读取：var.get();
- 写入：var.set(Value)。

在脚本工具内部选中该变量，右上角有针对该变量的详细参数说明以及使用说明。



其中，get 返回的数据类型和 set 写入的数据类型跟小程序变量的数据类型是对应的。比如例程中的 varInt 变量，其对应的是 Integer 类型，因此 set 写入的和 get 返回的必须是 integer 类型。详细举例如下所示：

- 读取变量：

以 varInt 变量为例：

```
void on_can_rx_OnRx_EID_DK_SAD2_Schalter_01(const PCAN ACAN) { // for identifier = 0x1BFD030
1  app.log("Received Msg 0x1BFD0301",lvlHint);
2  varInt.set(varInt.get() + 1);
3  int a = varInt.get();
4  varString.set("Test");
5  varCAN.set(*ACAN);
6  *ACAN = varCAN.get();
7
```

表示读取 varInt 的值，并赋值给变量 a；

- 写入变量：

以 vatString 变量为例：

```
1  app.log("Received Msg 0x1BFD0301",lvlHint);
2  varInt.set(varInt.get() + 1);
3  int a = varInt.get();
4  varString.set("Test");
5  varCAN.set(*ACAN);
6  *ACAN = varCAN.get();
7
```

表示给 varString 赋值为“Test”字符串。

以 varCAN 变量为例：

```
void on_can_rx_OnRx_EID_DK_SAD2_Schalter_01(const PLAN ACAN) { // for identifier = 0x1BFD0301
1  app.log("Received Msg 0x1BFD0301",lvlHint);
2  varInt.set(varInt.get() + 1);
3  int a = varInt.get();
4  varString.set("Test");
5  varCAN.set(*ACAN);
6  *ACAN = varCAN.get();
7
```

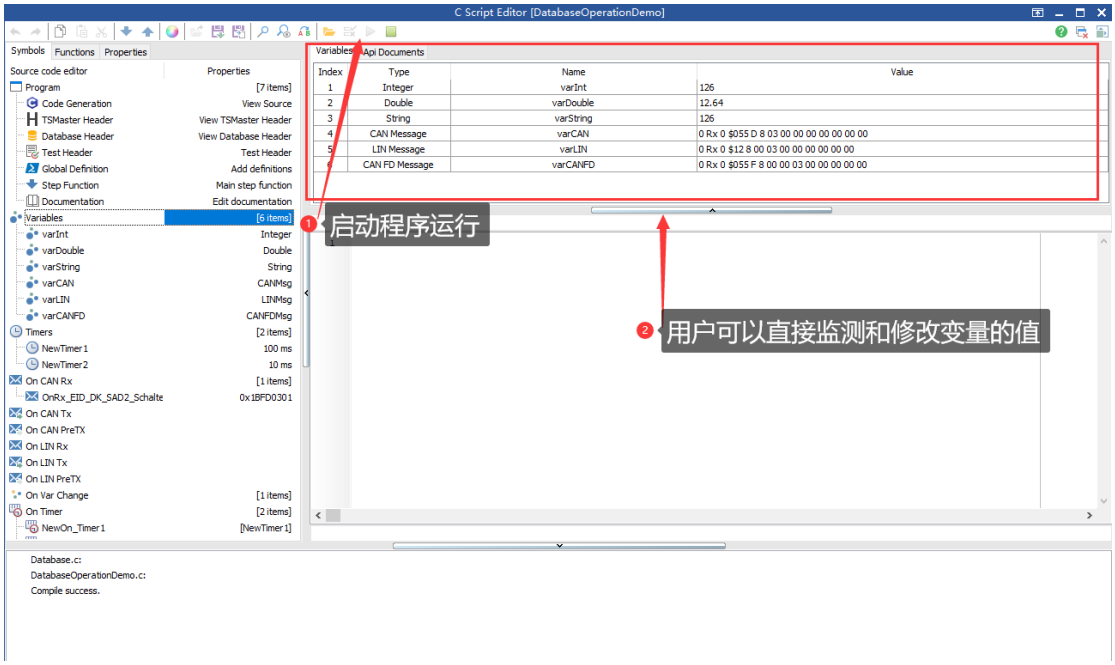
表示把收到的 ACAN 报文赋值给变量 varCAN。

以 varInt 为例：

```
app.log("Received Msg 0x1BFD0301",lvlHint);
varInt.set(varInt.get() + 1);
int a = varInt.get();
varString.set("Test");
varCAN.set(*ACAN);
*ACAN = varCAN.get();
```

表示实现了 vatInt 值自增 1.

Debug 小程序变量

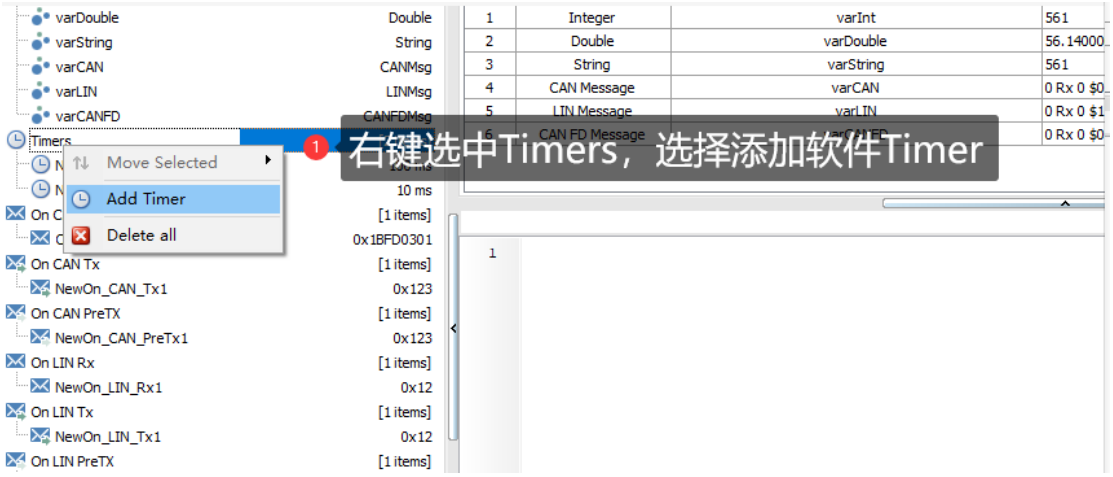


这个功能目前基本上是 TSMaster 所独有的功能，非常方便用户调试过程中监测和手动修改变量值。推荐用户把测试过程中需要动态监测和修改的变量申明为小程序变量。

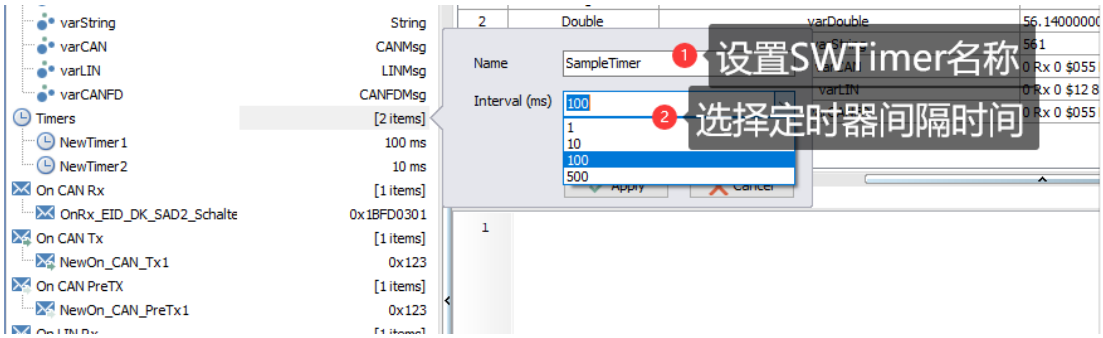
1.15.1.3. Timer 软件定时器变量

为了在事件机制中引入软件定时器事件，这里要先创建软件 timer 变量，也就是创建驱动软件 timer 事件的触发源。如下所示：

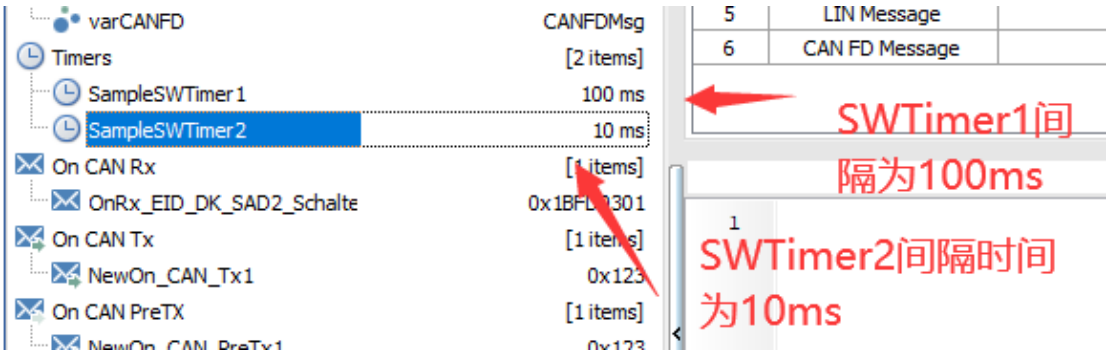
1. 添加 timer



2. 编辑 timer 属性：名称和定时时间

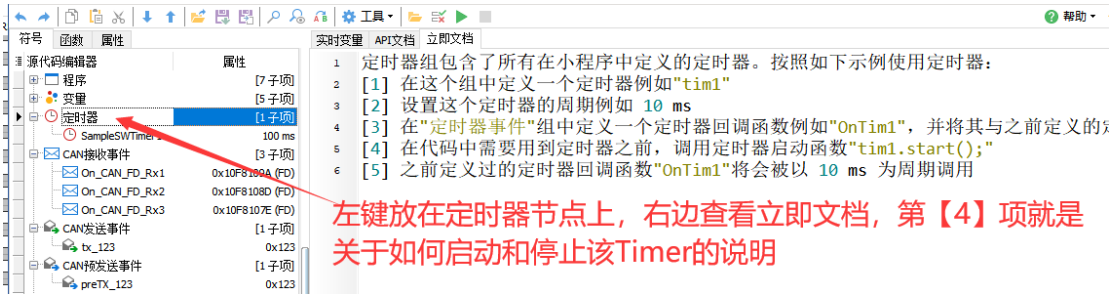


创建完成后，系统中增加几个软件定时器对象，作为后面的 OnTimer 事件的触发源。如下所示：



1.15.1.3.1. 启动停止 Timer

软件 Timer 变量创建好过后。通过调用该变量的属性函数直接启动和停止该 Timer。该说明可以通过如下路径看到：

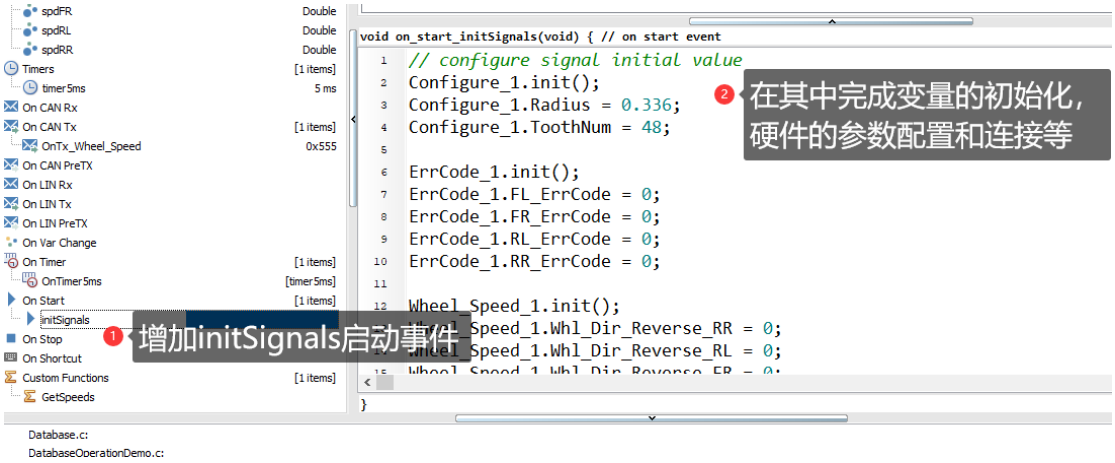


根据该说明，如果像启动该 Timer 的定时事件，则执行代码：SampleSWTimer1.start();停止该 Timer，则执行代码：SampleSWTimer1.stop();

1.15.1.4. 启动停止

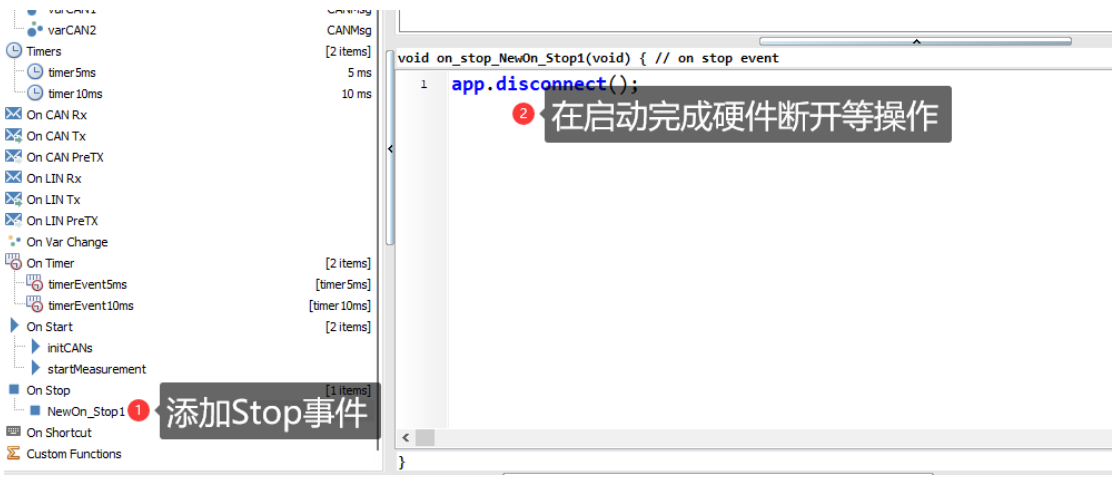
1.15.1.4.1. OnStart

启动整个 C 脚本程序的时候执行的函数事件。添加过程同上。



1.15.1.4.2. OnStop

停止整个 C 脚本程序的时候执行的函数事件。添加过程同上。

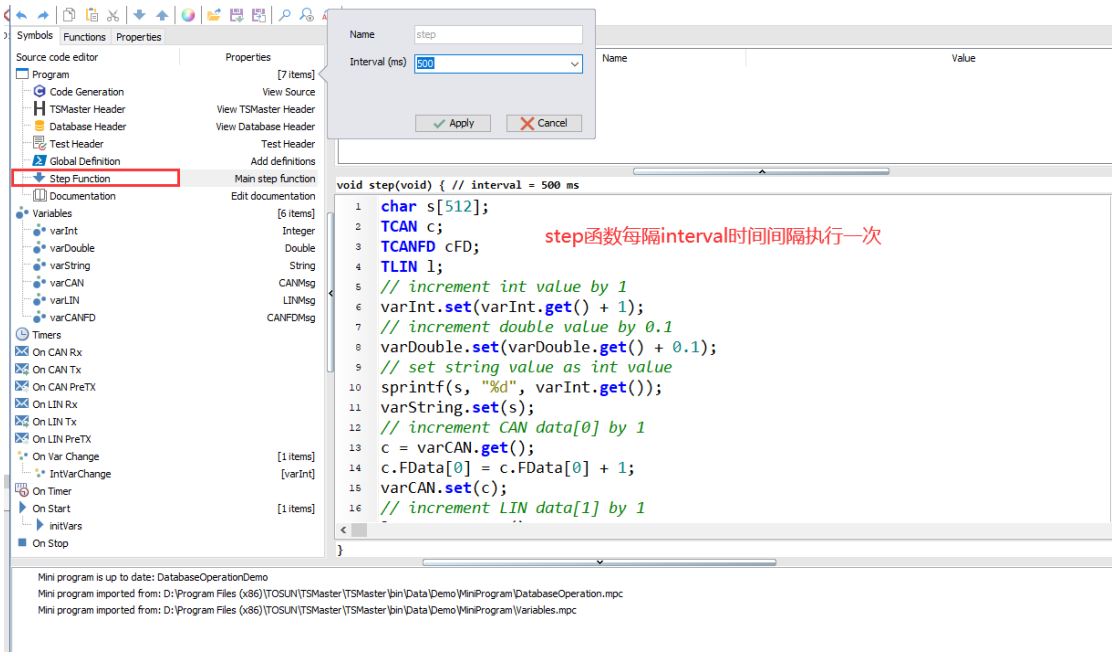


1.15.1.4.3. 注意事项（警告）：

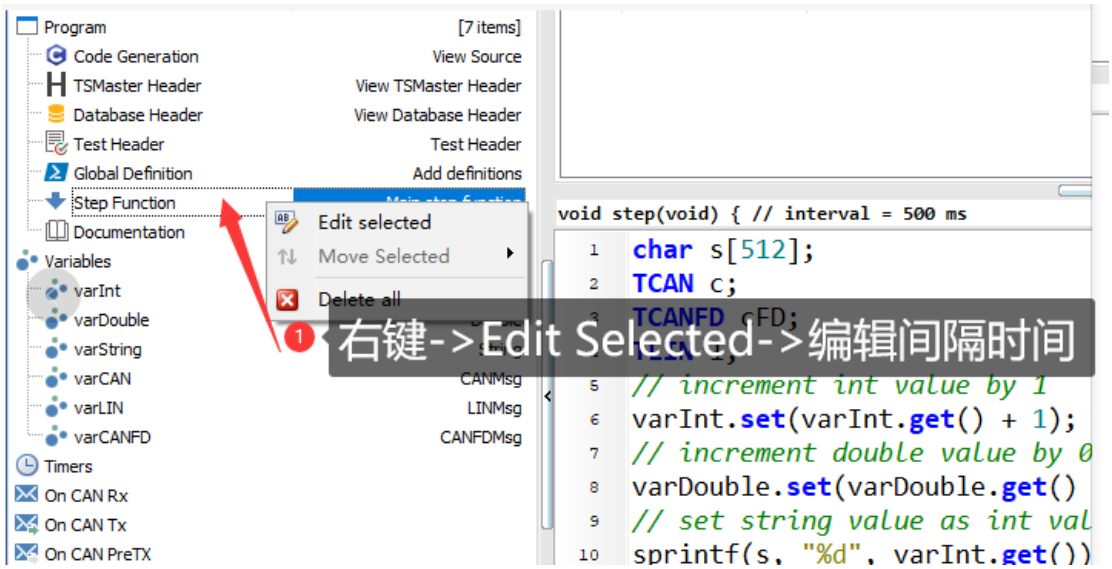
不能在 OnStart/OnStop 事件中，加入 While(1)类似机制，造成脚本卡在启动/停止步骤中，无法进入大循环，也就无法响应事件机制。

1.15.1.5. Step 机制

为用户的程序逻辑提供一个主循环调度接口。用户可以基于此周期性调度接口运行自己的算法。以例程为例，如下所示：

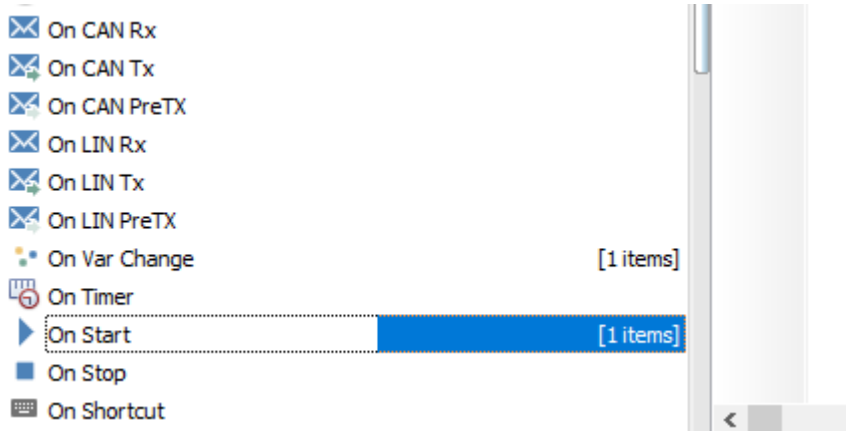


用户在 step 函数中添加代码，这部分逻辑每隔 intervaltime 时间间隔被调度一次。修改间隔时间如下所示：



1.15.1.6. 事件机制

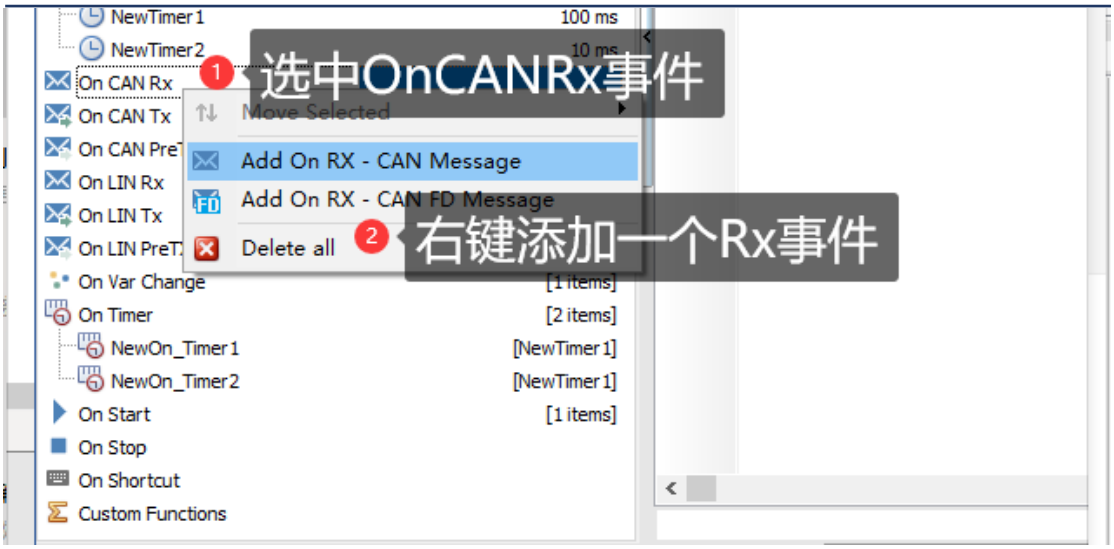
顾名思义，事件驱动机制指的是当预设的事件发生过后就立即触发函数调用的机制。TSMaster 的 C 脚本中主要包含了如下的事件机制：



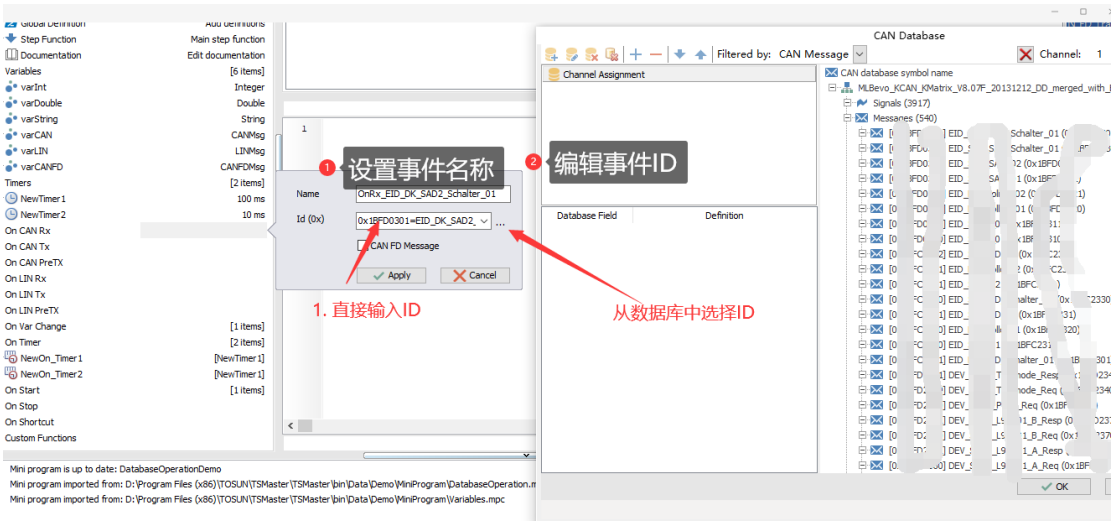
1.15.1.6.1. OnCANRx

指定 ID 的 CAN 报文接收到的时候触发执行的函数事件。举例如下：

1. 添加 CANRx 事件



2. 编辑事件的 CANID



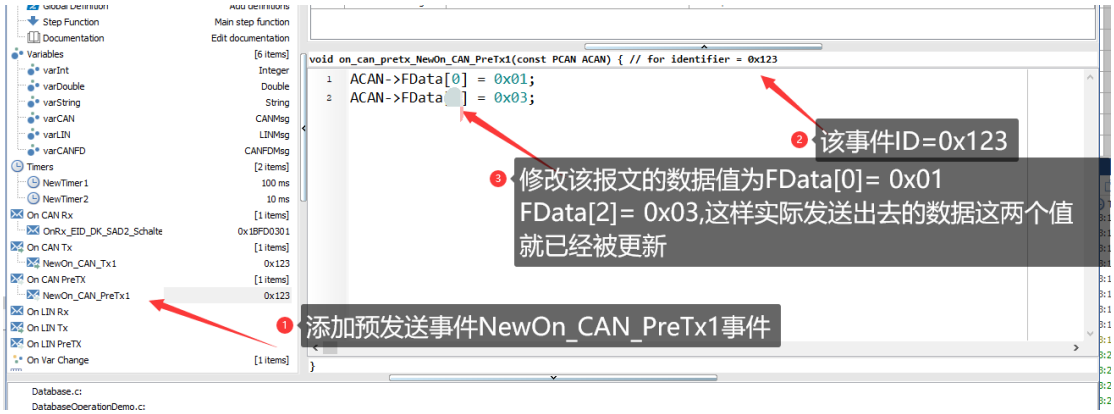
3. 在事件中加入逻辑

1.15.1.6.2. OnCANTx

指定 ID 的 CAN 报文发送成功后触发执行的函数事件。其添加事件的过程跟添加 OnCANRx 报文一样。

1.15.1.6.3. OnCANPreTx

指定 ID 的 CAN 报文发送之前触发执行的函数事件。添加过程同上。



1.15.1.6.4. OnLINRx

指定 ID 的 LIN 报文接收到的时候触发执行的函数事件。添加过程同上。

1.15.1.6.5. OnLINTx

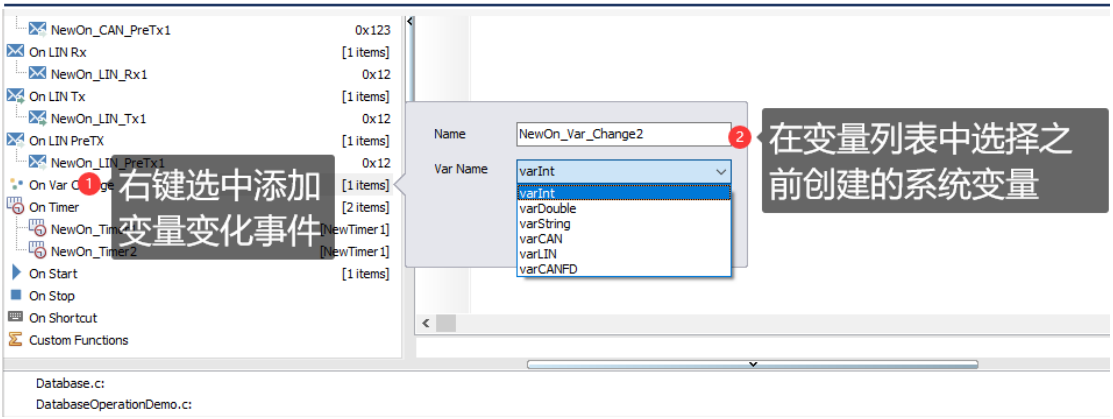
指定 ID 的 LIN 报文发送成功后触发执行的函数事件。添加过程同上。

1.15.1.6.6. OnLINPreTx

执行 ID 的 LIN 报文发送之前触发执行的函数事件。添加过程同上。

1.15.1.6.7. OnVarChange

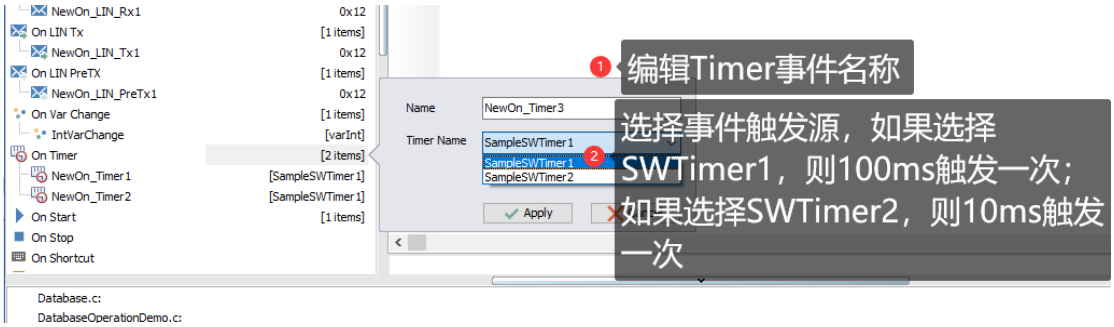
当小程序变量的值发生变化时触发执行的函数事件。添加过程如下：



在该变量变化事件中添加逻辑，如下所示：

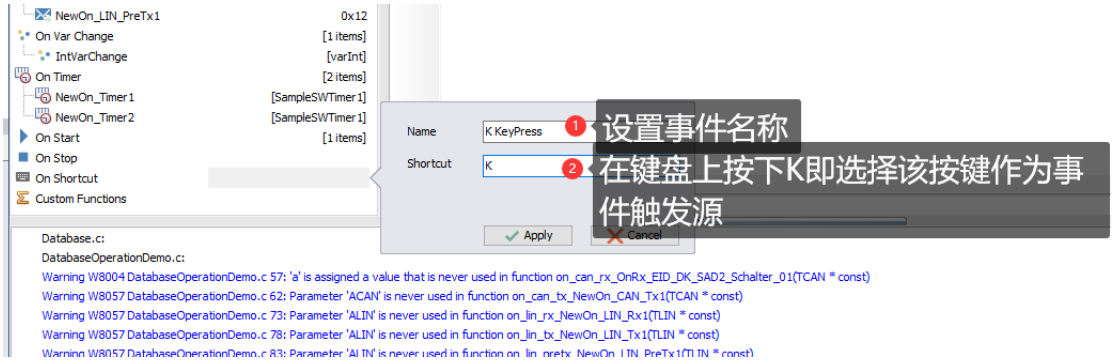
1.15.1.6.8. OnTimer

软件 Timer 事件，当软件定时器时间到了就触发执行的函数事件。添加过程同上。



1.15.1.6.9. OnShortCut

通过快捷键触发执行的函数事件。添加过程如下：



1.15.1.6.10. 注意事项（警告）：

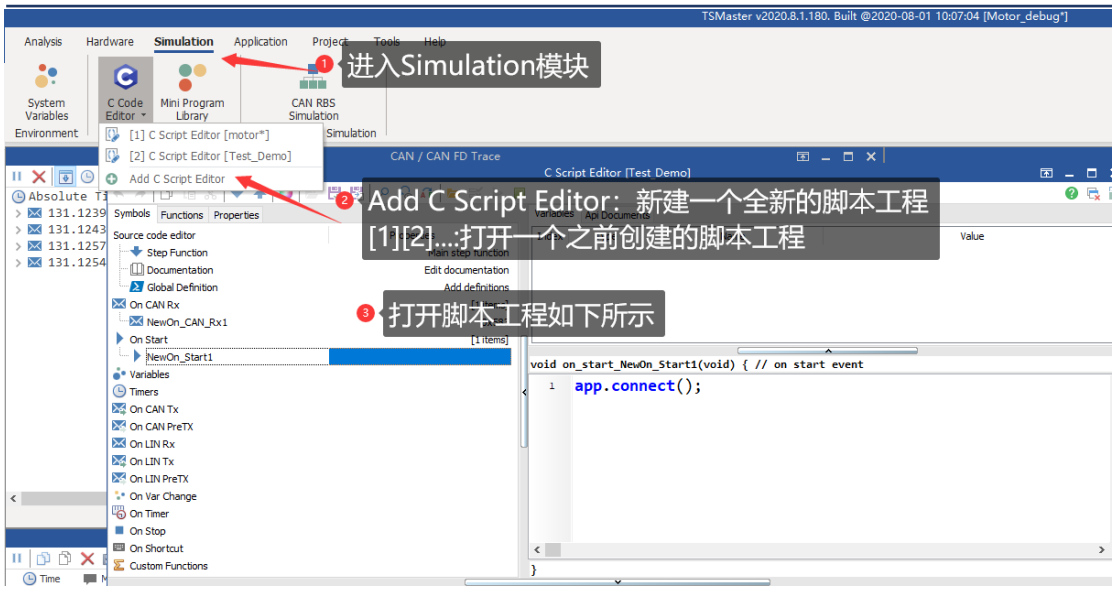
事件机制类似于单片机中的中断 Interrupt，在其中尽量避免执行非常耗时间的任务，更加禁止执行如 While(1)代码直接卡死在该中断中。

1.15.1.7. CAN 报文转发示例

1.15.2. 基本使用流程

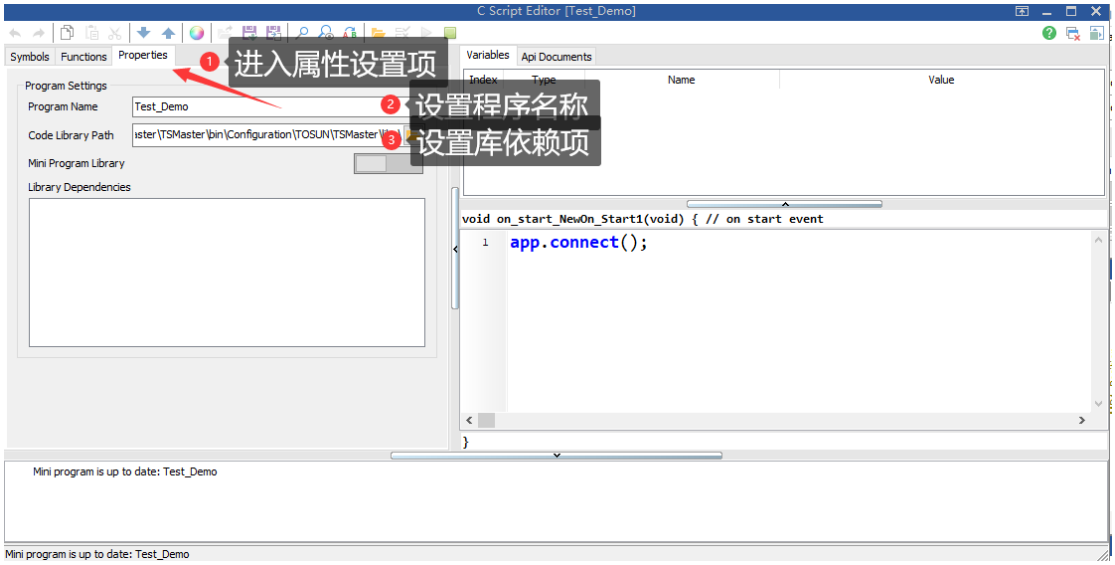
1.15.2.1. 创建（打开）脚本

创建(打开)脚本流程如下图所示：Simulation->C Code Editor->Open(Add) C Script Editor



1.15.2.2. 设置脚本属性

编辑和运行脚本之前，先设置脚本属性，主要包括：脚本名称，代码库路径，代码依赖项。如下图所示：



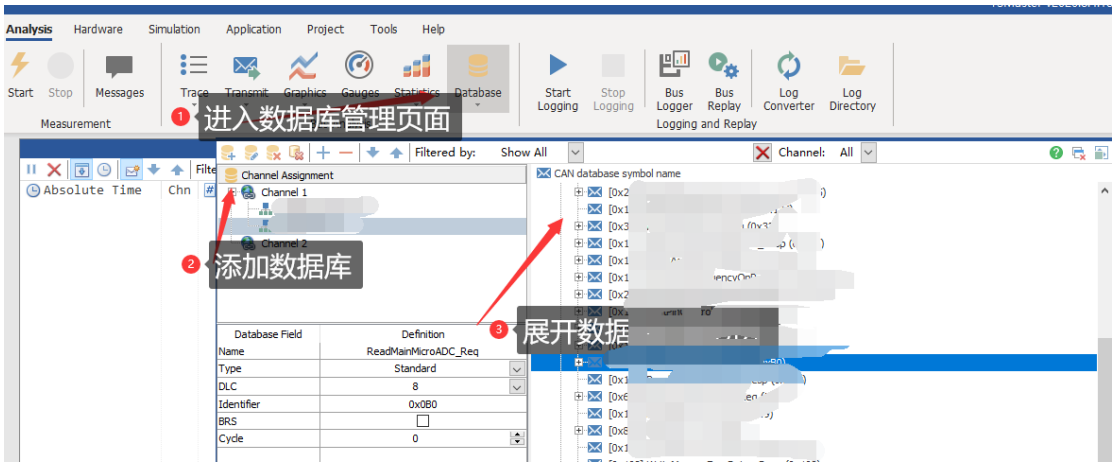
注意：C 脚本实质上就是一个独立的应用程序，和任何基于 TSMaster API 的程序（包括 TSMaster 本身）一样，都是通过应用程序名称来进行硬件映射的。因此，用户使用的时候需要明确的知道自身程序名称，并基于此名称来映射到实际的硬件设备上。如上图所示：设置本 C 脚本的名称为 Test_Demo。

1.15.3. 数据库信号操作（基于 RBS）

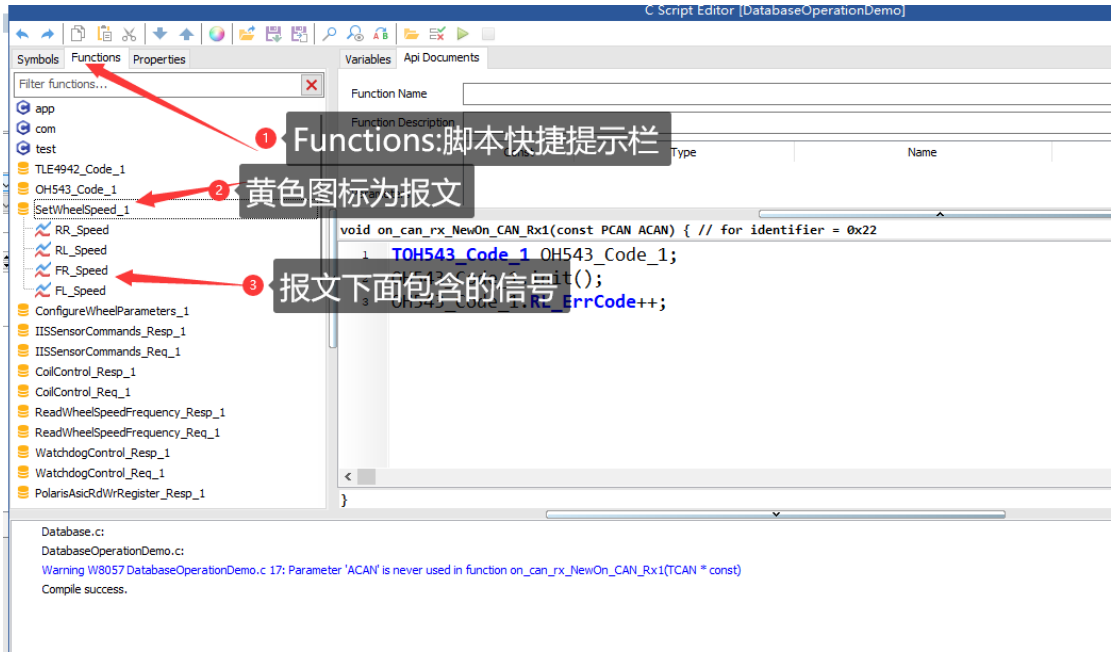
优缺点：

优点：直接根据信号路径读取和设置信号值，操作简单方便。默认推荐使用此方法来读取和写入数据库信号。
缺点：此操作方式跟数据库是强绑定的，在一些故意制造异常的测试场合，此操作方式就无法完全满足了。 采用这种方式的前提要启动 CANRBS 仿真，如果用户单纯想测试几个单独报文帧的收发，这种方式就无法满足要求了。

➢ 第一步：添加数据库：



加载完数据库过后，在脚本编辑器的快捷提示窗口可以看到所有的数据库报文以及信号，如下所示：



说明：添加数据库的目的是动态生成跟数据库中报文和信号相关的 C 语言数据类型定义，用户就不用自己去构造为数众多的各种 CAN 报文和信号的数据结构了。

➢ 第二步：读/写取 CAN 信号

读取 CAN 信号：

```
double d;
com.can_rbs_get_signal_value_by_address("0/CAN_FD_Powertrain/Engine/EngineData/EngSpeed", &d);
log("EngSpeed = %f", d);
```

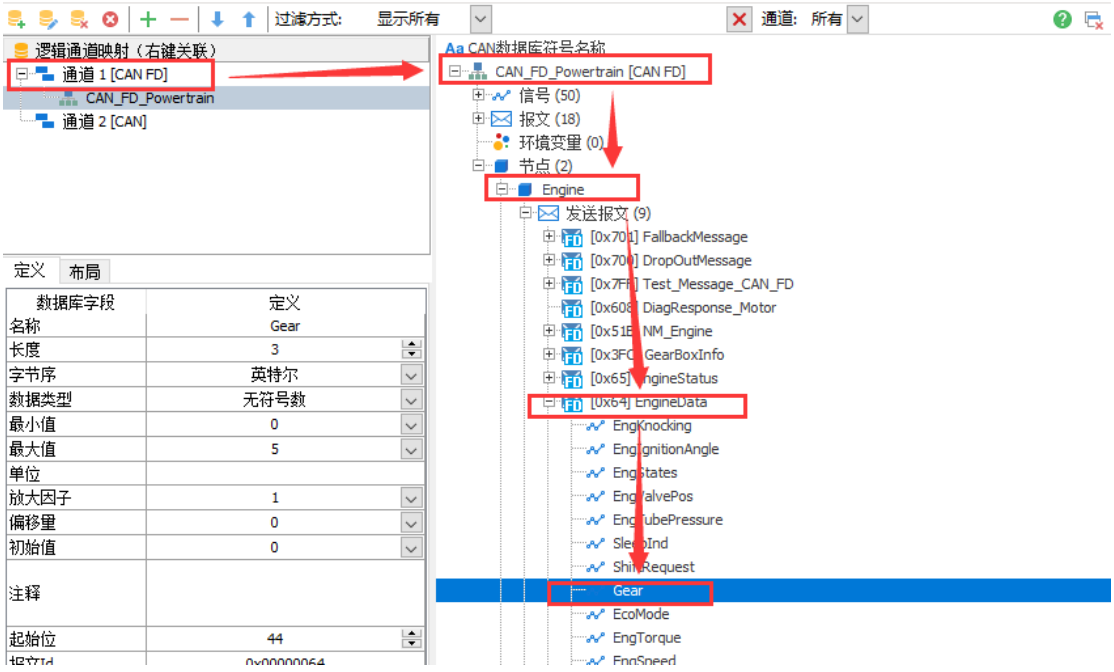
写入 CAN 信号：

```
double d;
com.can_rbs_set_signal_value_by_address("0/CAN_FD_Powertrain/Engine/EngineData/Gear", &d);
log("Gear = %f", d);
```

这两个函数的功能是：根据 CAN 信号在数据库中的路径，直接读取和写入 CAN 信号。CAN 信号字符串的组成，以 Gear 信号为例，其解析如下所示：

通道 (0)	数据库名称 (CAN_FD_Powertrain)	ECU 节点 (Engine)	报文 (EngineData)	信号 (Gear)

对应到数据库结构中，如下所示：

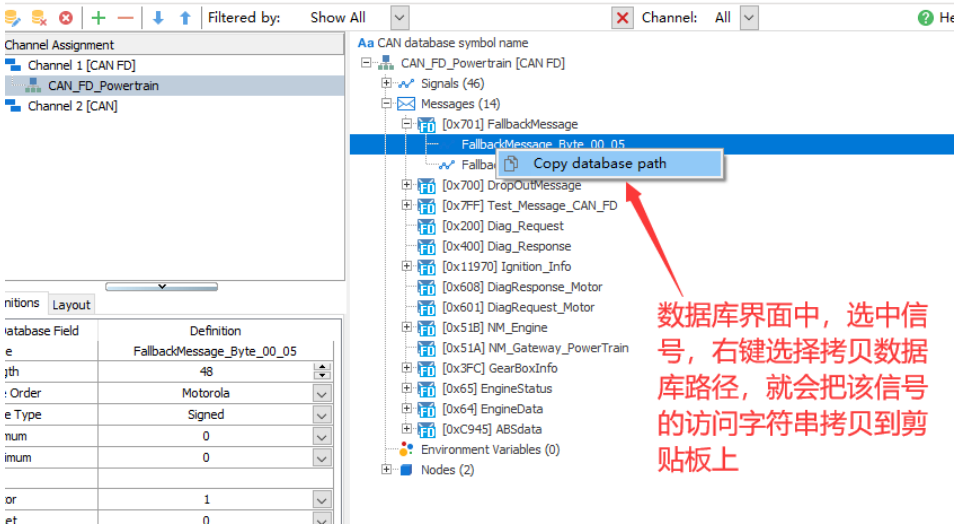


关于通道编号，其对应关系如下：

```
#define CH1 0
#define CH2 1
#define CH3 2
#define CH4 3
#define CH5 4
#define CH6 5
#define CH7 6
#define CH8 7
#define CH9 8
#define CH10 9
#define CH11 10
#define CH12 11
#define CH13 12
#define CH14 13
#define CH15 14
#define CH16 15
```

也就是应用程序中的 CHANNEL1 对应的编号是 0，CHANNEL2 对应的编号是 1，依次类推。因为对于程序开发来说，编号一遍从 0 开始（比如数组等变量）；对于现实生活中的描述习惯来说，一般从 1（比如通道 1 等）开始。

TSMaster 提供了图形界面方便用户直接提取信号的路径。如下图所示：



总结：

如上所示，采用这种操作方式，CAN 信号的读写就非常简单了，不需要预先申请信号变量等操作，直接基于 RBS 系统读取和写入信号到模拟变量中。

1.15.4. 支持的数据类型

TSMaster 内置的 C 脚本支持通用数据类型和自定义数据类型，主要包含如下数据类型：

1.15.4.1. 通用数据类型：

符号	名称	范围	说明

s8	有符号 8 位整数		
u8	无符号 8 位整数		
s16	有符号 16 位整数		
u16	无符号 16 位整数		
s32	有符号 32 位整形		
u32	无符号 32 位整形		
s64	有符号 64 位整数		
u64	无符号 64 位整数		
bool	布尔变量		
float	单精度浮点数		
double	双精度浮点数		
指针			
pvoid	无类型指针		
ps8	有符号 8 位整型指针		
pu8	无符号 8 位整形指针		
ps16	有符号 16 位整形指针		
pu16	无符号 16 位整形指针		
ps32	有符号 32 位整形指针		
s32*	有符号 32 位整数指针		
pu32	无符号 32 位整形指针		
u32*	无符号 32 位整数指针		
ps64	有符号 64 位整数指针		
s64*	有符号 64 位整数指针		
pu64	无符号 64 位整数指针		
u64*	无符号 64 位整数指针		
pbool	布尔指针		
bool*	布尔指针		
int*	整数指针		
uint*	无符号整数指针		
pfloat	单精度浮点数指针		
float*	单精度浮点数指针		
pdouble	双精度浮点数指针		
double*	双精度浮点数指针		
char*	字符指针		
pchar	字符指针		
指向指针的指针			
ppvoid	指向无类型指针的指针		
ppchar	指向字符指针的指针		
char**	指向字符指针的指针		
ppdouble	指向双精度浮点数指针的指针		
double**	指向双精度浮点数指针的指针		
float**	指向单精度浮点数指针的指针		
pps32	指向有符号 32 位整数指针的指针		

1.15.4.2. 内部自定义变量类型：

PCANSignal	CAN 信号指针		
PCAN	CAN 报文指针		
PCANFD	CANFD 报文指针		
PLIN	LIN 报文指针		
TSystemVar	系统变量类型		
PLIBTSMapping	硬件映射变量指针		
TCANFDControllerType	FD 控制器类型		
TCANFDControllerMode	FD 控制器模式		
TOnIoIpData	IP 数据类型		
PLIBSystemVarDef	系统变量指针		
Prealtime_comment_t	实时注释变量指针		
TLogLevel	打印注释的等级		
TCheckResultCallback	检测结果回调函数		

1.15.4.3. 用户自定义数据类型

如果用户自定义数据类型，如结构体，结构体指针等。TSMaster 的脚本默认是不支持的。但是有一种变通的处理方式。也就是通过无类型指针来传递数据地址，从而达到数据传递的效果。如下所示：

1.15.5. 数据库信号操作

优缺点：

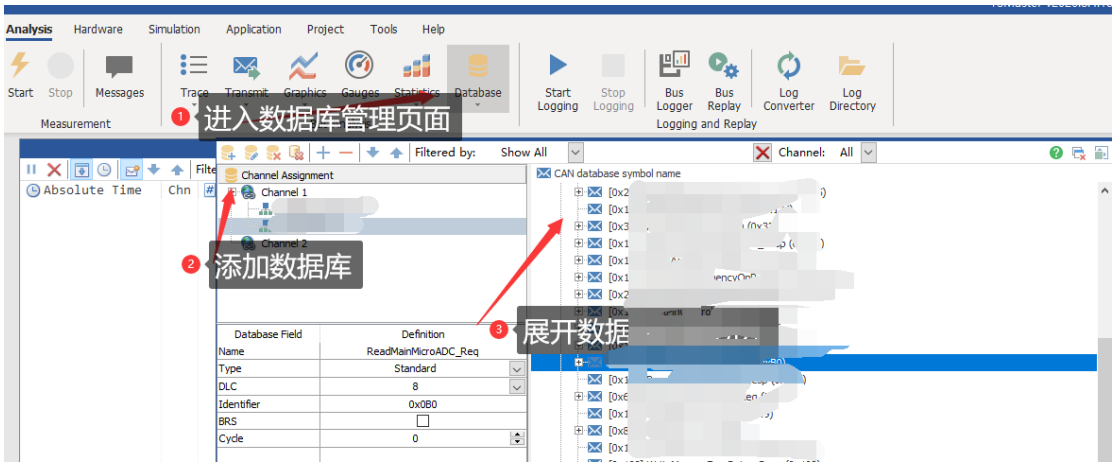
优点：

这种方式本质上是提供了一套 DBC 信息的解析和注入的机制，并没有跟数据库完全绑定，因此可以灵活使用。比如通过 Set_Data 函数，可以把任意报文注入到信号中，然后信号函数把相应的物理值计算出来，并不用管这个报文 ID 是否跟信号的报文 ID 是对应的。因此，用于需要灵活解析信号的场合，即使数据库中定义的报文跟收到的报文 ID 对不上，也可以用此方法进行信号解析。

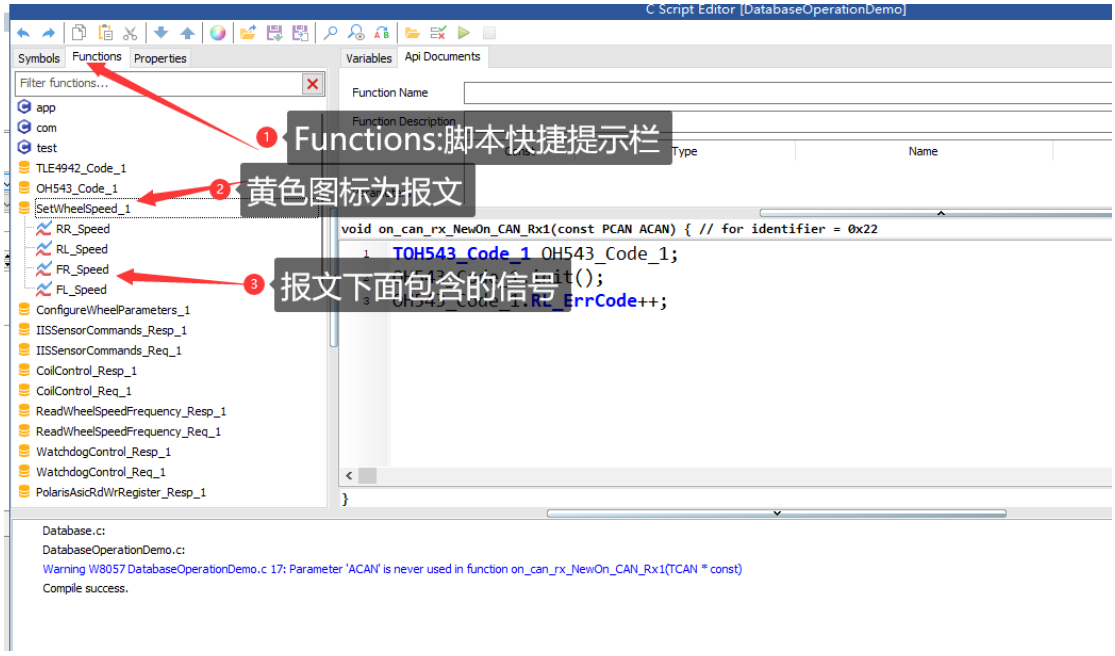
缺点：灵活的代码就是使用起来流程相对复杂一些，详细使用流程见本章节的第一—五步，因此除非有测试等特殊需求，一般不推荐使用此方式来使用信号。

下面步骤介绍了如何在 TSMaster 的 C 脚本中操作数据库变量。

➢ 第一步：添加数据库：



加载完数据库过后，在脚本编辑器的快捷提示窗口可以看到所有的数据库报文以及信号，如下所示：

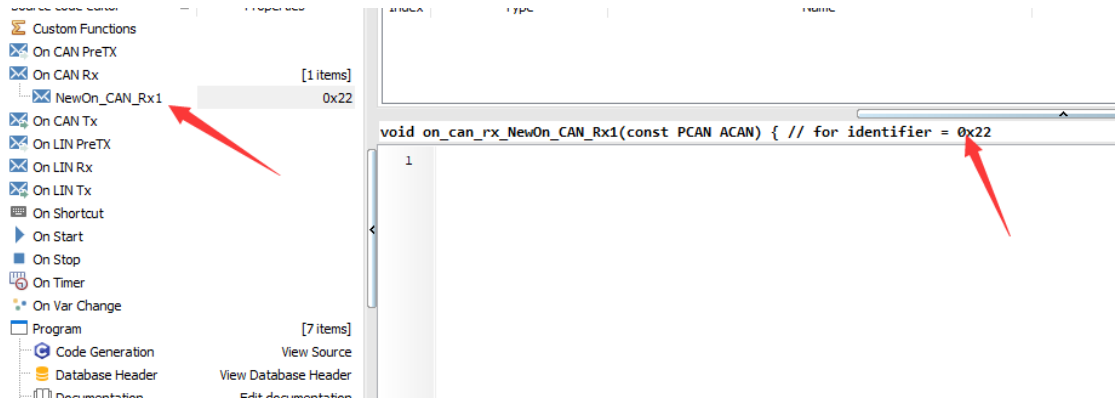


说明：添加数据库的目的是动态生成跟数据库中报文和信号相关的 C 语言数据类型定义，用户就不用自己去构造为数众多的各种 CAN 报文和信号的数据结构了。

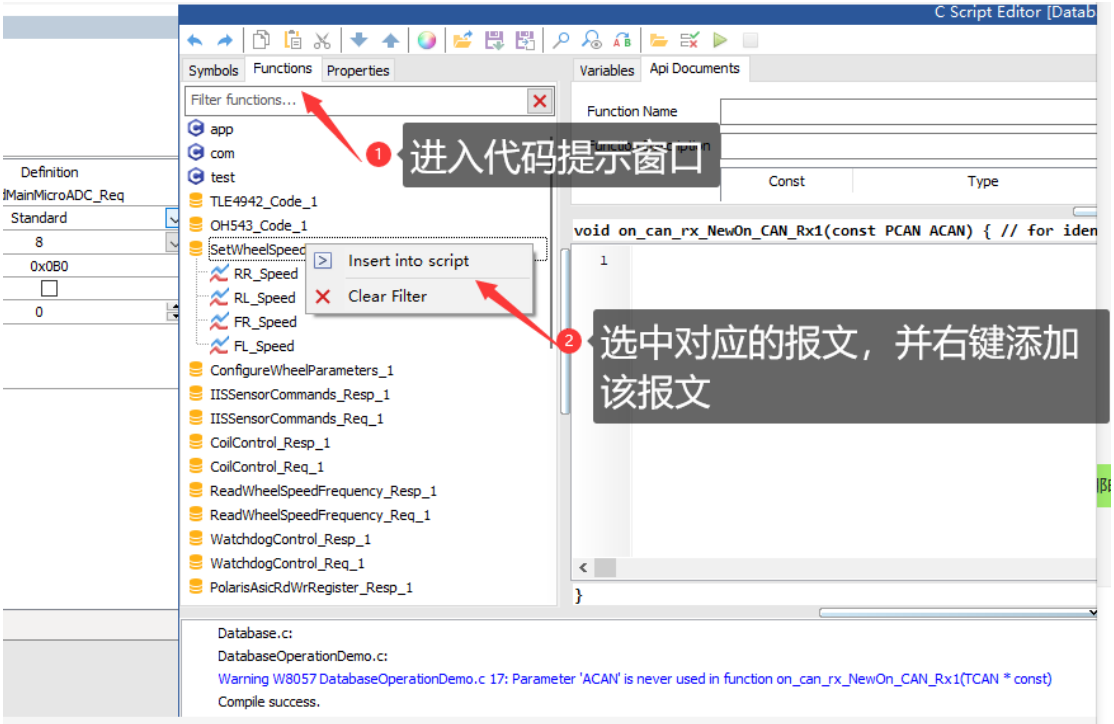
➢ 第二步：添加报文变量

局部报文变量:

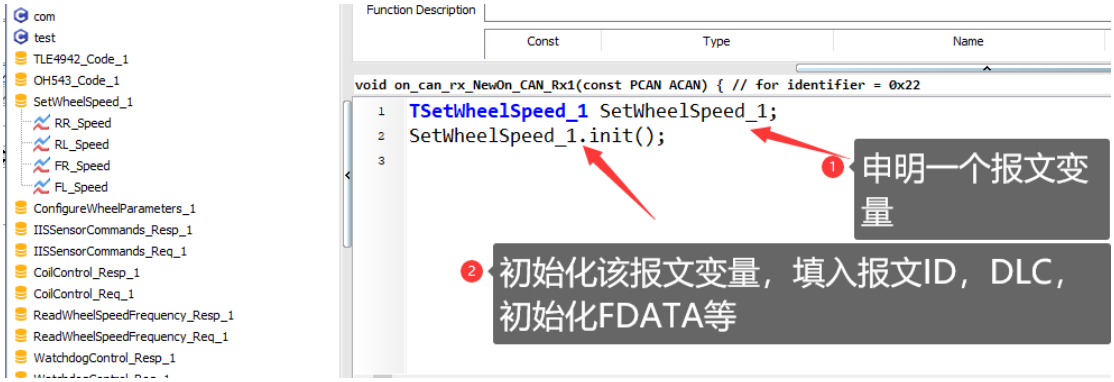
顾名思义，局部报文变量是只用在本函数内部的报文变量。其有效范围只在本事件函数内部，而且每次事件触发的时候该值都重新刷新。比如，在 CANRX 接收 ID=0x22 的接收报文事件中，创建一个局部变量，如下所示：



通过图形界面添加报文变量（如果记性好，手动直接输入也可以）：



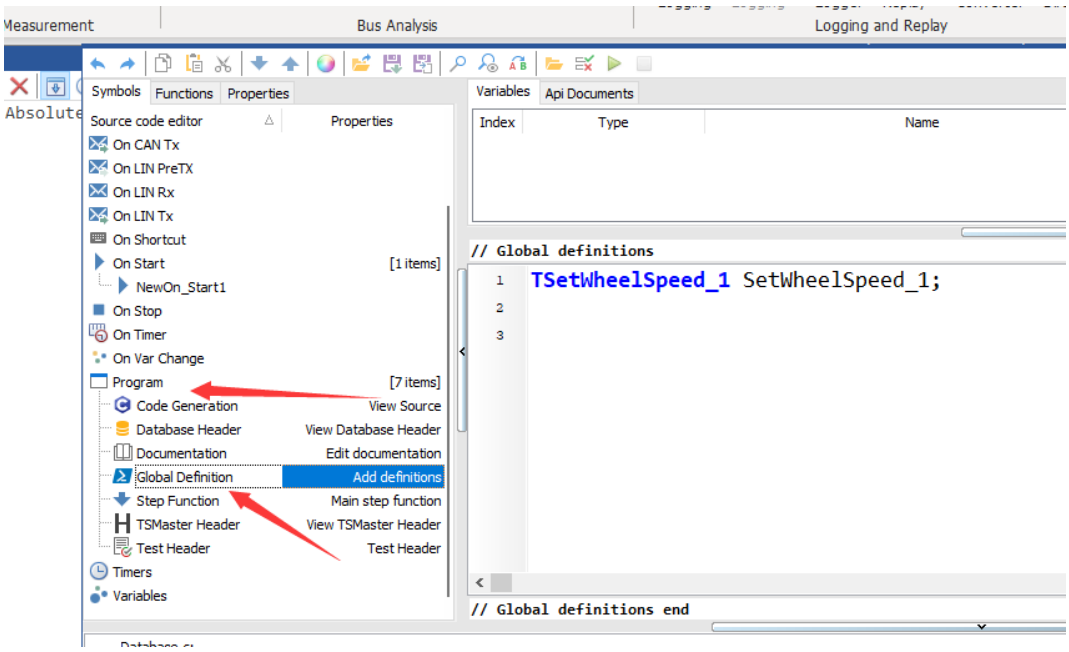
添加完成后，如下图所示：



全局报文变量

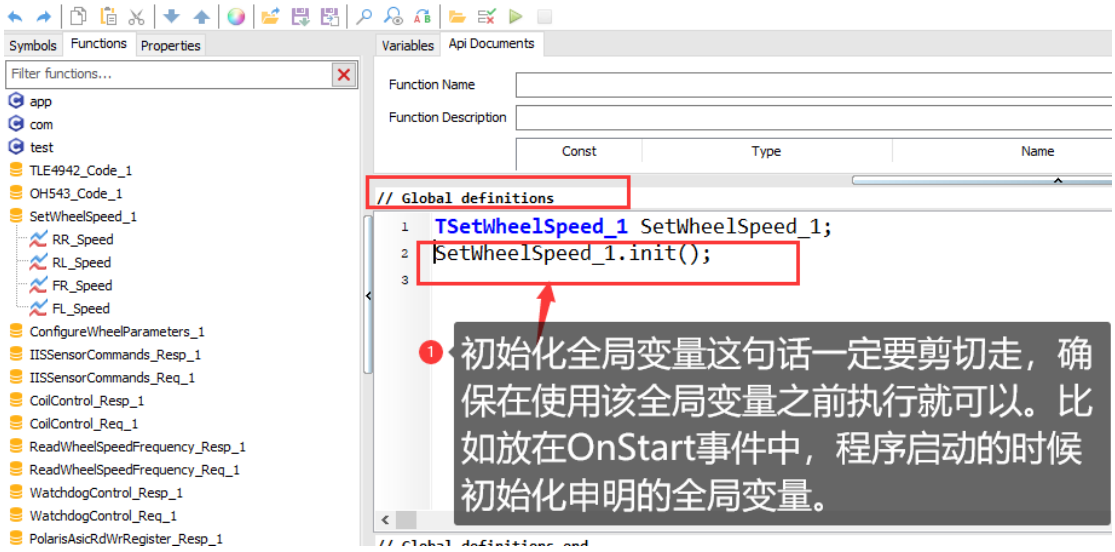
顾名思义，全局报文是整个 Program 都可以访问的数据，而且在使用过程中，程序会一

直保持该变量内存。其添加位置在：Program->Global Definition 目录下面：



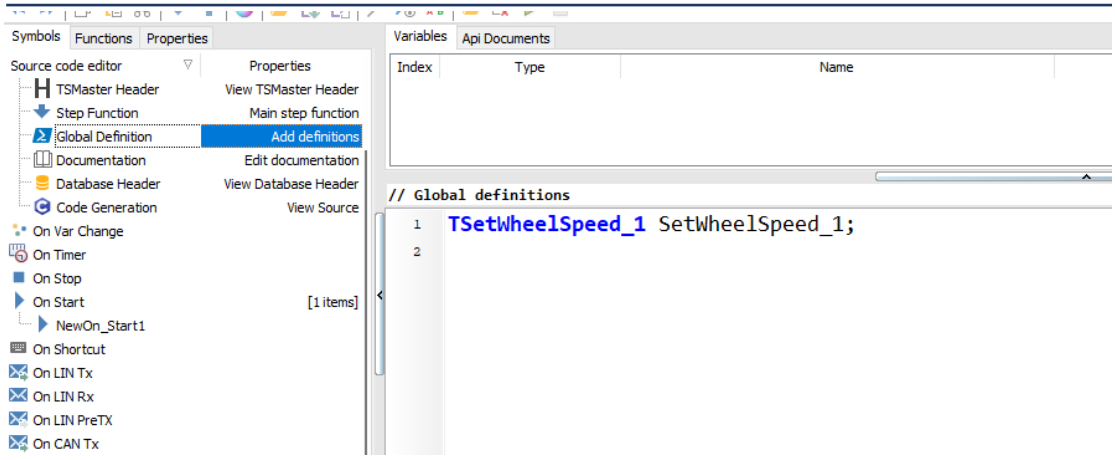
其添加方式跟局部变量是完全一样的，但是有个需要注意的是：全局变量的初始化函数比如 SetWheelSpeed_1.init()不能直接放在变量定义文件中，需要拷贝到比如 OnStart 这样的事件中，确保用户在使用这个变量之前初始化该变量。在新版的脚本编辑器中，这句话会加以提示，如果不剪切走，会触发重复定义错误提示。

详细添加过程如下：

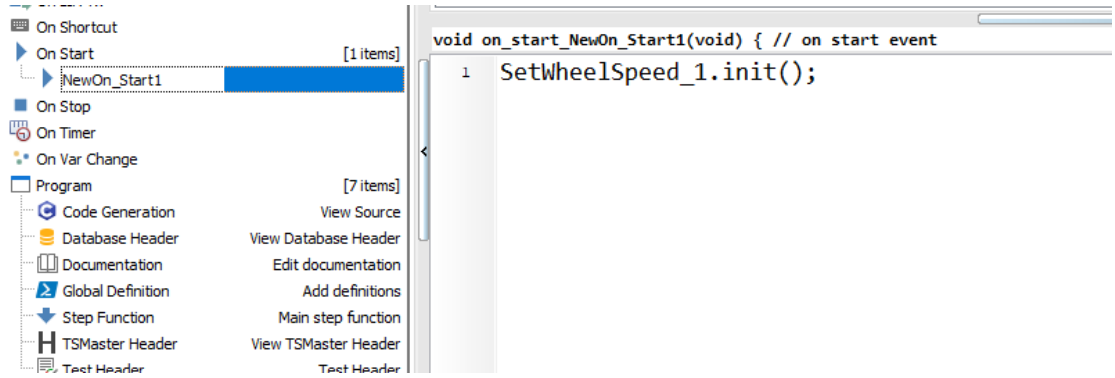


剪切初始化代码后，如下：

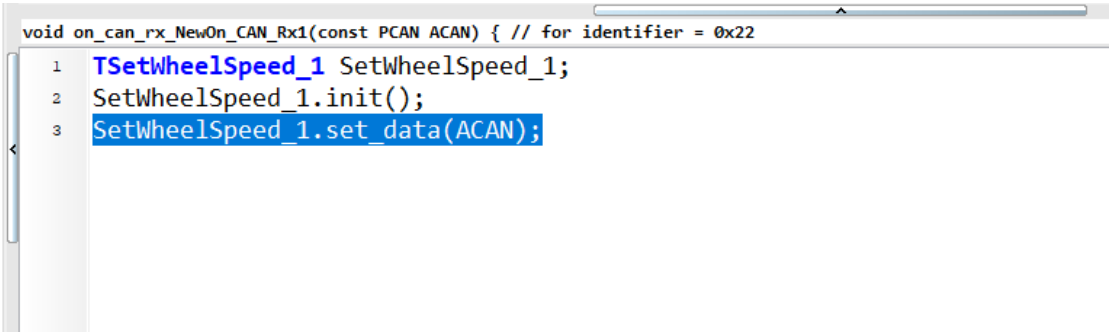
申明的全局变量：



初始化全局变量：

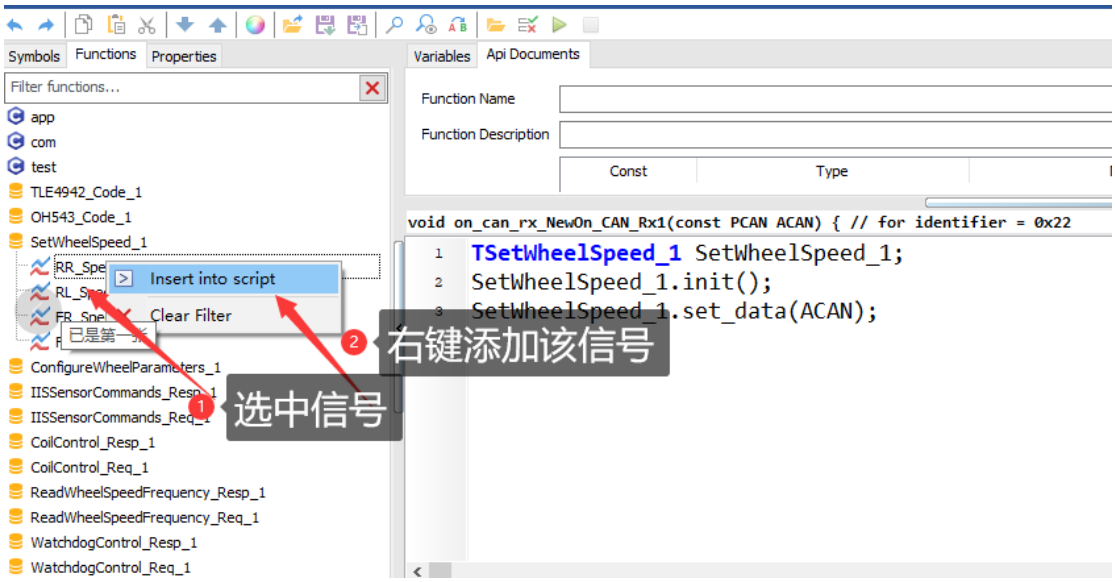


➢ 第三步：读取 CAN 原始数据到报文变量中

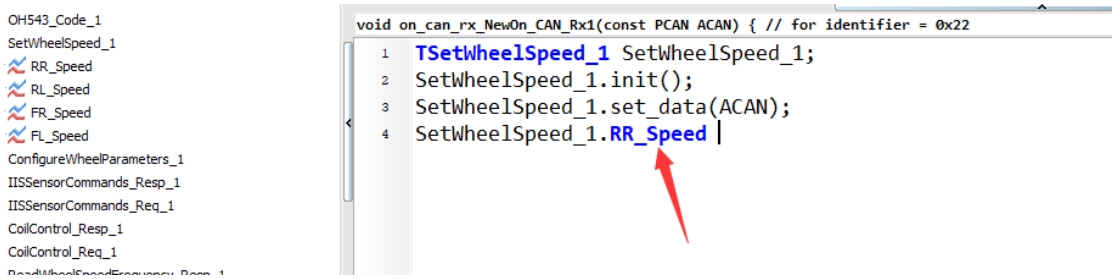


如上所示：这句话表示的意思就是把原始报文 TCAN 类型（PCAN 为 TCAN 的指针类型，定义如下： typedef TCAN* PCAN;）的报文，填充到数据库报文中（主要目的，是为了使用数据库里面定义的报文数据，信号数据类型。如果用户不需要数据库，直接操作 TCAN 数据结构当然也可以）。

➢ 第四步：基于信号操作数据

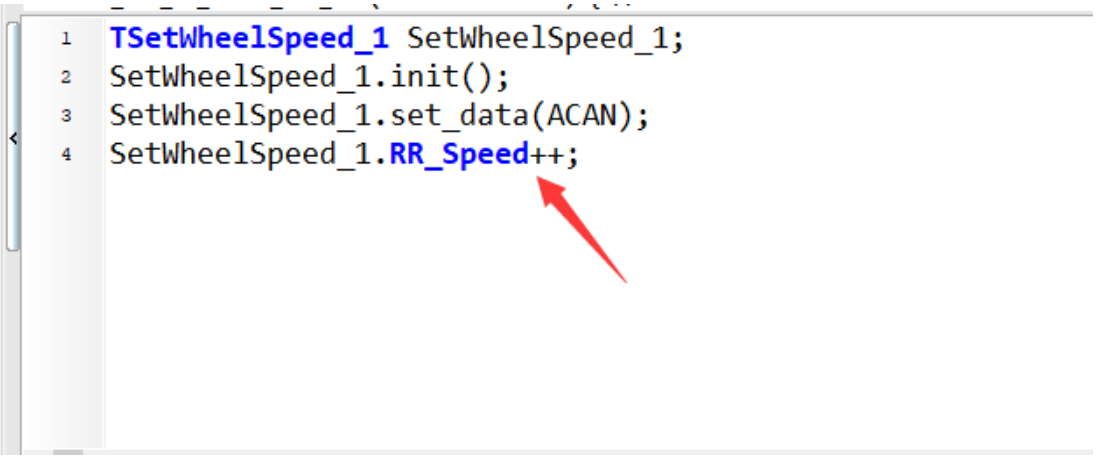


进入提示栏，选中信号，右键添加信号，会在代码中自动添加该信号变量。如下所示：



图形界面都是为了方便用户完成代码编辑，如果用户熟悉报文和信号数据结构定义，可以直接手动输入。

基于此信号，用户可以完成自己想要的运算，如下：



➢ 第五步：发送报文

把编辑完成后的报文再发送回总线。调用发送函数，把报文里面的 FCAN 数据发送出去即可。如下所示：

```
void on_can_rx_NewOn_CAN_Rx1(const PCAN ACAN) { // for identifier = 0x22
  TSetWheelSpeed_1 SetWheelSpeed_1;
  SetWheelSpeed_1.init();
  SetWheelSpeed_1.set_data(ACAN);
  SetWheelSpeed_1.RR_Speed++;
  com.transmit_can_async(&SetWheelSpeed_1.FCAN);
```

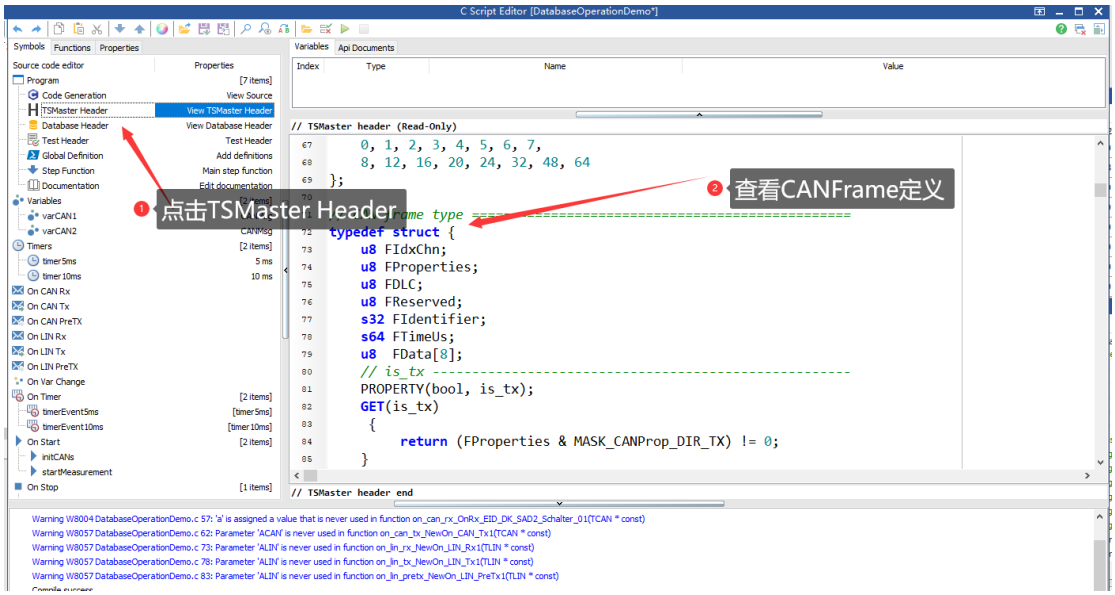
同样的：transmit_can_async 函数也可以通过帮助提示栏添加。

如果用户对发送报文函数编写不熟悉，可以参考章节：C 脚本操作技巧->通过 Trace 窗口添加发送报文/通过发送窗口添加发送报文。TSMaster 提供了代码生成工具，自动生成相应的发送报文。

1.15.6. C 脚本操作技巧

1.15.6.1. 查看系统变量类型定义（如 CAN，LIN 报文）

TSMaster C 脚本的所有内置的系统变量定义如 CAN，LIN 报文等都放在系统文件 TSMaster Header 中，如下所示：

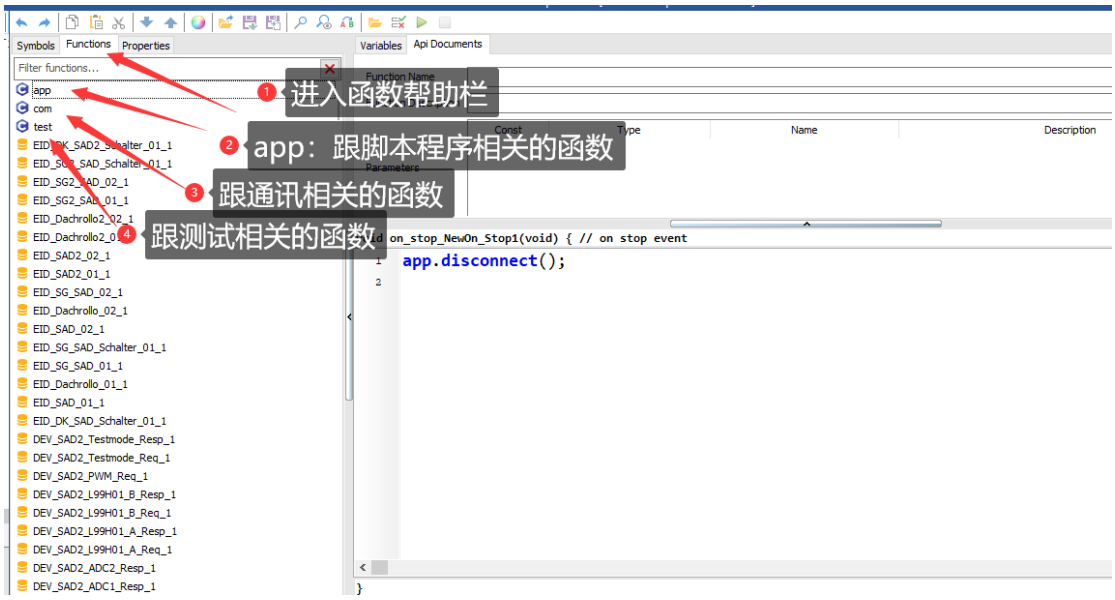


对于习惯直接处理原始报文数据的开发者来说，需要到此文件中查看数据类型定义。

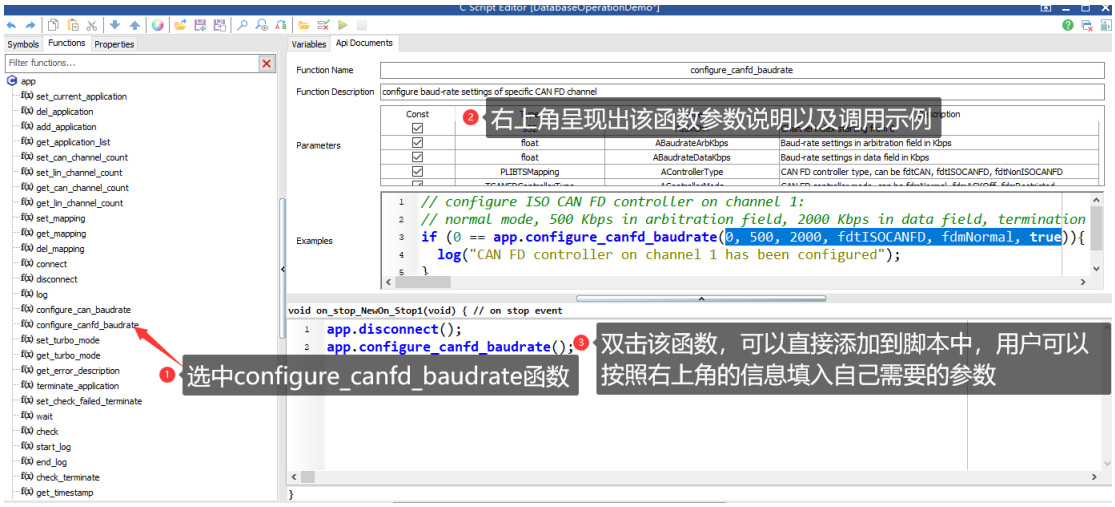
1.15.6.2. 添加系统函数

开发者在使用过程中会碰到系统函数不知道怎么调用的情况，TSMaster 的 C 脚本编辑器提供了帮助栏帮助用户理解和添加系统函数。其过程如下所示：

1. 进入系统函数栏目



2. 以添加一个 CANFD 的配置参数为例，讲解如何添加系统函数，如下所示：

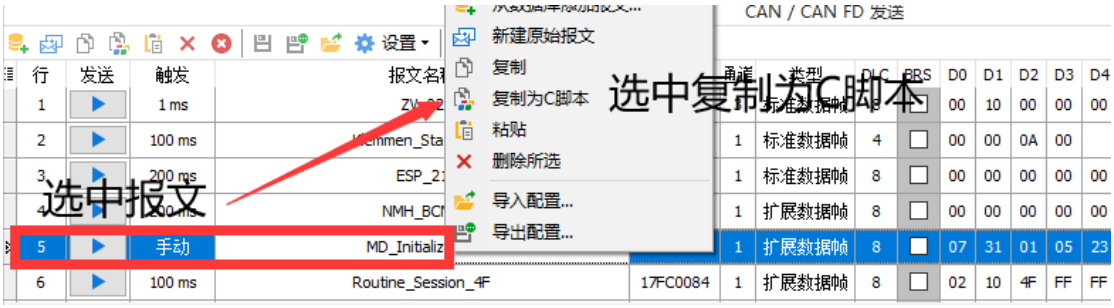


1.15.6.3. 基于数据库操作

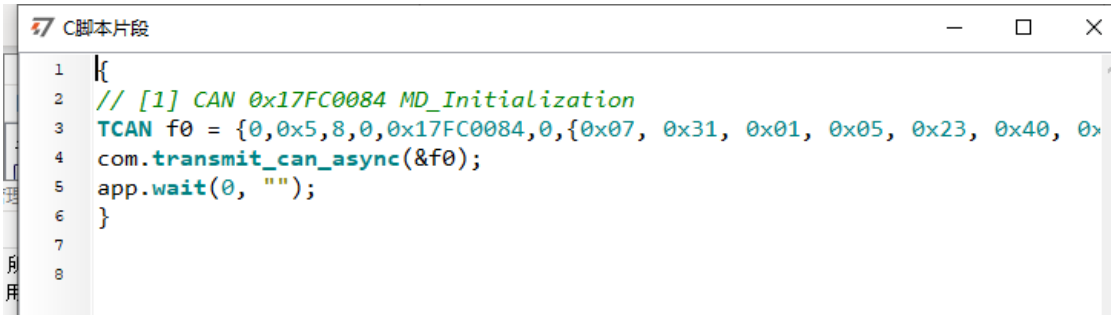
数据库的引入让用户不用直接处理原始 CAN 报文，便于用于直观的修改信号级的数据，详细的操作记录见章节：数据库变量操作。

1.15.6.4. 通过发送窗口添加报文发送函数

用户在发送窗口中验证结束过后，如果想把这部分代码转换为 C 脚本，提供了如下简洁的操作方式：

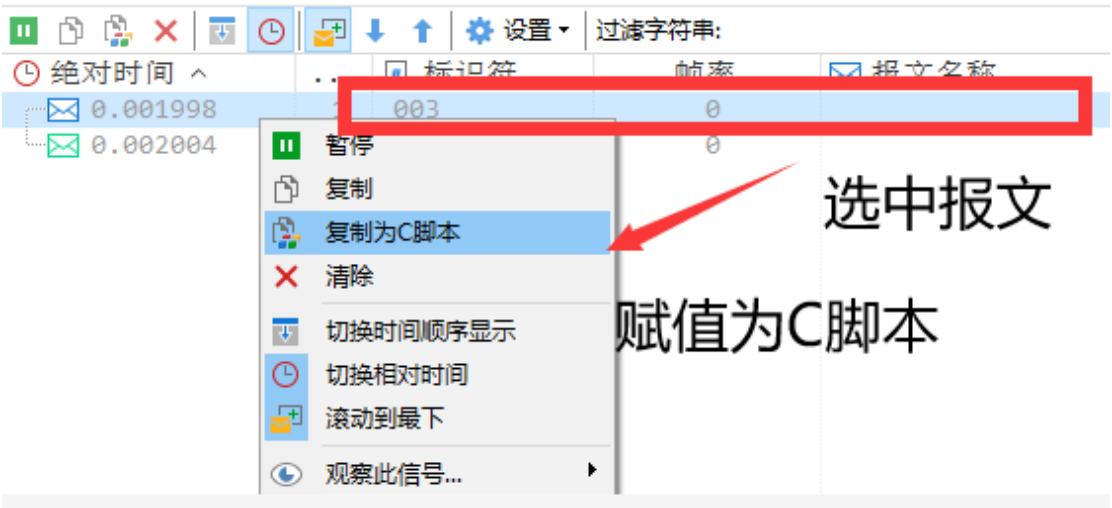


就会生成发送该报文的片段，如下所示，用户把该报文片段拷贝到 C 脚本编辑器中即可：



1.15.6.5. 通过 Trace 窗口添加报文发送函数

在不熟悉怎么添加发送报文函数的时候，可以通过如下机制快速编写 C 脚本。如下图所示：



系统自动生成调用的 C 代码，如下所示，用户把这部分代码拷贝到 C 脚本编辑器中即可：

C脚本片段

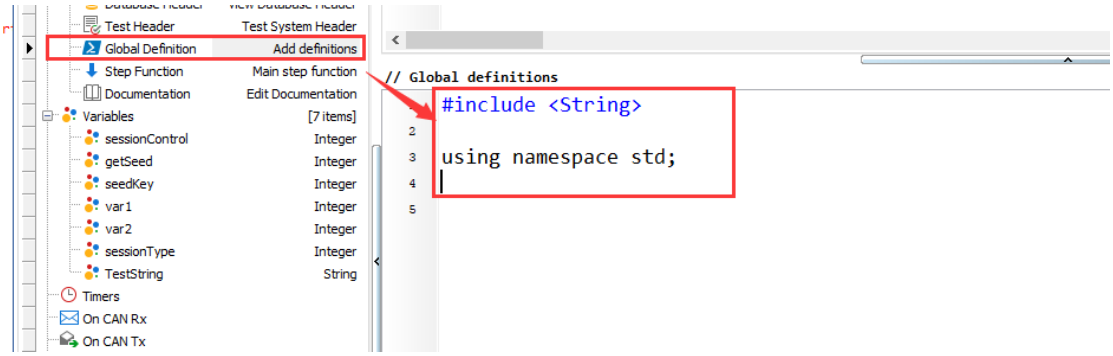
```
{
// [1] CAN 0x3
TCAN f0 = {2,0x0,3,0,0x3,1479918626,{0x01, 0x02, 0x03, 0x83, 0xCF, 0x7F,
com.transmit_can_async(&f0);
app.wait(0, "");
}
```

1.15.7. 引用 Windows 库文件

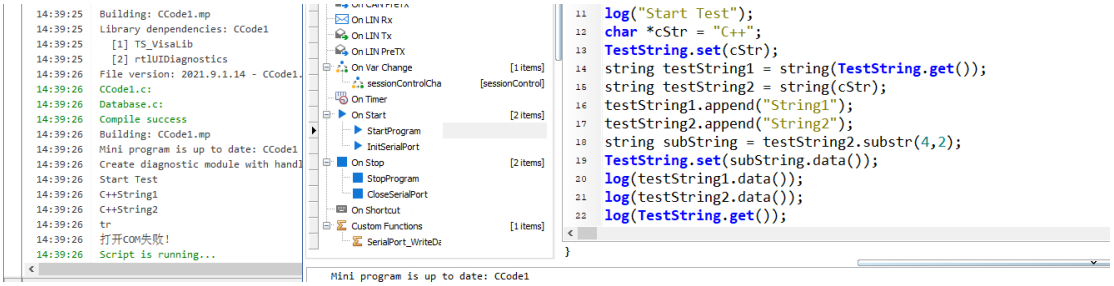
TSMaster 的 C 脚本采用的是标准的 C/C++编译器。因此，用户是可以引用 Windows 系统的资源进行任意扩展的。下面举两个例子来讲解如何使用 Windows 的 API 文件。

1.15.7.1. 使用字符串库文件

A. 第一步：在全局文件 Global Definiation 中添加引用库文件：



B. 第二步：此时就可以直接在脚本中使用 string 等变量类型，如下：



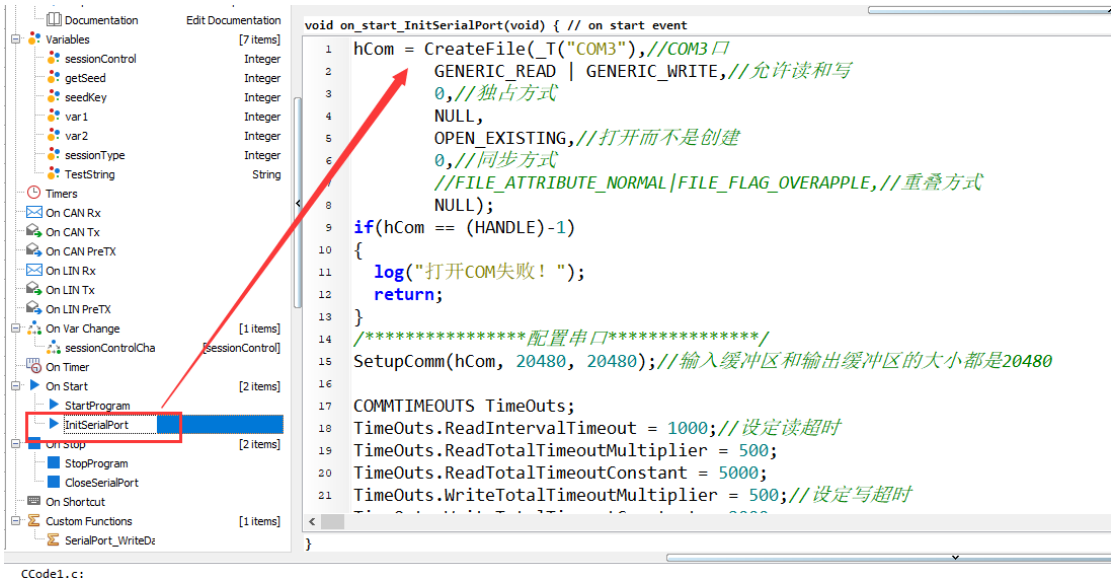

```
log("Start Test");
char *cStr = "C++";
TestString.set(cStr);
string testString1 = string(TestString.get());
string testString2 = string(cStr);
testString1.append("String1");
testString2.append("String2");
```

```
string subString = testString2.substr(4,2);
TestString.set(subString.data());
log(testString1.data());
log(testString2.data());
log(TestString.get());
```

1.15.7.2. 使用电脑串口

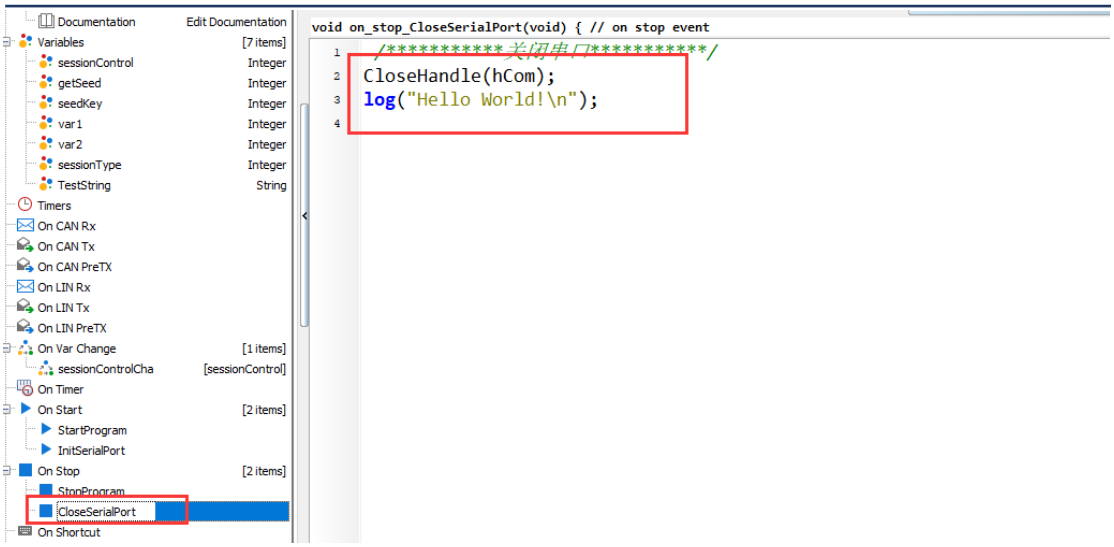
A. 第一步：在全局文件 Global Definiation 中添加引用库文件：

B. 第二步：在启动函数中添加初始化串口代码：

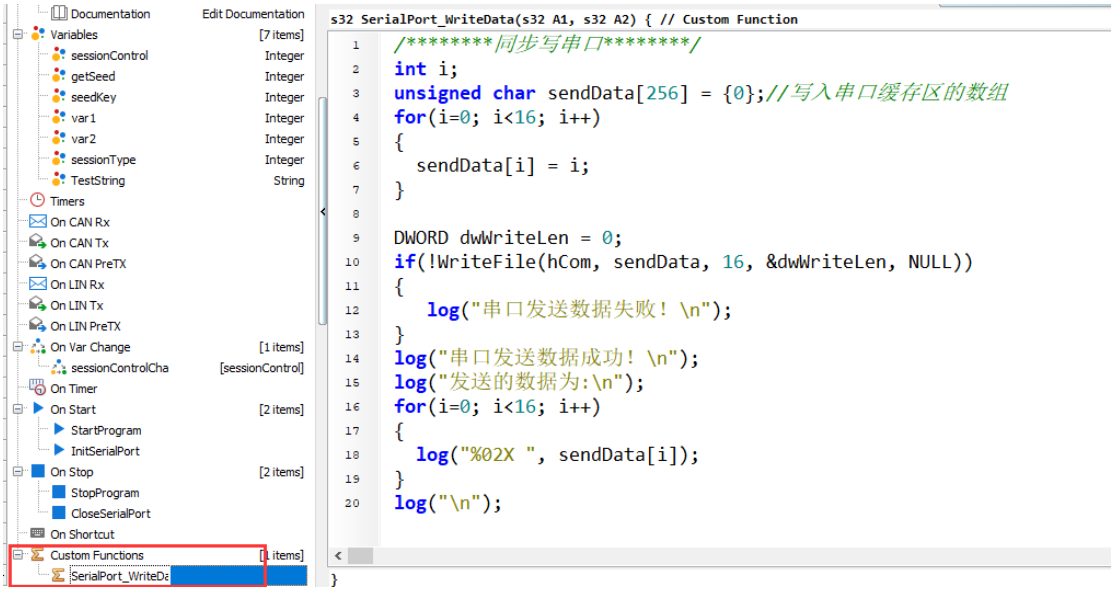


如上图所示：CreateFile，SetupComm 都为 windows 原生 API，通过添加头文件过后就可以直接使用。

C. 第三步：在退出函数中添加关闭串口代码：

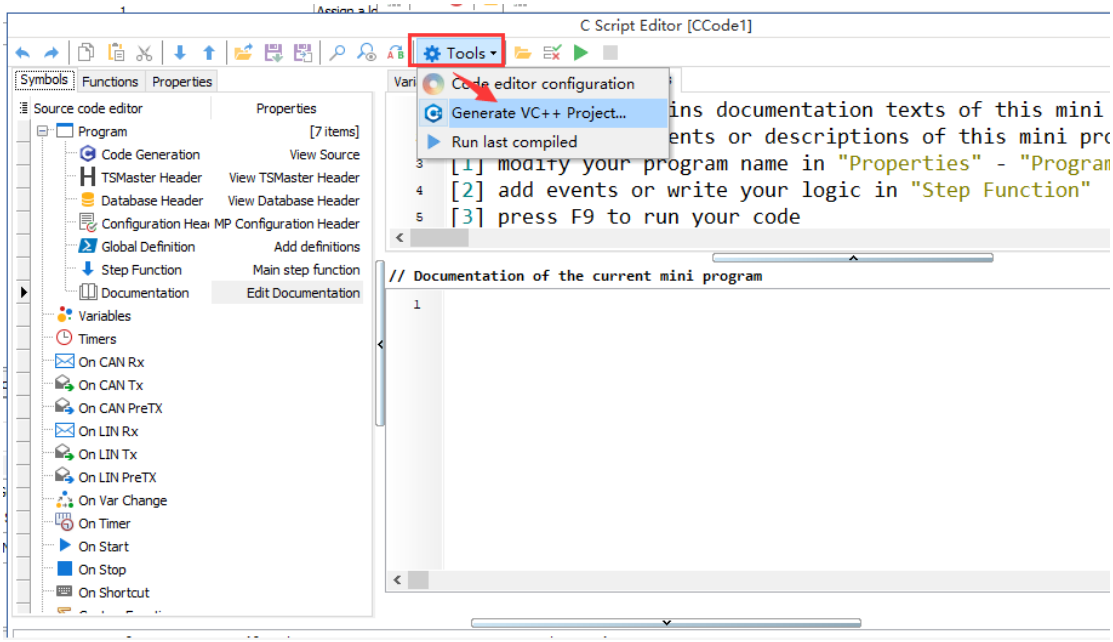


D. 第四步：添加发送串口数据函数



1.15.8. 导出 C 脚本到 Visual Studio 工程中调试

1.15.8.1. 生成代码调试工程

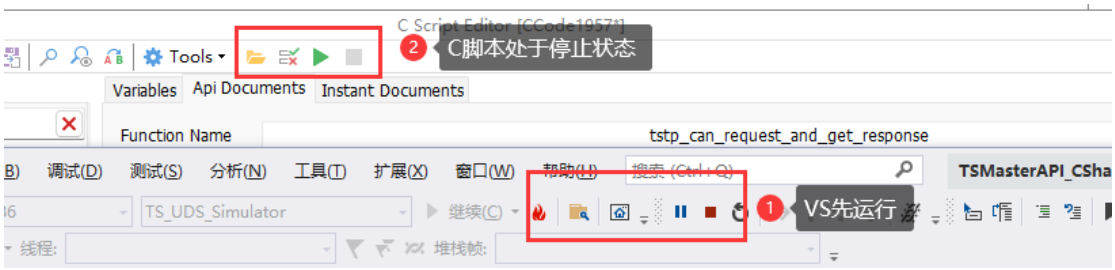


如上图所示，在 C 脚本面板中选择 Tool->Generate VC++ Project，选择文件夹存储生成的工程即可。生成的工程已经完成了相关的配置和关联。直接运行该工程即可附着到 TSMaster 上进行调试。

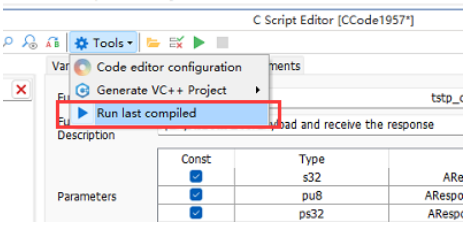
1.15.8.2. 在 VS 中直接调试 C 脚本

脚本到处到 VS 中过后，用户想在 VS 中直接调试 C 脚本代码，这样就可以充分利用 VS 的代码调试能力。采用如何步骤，就可以建立 VS 代码和 TSMaster 之间的关联：

- 首先在 VS 工程中完成代码的编译，并点击 Debug 运行。此时 TSMaster 中脚本保持不运行状态。



- 在 C 脚本中，选择 Tools->Run last compiled，此时 VS 调试环境和 TSMaster 之间建立关联，如下所示：



注意：一定要按照上述步骤来，否则无法在 TSMaster 和 VS 之间建立关联，也就无法使用 VS 的 Debug 功能了。

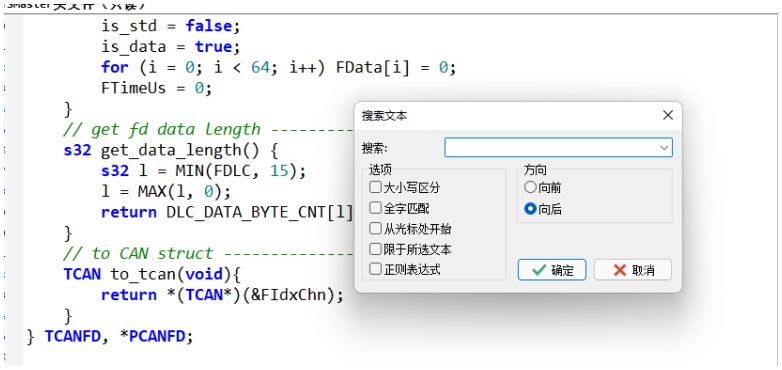
1.15.9. 代码检索/替换

1.15.9.1. C 脚本中检索代码

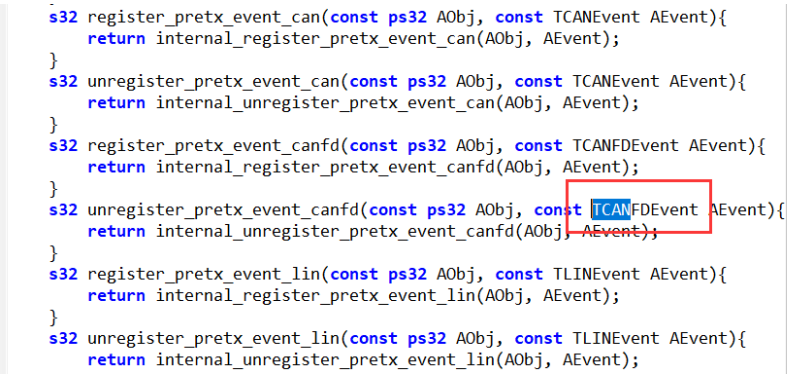
在进行程序开发的时候，检索代码是很常见的操作。在 TSMaster 的小程序脚本中，要实现代码的检索，主要通过如下的方法：

第一步：在代码编辑器中调出查询窗口，有两种方式：Ctrl+F，或者点击脚本工具上的快捷图标 

，这两种方式的效果是完全一样的，执行过后，弹出查询界面如下所示：



第二步：在搜索框中输入要检索的内容，比如检索 CAN 数据类型定义 TCAN，并点击确定，则检索到第一条符合要求的数据，如下所示：



```
s32 register_pretx_event_can(const ps32 AObj, const TCANEvent AEvent){
    return internal_register_pretx_event_can(AObj, AEvent);
}
s32 unregister_pretx_event_can(const ps32 AObj, const TCANEvent AEvent){
    return internal_unregister_pretx_event_can(AObj, AEvent);
}
s32 register_pretx_event_canfd(const ps32 AObj, const TCANFDEvent AEvent){
    return internal_register_pretx_event_canfd(AObj, AEvent);
}
s32 unregister_pretx_event_canfd(const ps32 AObj, const TCANFDEvent AEvent){
    return internal_unregister_pretx_event_canfd(AObj, AEvent);
}
s32 register_pretx_event_lin(const ps32 AObj, const TLINEvent AEvent){
    return internal_register_pretx_event_lin(AObj, AEvent);
}
s32 unregister_pretx_event_lin(const ps32 AObj, const TLINEvent AEvent){
    return internal_unregister_pretx_event_lin(AObj, AEvent);
}
```

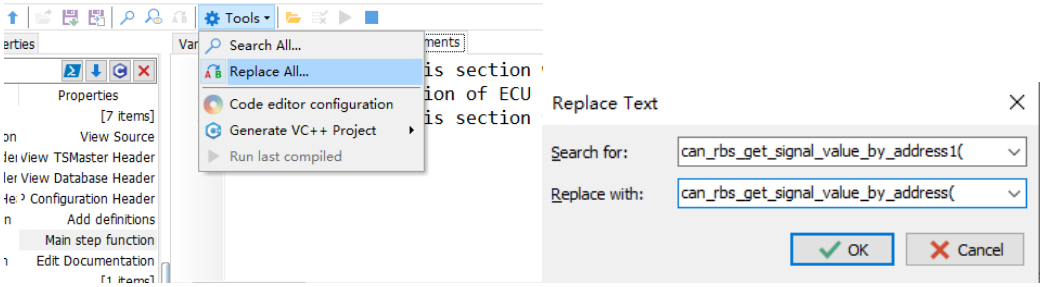
第三步：如果想持续检索此文本，有两种方式：F3 快捷键或者点击脚本工具上的快捷图标 

，这两种方式的效果是一样的。执行过后，就可以持续检索后面的文本工具，如下所示：

```
    void init_w_ext_id(s32 AId, s32 ADLC) {
        FIdxChn = 0;
        FIdentifier = AId;
        FDLC = ADLC;
        FReserved = 0;
        FProperties = 0;
        is_tx = false;
        is_std = false;
        is_data = true;
        *(u64*)(&FData[0]) = 0;
        FTimeUs = 0;
    }
} TCAN, *PCAN;
```

1.15.9.2. C 脚本中替换代码

在 C 脚本中，如果用户函数接口或者变量名称发生变化的时候，如果该变量使用的地方比较多，一个一个替换就非常麻烦，因此 TSMaster 的 C 脚本编辑器提供了全局替换的方法。基本路径：Tools->Replace All, 填写替换的参数，如下图所示：



1.15.10. 释疑

1.15.10.1. MiniProgram 之前运行是 OK 的，升级版本过后点击无法运行了。

解决办法：升级工程引入了新的库文件，重新编译 MP 程序，然后再次点击运行。

1.15.10.2. 为什么通过 get 函数无法读取信号值

情况描述：

在 Transmit 窗口中发送报文，但是在脚本或者 Panel 界面上无法观测到信号变化。

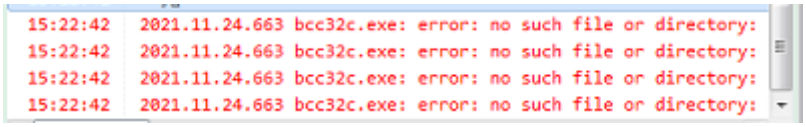
解释：

这是因为 Transmit 发送报文主要用于调试，发送的报文不能反向同步到 RBS 引擎中。因此，通过 Transmit 发送的报文，CAN RBS 里面的信号不会跟着变化。CAN RBS 信号跟 RBS 引擎，UI 界面，C 脚本是一体的。

1.15.10.3. 编译报错

情况描述：

编辑好脚本过后，编译脚本，报错如下：no such file or directory:...



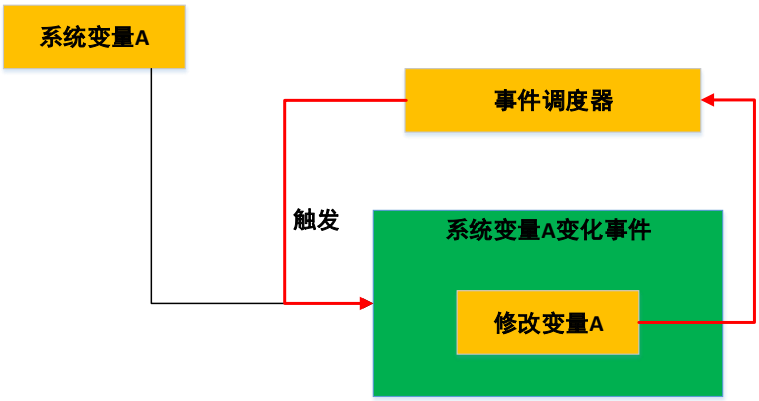
解释：

因为 TSMaster 内置 C 脚本解释器不支持中文路径，解决办法就是把文件夹拷贝到非中文路径下，即可解决问题。

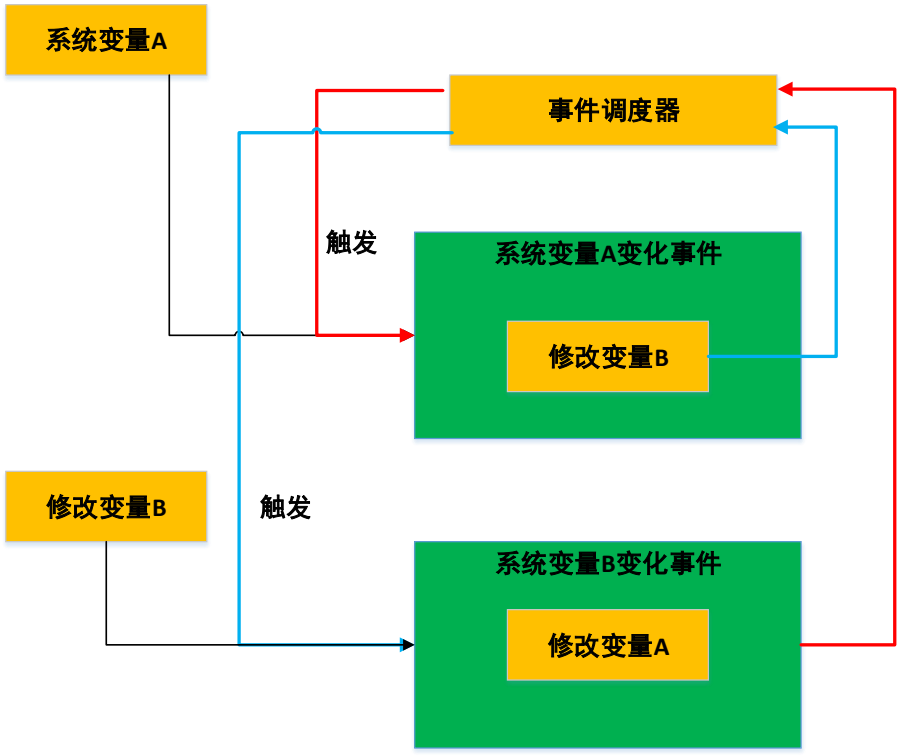
1.15.10.4. 系统变量触发死循环逻辑

TSMaster 的 C 脚本通过变量变化可以触发变量变换事件，但是如果发生如下情况的时候，会造成逻辑上的死循环。

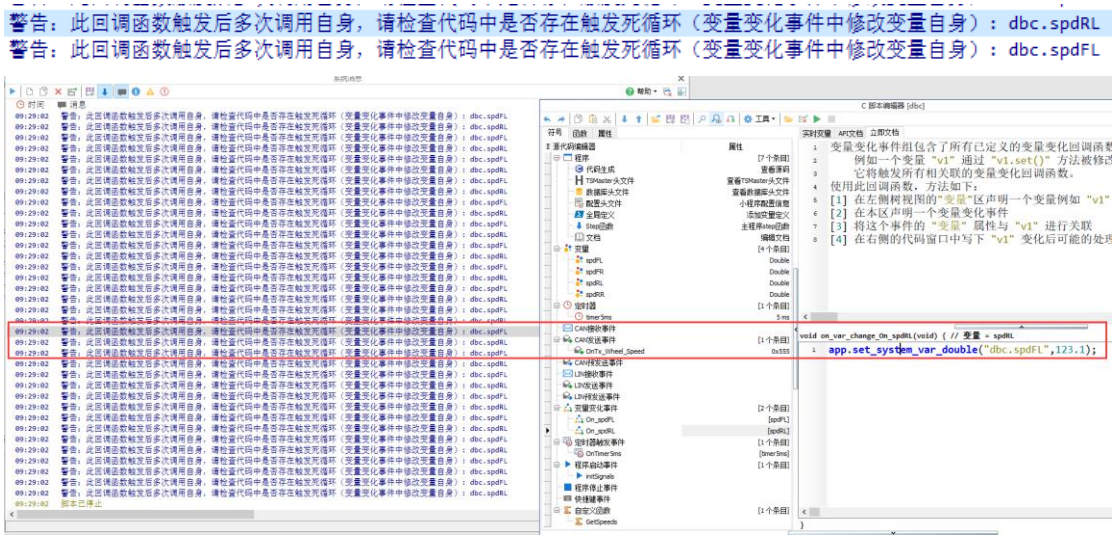
- 系统变量 A，其对应了变量变化事件 A_Event。如果在 A_Event 中直接执行修改系统变量 A 自身的函数。这就会造成死循环的逻辑。如下图所示：



- 系统变量 A，其对应了变量变化事件 A_Event；系统变量 B，其对应的系统变量变化事件为 B_Event；如果在 A_Event 中直接执行修改系统变量 B 自身的函数，然后在 B 中执行修改系统变量 A 的函数。这就会造成死循环的逻辑。如下图所示：



发送上述代码死循环过后，如果不做处理，会造成软件逻辑卡死。TSMaster 内置了代码检测机制。当发生逻辑死循环时，会在系统消息中打印消息进行提示。如下图所示：



当发生如上所示的提示的时候，用户一定要检查 C 脚本代码，去掉逻辑死循环的部分。否则会影响程序执行的效率。

1.15.10.5. 初始化函数(init_w_std_id)调用位置

```
void on_start_startCANTransmit(void) { // on start event
1  TCAN c;
2  c.FIdxChn = CH2;
3  c.init_w_std_id(0x123, 8);
4  txCAN.set(c);
5
```

如上图所示的脚本，本意是：设置该报文并且发送到通道 2 上面。但是实际测试的时候，发现报文发送到了通道 1 上面。如果通道 1 没有设置实物通道，该报文就发送失败了。

主要原因是，报文初始化函数如 init_w_std_id，会重置（reset）该报文内部所有数据，包括把 CAN 通道设置为 0。因此，用户初始化报文，设置报文属性的时候，一定要先调用 init_w_std_id 函数，再设置其他属性，这样设置的属性才是有效的。因此，整改代码如下所示：

```
void on_start_startCANTransmit(void) { // on start event
1  TCAN c;
2  c.init_w_std_id(0x123, 8);
3  c.FIdxChn = CH2;
4  txCAN.set(c);
5
```

1.15.10.6. 同名小程序触发异常

在 TSMaster 中，创建了同名称的小程序。就会触发如下异常：

CCode1.c:

Database.c:

Fatal: Could not open C:\Users\XYY\Desktop\FT-Tech\ECHO_R\bin\CCode1.mp (program still running?)

bcc32c.exe: error: unable to remove file: C:/Users/XYY/Desktop/FT-Tech/ECHO_R/bin/CCode1.mp: Can't destroy file: 拒绝访问。

bcc32c.exe: error: linker command failed with exit code 2 (use -Xdriver -v to see invocation)

这是因为，在 TSMaster 中，每一个小程序根据自己的名称拥有独立的运行空间。如果两个小程序使用同一个名称，则他们会同时访问同一个系统资源，肯定会造成冲突错误。

解决办法：

修改其中一个小程序的名称，防止出现同名称的情况。

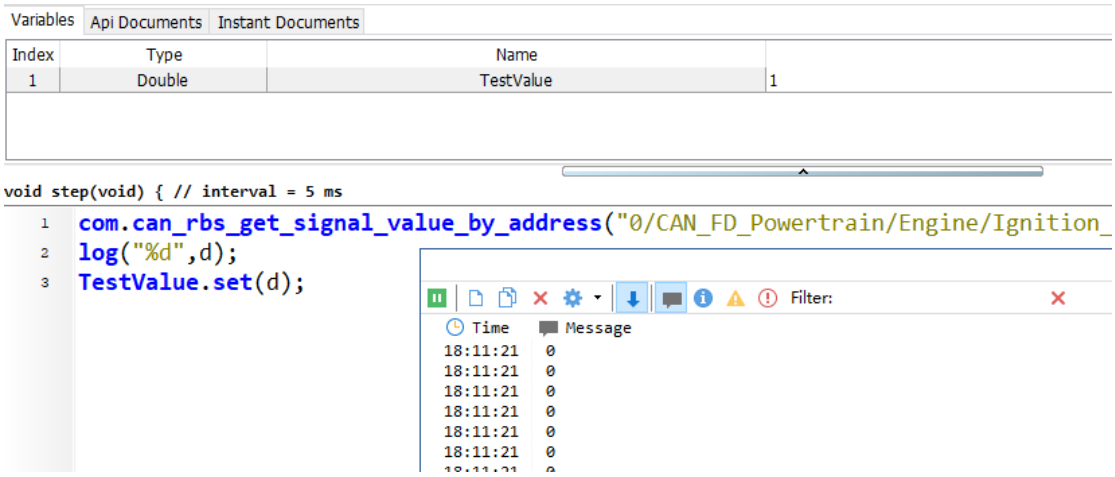
1.15.10.7. log 函数打印，无论是否有数据，打印出来总是 0

在 C 脚本中，代码如下图所示：

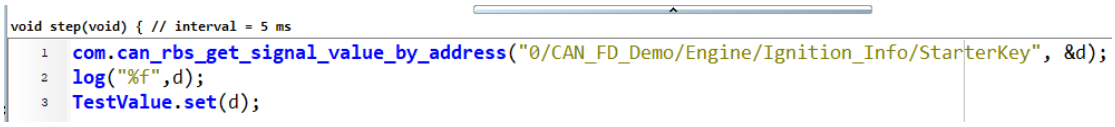
```
// Global definitions
1  double d ;
```

```
void step(void) { // interval = 5 ms
1  com.can_rbs_get_signal_value_by_address("0/CAN_FD_Demo/Engine/Ignition_Info/StarterKey", &d);
2  log("%d",d);
3  TestValue.set(d);
```

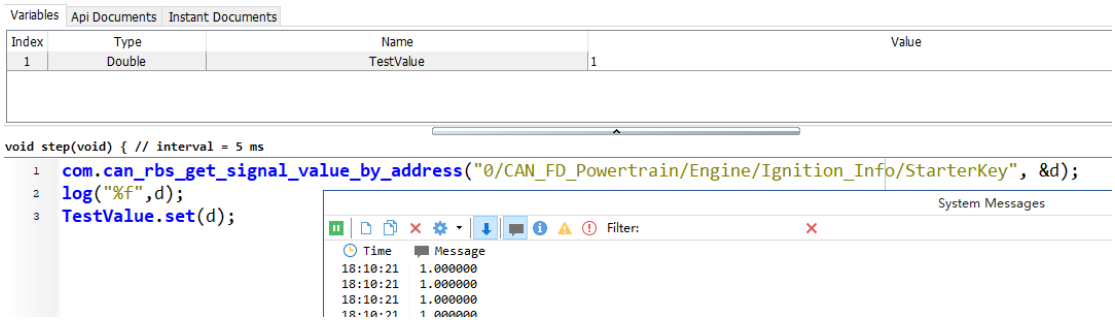
观测打印报告记录，发现无论读取出来的信号值 d 是否有其他值，打印出来的值总是为 0。打印数值和实际的数值不匹配，如下图所示：



这是因为，d 是 double 类型，但是 log 函数打印的时候，格式字符串是针对 10 进制数字的。解决办法，修改打印的格式字符串从"%d"修改为"%f". 解决此问题。



整改代码后，通过系统消息打印出来的字符串跟实际数值是匹配的，如下图所示：



总结：使用 log 函数的时候，注意格式字符串。

1.15.11. 附件

1.15.11.1. TSMaster Header

1.16. 小程序（MiniProgram）

为了扩展 C 脚本的功能，TSMaster 提供了一种被称为小程序的库文件封装机制，用户

在其他开发环境下（visual studio 或者其他开发环境）采用 C/C++，Pascal 等封装的库文件，也能够便捷到应用到 C 脚本中。这种机制极大的增大了 C 脚本的灵活性，比如用户想把数据存储到数据库中，或者从 Excel 中解析数据，这些函数接口都可以封装到小程序库中，并被 C 脚本所调用。

1.17. 调用外部 DLL/LIB 程序

在用户自定义开发过程中，常常会遇到需要调用外部 DLL/LIB 程序文件的需求，这些文件可能是用户自己编写的，也可能是其它供应商提供的。TSMaster 支持调用外部二进制程序库，但必须通过一定的方法进行封装。本节内容以调用 NI 公司的 TDMS 文件记录程序为例，演示封装 DLL 库的方法，而 LIB 库的使用与 DLL 类似，用户可以在 visual studio 工程中执行类似操作实现。

1.17.1. 获取外部程序库

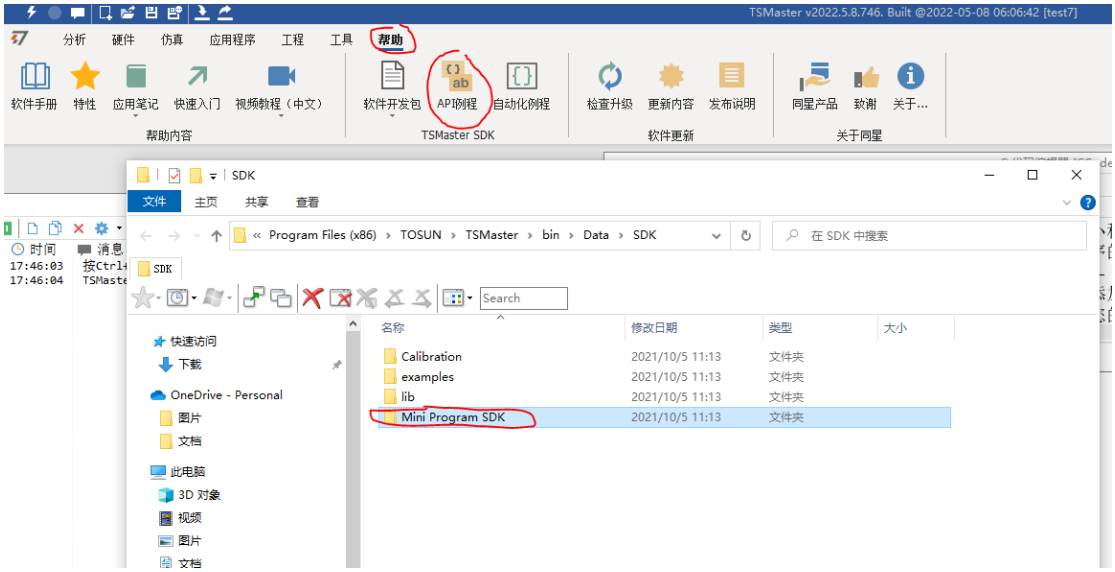
NI 公司的 TDMS 库可以通过链接 https://www.ni.com/content/dam/web/product-documentation/c_dll_tdm.zip 下载，对于外部库的使用，请注意以下限制：

1. 在下载外部程序库的过程中，请注意发布程序库的使用协议，同星公司仅提供外部程序库的使用环境，不为违反外部库使用协议的行为负责；
2. 外部库被 TSMaster 载入后，即成为了主程序的一部分，如果外部库发生了崩溃/内存溢出等行为，会导致 TSMaster 程序不稳定或崩溃，此时需要重新打开软件并卸载有问题的外部库；
3. TSMaster 仅支持 32bit-msvc 版本的外部库，请使用合适版本的 DLL/LIB 文件，否则会导致编译过程出错。

在附件的“tdms_example\TDM C DLL”目录下，可以找到解压后的 TDMS 外部库相关文件。

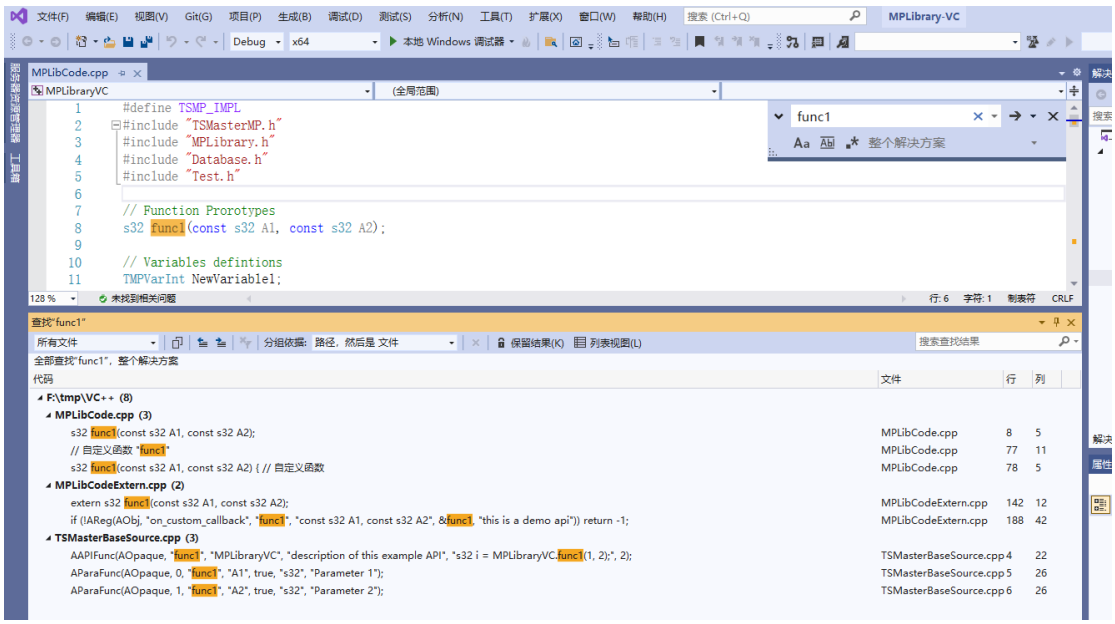
1.17.2. 准备外部库调用模板

该模板可以从 TSMaster 程序中拷贝，通过 帮助->api 例程-> Mini Program SDK 目录下名为 VC++的工程，将该工程复制到其它用户自定义的位置备用即可。在附件“tdms_example\tdms_sdk”可以看到为 TDMS 功能准备的工程。



1.17.3. 编辑模板并生成 DLL

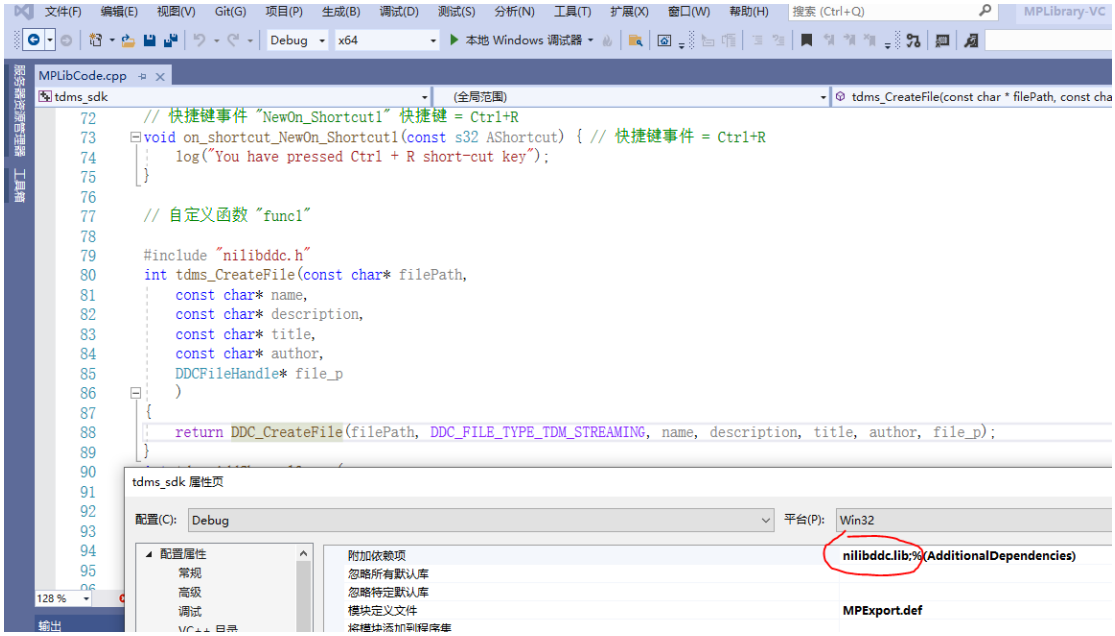
无论外部库是 DLL 还是 LIB，或者两者均有，都可以在模板工程中被调用。需要注意的是，为了能够使得 TSMaster 能够正常识别，用户在准备模板工程过程中，除了实现自己的逻辑外，还需要提供函数的注释，参数说明等信息。具体方法可以打开默认模板文件，在全局搜索 fun1 关键字，该函数为一个示例函数，它出现的地方，也就是用户需要添加自定义内容的地方。



如上图所示，可以看到在 MPLibCode.cpp 文件中，实现了 fun1 函数的声明和实现，在 MPLibCodeExtern.cpp 文件中，向 dll 管理模板注册了函数 fun1 的存在，在 TSMasterBaseSource.cpp 文件中，向 dll 管理模板注册了 fun1 函数的相关参数信息。

在集成 TDMS 功能过程中，首先需要将编译过程需要的 h 文件和 lib 文件拷贝到工程目录下，并在工程链接器中将 LIB 文件作为输入。对于外部函数，例如 TDMS 库自带的

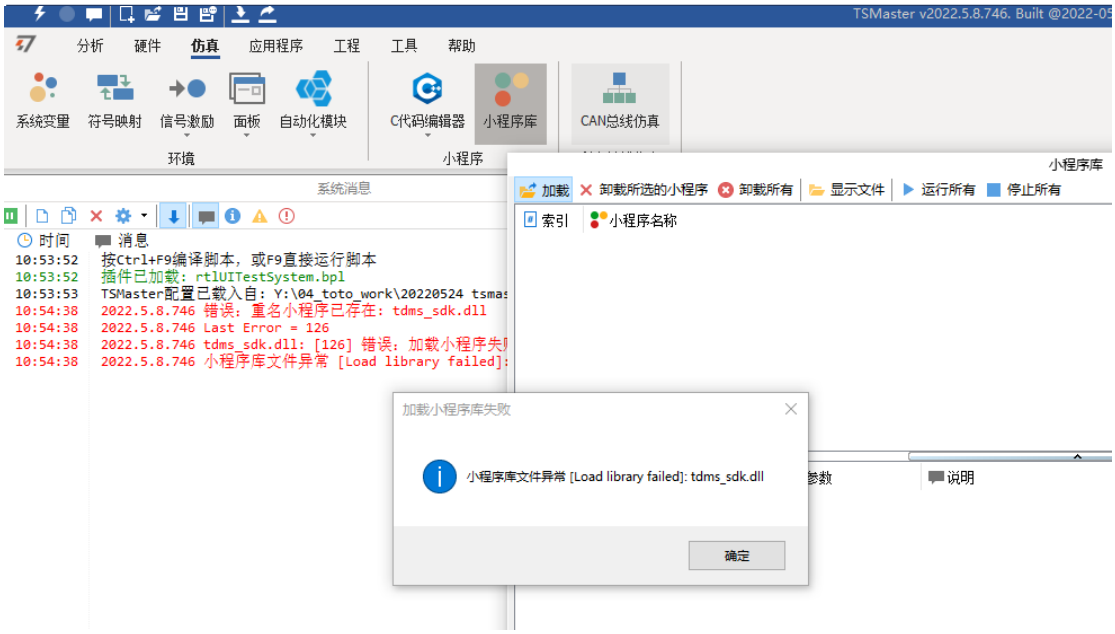
DDC_CreateFile 函数，在模板中使用一个新的函数进行封装，将其命名为 tdms_CreateFile。尽管 LIB 文件的函数可以直接导出，但通常建议新建一个函数对它进行封装，一是可以统一函数的名称，便于用户区分，而是所有 API 的函数返回值必须为 int 类型，如果原生外部库不是该返回类型，则必须通过传递指针等方式获取返回值，此时必须通过封装的形式使用。



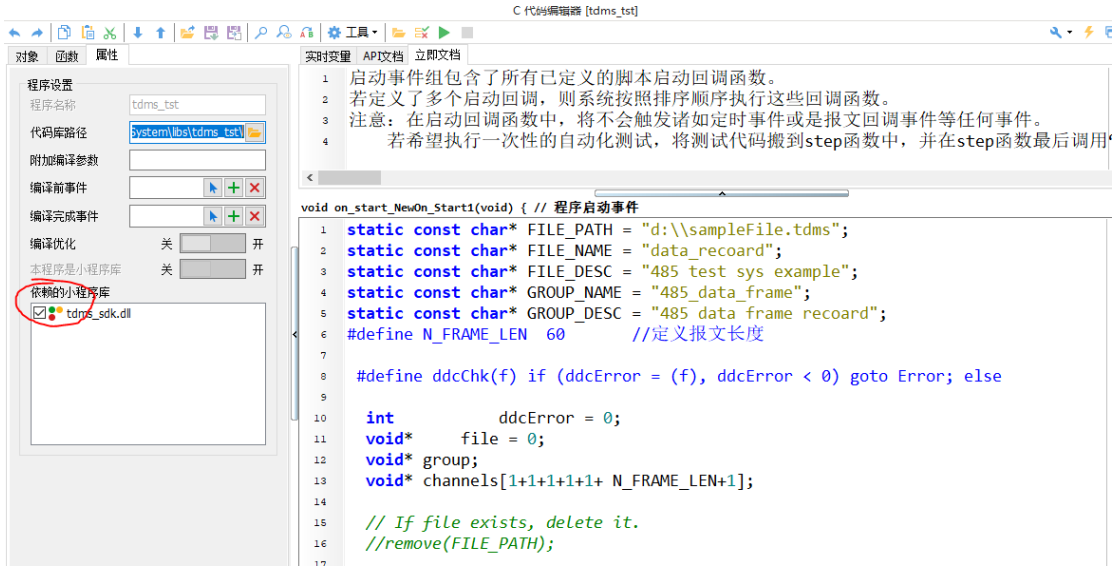
TDMS 其它的函数封装过程请参考示例工程，请务必注意封装函数代码和函数注册代码需要匹配，才能正确工作。基于该模板，在 Debug/Release-x86 模式下，可以生成所需的“tdms_sdk.dll”。

1.17.4. 在 TSMaster 工程中调用模板 dll

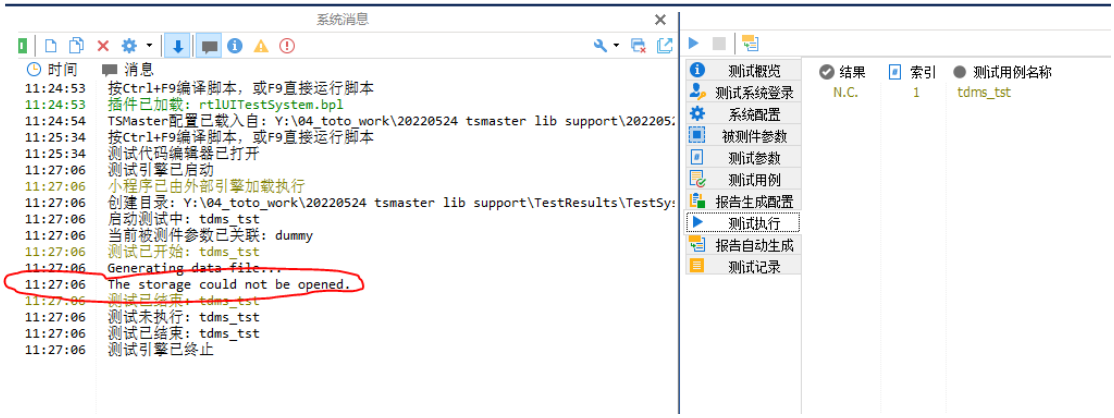
为了在 TSMaster 中调用“tdms_sdk.dll”，可以将该 dll 直接拖入，或者通过仿真->小程序库->加载的方式载入，如下图所示，直接加载会失败，这是由于模板 dll 依赖了 TDMS 运行时 DLL 文件，依赖文件在“tdms_example\TDM C DLL\dev\bin\32-bit”目录下，由 NI 公司提供，将所有文件拷贝到 TSMaster 工程\Plugins\Dependencies 目录下（这个目录需要手动创建，默认 TSMaster 不会创建），再载入模板 dll，即可正确载入。



打开示例工程“tdms_example\tdms_example”，在测试系统的第一条用例，为 TDMS 文件生成测试代码。用户也可以新建小程序来调用，两者在操作上没有区别，都需先在属性窗口中，勾选所需外部库，然后在脚本程序中调用所需函数。如下图所示，脚本运行后会在 D 盘根目录下创建一个示例 TDMS 文件。



直接运行该脚本，可以发现程序能够运行但并未按需创建 TDMS 文件，通过运行记录文件查看错误消息，可以看到提示未能打开存储设备。这个问题是 TDMS 库所特有的，因为它的依赖文件中，除了 dll 外，还包含一个名为 DataModels 的文件夹，TSMaster 在使用“tdms_example\Plugins\Dependencies”目录下的依赖文件时，不会对文件夹进行关联，因此需要手动把该文件夹复制到 TSMaster 安装程序目录，例如“C:\Program Files (x86)\TOSUN\TSMaster\bin”。有些外部库只有 dll 依赖，则不需要手动复制操作。



解决依赖的文件夹后，即可正确生成 TDMS 文件。用户可以参考以上过程，实现自己的逻辑。整个过程需要对 visual stuido 环境具备一定了解，如有不清楚的地方，请参考示例工程。

1.17.5. 在 TSMaster 工程中调试模板 dll

在使用模板 dll 过程中，不可避免存在调试过程。用户可以修改模板 visual stuido 中项目属性->常规->输出目录，将 dll 的目录输出到调试使用的工程中“MPLibrary”目录下。例如将“tdms_example”示例中 dll 生成到“tdms_example\tdms_example\MPLibrary”下，然后先运行 TSMaster 工程，再启动 visual stuido 调试功能。调试过程与小程序调试方法一致，可以参考小程序的调试过程。

1.18. 图形编辑面板(Panel)

TSMaster 图形编辑面板让用户能够开发自己的图形窗口，用于处理报文的收发，信号解析显示等功能。

1.18.1. 工具栏

1.18.1.1.  模式选择按钮，主要包含如下模式：

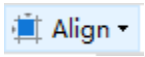
(1)  按下状态时候，当前 Panel 处于编辑模式下，用户可以增加删除控件，编辑控件属性的。

(2)  弹起状态，当前 Panel 处于测试运行模式，显示的是该面板实际运行时的状态，用户不可进行编辑。

(3) 灰色状态，当前 Panel 处于运行状态。意味着当前 TSMaster 处于设备连接运行状态。如果用户想重新编辑界面，必须断开 TSMaster 连接，才可以进入编辑状态。

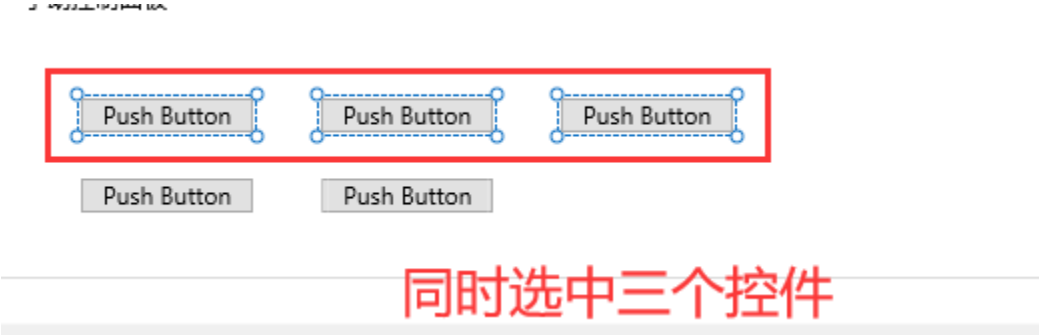
1.18.1.2.  层叠控制

当出现控件层叠情况时，把控件移动到前面和把控件移动到后面。

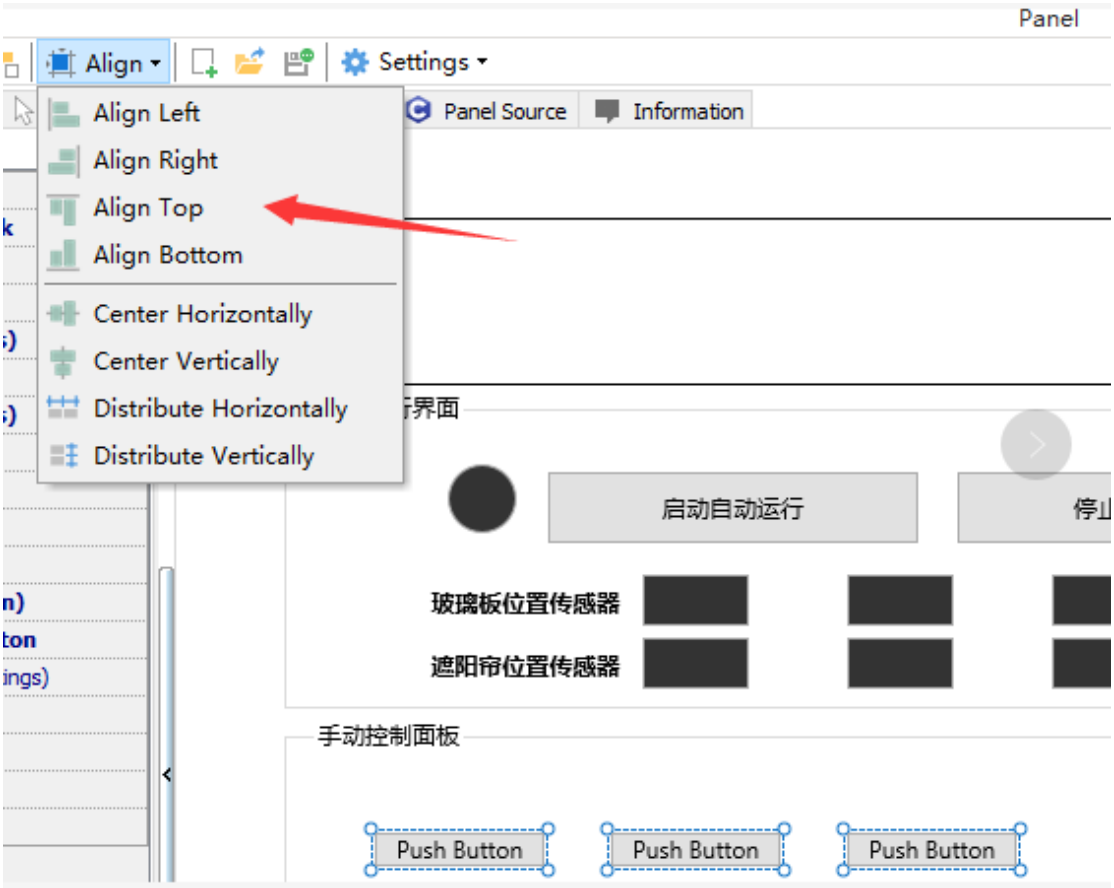
1.18.1.3.  对齐控件

控件对齐分为两步：

1. 首先选择需要对齐的多个控件：按住 Ctrl 键，鼠标点击多个按钮，则选中多个控件。如下所示：



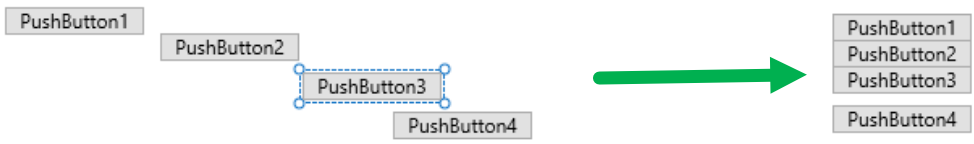
2. 选择顶部菜单栏的对齐按钮，如下所示：



对齐选项有：

- Align Left：左对齐
- Align Right：右对齐
- Align Top：上对齐
- Align Buttom：下对齐
- Center Horizontally：以中间模块为准水平对齐

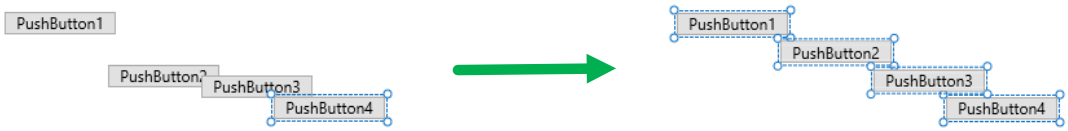
- Center Vertically：以中间模块为准垂直对齐



- Distribute Horizontally:：横向均匀排布

以最左边和最右边一个控件的坐标为准，计算控件之间间隔的平均值，然后在横向上均匀排列。

- Distribute Vertically:：纵向均匀排布



以最上面和最下面一个控件的坐标为准，计算控件之间间隔的平均值，然后在纵向上均匀排列。

1.18.1.4. 新建 Panel

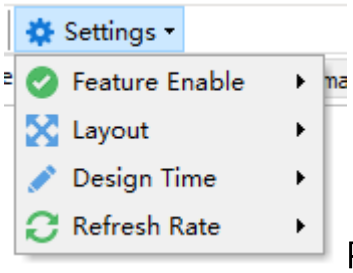
创建全新的 Panel，此操作将删除 Panel 所有现有的控件。

1.18.1.5. 加载配置

载入现有 Panel 配置文件。

1.18.1.6. 存储配置

存储当前 Panel 的配置文件。

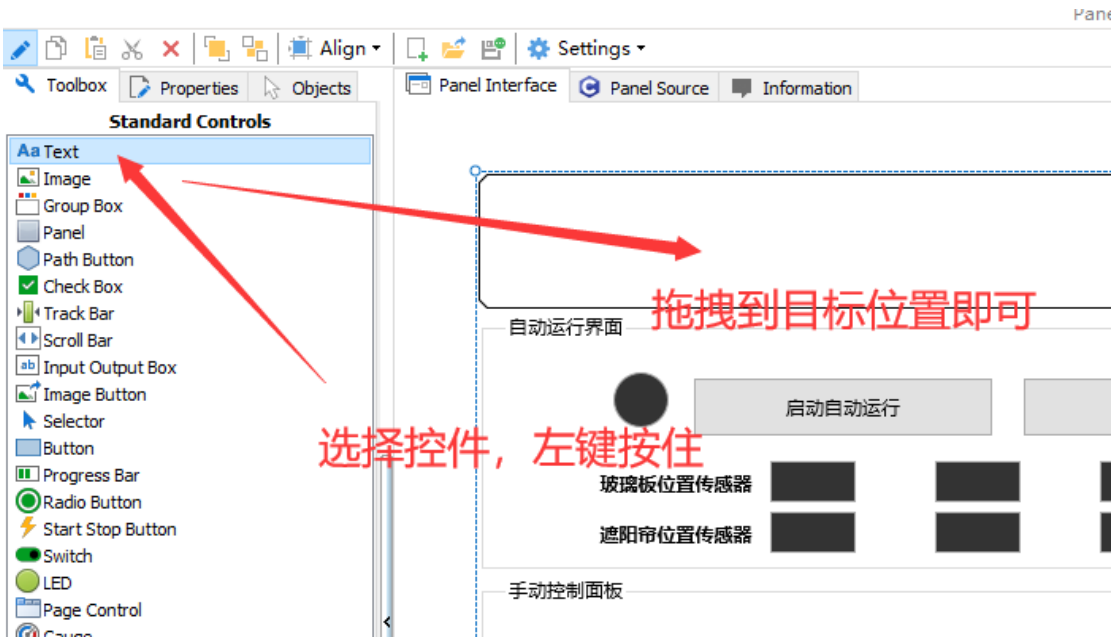
1.18.1.7.  Panel 参数配置

主要包含如下参数配置：

- Feature Enable：是否使能 Panel
- Layout：Panel 内部控件整体布局。
- Design Time：设计时是否显示链接标签和控件名称。
- Refresh Rate：配置 Panel 数据的刷新时间。推荐刷新时间为 300ms。如果电脑配置较高，可以配置更高的刷新率。

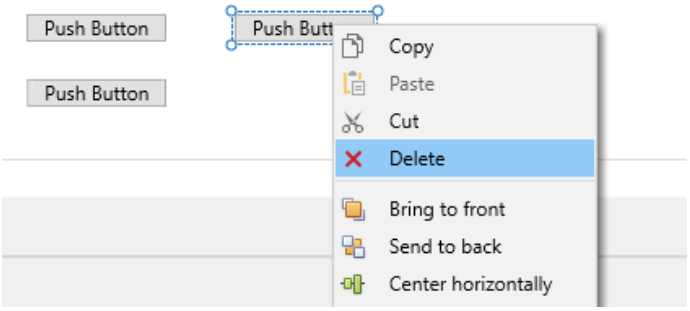
1.18.2. 控件基本操作

1.18.2.1. 添加控件



1.18.2.2. 删除控件

直接 Delete 键，或者右键，快捷菜单选择删除。



1.18.2.3. 移动控件

控件都会有自己所在的容器，在容器范围内，想调整位置，直接选中控件，左键按住，移动即可。如下所示：

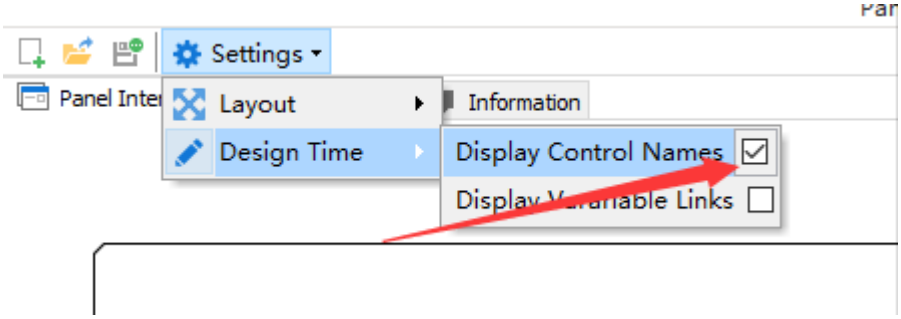
1.18.2.4. 移动到容器外面

TSMaster 不支持直接拖拽到容器外面，如果要把控件移出容易，可以采用剪切（Ctrl+X）+ 粘贴（Ctrl+V）的方式可以把控件移动到容器外面。

1.18.2.5. 显示控件类型名

在设计时（Design Time）阶段，默认是显示控件的名称（这个名称是控件的唯一 ID，是系统默认分配的，不能修改，在运行时不可见）的，如下图所示：

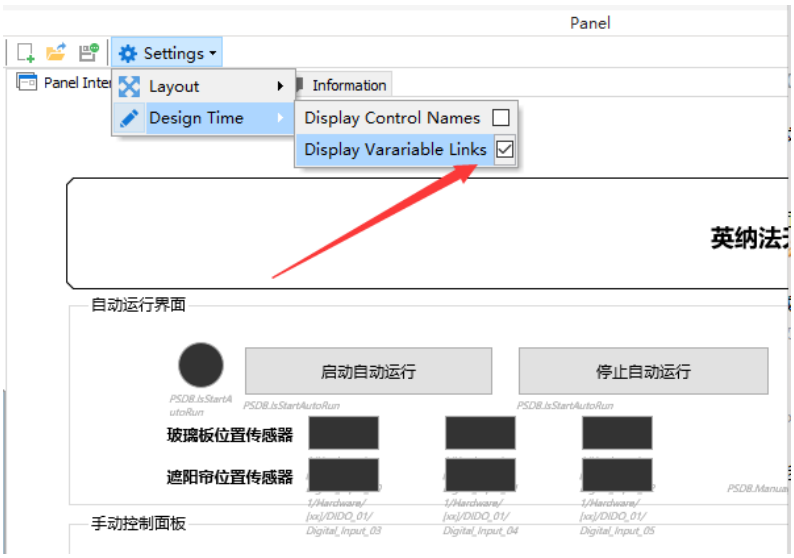
如果在设计阶段不想看到控件的唯一名称，可以到 Settings 界面中进行设置，如下所示：



1.18.2.6. 显示控件关联的变量链接

在设计时（Design Time）阶段，用户可以设置显示当前控件所关联的变量(CAN/LIN 信号或者系统变量等)，便于设计者清晰的知道当前该控件所关联的信号值，如下图所示：

在设计时阶段，打开和关闭该关联信号的显示，跟打开和关闭控件名称的显示操作是一样的，如下图所示：



1.18.3. 控件介绍

1.18.3.1. 曲线控件（Graphic）

1.18.4. 控件关联变量

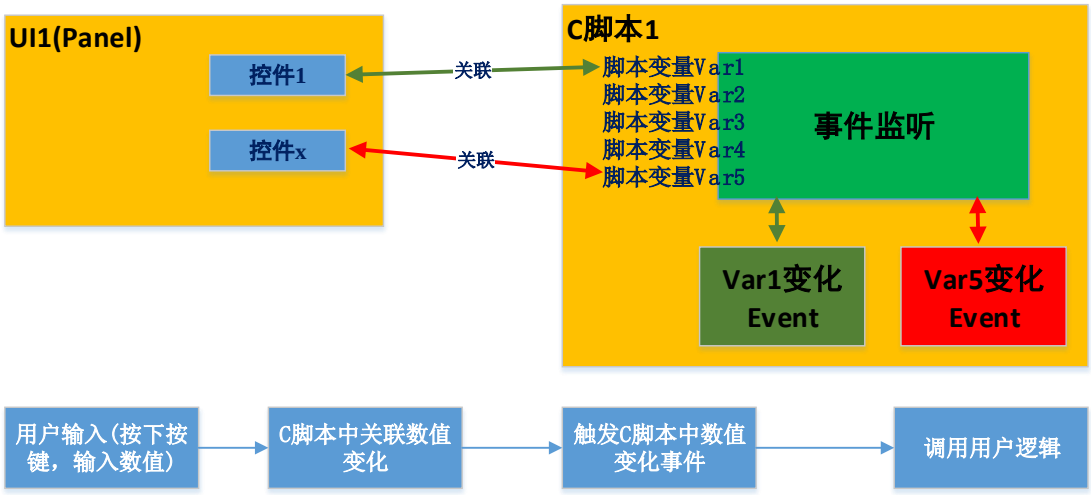
开关只是做显示
TextInputBox 只做显示

1.18.5. UI 事件

TSMaster 提供了丰富的 Panel 界面。通过关联变量，可以实现变量值的修改，或者变量值修改过后在 UI 界面上呈现，这些都比较好理解。如果用户要实现按下按键过后，TSMaster 软件发送一串报文，则需要配合 C 脚本实现 UI 事件机制。

1.18.5.1. UI 事件机制

Panel 的事件机制架构图如下图所示：



Panel 事件机制的实现，简单来说可以概括如下：Panel 中用户输入（按下按键，输入数值等）->改变关联的小程序变量的值->触发 C 脚本中数值变化事件->在事件中执行用户想执行的代码即可。下面以按键发送报文为例讲解 UI 事件的添加过程。

1.18.5.2. 按键发送报文示例

本示例实现的效果是：在 UI 界面上按下按键发送一帧报文；放开按键的时候，再发送另一帧报文。

- **第一步**：准备好脚本和 Panel。

- **第二步：**

1.18.6. 释疑

1.18.6.1. 为啥 DBC 解析是对的，Panel 上控件显示不对

问题描述：

为什么信号值 DBC 解析出来是 12%，但是仪表盘上总是显示 1.0（100%）

查看信号定义，因为信号值定义的范围是 0—100，单位为"%",解析出来的信号值为 12%，代表信号值为 12，然后单位是字符串"%"，并不是信号值是 0.12。因此，仪表盘的范围应该是 0—100，跟信号定义中的范围是一致的。如果仪表盘的范围设置为 0—1,则当信号值显示为 12%的时候，仪表盘当然会显示到最大值 1.

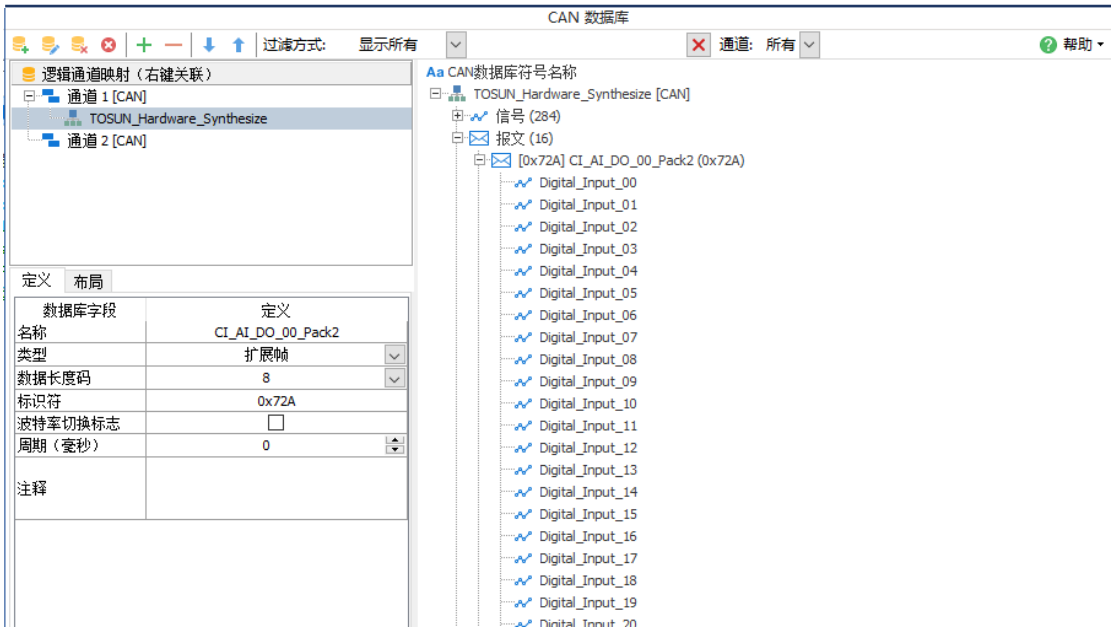
解决办法：

设置仪表控件属性，调整该控件的显示范围跟信号的范围一样。从 0-1 调整为 0-100，调整过后，仪表盘显示正常。

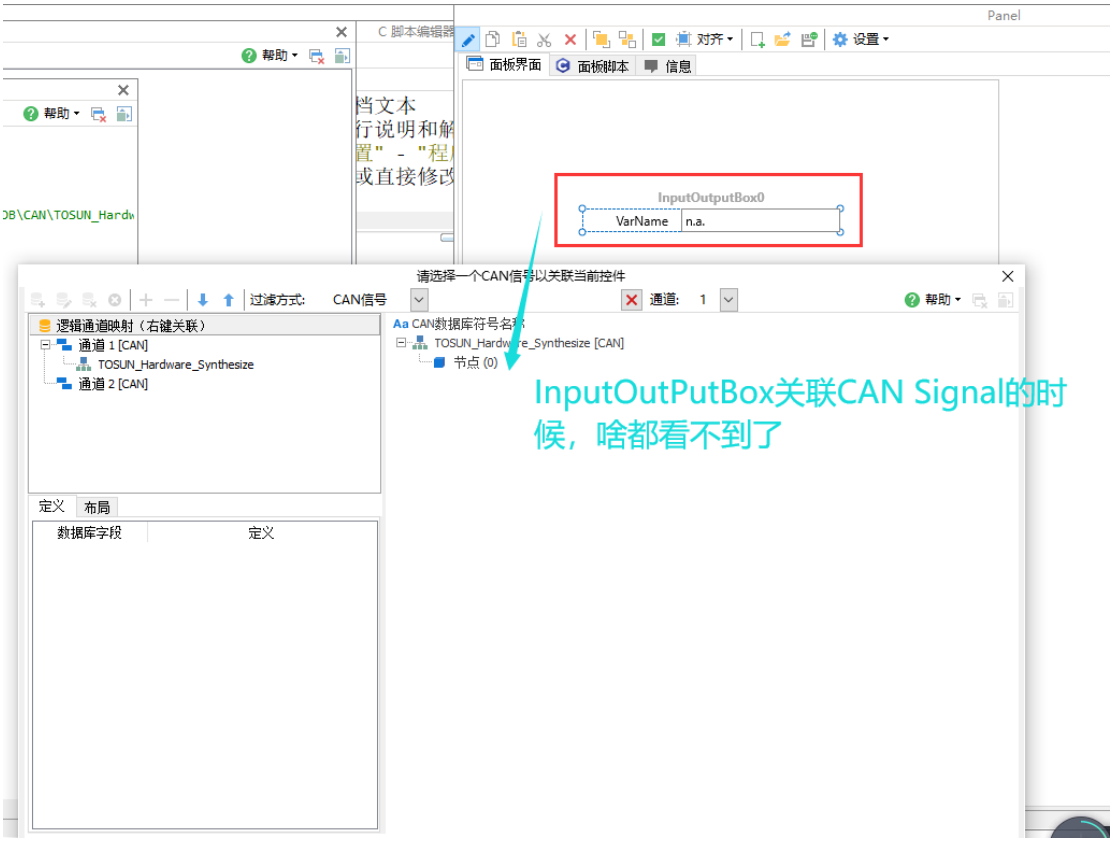
1.18.6.2. 添加了 DBC，为啥通过 Panel 关联信号的时候看不到任何信号？

情况描述：

加载了示例数据库：TOSUN_Hardware_Synthesize



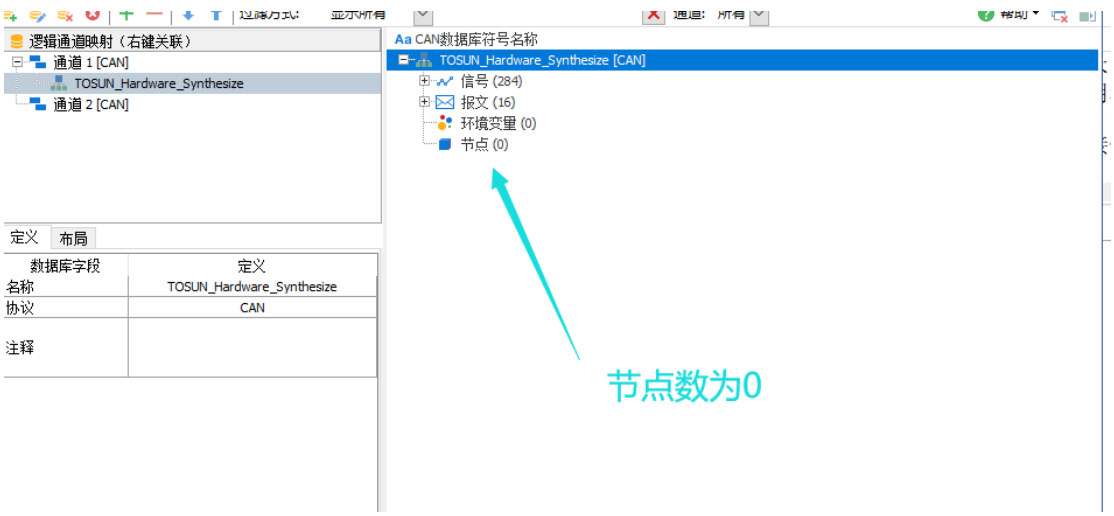
可以看到，有报文，有信号等。在面板中，添加 InputOutputBox，选择关联一个 CAN 信号，结果界面如下：



此时如果用单纯显示控件，如 Progressbar 等关联信号，是可以看到信号的。

原因分析：

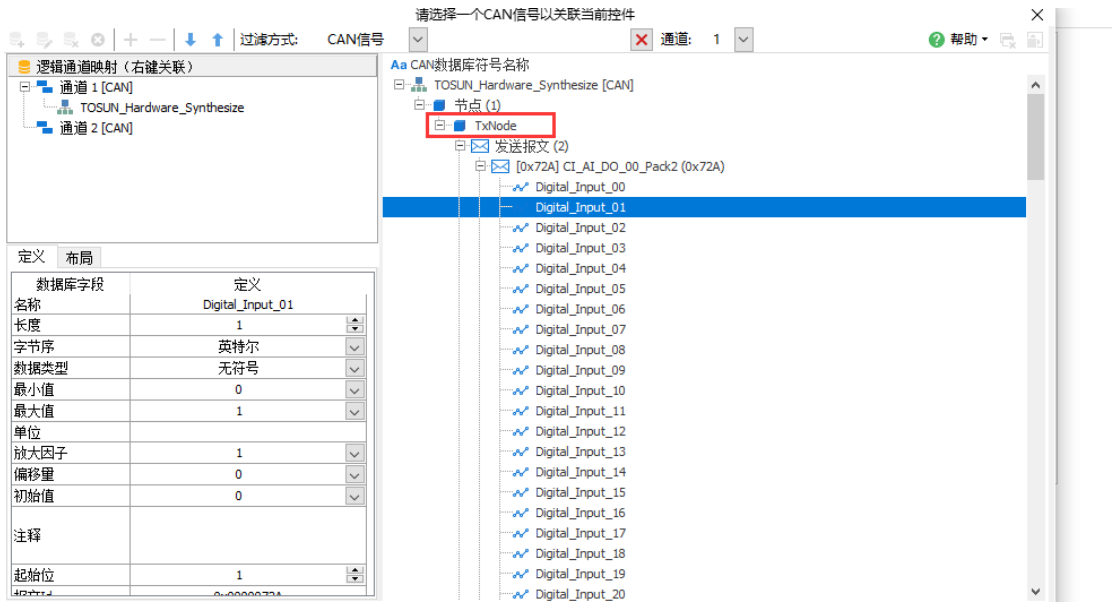
InputBox 是输入信号，这种信号从逻辑上来说需要被修改并发送到总线上，因此必须关联到一个发送节点的 CAN 信号上。然后我们继续看这个数据库：



可见，节点数量为 0，也就是没有任何发送和接收节点。这种情况下是无法启动 RBS 仿真的，因为没有发生节点，也就看不到任何跟发送节点相关的信号了。

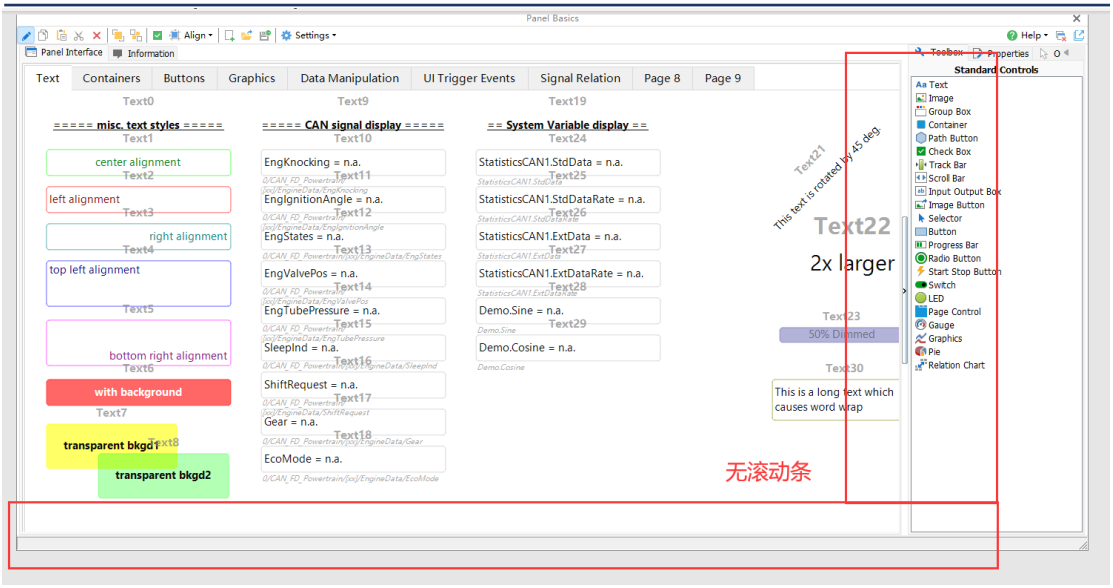
解决办法：

添加发送节点，并把报文和信号关联到发送节点上，就可以看到信号了。



1.18.6.3. Panel 中看不到滚动条

在设计 Panel 的时候，出现看不到窗体看不到滚动条（垂直+水平）的情况。如果窗体面积很大，超出了屏幕显示范围，就会出现一部分窗体被遮挡的情况。如下所示：

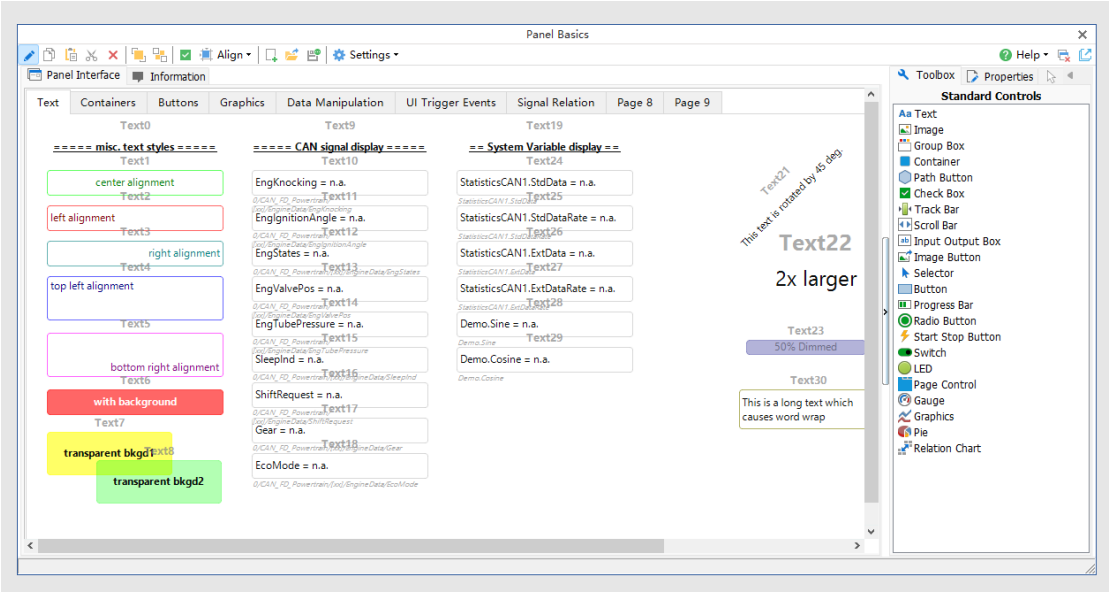


原因：

电脑屏幕设置了缩放，造成控件内部不能正确计算相对屏幕的尺寸，因此无法正确显示滚动条。查看电脑设置如下：

解决方案：关闭屏幕缩放，设置缩放为 100%，**重启软件**，即可重新看到窗体的滚动条。

可以看到，在窗体上重新出现了滚动条（垂直+水平），如下图所示：

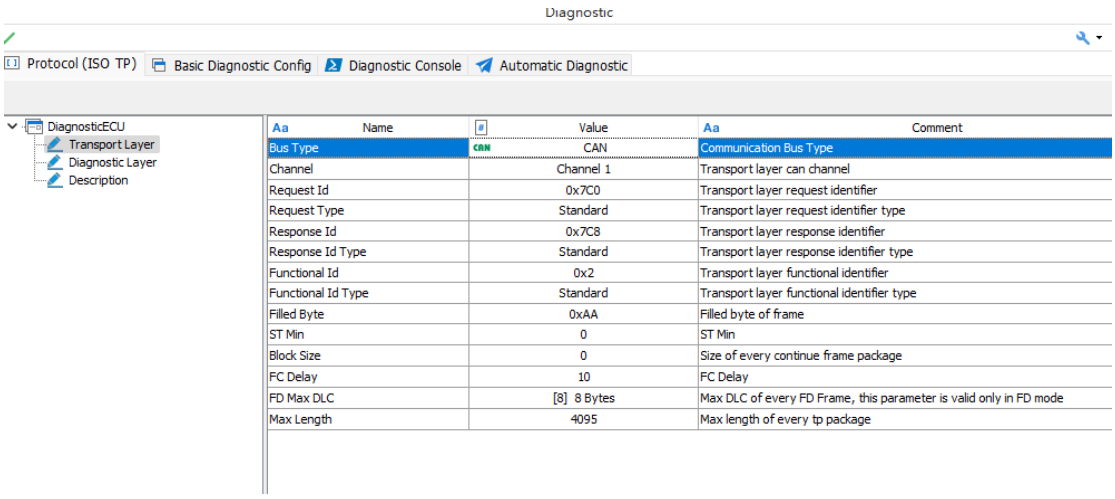

Panel Basics
Align
Settings
Help
Panel Interface
Information
Text
Containers
Buttons
Graphics
Data Manipulation
UI Trigger Events
Signal Relation
Page 8
Page 9
Text0
===== misc. text styles =====
Text1
center alignment
Text2
left alignment
Text3
right alignment
Text4
top left alignment
Text5
bottom right alignment
Text6
with background
Text7
transparent bkgd1
Text8
transparent bkgd2
Text9
===== CAN signal display =====
Text10
EngKnocking = n.a.
Text11
EngIgnitionAngle = n.a.
Text12
EngStates = n.a.
Text13
EngValvePos = n.a.
Text14
EngTubePressure = n.a.
Text15
SleepInd = n.a.
Text16
ShiftRequest = n.a.
Text17
Gear = n.a.
Text18
EcoMode = n.a.
Text19
== System Variable display ==
Text24
StatisticsCAN1.StdData = n.a.
Text25
StatisticsCAN1.StdDataRate = n.a.
Text26
StatisticsCAN1.ExtData = n.a.
Text27
StatisticsCAN1.ExtDataRate = n.a.
Text28
Demo.Sine = n.a.
Text29
Demo.Cosine = n.a.
Text21
This text is rotated by 45 deg.
Text22
2x larger
Text23
50% Dimmed
Text30
This is a long text which causes word wrap
Toolbox
Properties
Standard Controls
Text
Image
Group Box
Container
Path Button
Check Box
Track Bar
Scroll Bar
Input Output Box
Image Button
Selector
Button
Progress Bar
Radio Button
Start Stop Button
Switch
LED
Page Control
Gauge
Graphics
Pie
Relation Chart

1.19. CAN/CANFD 诊断(Diagnostic_CAN)

1.19.1. Diagnostic TP 参数配置

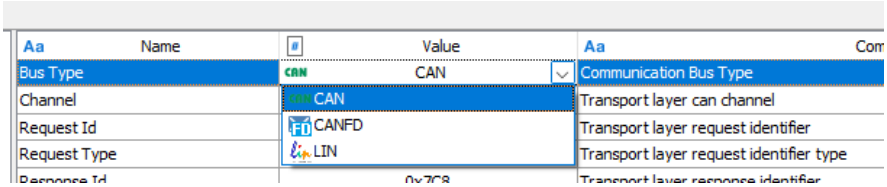
TSMaster 提供了诊断控制台基础功能，用户可以根据需求配置自己的发送和应答请求。按照如下步骤操作即可。

1.19.1.1. 传输层参数：

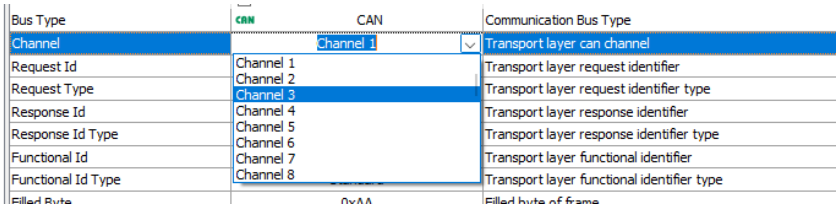


其中，各个参数解释如下：

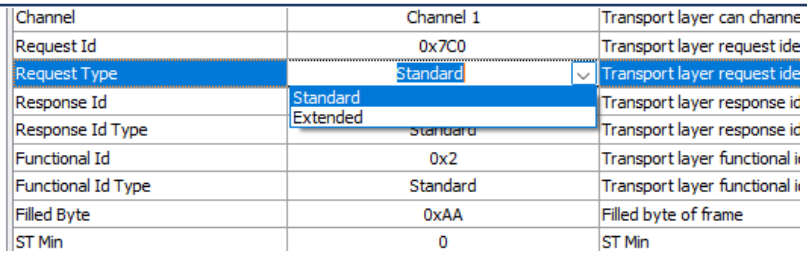
- Bus Type: 诊断传输层类型，目前已经支持 CAN/CANFD/LIN，接下来支持以太网和 Flexray 等。通过下拉列表可以选择，如下图所示：



- Channel：诊断模块用到的逻辑通道。TSMaster 支持多个诊断模块同时在线工作，这里用于选择当前诊断模块使用系统的哪一个逻辑通道。通过下拉列表进行选择，如下图所示：



- Request ID/Response ID/Function ID：设置诊断模块 PC 工具端的诊断请求/应答/功能帧的 ID。
- Request ID Type/Response ID Type /Function ID Type：设置诊断模块 PC 工具端的诊断请求/应答/功能帧的 ID 类型，是标准帧（11 位）还是扩展帧（29 位），如下图所示：

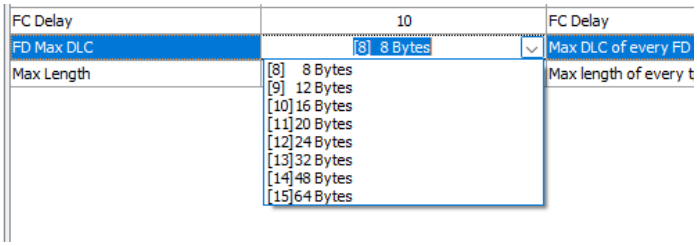


➢ Filled Byte：传输过程中，实际有效字节不足一个 CAN 报文数据端的时候，剩余数据段的填充字节。比如一帧 CAN 报文 8 个字节，如果有效传输字节是【0x02，0x10，0x02】,填充字节是 0xAA，则实际的报文字节是【0x02，0x10，0x02，0xAA，0xAA，0xAA，0xAA，0xAA】。

➢ STMin：最短接收时间间隔。TSMaster 诊断模块作为接收端，在接收连续帧报文的时候能够支持的诊断帧之间的最短时间间隔，这个参数是回复给诊断客户端的。设置为 0，表示支持以最短的时间间隔接收。

➢ BlockSize：接收 Block 的大小。TSMaster 诊断模块作为接收端，在接收连续帧报文的时候一次能够接收的数据块的大小。这个参数是回复给诊断客户端的。设置为 0，表示一次性能够接收任意大小的数据块。

➢ FD Max DLC：当传输层设置为 CANFD 的时候。此时，传输层单帧的最大传输字节数量是 64 字节（DLC=15），但是这个参数是可以调节的，调节范围如下所示：



➢ Max Length: 该参数对于普通 CAN/LIN 是无意义的。多帧传输的时候。当 DLC 长度= 8 字节的时候，首帧（First Frame）采用第 0 字节低四位+第一个字节的 8 位，共 12Bit 表示一次传输的包的大小，也就是最多 4095 个字节，如下图所示：

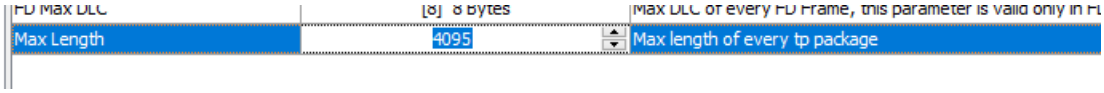
Byte0		Byte1	Byte2--7
0001	FF_DL(12 Bits)		数据段

但是 FDCAN 中，设置 DLC 长度>8 字节的时候，可以采用更多的 Bits 来传输信息。因此，FDCAN 的传输层支持采用第 2,3,4,5 四个字节共 32bit 来传输一个数据 Block 的长度。也就是说 FDCAN 的传输层一次支持传输最多 4 个 G 的数据。但是具体支持多少，让用户可以配置。

Byte0		Byte1	Byte2	Byte3	Byte4	Byte5	Byte6-Byte63
0001	000000000000		FF_DLC(32Bits)				数据段

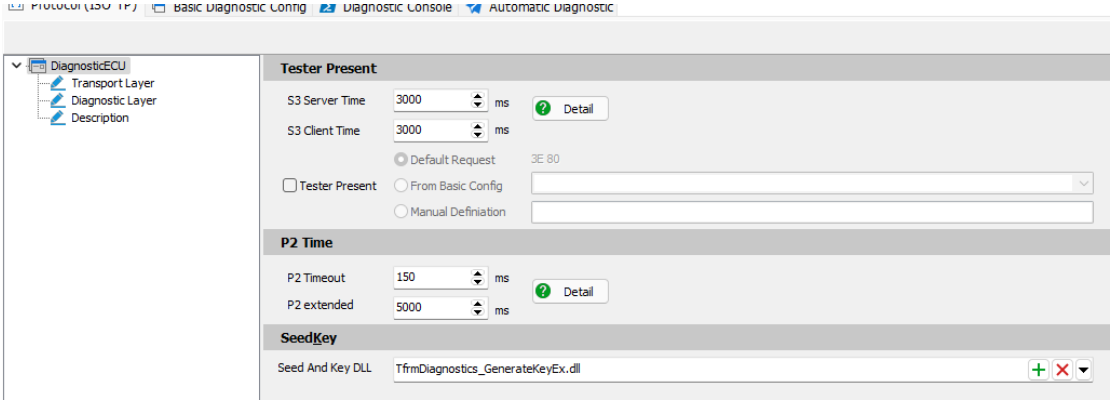
注释：第一个字节的高四位 = 1，表示该帧是首帧（First Frame），无论是 FDCAN 还是 Class CAN 的传输层都是如此。

比如，如下图所示配置位 4095 个字节，则跟普通传输层一样。如果配置为大于 4095，则要使用 FD 帧扩容的传输层。



1.19.1.2. 服务层参数：

服务层参数主要包含 S3，P2 时间参数，以及加载 SeedKey 的 dll。如下图所示：

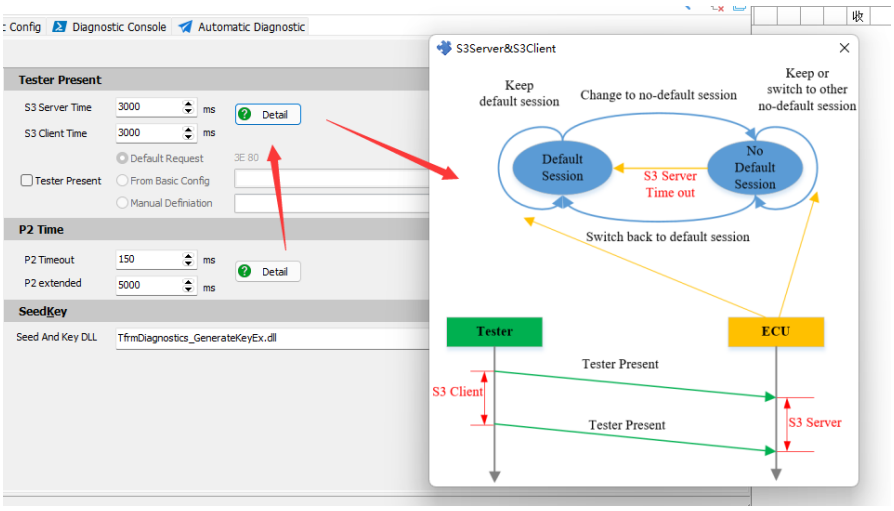


➢ S3 参数：包括 S3 ServiceTime 和 S3 Client Time。

S3 Service Time：表示该 ECU 从 Default 会话被切换到其他会话过后，经过多场时间会自动切换回默认会话的超时时间。

S3 Client Time：表示作为诊断 Tester 端，发送 TesterPresent 帧的时间间隔。

上述两个参数的示意图，可以点开 Detail 按钮，查看图示说明，如下图所示：

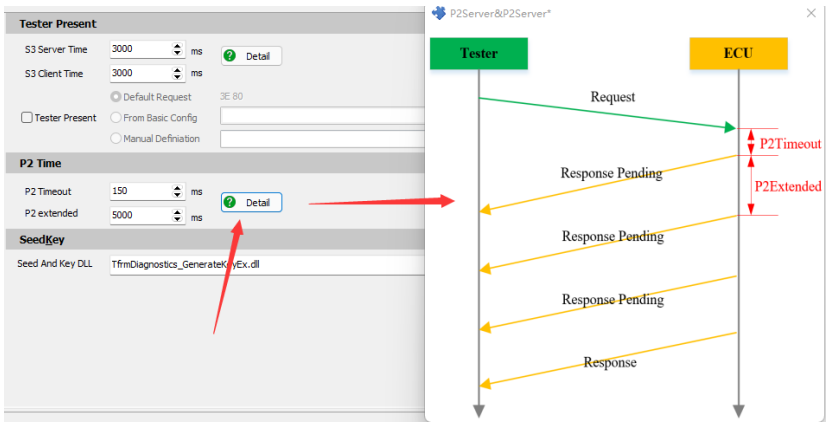


➢ P2 参数：包括 P2 Timeout 和 P2 Extended 参数。

P2 Timeout: 表示 ECU 收到诊断请求帧过后，最短回复的时间间隔。对于诊断工具端，该参数可以作为发送请求过后，等待回复的超时判断参数。比如诊断工具发送了一个诊断报文，P2Timeout 时间段内都没有收到回复，则认为请求失败，超时退出。

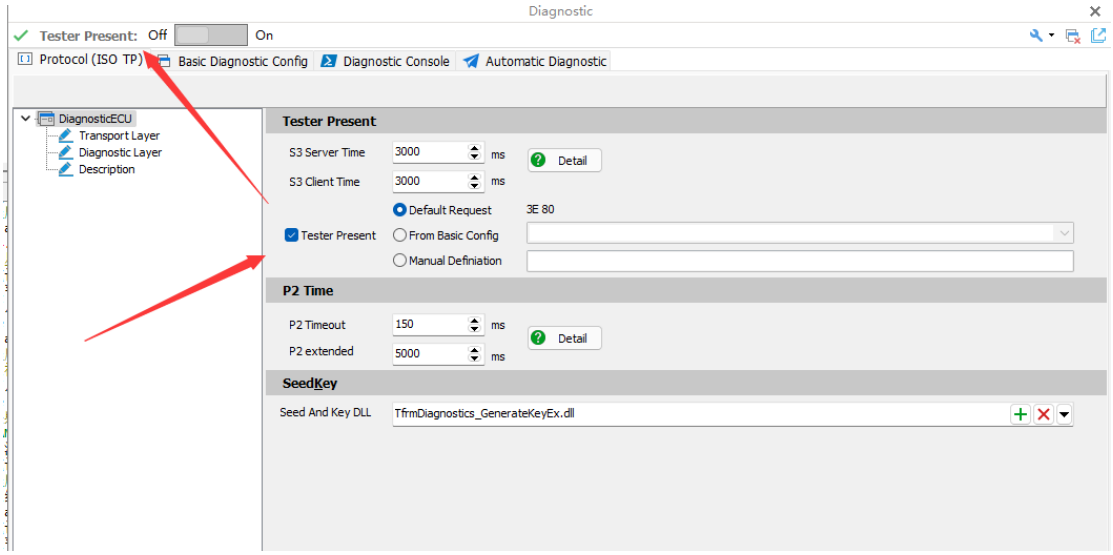
P2 Extended：当诊断工具发出诊断报文过后，被测 ECU 来不及在 P2 Timeout 时间段内做出应答，则回复一帧 7F XX 78 报文，告诉诊断工具端自己来不及响应，需要延长等待时间再回复。ECU 发送了延迟等待报文后，则把等待时间参数切换为 P2Extended。诊断工具端的超时判断参数在收到延迟等待报文后，需要切换到 P2Extended。

上述两个参数示意图如下所示：



➢ 使能 Tester Present 命令：

TSMaster 诊断模块中，可以选择配置并使能 TSMaster Present 命令，如下图所示：



当使能了该命令过后，在模块的最上方会出现启动 Tester Present 命令的开关。打开 Tester Presnet，则按照设定的 S3ClientTime 时间间隔发送该报文。

Tester Present 的发送字节是可选的。支持三种类型：

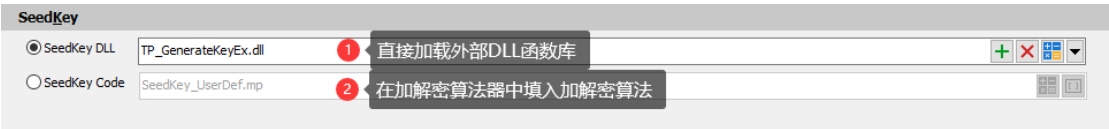
【Default Request】：也就是最常用的 0x3E 0x80

【From Basic Config】：从 Basic Config 中选择配置好的 3E 命令

【Manual Definition】：用于自定义的字节

1.19.1.3. Seed&Key

TSMaster 中提供了两种 Seed&Key 的处理方法：第一种就是最常规的加载主流的 SeedKey 的 DLL 函数；第二种是提供了内置的 Seed&Key 的解释器，用户可以在里面填入加解密的算法。在 TP 参数配置界面中，用户根据需要选择采用加载 dll 的方式还是直接编辑 seed&Key 源码的方式，如下所示：

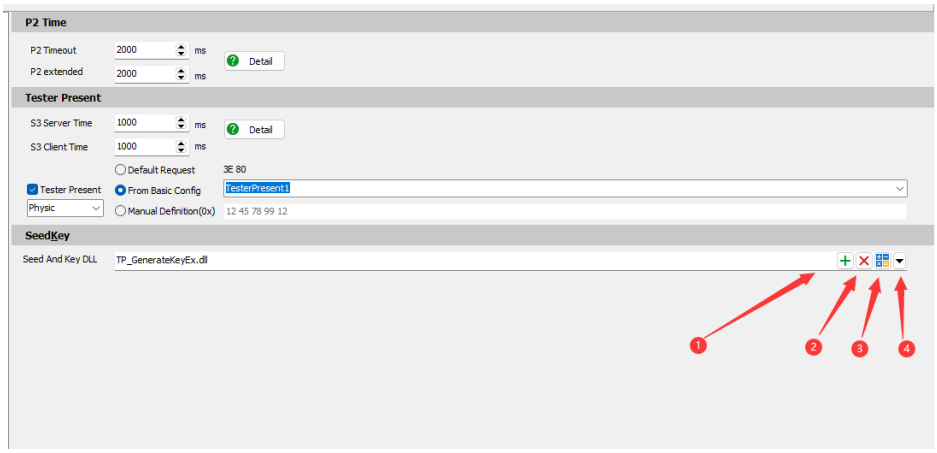


1.19.1.3.1. 加载外部 Seed&Key DLL

在诊断过程中，会设计到安全访问的问题，也就是所说的 Seed&Key。TSMaster 诊断模块支持通过 dll 载入 Seed&Key 算法，该算法 dll 跟主流工具的计算接口兼容，接口定义如下图所示：

```
KEYGEN_API GenerateKeyEx(
    const unsigned char* ipSeedArray,          /* Array for the seed [in] */
    unsigned int iSeedArraySize,               /* Length of the array for the seed [in] */
    const unsigned int iSecurityLevel,         /* Security level [in] */
    const char* ipVariant,                     /* Name of the active variant [in] */
    unsigned char* iopKeyArray,                /* Array for the key [in, out] */
    unsigned int iMaxKeyArraySize,             /* Maximum length of the array for the key [in] */
    unsigned int& oActualKeyArraySize)         /* Length of the key [out] */
```

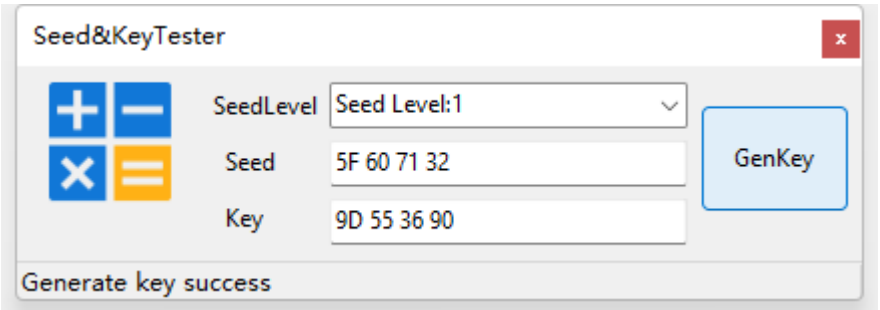
DLL 加载界面如下图所示：



【1】 加载 DLL

【2】 删除 DLL

【3】 DLL 校验器，通过此按钮，用户可以判断自己加载的 dll 接口是否正确，算法是否符合设计要求。如下图所示：



如上图所示界面，用户选择 Seed 的 Level 过后，输入 Demo Seed 值，点击 GenKey 进行判断。如果该 DLL 接口跟模板定义接口统一，则会输出提示信息：Generate Key Success，然后用户根据 Key 值跟目标值对比，进一步确认 DLL 中的算法是否符合设计要求。

【4】 打开 TSMaster 安装目录下 Seed&Key 接口工程所在的路径。用户可以拷贝该

工程添加自己的 Seed&Key 算法。

默认 SeedKey 函数接口

目前，要想被 TSMaster 的诊断模块直接加载，该 DLL 必须实现如下三种函数接口中的一种：

【1】 接口 1：

```
unsigned int GenerateKeyEx(
    const unsigned char* ipSeedArray,       /* Array for the seed [in] */
    unsigned int iSeedArraySize,            /* Length of the array for the seed [in] */
    const unsigned int iSecurityLevel,      /* Security level [in] */
    const char* ipVariant,                  /* Name of the active variant [in] */
    unsigned char* iopKeyArray,             /* Array for the key [in, out] */
    unsigned int iMaxKeyArraySize,    /* Maximum length of the array for the key [in] */
    unsigned int& oActualKeyArraySize);     /* Length of the key [out] */
```

【2】 接口 2：

```
unsigned int GenerateKeyExOpt(
    const unsigned char* ipSeedArray,       /* Array for the seed [in] */
    unsigned int iSeedArraySize,            /* Length of the array for the seed [in] */
    const unsigned int iSecurityLevel,      /* Security level [in] */
    const char* ipVariant,                  /* Name of the active variant [in] */
    const char* iPara,                      /* */
    unsigned char* iopKeyArray,             /* Array for the key [in, out] */
    unsigned int iMaxKeyArraySize,    /* Maximum length of the array for the key [in] */
    unsigned int& oActualKeyArraySize)      /* Length of the key [out] */
```

【3】 接口 3：

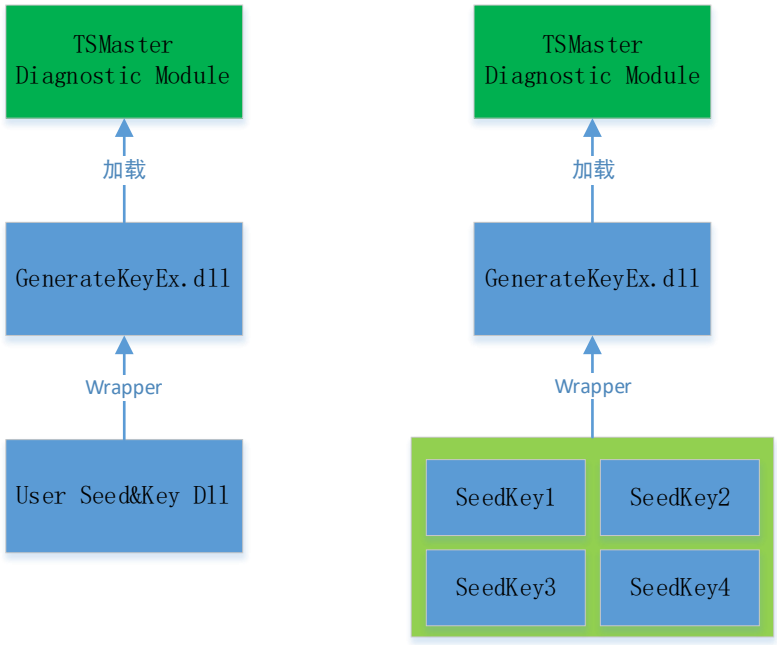
```
bool ASAP1A_CCP_ComputeKeyFromSeed(
    const unsigned char* ipSeedArray,       /* Array for the seed [in] */
    unsigned short iSeedArraySize,          /* Length of the array for the seed [in] */
    unsigned char* iopKeyArray,             /* Array for the key [in, out] */
    unsigned short iMaxKeyArraySize, /* Maximum length of the array for the key [in] */
    unsigned short* opSizeKey)              /* Length of the key [out] */
```

用户的 DLL 只要实现了上述任意一种函数接口，即可直接加载到 TP 层模块中。如果出现加载失败的情况，主要检查如下情况：

1. 是否用 Release 模式发布，如果是 Debug 模式，常常出现以来调试 DLL 的情况。
2. 是否采用 x86 平台发布，目前 TSMaster 为支持 X86 的版本，以来 DLL 也必须为 x86 模式。

如何兼容其他函数接口

但是日常使用中，经常出现用户已经开发好了 dll，但是该 DLL 的接口不是上述三种中的任何一种，这就无法直接加载到 TSMaster 的诊断模块中了。对于这种情况，推荐采用如下方案来解决此问题。



下面以一个实际的实例来讲解如何兼容用户现有的 DLL 文件。

1. 用户现有的 DLL，名称为 UserSeedKey.dll。该函数内部的 API 函数有：

- Seed 等级为 1 的时候，调用函数 void GetKeyFromSeed01(byte* ASeed, byte* AKey);
- Seed 等级为 3 的时候，调用函数 void GetKeyFromSeed03(byte* ASeed, byte* AKey);
- Seed 等级为 11 的时候，调用函数 void GetKeyFromSeed11(byte* ASeed, byte* AKey);

该 dll 不支持上述默认加载接口，无法直接加载到 TSMaster 中使用的。因此，需要把这些 DLL 再包装一层，才能载入到 TSMaster 的诊断模块中。

2. 选择 TSMaster 安装目录中提供的 GenerateKeyEx 的模板工程，在该工程中调用上述 DLL 的函数接口。基本思路是：

- 采用 Loadlibrary 动态用户现有的 dll。
- 根据传入的 Level 参数，采用 GetProcAddress 函数动态获取实际的用于计算 Key 的函数指针。
- 如果获取函数指针成功，则使用该函数指针传输 Seed 值，并计算对应的 Key 值。

详细调用示例函数如下图所示：

```
KEYGEN_API GenerateKeyEx(
    const unsigned char* ipSeedArray,          /* Array for the seed [in] */
    unsigned int iSeedArraySize,               /* Length of the array for the seed [in] */
    const unsigned int iSecurityLevel,         /* Security level [in] */
    const char* ipVariant,                     /* Name of the active variant [in] */
    unsigned char* iopKeyArray,                /* Array for the key [in, out] */
    unsigned int iMaxKeyArraySize,             /* Maximum length of the array for the key [in] */
    unsigned int& oActualKeyArraySize)         /* Length of the key [out] */
{
    HMODULE hDll = LoadLibrary(TEXT("UserSeedKey.dll"));   //Existing SeedKey dll Name
    if (hDll == NULL)
        return 1; //ErrorCode = 1; Security not existing;

    TSeedKeyFunction fSeedKey = NULL;

    //begin calculate key from seed-----------------------------------------------------------
    switch (iSecurityLevel)
    {
    case 0x01://for security access with Services 0x27 01 ->0x27 02
        fSeedKey = (TSeedKeyFunction)GetProcAddress(hDll, "GetKeyFromSeed01");
        break;
    case 0x03://for security access with Services 0x27 03 -> 0x27 04
        fSeedKey = (TSeedKeyFunction)GetProcAddress(hDll, "GetKeyFromSeed03");
        break;
    case 0x05://for security access with Services 0x27 05 -> 0x27 06
        fSeedKey = (TSeedKeyFunction)GetProcAddress(hDll, "GetKeyFromSeed05");
        break;
    case 0x0B://for security access with Services 0x27 09 ->0x27 0A
        fSeedKey = (TSeedKeyFunction)GetProcAddress(hDll, "GetKeyFromSeed11");
        break;
    default:break;
    }
    if(fSeedKey == NULL)
        return 2; //ErrorCode = 2; Function not existing
    fSeedKey((byte*)ipSeedArray, (byte*)iopKeyArray);
    //setting length of key
    oActualKeyArraySize = 4;
    return KGRE_Ok;
}
```

3. 该 GenerateKeyEx 工程开发结束后，TSMaster 直接加载 GenerateKeyEx 所在的 dll。需要注意的时候，用户需要把现有的如 UserSeedKey.dll 拷贝到 TSMaster 根目录或者 GenerateKeyEx.dll 所在的目录。如果不拷贝过去，GenerateKeyEx.dll 执行的时候会出现找不到对应依赖 dll 的情况，解锁失败。

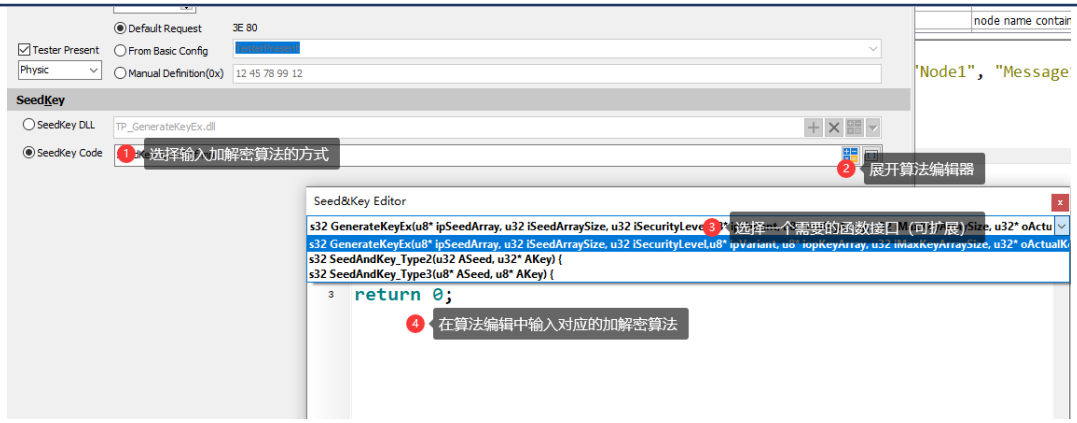
总结:

在 TSMaster 安装目录中，提供了封装 Seed&Key 算法的模板工程。如 GenerateKeyEx，GenerateKeyExOpt，ASAP1A_CCP_ComputeKeyFromSeed，用户基于此模板工程开发即可得到能够直接加载的 dll 函数。

同时，也提供了二次封装的 dll 的工程，比如 GenerateKeyEx_Wrapper_Demo，该工程演示了如何基于已经存在的 SeedKey 算法库进行包装，生成可以直接加载到 TSMaster 诊断模块中的 dll 的过程。

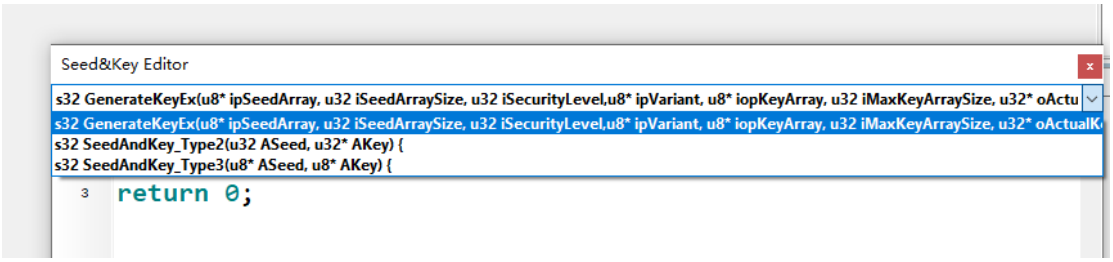
1.19.1.3.2. 采用内置的算法编辑器

基本步骤如下所示：

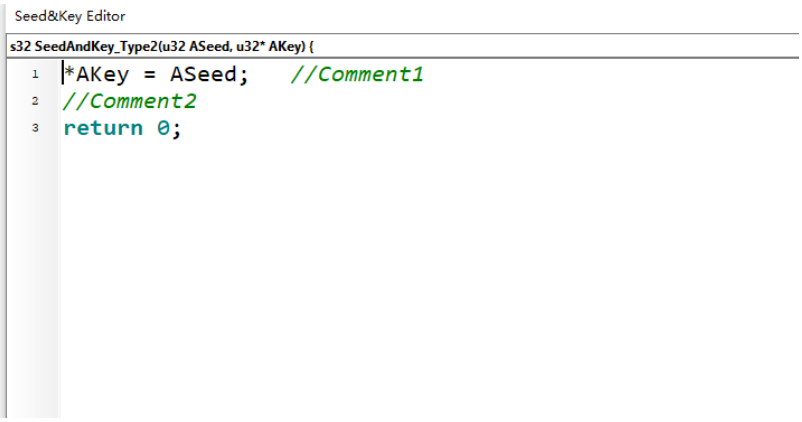


注意事项：

【1】 算法函数的接口，TSMaster 目前提供了最常用的接口形式，如果用户有自己特殊的接口形式，无法覆盖住，请联系上海同星工具把此接口增加到选项中。



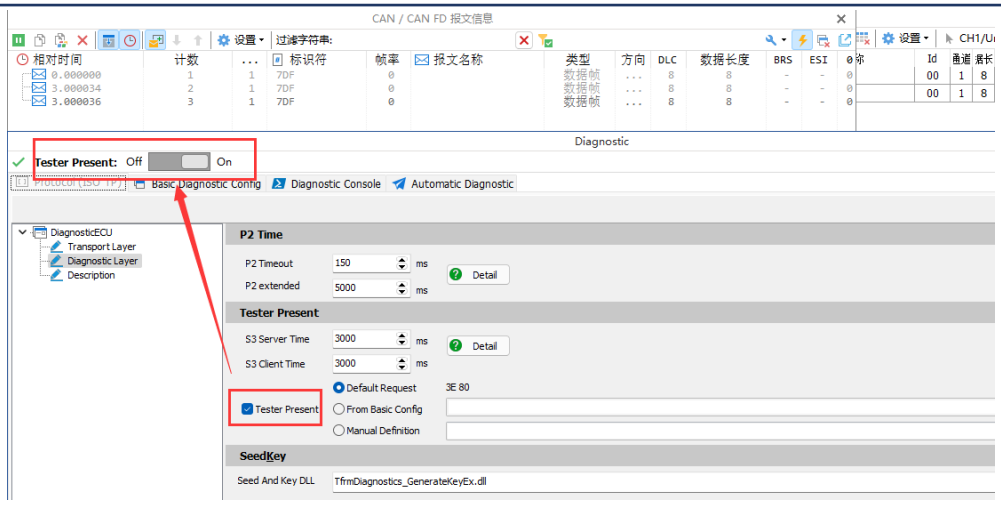
【2】 所有的接口函数都定义了返回值 s32。增加此约束，主要是增加函数的严谨性。返回值为 0 表示成功，为其他值则有对应的错误码。用户在编辑代码的时候，最后一行一定不要忘了输入返回值，否则系统执行函数过后，会认为算法执行失败，不予往后面执行。如下所示：



【3】 添加算法过后，点击 OK 退出。TSMaster 内置编译器会自动解释该算法，并准备好在执行诊断的过程中使用。

1.19.1.4. TesterPresent

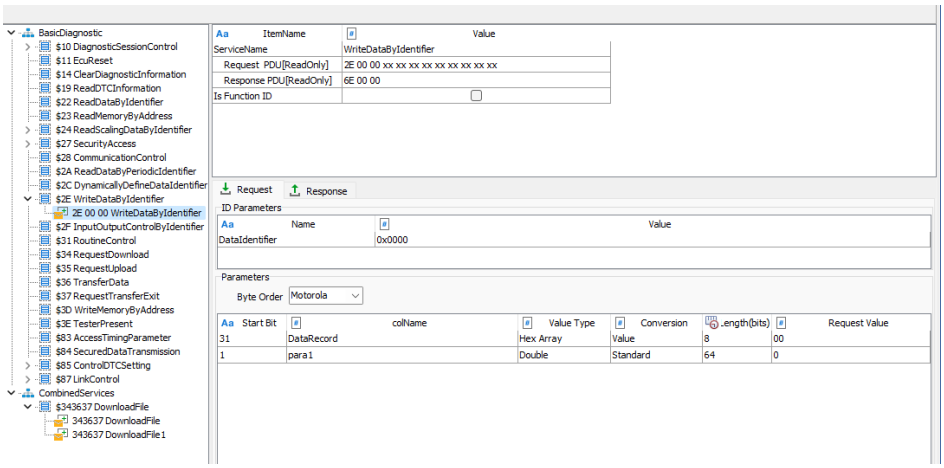
在 Diagnostic Tp 参数配置中使能 TesterPresenter，TSMaster 会提供一个全局的开关。用户通过该开关，可以直接打开和关闭 TesterPresent 命令，如下图所示：



除了该全局开关，如果用户想更加灵活的控制 TesterPresent 命令的打开和关闭，在后续的自动化流程步骤中，TSMaster 也提供了基于步骤配置该命令的方式，让用户选择在需要的步骤打开和关闭 TesterPresent 命令。

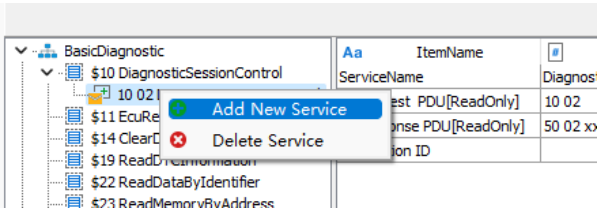
1.19.2. 基础诊断配置

该模块包含 BasicDiagnostic 参数和 CombinedService。对于执行过程完全独立的命令，则放入 BasicDiagnostic 中；对于必须多个命令组合才能够完成的命令，则放入 CombinedService 中。如下图所示：



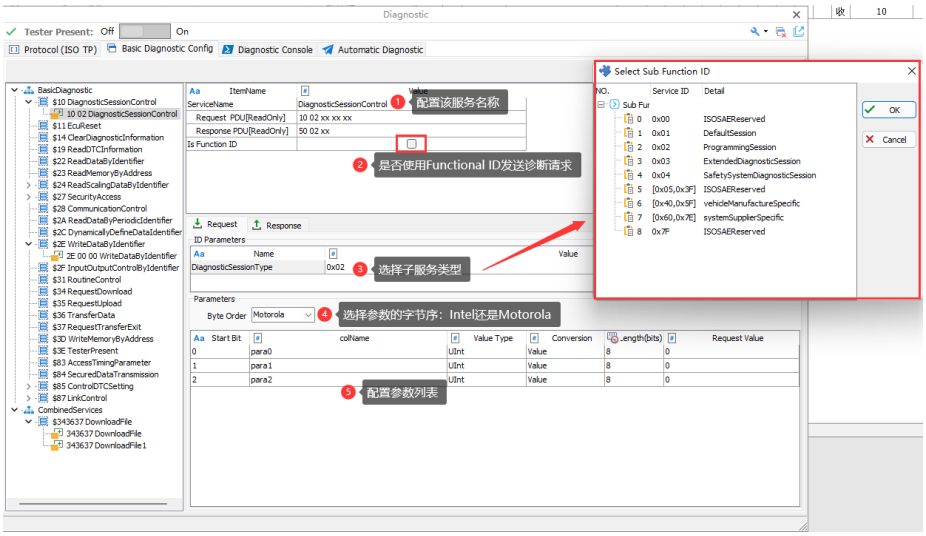
1.19.2.1. 添加删除服务命令：

把鼠标放到需要添加和删除的服务命令上方，右键展开，选择是否需要添加和删除该服务，如下图所示：



1.19.2.2. 配置 BasicDiagnostic 服务参数：

以 Session Control 为例，主要包含如下参数的配置：



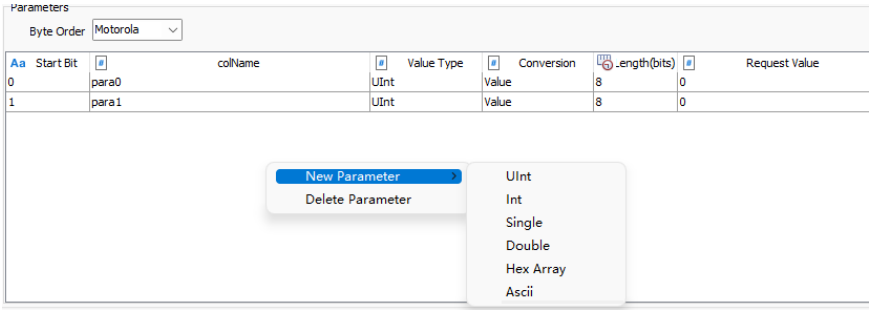
【1】 配置服务名称：用户可以配置一个易于理解和管理的服务名称。

【2】 Is Function ID：本诊断服务是否采用 Functional ID 发送诊断请求。

【3】 选择子服务类型：比如 Session Control 中的 DiagnosticSessionType 就包含如上图所示的 Session 类型。

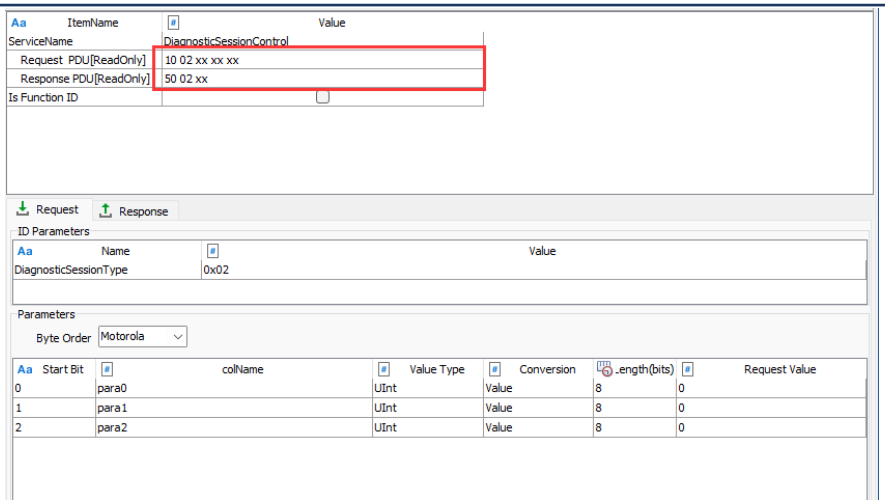
【4】 参数列表的字节序：支持 Motorola 和 Intel 字节序。

【5】 参数列表：诊断服务除了诊断 ID 和子服务类型 ID，还可以带着参数发送给被测 ECU。参数列表包含请求和应答帧的参数列表，其配置方法如下所示，用户可以选择增加/删除多种类型的参数。



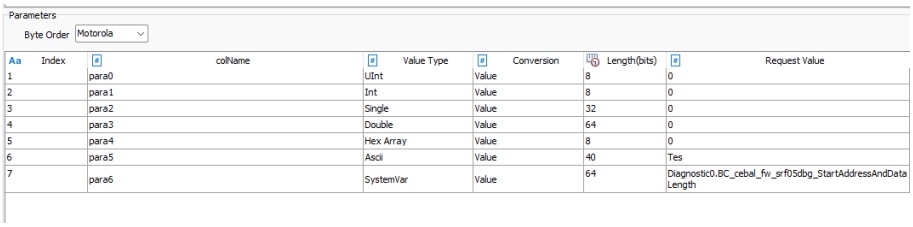
其中，服务 ID 和子服务类型 ID，如 SessionControl 里面的 DiagnosticSessionType 参数是必须的，而参数列表是可选的。

在修改配置后，界面上方会实时显示实际诊断报文的示例报文，如下图所示，完成如下所示的配置过后，诊断仪将要发出的服务报文是：【10 02 xx xx xx】：xx 表示该参数是可变的，根据用户实际填入的数据确定；诊断仪将要收到的肯定响应报文是【50 02 xx】。



1.19.2.2.1. 诊断服务参数

诊断模块参数支持 7 种数据类型。包括：UInt，Int，Single，Double，HexArray，Ascii 和 SystemVar。



【1】 UInt：无符号整型，其数据长度必须小于 32bits，并且为 8 的倍数，可以为 8,16,24,32bits。

【2】 Int：有符号整形，其数据长度必须小于 32bits，并且为 8 的倍数，可以为 8,16,24,32bits

【3】 Single：单精度浮点数，数据长度为固定的 32bits。用户直接输入输出浮点数据。

【4】 Double：单精度浮点数，数据长度为固定的 64bits。用户直接输入输出浮点数据。

【5】 Hex Array：十六进制数组，数据长度为 8 的倍数。输入数据满足 16 禁止数据类型。

【6】 ASCII：ASCII 字符串，数据长度为 8 的倍数。输入数据为 ASCII 字符数组，转化为 16 进制后进行发送。

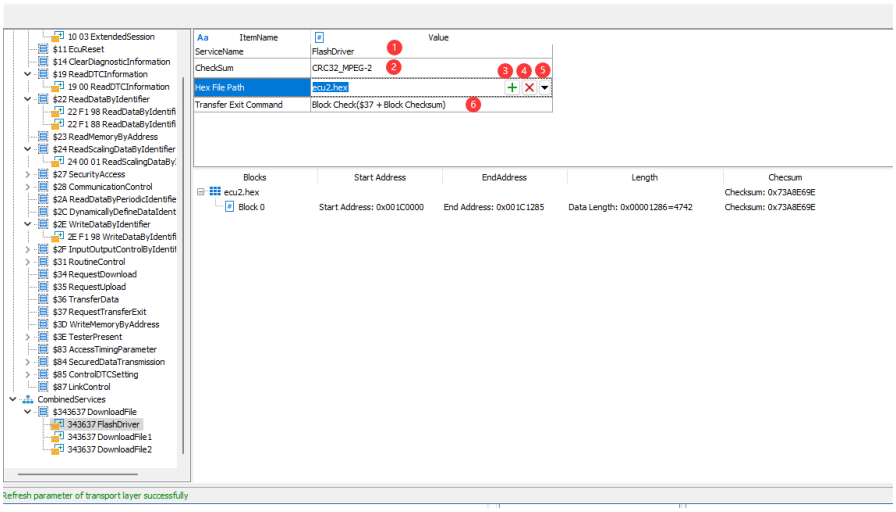
【7】 SystemVar：系统变量，数据长度为 8 的倍数。TSMaster 系统变量可支持 Uint，Int，Single，Double，UintArray，DoubleArray，HexArray，String 等各种数据类型。其具体的数据类型由系统变量自身定义所决定。

关于上述 7 种类型输入数据和实际发送字节数据转换过后的效果，请查看后续章节：输入标定参数章节。

1.19.2.3. 配置 CombinedServices 报文

1.19.2.3.1. Download File：

组合服务目前只支持了下载文件服务，如果用户有其他组合需求，可以反馈给上海同星，合理的需求可以作为标准服务模块添加到软件中。



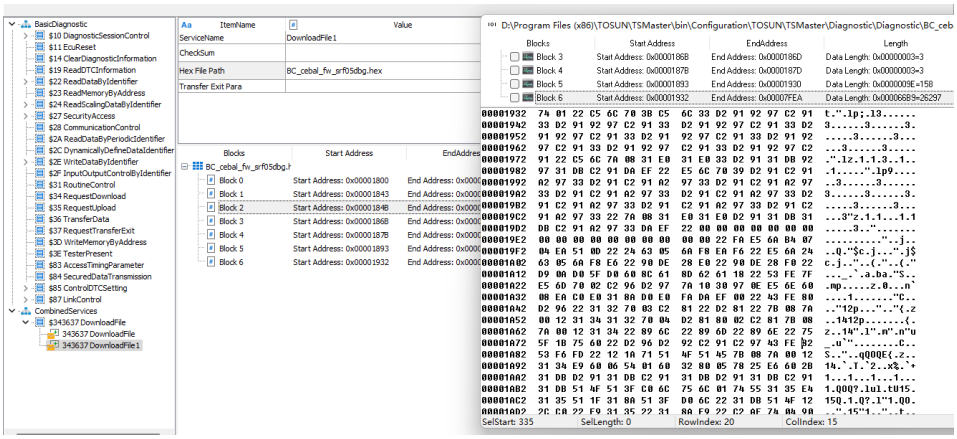
【1】 配置该服务的名称

【2】 选择文件进行 CRC 的校验算法，关于 CRC 校验，后面会详细介绍。

【3】 加载可执行文件。TSMaster 支持 Hex，S19，Mot，bin 文件的加载。加载过后，该文件包含的段落，地址，长度等信息见界面下方。

【4】 删除该可执行文件

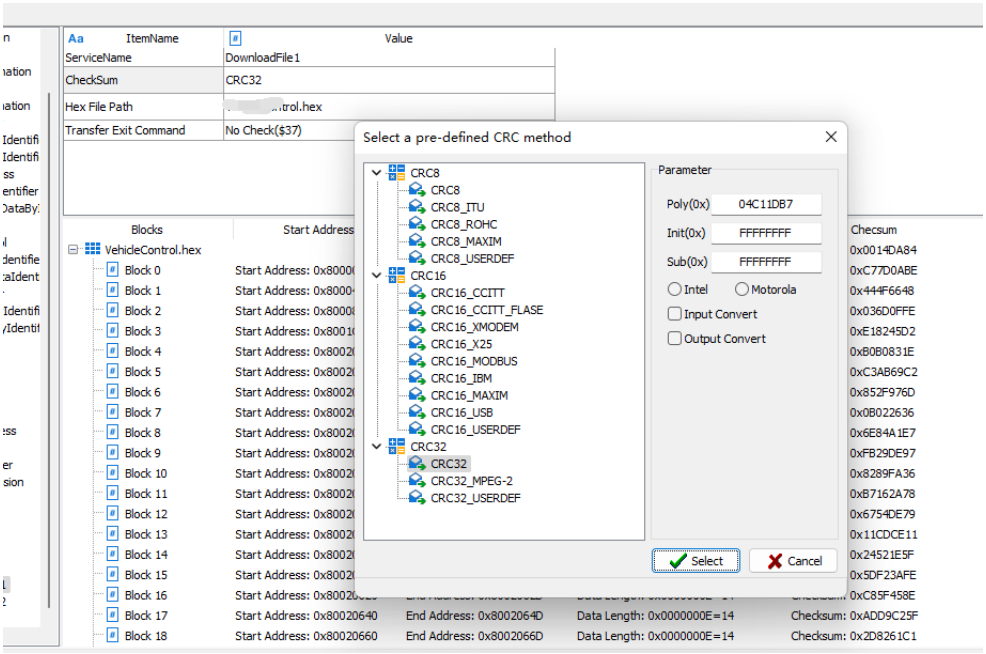
【5】 打开 hex viewer。TSMaster 内置了可执行文件查看编辑器 TSHexViewer，如下图所示，用户可以用该工具，查看载入 Hex 文件的详细信息。



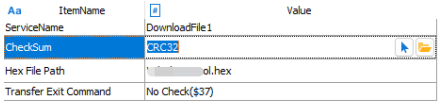
【6】 选择 TransferExit(0x37)命令的参数类型。

1.19.2.3.2. Checksum

在程序下载过程中，为了保证数据的完整性，需要引入 Checksum 算法对数据的完整性和有效性进行校验。TSMaster 诊断模块的符合服务中，引入了主流的 CRC 算法进行校验。其选择框如下图所示：



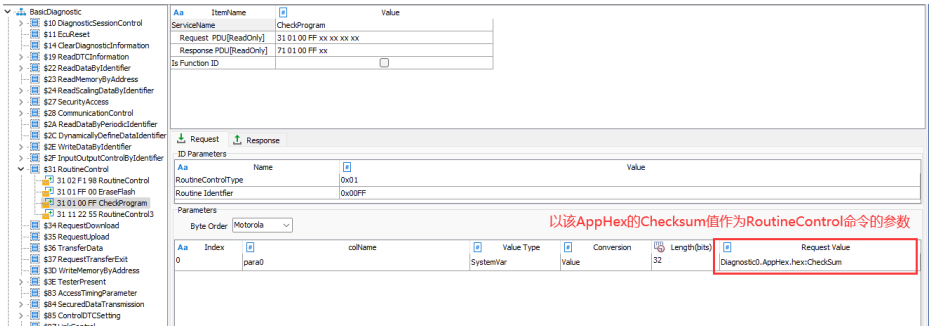
用户选择指定的算法过后，诊断模块会对可执行文件计算其 Checksum 值，包括该可执行文件每一个 Block 的 Checksum 值以及该文件整体的 Checksum 值，如下图所示：



在计算好每一个 Block 和程序整体的 Checksum 值过后，会进一步将这些值注册到系统变量中，如下图所示：

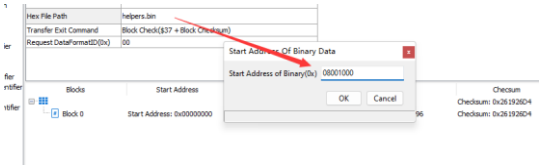
Internal Variable	Type	Value	Owner	Comment
Diagnostic0.AppHex.hex:CheckSum[Block]13	UInt32	0x11CDCE11	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum[Block]14	UInt32	0x24521E5F	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum[Block]15	UInt32	0x5DF23AFE	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum[Block]16	UInt32	0xC85F458E	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum[Block]17	UInt32	0xADD9C25F	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum[Block]18	UInt32	0x2D8261C1	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum[Block]19	UInt32	0xA4F5452B	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum[Block]20	UInt32	0x6C1352A2	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum[Block]21	UInt32	0x0E7C2468	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum[Block]22	UInt32	0x222DD2D5	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum[Block]23	UInt32	0x71DF9D61	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum[Block]24	UInt32	0x9CD27CFC	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum[Block]25	UInt32	0xCECA5FFB	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum[Block]26	UInt32	0x3FE11D16	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum[Block]27	UInt32	0x3C2EC20E	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum[Block]28	UInt32	0x5901BCEC	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum[Block]29	UInt32	0xFA55E9F2	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum[Block]30	UInt32	0x8E1901D7	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum[Block]31	UInt32	0x9906CABF	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum[Block]32	UInt32	0xC85300BB	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum[Block]33	UInt32	0xB8580A64	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum[Block]34	UInt32	0x181872B3	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum[Block]35	UInt32	0x2144DF1C	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum[Block]36	UInt32	0xF288B395	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum[Block]37	UInt32	0xF86ACFDA	Diagnostic	AppHex.hex:CheckSum[32 bits]
Diagnostic0.AppHex.hex:CheckSum	UInt32	0x0014DA84	Diagnostic	AppHex.hex:CheckSum[32 bits]

TSMaster 的诊断模块能够直接把系统变量作为参数。以诊断命令中，常用的校验可执行文件的有效性为例，此时就可以配置如下的 RoutineControl 命令，就可以实现对文件有效性的检查，如下所示：

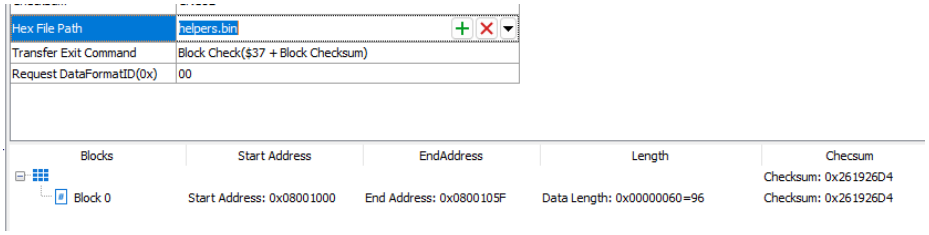


1.19.2.3.3. 载入 Binary 数据文件

Hex，S19 文件格式内部是带数据起始地址和长度的，但是对于 Binary 类型的二进制文件来说，他内部是不带数据的起始地址的，因此，再载入 Binary 数据文件的时候，用户需要手动输入数据起始地址段的长度，如下所示：



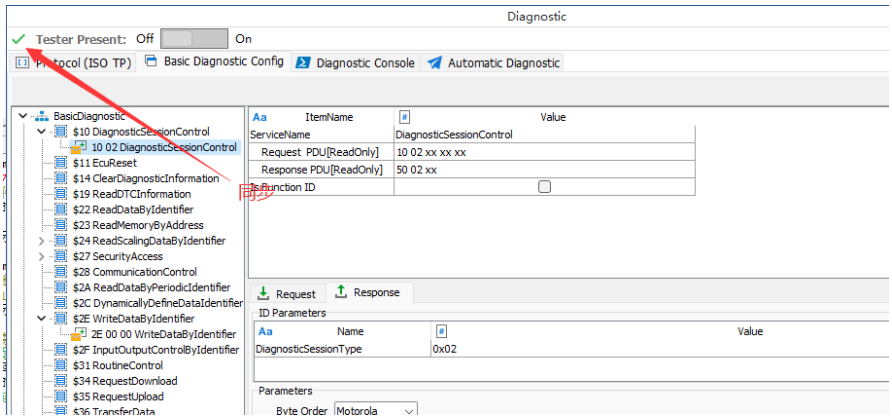
加载 Binary 的时候，软件会自动弹出起始地址设置的窗体如上图所示。地址格式为 16 进制，地址范围为[0x00000000,0xFFFFFFFF]，用户再此范围内设置，并点击确认即可。如果选择取消，则使用默认的地址 0x00000000。设置地址并载入后，如下图所示：



可见，该 Binary 文件现在有数据段的起始地址了。

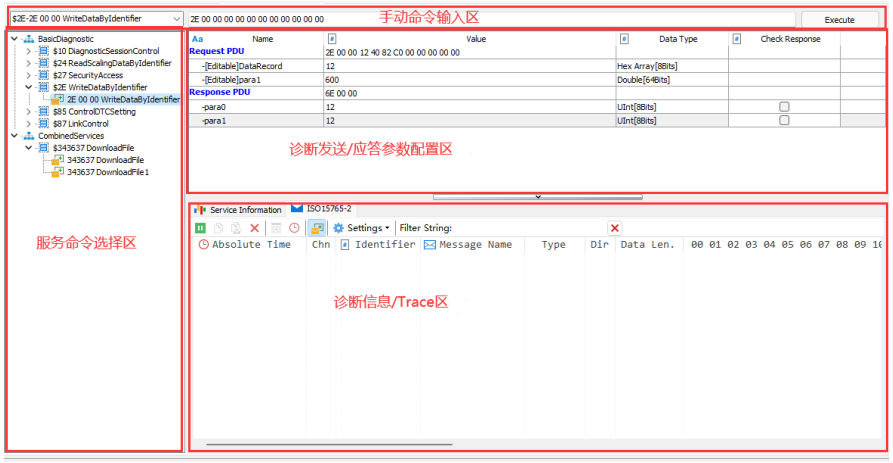
1.19.2.4. 同步到控制台

在完成上述配置操作后，点击界面左上方的同步按键，把上述配置同步到 Diagnostic Console 控制台模块中，为接下来具体的诊断操作做准备。如下所示：



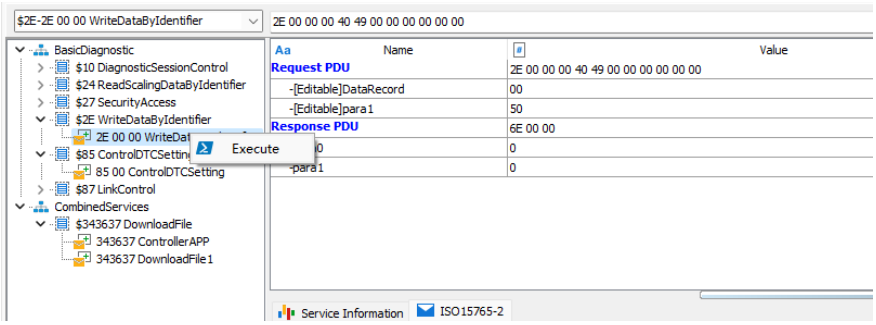
1.19.3. 诊断控制台

诊断控制台作为诊断命令调试器，可以让用户选择每一条单独的服务命令，编辑发送服务报文和接收服务报文，进行测试验证。主要包含四块工作区域，如下图所示：



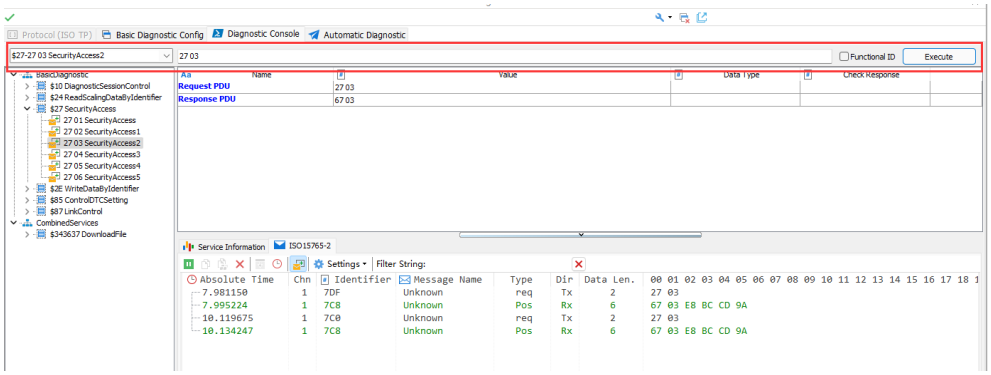
1.19.3.1. 服务命令选择区

服务命令选择区中是根据基础配置（后续 Odx/Cdd）等加载生成的可执行服务列表。用户可以双击执行选中的服务或者右键选择执行该服务，如下图所示：



1.19.3.2. 手动命令输入区

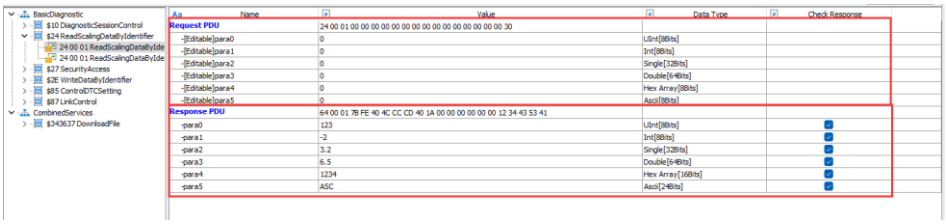
测试过程中，如果用户想发送任意的诊断命令，则可以在手动命令输入区中输入自己想要发送的任意报文，如下图所示：



在输入诊断报文过后，点击右边的 Execute 按钮，就可以完成诊断报文的发送。为了增加测试灵活性，用户可以通过选择框选择采用物理地址发送还是功能 ID 发送诊断请求报文。

1.19.3.3. 诊断命令发送/应答区

在本区域中，用户可以编辑发送数据段以及期望接收数据段，启动执行来验证被测 ECU 的诊断响应是否符合实际要求。下面以 24 服务为例，设计了 6 种不同数据类型的发送参数，也同时设计了 6 种不同数据类型的应答参数，如下图所示：



1.19.3.3.1. 输入诊断参数

输入标定参数示例如下：

Request PDU	24 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 31 74 73 65 54 00 00 18 00 00 0	
-[Editable]para0	0	UInt[8Bits]
-[Editable]para1	0	Int[8Bits]
-[Editable]para2	0	Single[32Bits]
-[Editable]para3	0	Double[64Bits]
-[Editable]para4	0	Hex Array[8Bits]
-[Editable]para5	Test1	Ascii[40Bits]
-[Editable]para6	Diagnostic0.BC_cebal_fw_srf05dbg_StartAddressAndDataLength	SystemVar[64Bits]

Request PDU：诊断模块要发送的诊断数据包字节，该部分数值是不可编辑的，用户在填入参数值过后，该部分数据自动生成对应的诊断数据值。

诊断参数：对应关系如下：

【1】 Para0，数据类型为 UInt，数据长度为 8Bits，输入 12，则对应字节为 0x0C。

【2】 Para1，数据类型为 Int，数据长度为 8Bits,输入为-1，则对应字节为 0xFF。

【3】 Para2，数据类型为 Single，数据长度为 32Bits，输入为 3.1，则对应字节为 0x40， 0x46，0x66，0x66。

【4】 Para3，数据类型为 Double，数据长度为 64Bits，输入为 3.2，则对应字节为 0x40， 0x09，0x99，0x99，0x99，0x99，0x99，0x9A。

【5】 Para4，数据类型为 Hex 数组，数据长度为 8Bits，输入为 0x11，则对应字节为 0x11。

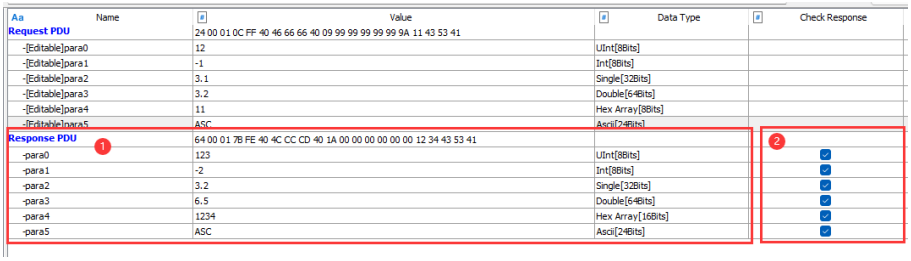
【6】 Para5，数据类型为 ASCII 字符串，数据长度为 24Bits，输入字符串为"ASC"，则对应字节为 0x43，0x53，0x41。

【7】 Para6，数据类型为系统变量。数据长度根据提取的系统变量的值为 64bits，系统变量名称为 Diagnostic0.BC_cebal_fw_srf05dbg_StartAddressAndDataLength，在执行过程中，系统会根据该名称自动提取系统变量的实际值，并解析到发送报文中。

完成上述诊断参数的输入过后，生成的诊断请求数据包为：0x24 0x00 0x01 0x0C 0xFF 0x40 0x46 0x66 0x66 0x40 0x09 0x99 0x99 0x99 0x99 0x99 0x9A 0x11 0x43 0x53 0x41，正如上图所示。

1.19.3.3.2. 输入应答参数

输入应答参数值如下图所示：



其中，第 1 部分跟前一个章节输入诊断参数完全一样，这里不再讲解。但是应答参数增加了一个可选命令，是否检查（Check）这部分参数。如果勾选了 Check，则 ECU 的应答必须等于配置的应答参数，本诊断测试才算通过。如果不勾选，则诊断模块不检测 ECU 应答中这部分字节的内容。

【1】 当上述所有配置应答都勾选上的时候，ECU 应答的报文必须等于：

0x64 0x00 0x01 0x7B 0xFE 0x40 0x4C 0xCC 0xCD 0x40 0x1A 0x00 0x00 0x00 0x00 0x00 0x00

0x12 0x34 0x43 0x53 0x41 才被系统认定为通过诊断测试。

【2】 去掉勾选判断 Para1 和 Para2，如下图所示：

Response PDU	64 00 01 7B xx xx xx xx xx 40 1A 00 00 00 00 00 00 00 12 34 43 53 41		
-para0	123	UInt[8Bits]	☑
-para1	-2	Int[8Bits]	☐
-para2	3.2	Single[32Bits]	☐
-para3	6.5	Double[64Bits]	☑
-para4	1234	Hex Array[16Bits]	☑
-para5	ASC	Ascii[24Bits]	☑

此时，ECU 应答的报文必须等于：0x64 0x00 0x01 0x7B **0xXX 0xXX 0xXX 0xXX 0xXX** 0x40 0x1A 0x00 0x00 0x00 0x00 0x00 0x00 0x12 0x34 0x43 0x53 0x41。其中红色部分 0xXX 表示该部分字节不予判断，其他字节必须等于上述配置字节才被系统认定通过诊断测试。

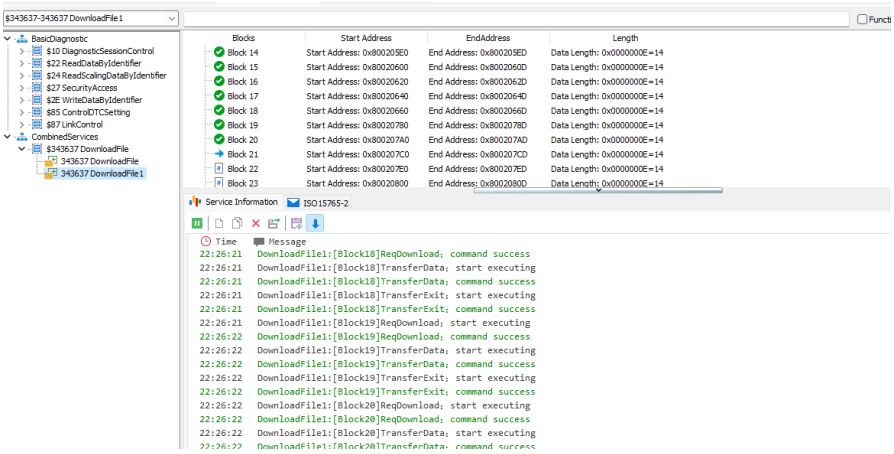
【3】 去掉勾选判断 Para0-Pra5，如下图所示：

Response PDU	64 00 01		
-para0	123	UInt[8Bits]	☐
-para1	-2	Int[8Bits]	☐
-para2	3.2	Single[32Bits]	☐
-para3	6.5	Double[64Bits]	☐
-para4	1234	Hex Array[16Bits]	☐
-para5	ASC	Ascii[24Bits]	☐

此时，ECU 应答的报文必须等于：0x64 0x00 0x01 才被系统认定通过诊断测试。

1.19.3.4. 诊断运行

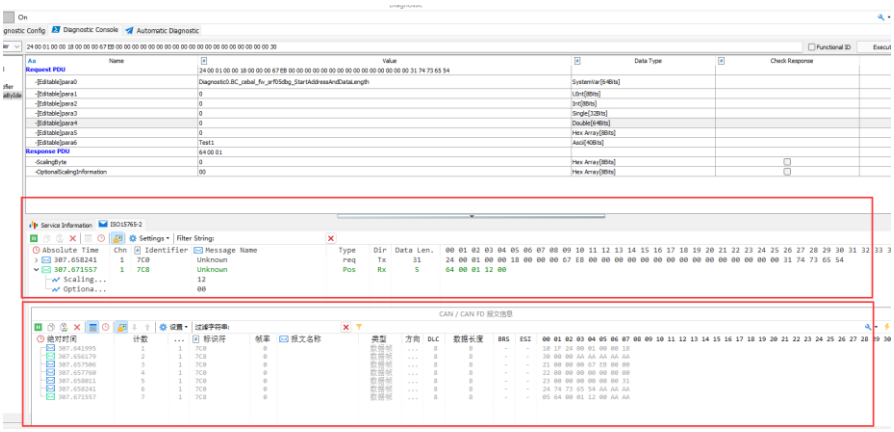
以 CombinedService 为例，诊断运行过程中，会显示当前下载完成的 Block 块区域，并显示每一个 Block 写入的执行时间等，如下图所示：



1.19.3.5. 诊断信息/Trace 区

1.19.3.5.1. 服务/原始报文 Trace 对比

在诊断中，用户会碰到最原始的 CAN/CANFD/LIN 报文，以及经过传输层传输过后的服务层报文。在 TSMaster 诊断模块中，原始 CAN 报文在基本的 Trace 模块中查看，而经过传输层处理过后的服务报文，则直接在诊断模块的 Trace 区域查看。如下图所示：

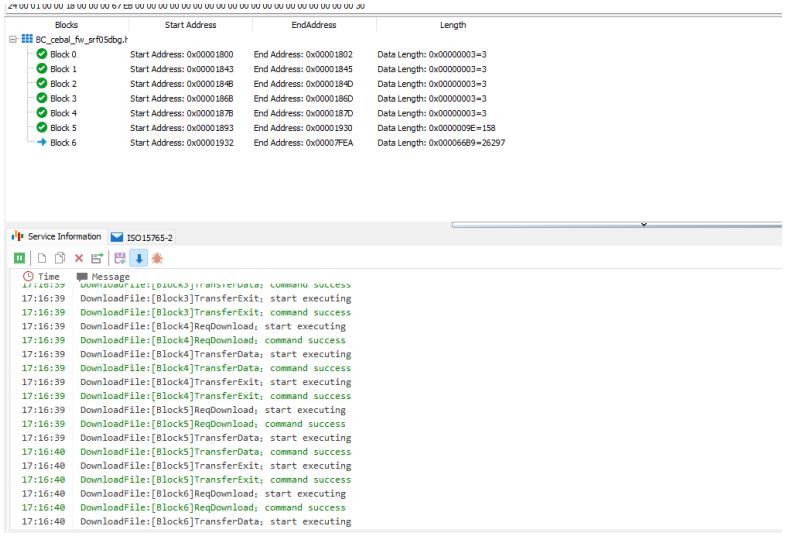


通过上图对比可以看到：

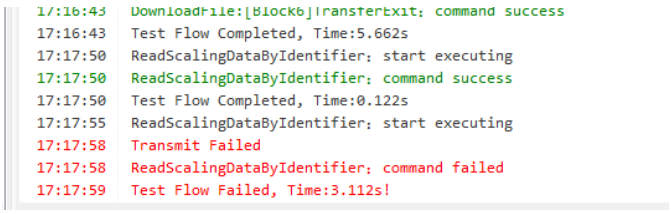
- 原始的 CAN/CANFD 报文区还可以看到多帧，单帧，首帧等传输层信息。
- 诊断模块中的 Trace 呈现给用户的是直接的服务层报文。对于用户来说，只需要关心自己发送的服务内容即可，不需要关心这些内容具体是怎么拆分发送的。因此，做诊断服务的时候，重点观察诊断模块内部 Trace 界面即可。

1.19.3.5.2. 操作提示区

该区域显示当前在诊断模块中的操作步骤。如下图所示，显示的是下载一个 hex 文件，程序内部的传输步骤流程。

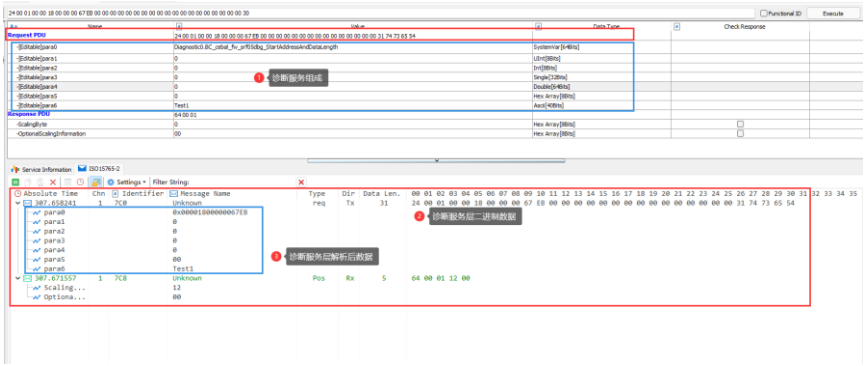


当诊断服务没有得到肯定响应或者无响应时，报错提示信息等如下所示：



1.19.3.5.3. ISO15765-2 服务报文区

本区域用于显示诊断模块详细的服务层报文信息。结合前面配置的诊断数据库，还可以把原始的报文数据解析成物理信号等呈现给用户，如下图所示：

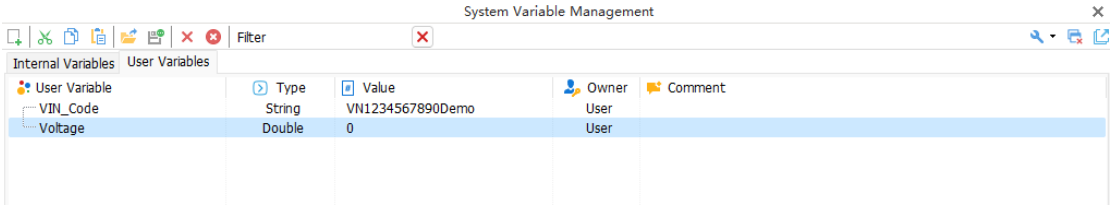


1.19.4. 系统变量的灵活应用

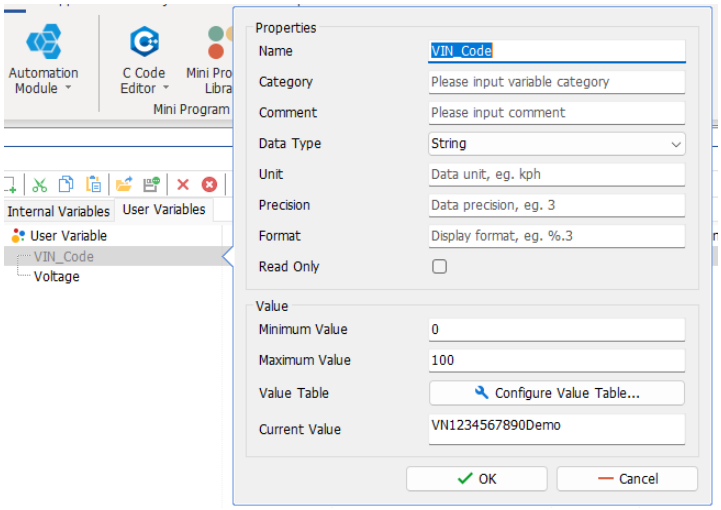
1.19.4.1. 系统变量作为参数

系统变量具有软件内部和外部模块之间数据交互的能力，TSMaster 把系统变量作为参数引入诊断模块中，极大的拓展了诊断模块跟其他模块数据交互的能力。下面以几个典型的应用场景为例说明其功能：

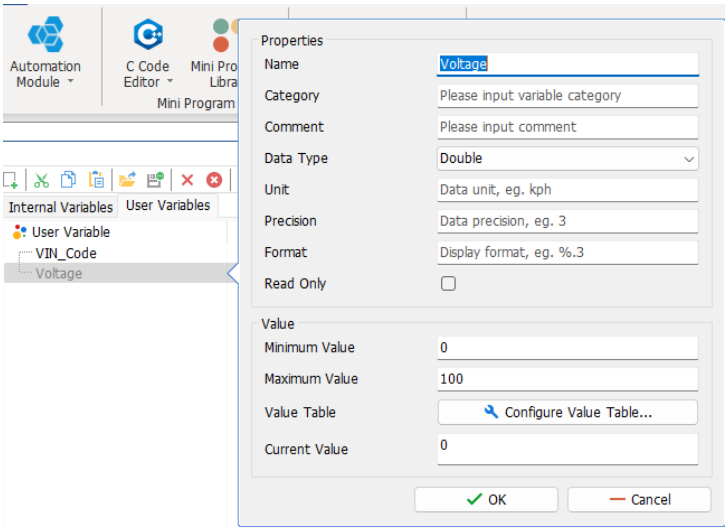
首先，在系统中创建两个系统变量，VIN_Code，Voltage，如下所示：



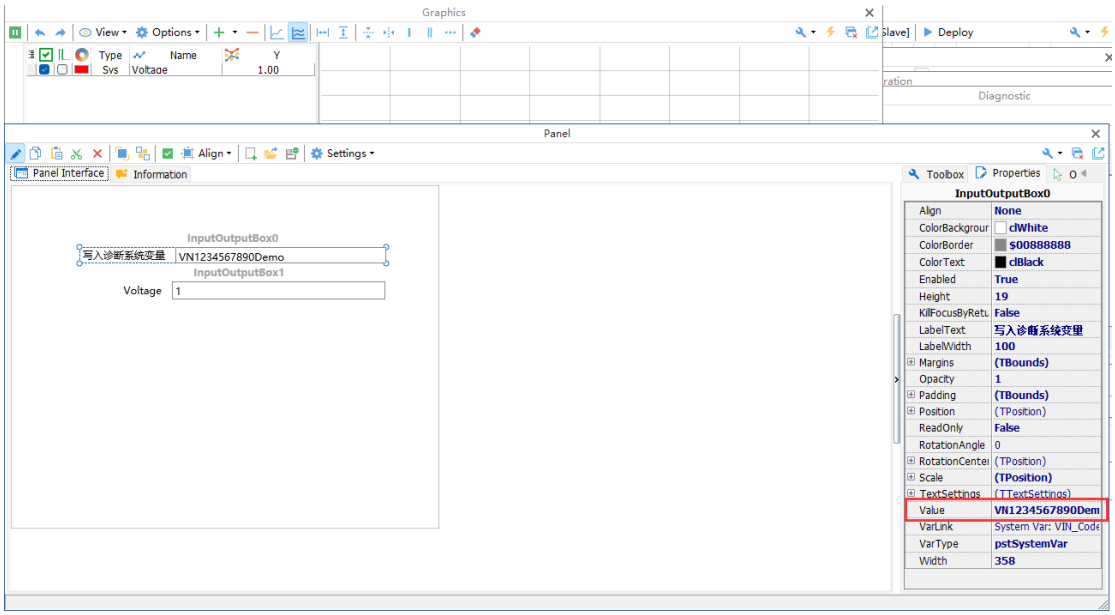
VIN_Code 变量为 string 类型：



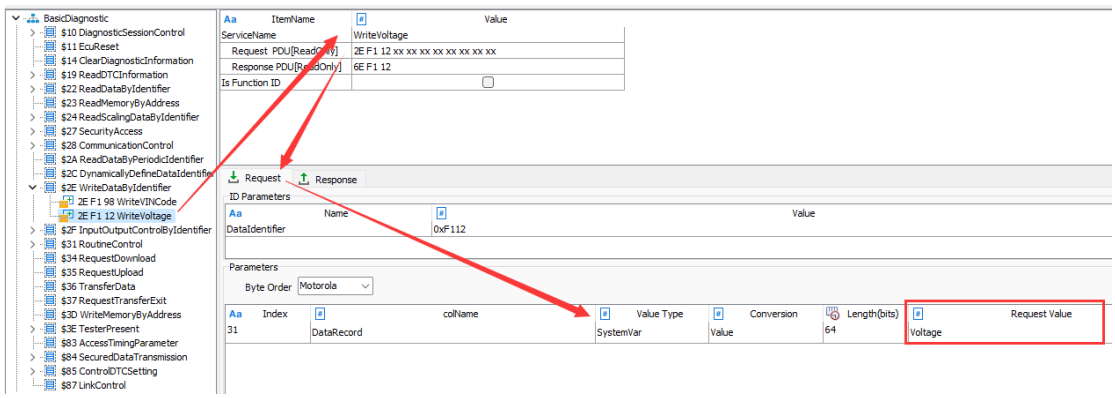
Voltage 变量为 Double 类型：



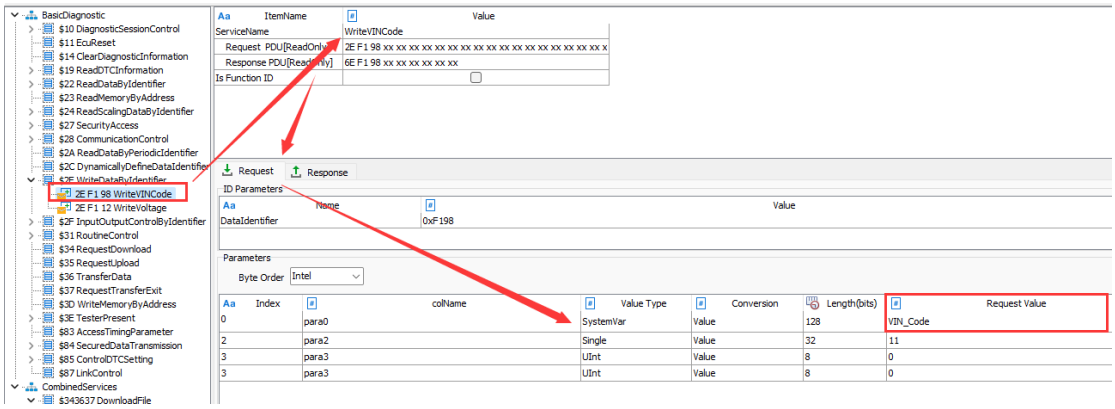
把系统变量关联到 Panel 和 Graphic 中，如下所示：



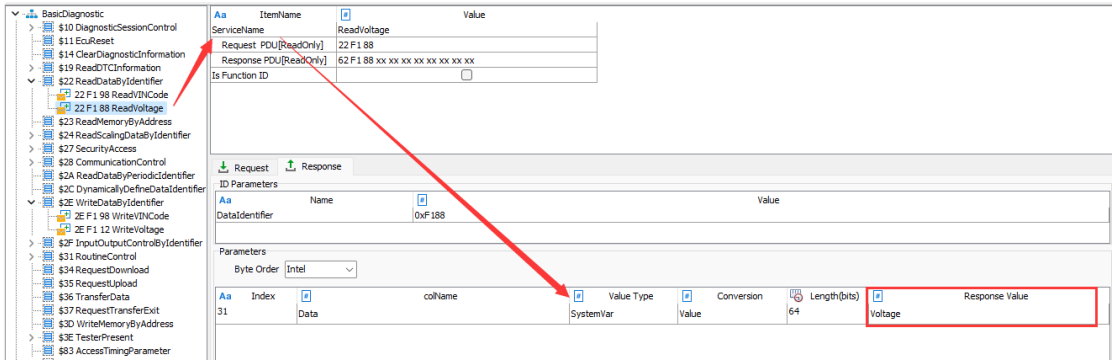
【1】 在 Panel 中设置电压值 Voltage，通过诊断写入到 ECU 中。



【2】 在 Panel 中设置 VIN 码，通过诊断写入到 ECU 中。

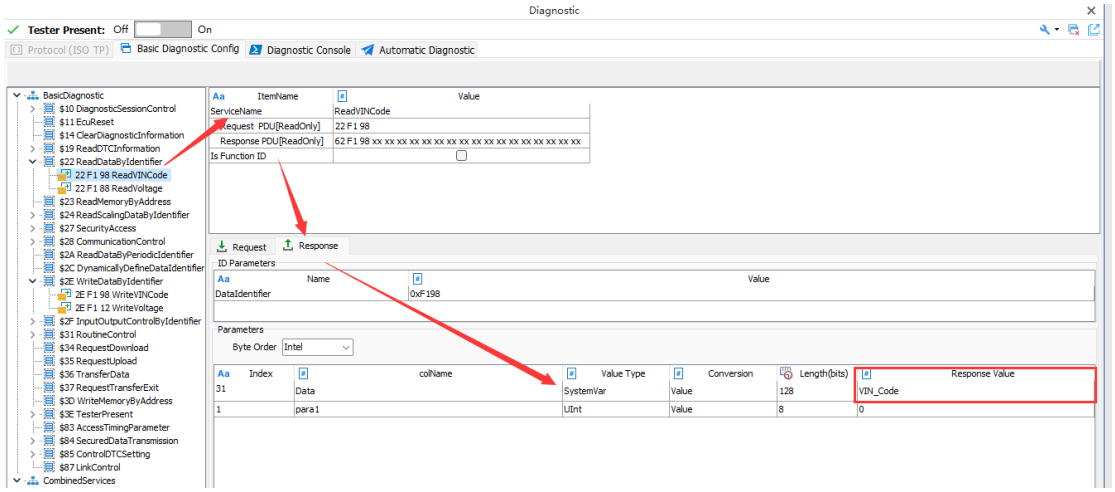


【3】 通过诊断读取 ECU 内部电压值，并显示到 Graphic 中。



注意，读取的变量，需要用户手动设置，才会同步到系统变量中。因为

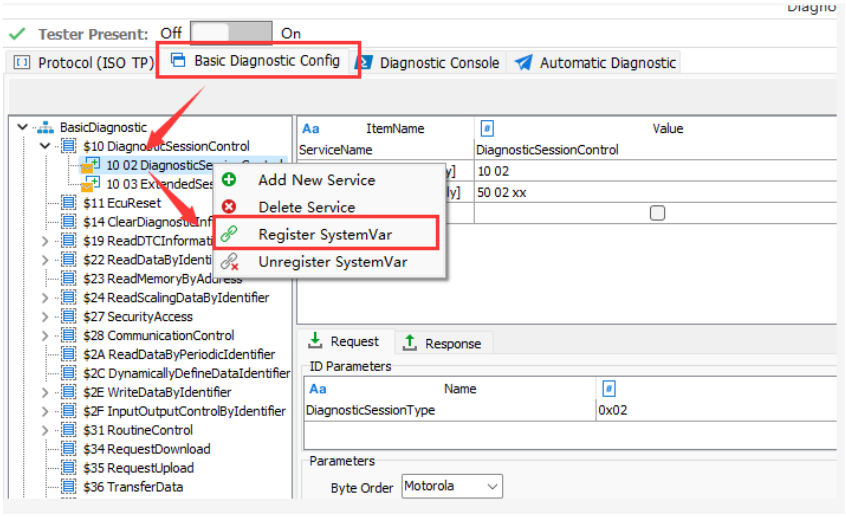
【4】 通过诊断读取 ECU 内部 VIN 码，并显示到 Panel 中。



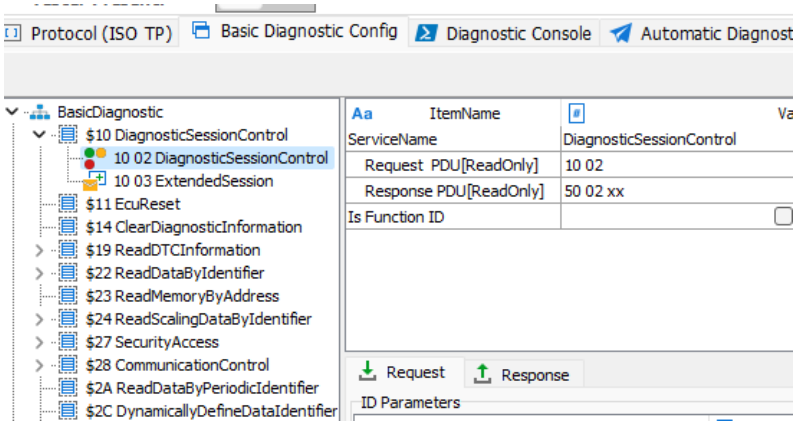
1.19.4.2. 系统变量关联 Console 服务

在前面章节中，用户在诊断控制台中可以根据需要灵活配置诊断服务。这些诊断服务配置好过后，用户需要在诊断控制台中双击启动该诊断服务。但是如果用户想在 Panel 界面中启动该诊断命令，则还需要借助系统变量。步骤如下：

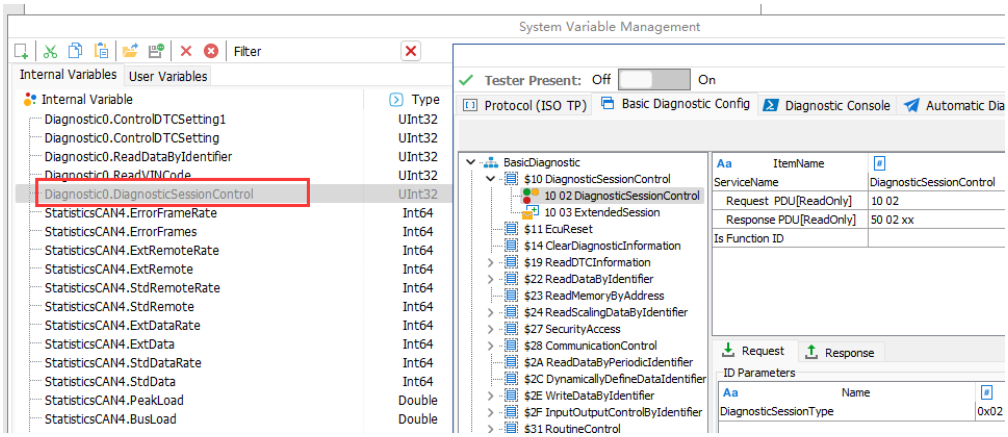
【1】 首先在诊断 BasicConfig 窗体中，选中目标服务，然后右键菜单中把该诊断服务注册为系统变量，如下所示：



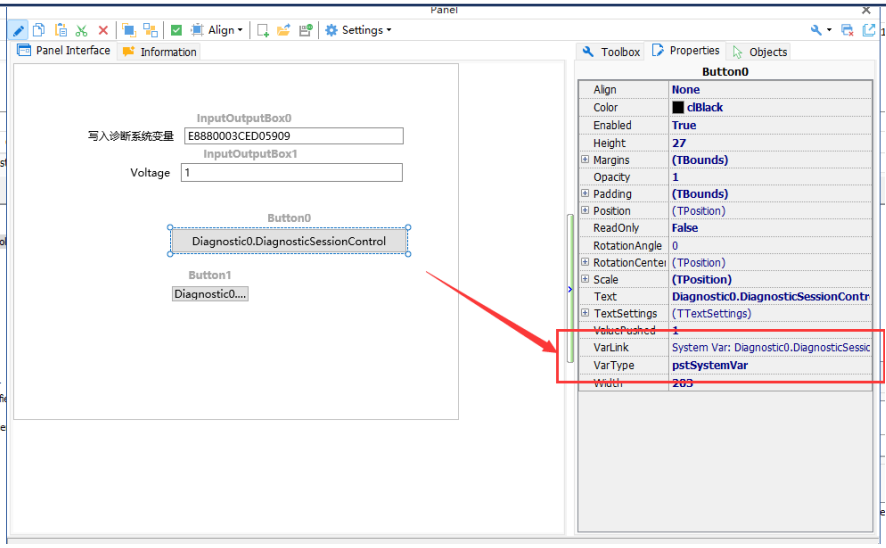
注册完成后，该服务项的图标变成如下图标，表示成为一个注册了系统变量的服务，如下所示：



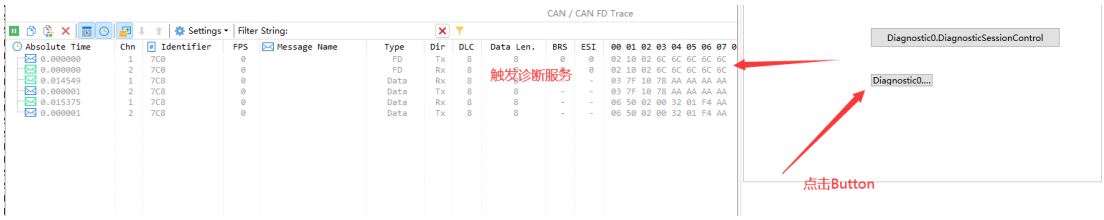
【2】 注册完成后，在系统变量管理器中，就可以看到该系统变量了，如下所示：



【3】 在 Panel 中添加 Button，并关联该系统变量，如下所示：



【4】 运行程序，点击 Panel 上的测试按钮，可以看到，诊断模块执行了 DiagnosticSessionControl 服务。如下所示：



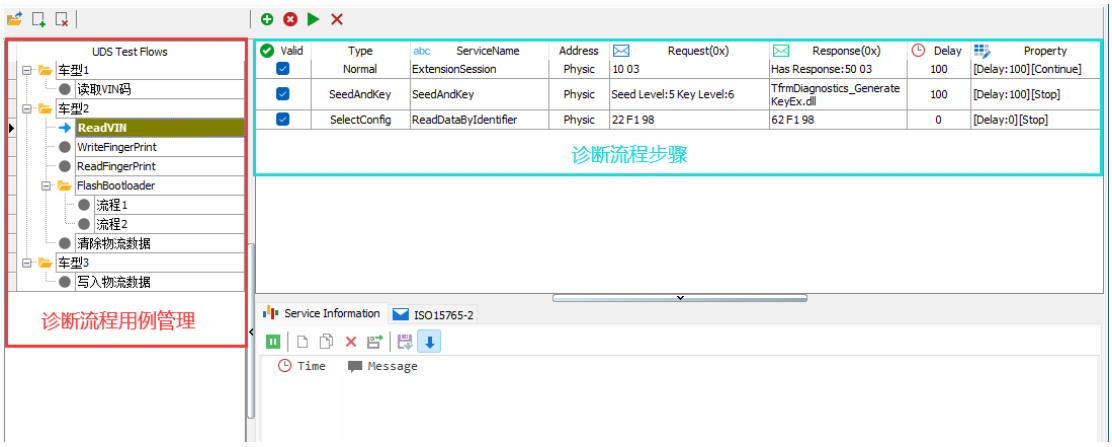
1.19.4.3. 外部程序控制诊断

外部程序可以通过系统变量跟 TSMaster 进行数据交换。

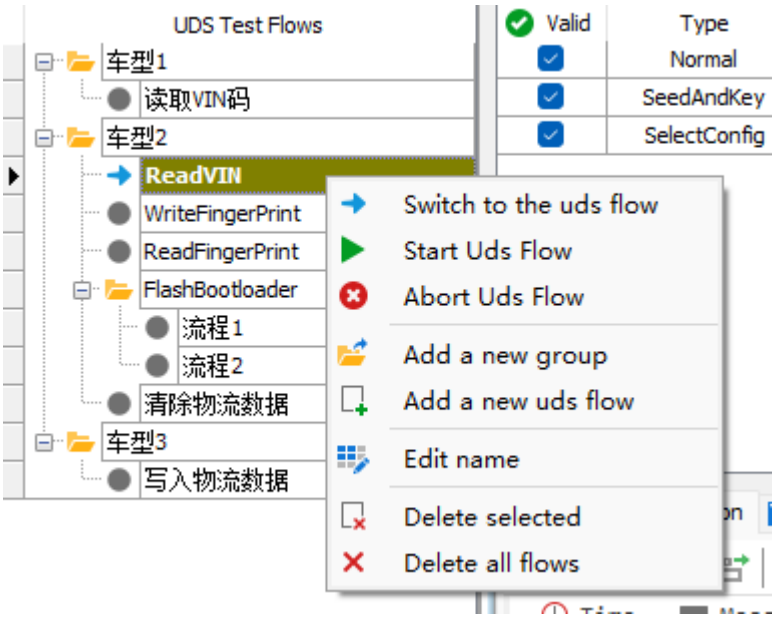
1.19.5. 自动诊断流程

1.19.5.1. 流程用例管理

TSMaster 的自动化诊断流程不仅仅是针对某一个具体的应用，而是针对整个项目的诊断流程进行管理。用户可以根据完整项目的需求，配置测试诊断流程组，每个组里面可以包含多个不同的诊断流程，在一个诊断流程中才包含具体的诊断步骤。如下图所示：



在 UDS 流程管理栏右键鼠标，展开流程用例管理的操作菜单，如下图所示：

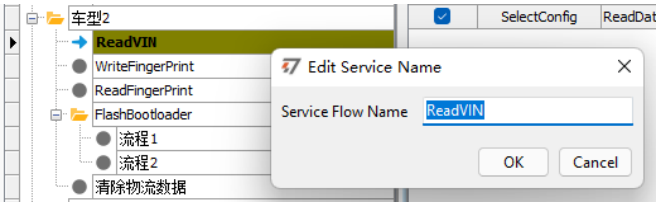


主要包含如下操作：

【1】 Add a new group：新增诊断流程组。比如新增车型 1。诊断组下面可以再增加诊断流程用例，其本身不包含诊断步骤。

【2】 Add a new uds flow：新增一个诊断流程用例，在诊断流程用例下面可以增加详细的诊断步骤。

【3】 Edit name：选中一个流程组或者流程用例，右键选中 Edit name 编辑该节点的名称，如下图所示：



【4】 Switch to the uds flow：切换到当前 UDS 流程节点。双击该节点，也可以达到切换到该流程节点的效果。切换到该节点过后，节点图标和背景色如下图所示，同时右边的节点流程中展开显示该 uds 流程包含的详细诊断步骤。

【5】 Start Uds flow：启动该节点的诊断流程。点击该选项后，诊断模块按照右边的配置，从上往下自动执行诊断步骤。

【6】 Abort Uds flow：点击该节点后，中断正在执行的诊断流程步骤。

【7】 Delete selected：删除选中的节点。

【8】 Delete all flows：清除所有的节点。

1.19.5.2. 配置诊断流程（UDS Flow）

1.19.5.2.1. 基本配置步骤

配置诊断流程，基本步骤如下图所示：

【1】 在左边的管理栏中选中一个诊断流程节点。

【2】 在右边的编辑区域，添加，删除，编辑诊断步骤。

【3】 添加步骤后，编辑步骤名称。

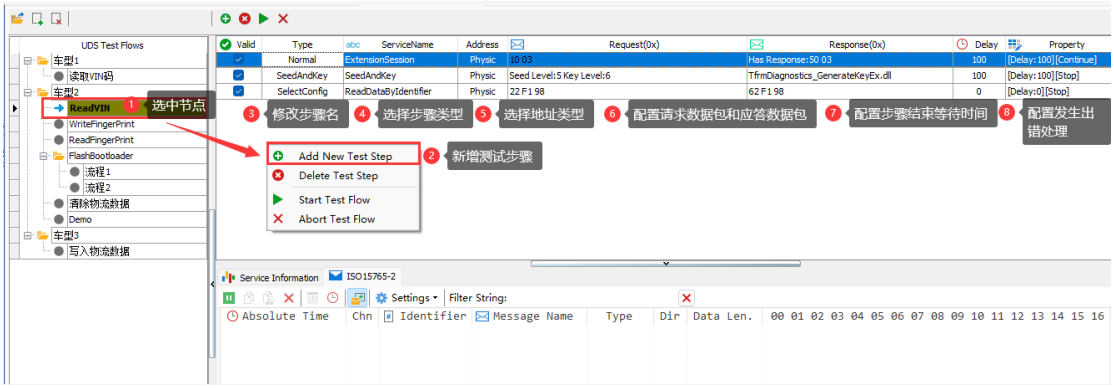
【4】 选择该步骤的类型。

【5】 选择该步骤地址类型，物理地址还是功能地址。

【6】 配置详细的诊断请求数据包和应答数据包。

【7】 配置本步骤结束后步骤之间的等待时间。

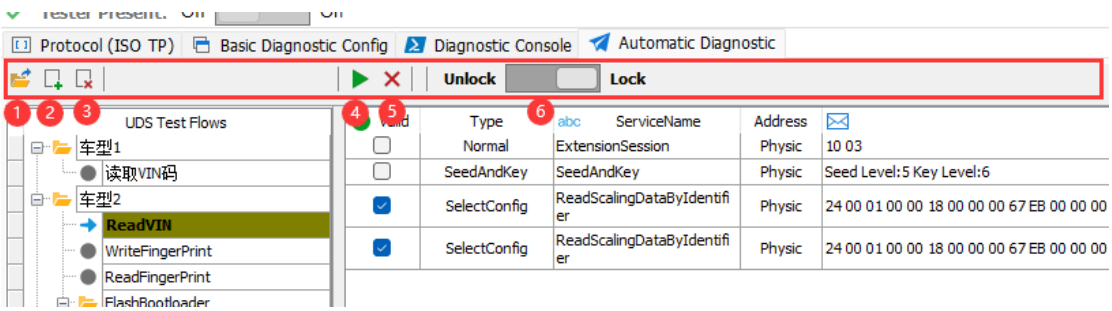
【8】 配置本步骤发生错误的错误处理方法。



上面几个步骤是配置诊断流程的基本步骤，实际使用的时候，根据应用场景还提供了更灵活的机制，接着看后续章节。

1.19.5.2.2. 工具栏

诊断流程配置工具栏如下图所示：



【1】 新增诊断流程组。

【2】 新增诊断流程用例。

【3】 删除选中的诊断流程组/用例。

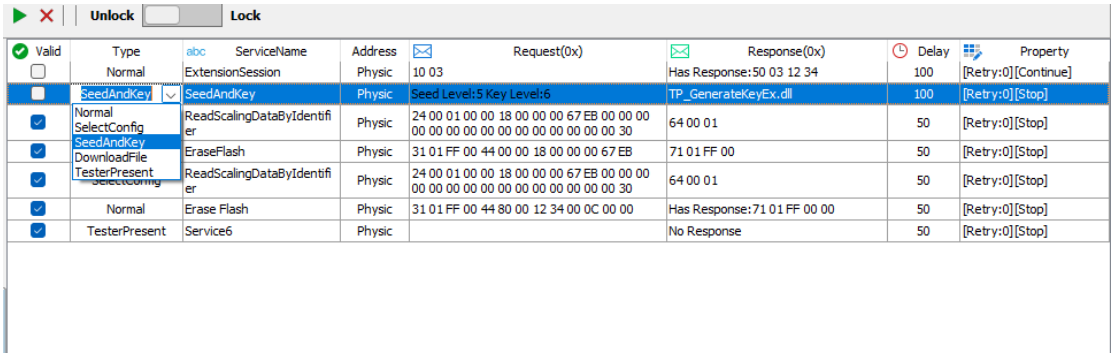
【4】 启动配置好的诊断流程。

【5】 中断正在运行的诊断流程。

【6】 锁定/解锁流程配置区域。如果锁定该区域，在诊断流程区域变得不可编辑。

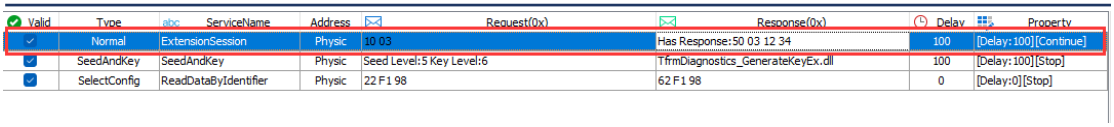
1.19.5.2.3. 诊断步骤类型

测试步骤中，为了增加诊断配置的灵活性，设计了 5 中类型可供选择，如下图所示，主要包含：Normal，SelectConfig，SeedAndKey，DownloadFile，TesterPresent，RoutineControl。通过这 5 种类型，基本上涵盖住了市面上所有主流的诊断流程需求，下面详细介绍每种类型的特点。

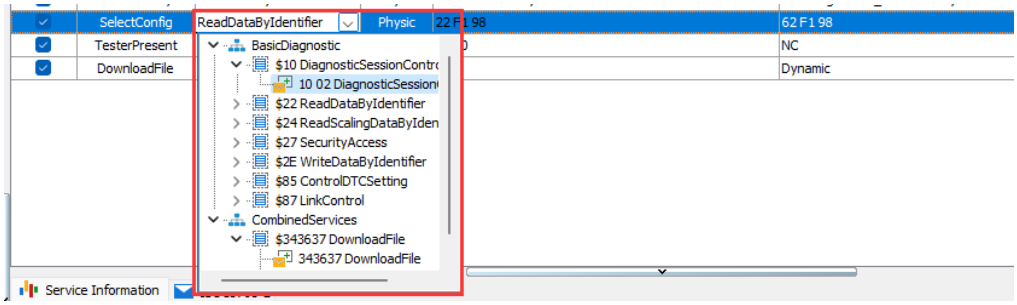


【1】 **Normal**：普通配置。该配置主要用于一些简单的，请求数据和应答数据都很清晰的场合。比如服务请求数据为 【10 03】，服务应答数据为 【50 03 12 34】，则可以选择 Normal 类型。Normal 的配置是最简单的，直接在 Request 中填入想要发送的请求数据【10 03】，在 Response 中填入期望的应答报文【50 03 12 34】即可。配置 Response 报文的时候，展开如下所示：

因为有些测试用例中，ECU 是没有应答的，对于这种情况，用户只需要去掉勾选 Has Response 即可。完成配置后效果图如下所示：



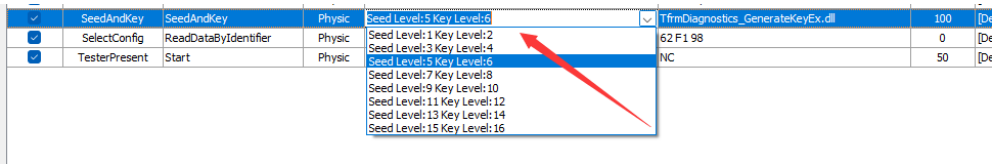
【2】 **SelectConfig**：选择已有配置，该配置设计的目的，就是让用户选择在 Diagnostic Console 控制台中已经调试好的诊断步骤。选择过程如下图所示：



这种方式是 TSmaster 最推荐的配置方式。用户可以现在 Diagnostic Console 中把各个子流程全部配置并测试好，然后在自动诊断流程中引用该配置即可，其逻辑如下图所示：

在自动诊断流程的执行过程中，其执行效果跟 Diagnostic Console 中将完全一样。

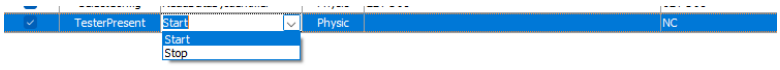
【3】 **SeedAndKey**：SeedAndKey 是一个组合命令，直接用 Normal 命令无法配置出来。用户可以通过 SelectConfig 直接从已有配置中选择，也可以通过选择 SeedAndKey 类型，在自动流程中直接配置解密步骤。SeedAndKey 只需要选择 SeedLevel 参数即可，解密的 DLL 直接关联到 TP 参数配置中载入的 SeedAndKey 的 Dll 中，如下图所示：



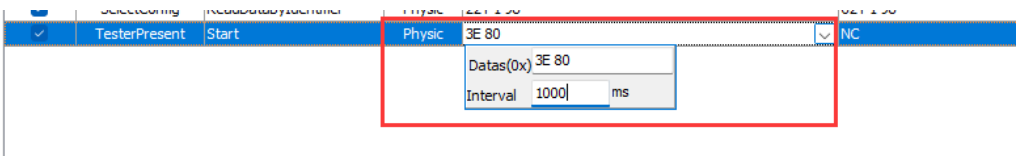
可见，无论是在 Diagnostic Console 模块中，还是 Automatic Diagnostic 模块中正确运行的前提需要用户正确完成 TP 层参数的配置。

【4】 **TesterPresent:** 如前文所讲，TSMaster 提供了一个 TesterPresent 的全局开关，通过该开关，用户可以直接打开和关闭该命令。同时为了支持更加灵活的测试需求，在自动化流程步骤中，也提供了基于步骤配置该命令的方式，让用户选择在需要的步骤打开和关闭 TesterPresent 命令。选择该类型过后，主要有两个参数需要配置：

- 是否启动/停止该命令，如下：

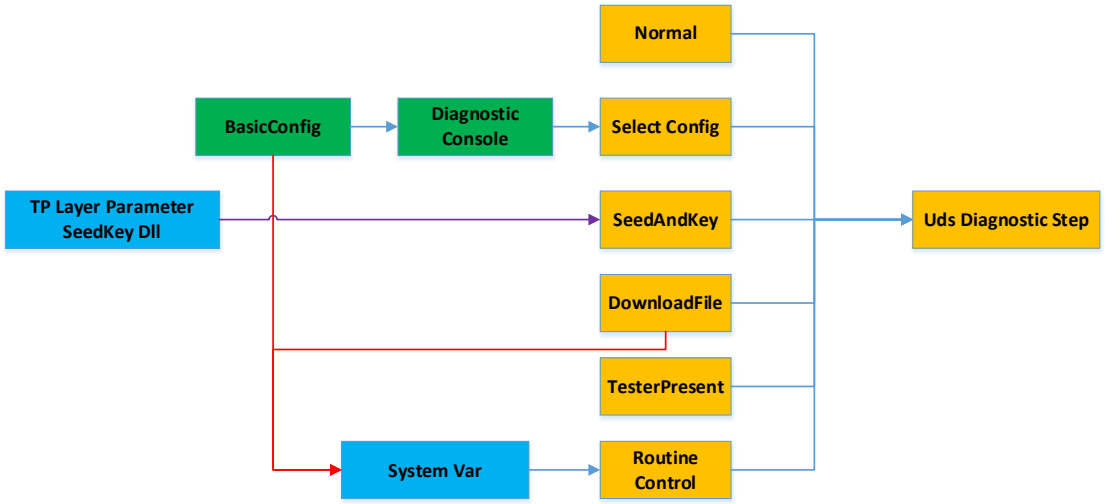


- 配置该命令数据，以及周期间隔，如下：



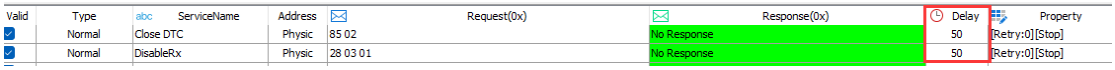
【诊断步骤配置汇总】

总结上个章节测试步骤的配置流程，其逻辑流组成如下图所示：



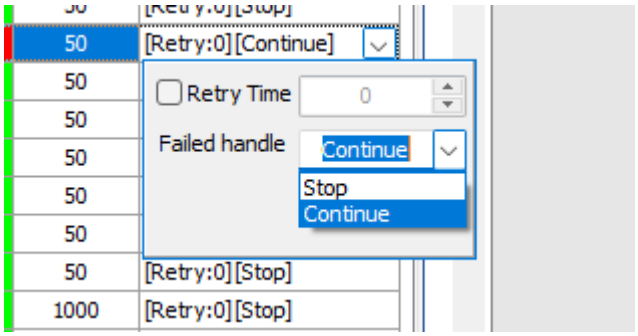
1.19.5.2.4. 步骤间隔时间

诊断流程模块步骤跟步骤之间的时间间隔是可以设置的，如下图所示，单位为 ms：



1.19.5.2.5. 出错处理

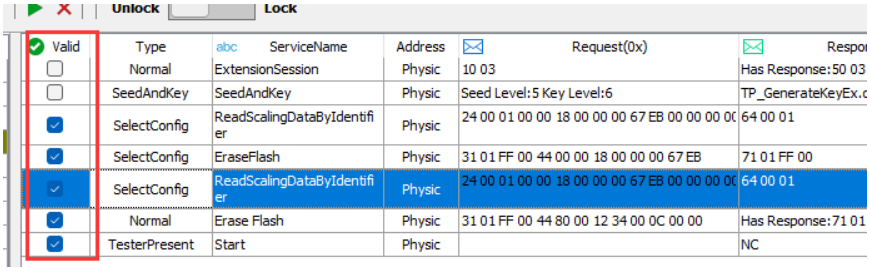
现阶段出错处理，主要包含两个参数：错误后重试次数以及错误后停止还是继续运行，如下图所示：



在后续产品规划中，出错后允许跳转到指定的流程中（比如跳转到擦除流程中），进一步增加自动运行流程模块的灵活性。

1.19.5.2.6. 使能步骤/位置调整

对于已经完成配置的诊断流程步骤，用户根据左边的选择框来勾选想要执行的诊断步骤，如下图所示：

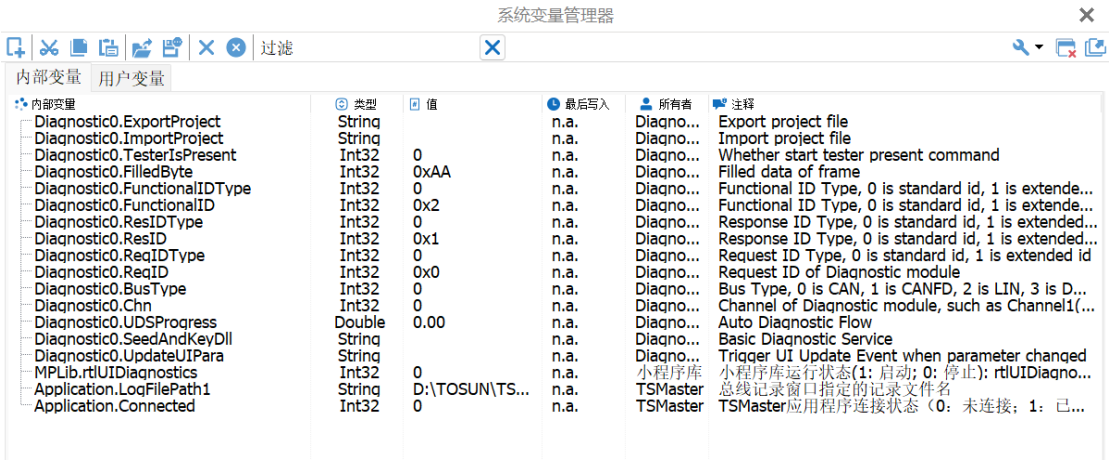


关于执行顺序调整：无论是测试用例组，测试用例还是测试用例中的具体步骤，用户想调整相互之间执行顺序的时候，直接拖拽对应的测试用例到相应位置即可。

1.19.5.3. 程控自动诊断流程

1.19.5.3.1. 诊断模块通用系统变量

在 TSMaster 新添加基础诊断模块后，系统变量管理器会自动生成该诊断模块的系统变量（如下图 1-1 所示）。通过修改系统变量可以配置对应的参数。



诊断模块对应的系统变量

创建一个诊断模块过后，其在系统变量管理器中会自动生成相应的系统变量，主要包含如下这些类型：

- 总线类型——BusType（Int32）：CAN = 0，CANFD = 2，LIN = 2，DOIP = 3；
- 通道——Chn（Int32）：CH1 = 0，CH2 = 1， CH3 = 2，……；
- 请求 ID——ReqID（Int32）：物理寻址 ID；
- 请求 ID 类型——ReqIDType（Int32）：标准帧 = 0， 扩展帧 = 1；
- 应答 ID——ResID（Int32）：诊断应答 ID；
- 应答 ID 类型——ResIDType（Int32）：标准帧 = 0， 扩展帧 = 1；
- 功能 ID——FunctionalID（Int32）：功能寻址 ID；
- 功能 ID 类型——FunctionalIDType（Int32）：标准帧 = 0， 扩展帧 = 1；
- 填充字节——FilledByte（Int32）：除有效数据外的填充字节数据
- 诊断仪在线——TesterPresent（Int32）：是否使能诊断仪在线

- 安全访问种子和密钥——SeedAndKeyDll（String）：SeedKey 算法 dll 的物理地址，使用时注意转义字符；
- 自动化流程进度——UDSProgress（Double）：自动诊断流程的实时进度，该变量用来获取自动诊断流程的运行状态。

1.19.5.3.2. 诊断服务特定系统变量

组合服务下载文件的载入和切换：

在基础诊断配置-复合诊断服务添加新的服务后，系统变量管理器同样会生成对应的系统变量（服务名_DataFile），此变量为下载文件的物理地址，修改此变量可控制下载文件的载入与切换。

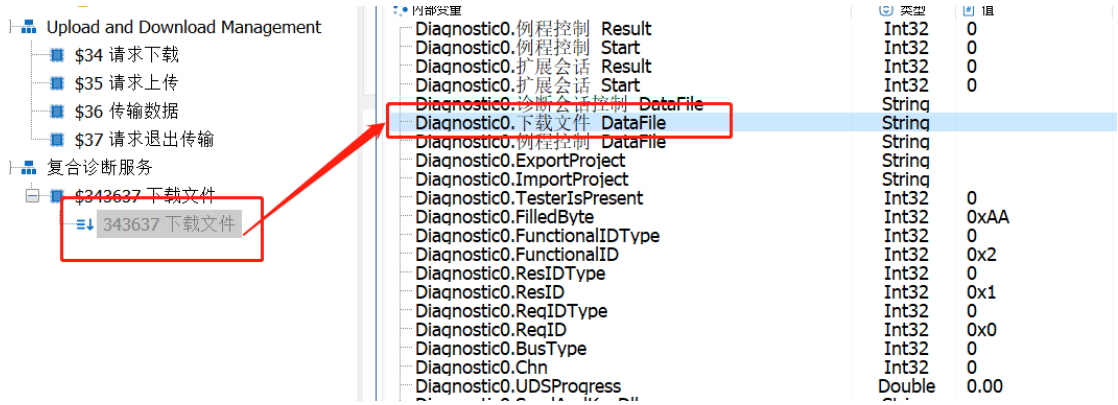


图 2-1 下载文件系统变量

Notes：载入下载文件后，系统变量控制器会根据选择的校验和算法生成每块校验和及总校验和、下载文件的首地址和长度，如果已经添加了复合诊断服务，载入了下载文件，并且在基本诊断服务中关联了下载文件相关变量，那么在替换下载文件的同时，这些关联的变量也会随之改变。

1.19.5.3.3. 基本诊断服务和自动诊断流程的自动化程控

已有的诊断服务和自动诊断流程可右击注册为系统变量，注册后系统变量管理器会生成对应服务或流程的程控变量（服务或流程名_Start、服务或流程名_Result）。*Start 变量用于服务或流程的启动执行（赋值为 1 即可），*Result 表示服务或流程的执行结果（0 为默认，1 为运行中，2 为成功，3 为失败）。

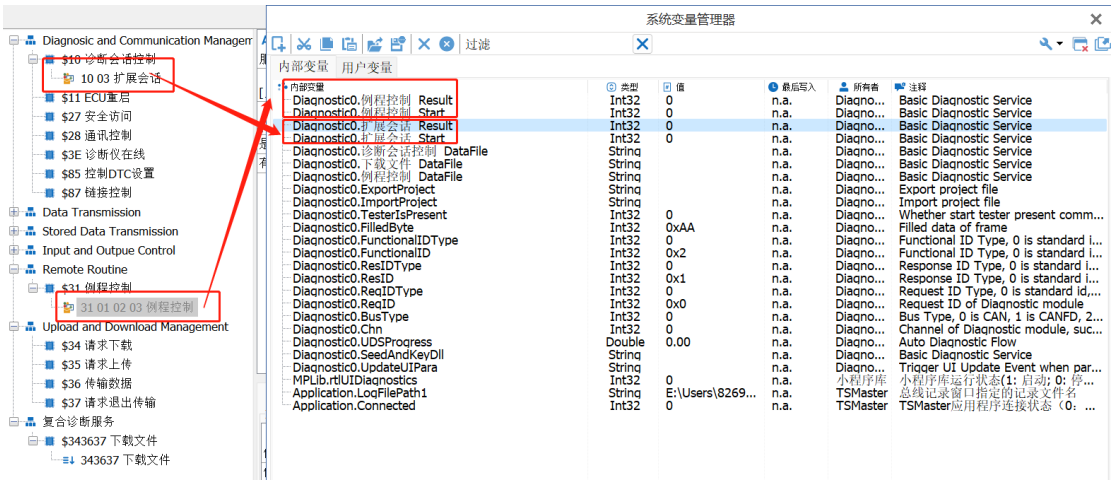


图 3-1　基本诊断服务系统变量

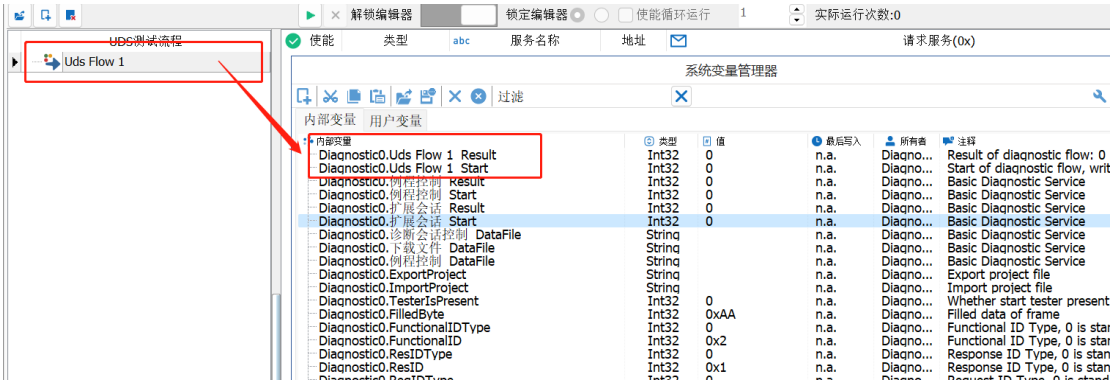
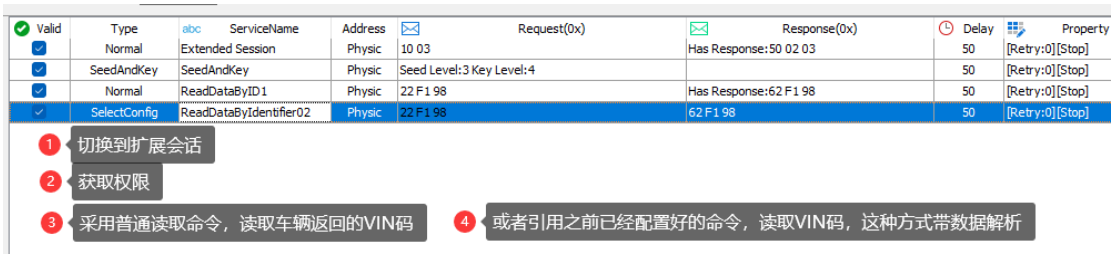


图 3-2　自动诊断流程系统变量

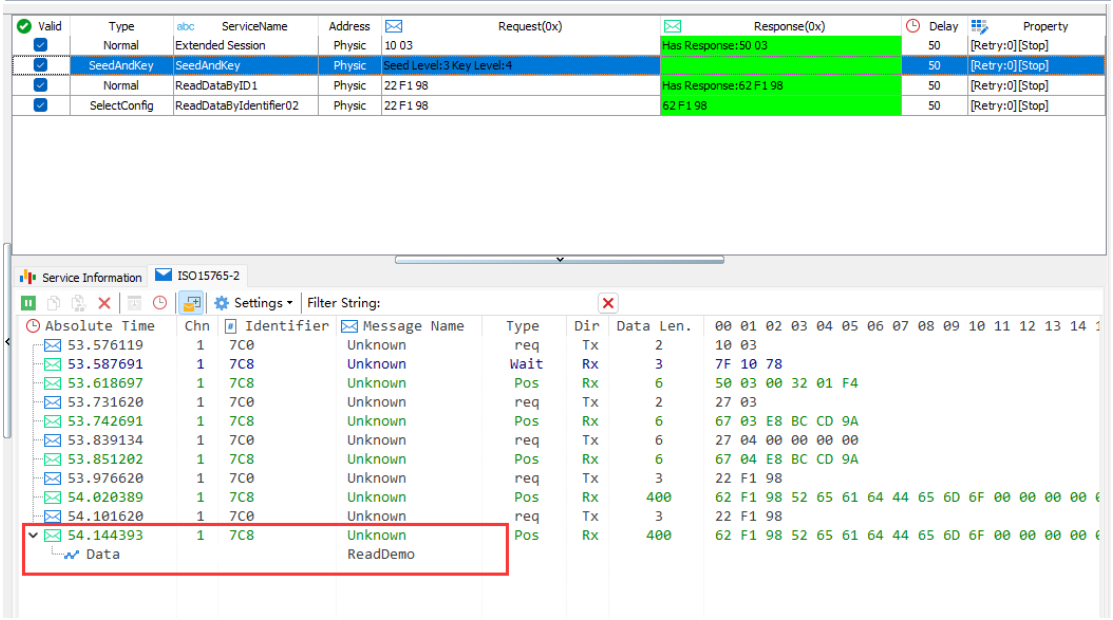
Note：自动诊断流程通过系统变量控制需要 Diagnostic License。

1.19.6.　典型应用

1.19.6.1.　读取车辆 VIN 码

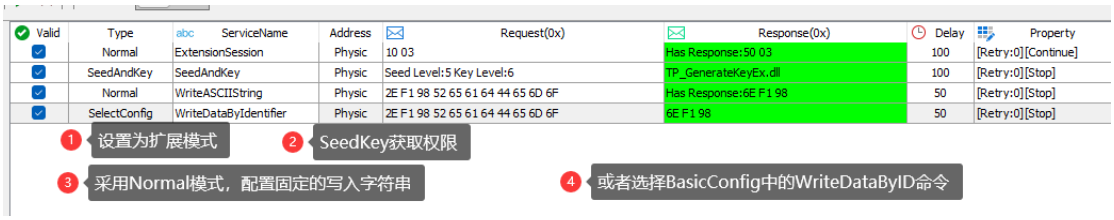


执行效果如下所示：

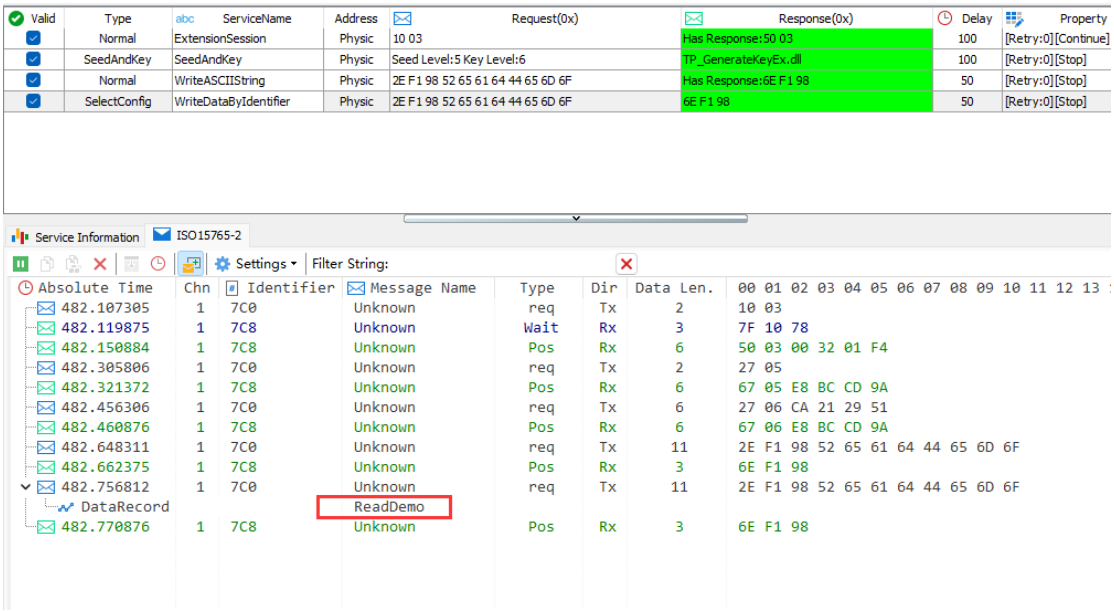


可见，同样是 ReadDataByID，引用 BasicConfig 的测试步骤中因为带有相应的解析信息，可以直接出读取的字符串为“ReadDemo”。

1.19.6.2. 写入配置信息



执行效果如下：



可见，同样是 WriteDataByID，引用 BasicConfig 的测试步骤中因为带有相应的解析信

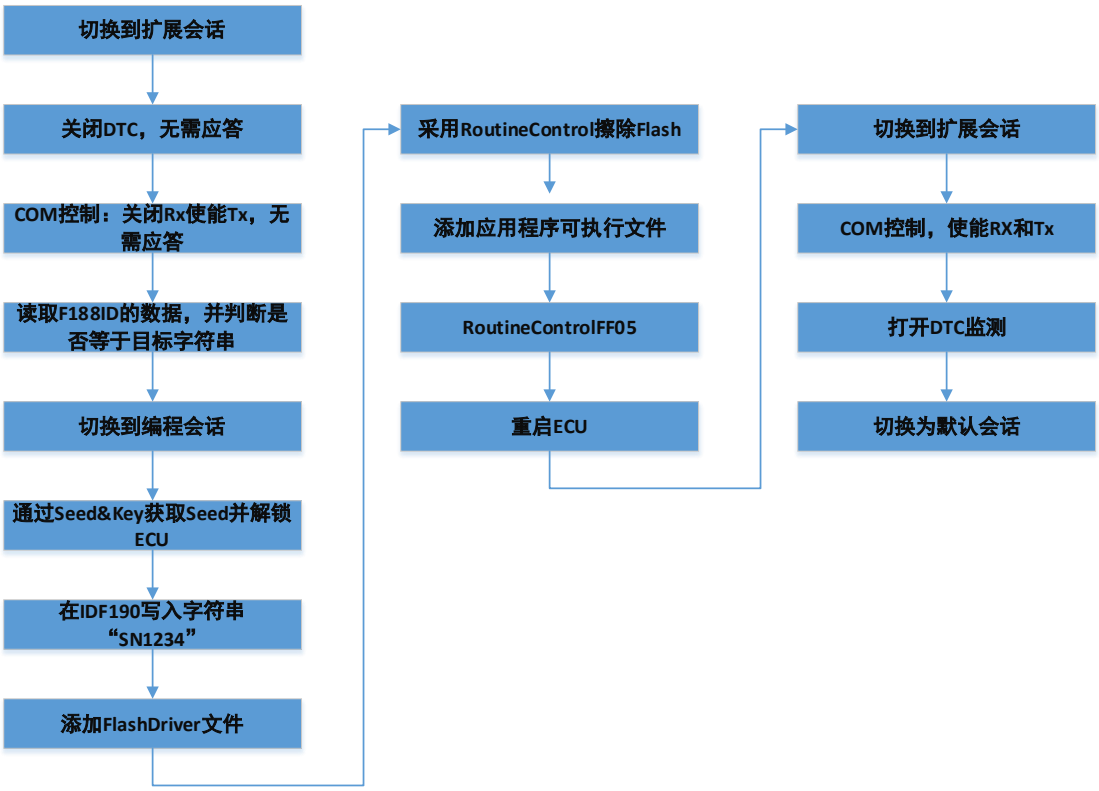
息，可以直接出读取的字符串为“ReadDemo”。

1.19.6.3. Flash Bootloader

本文设计了一个简单的 Bootloader 流程来说明如何基于 TSMaster 诊断模块配置一个 Flash Bootloader 流程。

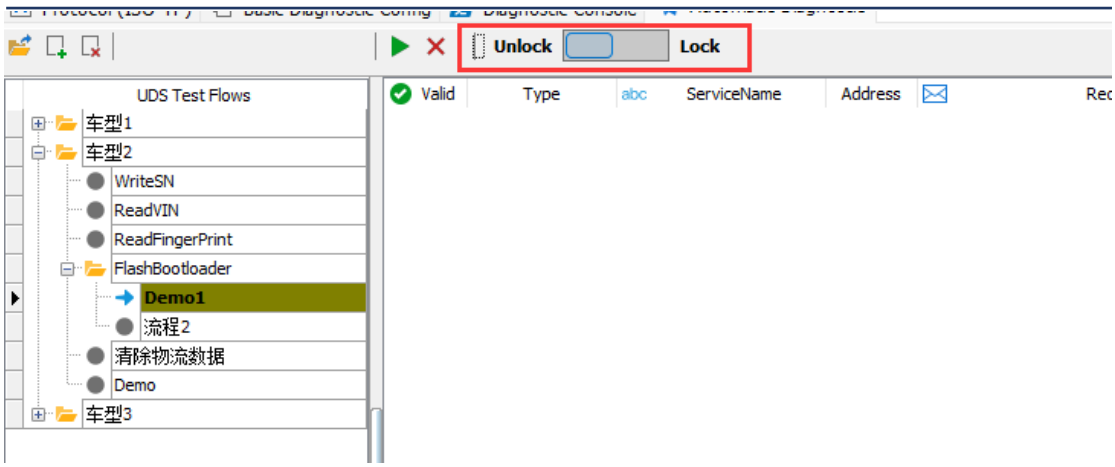
1.19.6.3.1. Flash Bootloader 流程

首先，设计 FlashBootloader 流程如下所示，这是一个示例流程，用户可以根据自己的实际设计规范整形调整。

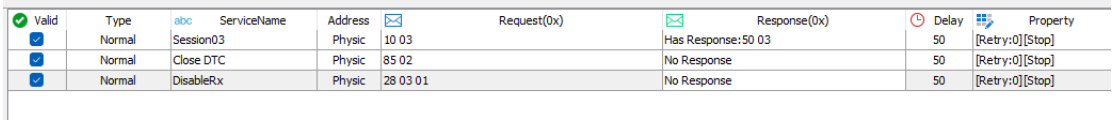


1.19.6.3.2. 配置刷写例程

【1】 首先创建 Demo1 流程：注意切换编辑器为 Unlock 状态，否则无法添加新的流程步骤。



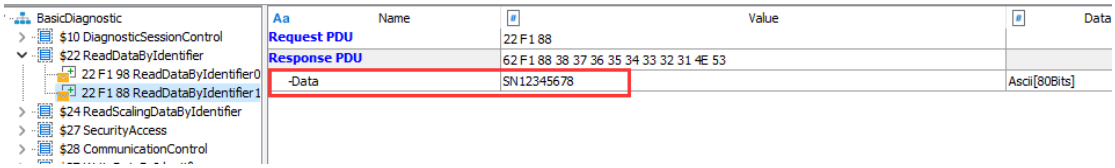
【2】 对于流程图所示的切换会话，关闭 DTC，COM 控制等命令，直接配置位 Normal 类型的命令即可(注意，这些命令当然也可以在 BasicConfig 中配置出来这里引用)。如下所示：



【3】 基于 ReadDataByID 读取 ID=F188 位置处的数据，并判断该数据是否等于比如 SN12345678。如果符合，则判断零件号匹配，进入下一个步骤，否则退出流程。配置如下：

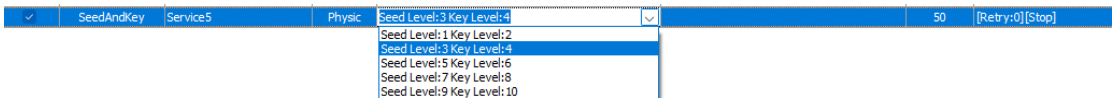
方式 1：直接配置位 Normal 形式，如下所示：

方式 2：在 BasicConfig 中配置好，然后在流程中引用：

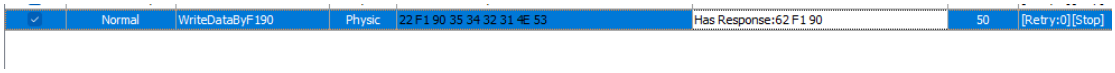


【4】 切换到编程会话

【5】 添加 Seed&Key 步骤，解锁 ECU，配置如下：



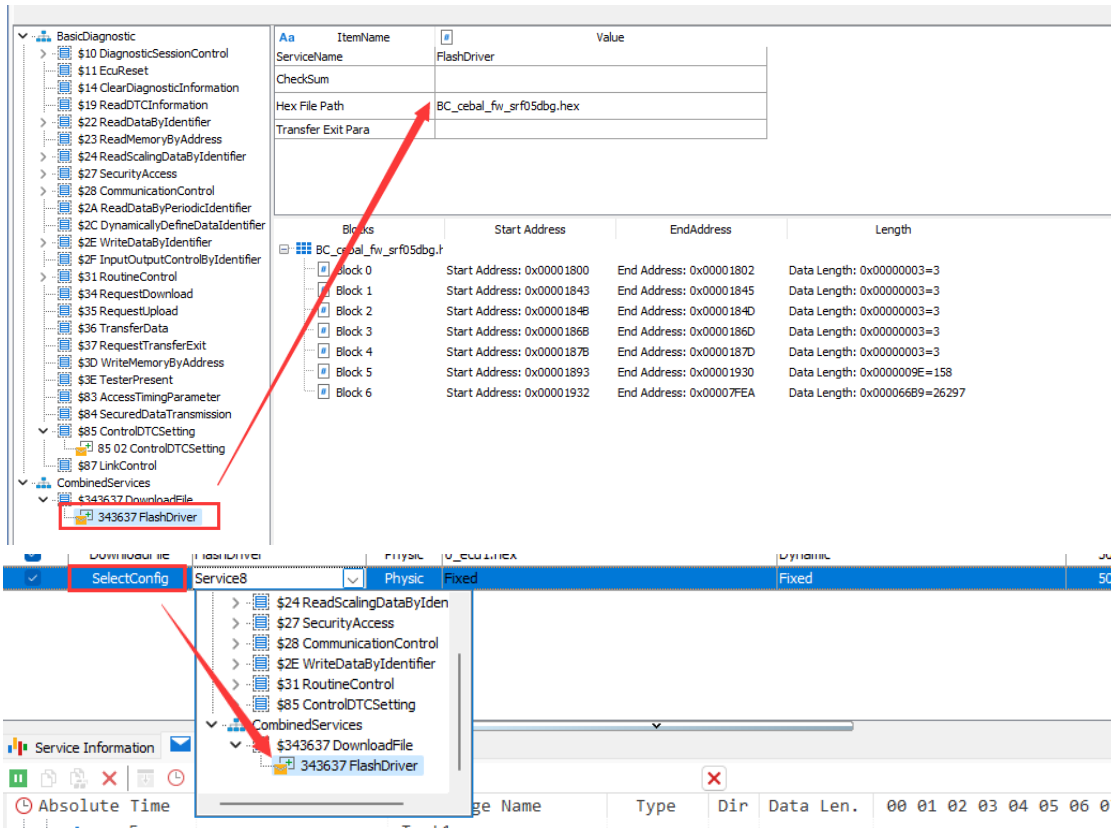
【6】 获取权限后，在 IDF190 处写入字符串“SN1234”，对于这种固定写入的字符串，最省事儿还是直接配置 NormalStep，如下所示：



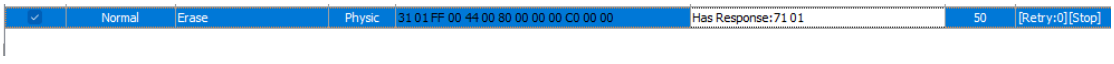
【7】 添加 FlashDriver/应用程序文件。不论是 FlashDriver 还是应用程序文件，添加方式都是一样的。两种方式：

方式 1： 直接添加 DownloadFile 步骤，如下所示：

方式 2：配置 BasicConfig，然后在 Flow 中引用，如下所示：

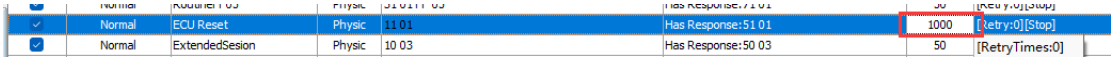


【8】 采用 RoutineControl 擦除 Flash。这里的地址和长度都是固定的，因此直接配置固定值即可，如下：



如果地址和长度是动态变化的，请看后续章节，通过引入系统变量解决此问题。

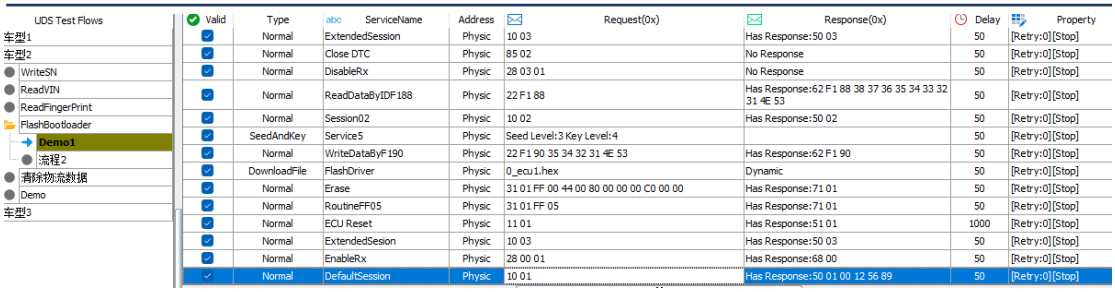
【9】 重启 ECU，直接添加 NormalStep 如下所示，需要注意的是 ECUReset 到重新开始诊断，步骤之间的等待时间需根据 ECU 设计规范调节，这里设置为 1000ms：



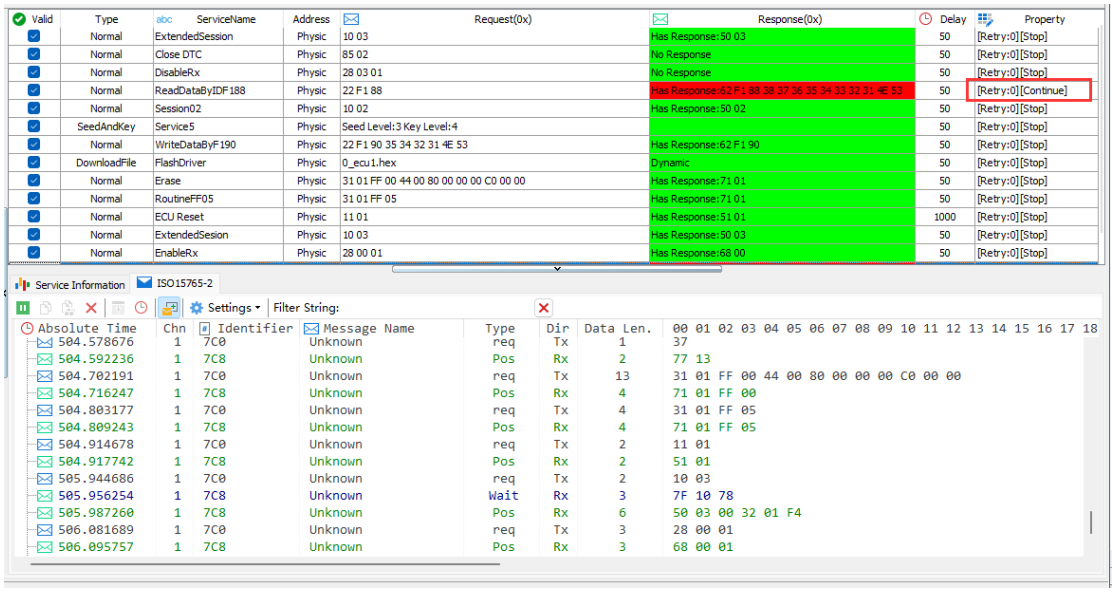
【10】 剩下切换会默认会话，COM 控制，DTC 控制等操作，按照前面步骤完全即可。

1.19.6.3.3. 总结

在完成配置后，总的配置流程如下所示，可见借助于 TSMaster 的诊断模块，开发诊断流程如 Bootloader 等应用就是一个非常简单的事情了：



实际运行效果图如下所示：



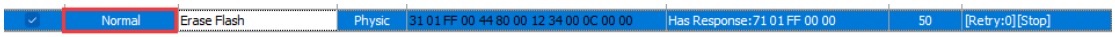
因为被测零件号不匹配，为了让测试流程走下去，这里设置发生错误后继续执行。

1.19.7. 常见问题汇总：

1.19.7.1. 擦除地址配置

【1】 固定地址和长度

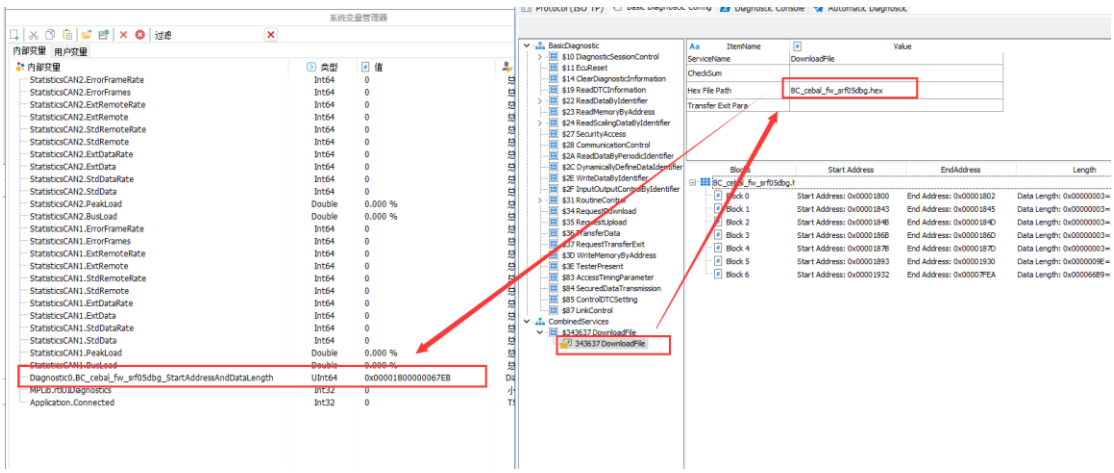
如果擦除地址是固定的地址，处理方法最简单。直接配置一个 Normal 内容的服务，里面直接填入原始数据即可。如果擦除地址为 0x00801234，擦除长度为 0x0000C000。则填入的固定值如下所示：



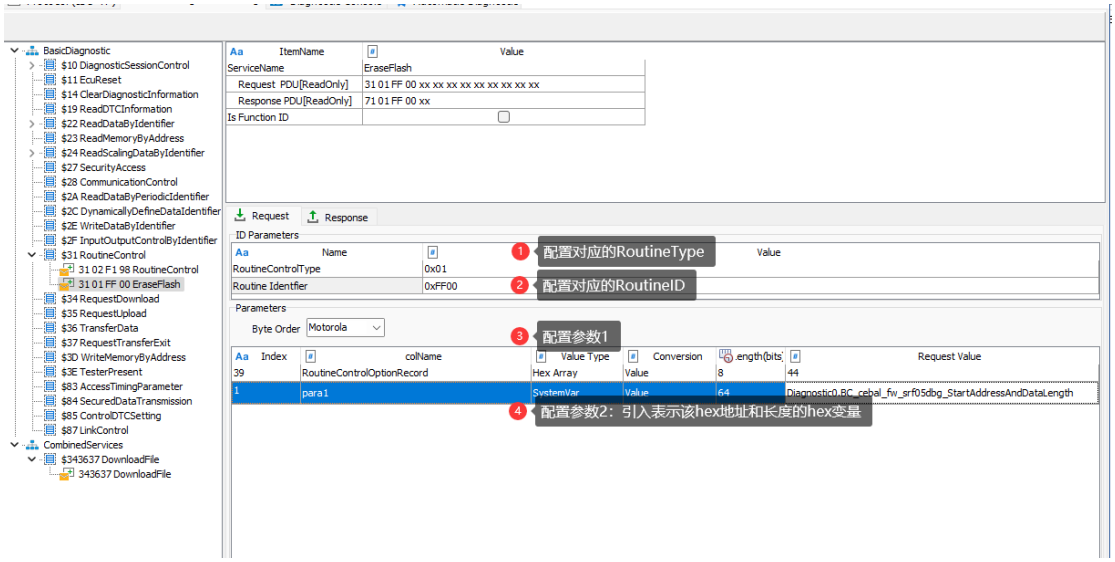
也就是把需要发送的值和期望的应答值直接填入到服务队列中。

【2】 可变地址和长度

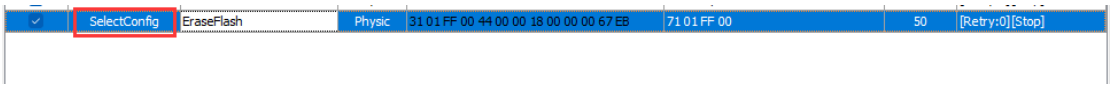
地址根据载入的不同 Hex 文件，对应的数据地址和长度是可变的。对于这种情况，则需要用到系统变量。则需要用到系统变量。以示例 hex 文件为例。诊断模块每次载入 Hex 文件的时候，会自动提取 Hex 一些特征信息作为系统变量注册到系统中（目前只注册了地址 + 长度，如有其他需求请直接反馈到同星进行评估）。如下图所示：



然后在 BasicConfig 中配置如下：



最后在自动化流程中，引用该配置即可，如下所示：

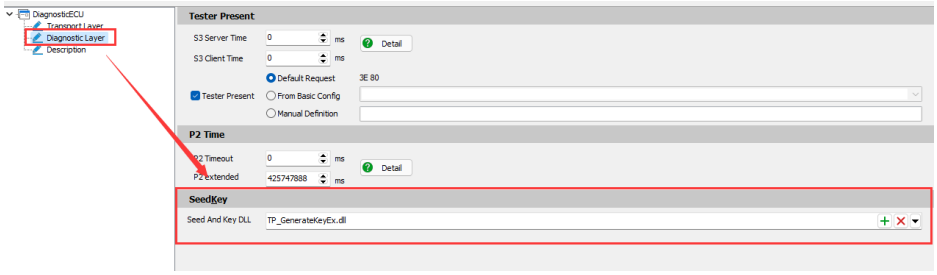


自动运行过程中，系统会自动读取当前系统变量的值，并填入到发送服务中，这样就实现了动态参数的载入。

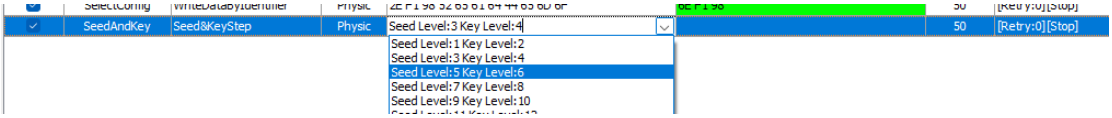
1.19.7.2. Seed&Key 的值

如果 Seed&Key 值是固定值，直接采用选择 Normal 模式，填入固定的值即可。本章节主要讲解基于 Seed&Key 动态计算 Key 值。主要包含如下步骤：

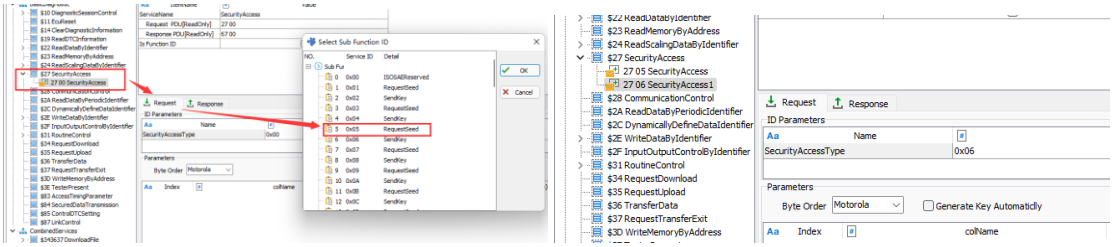
【1】 首先是在配置传输层参数的时候，载入相应的算法 DLL，这个 DLL 是本诊断模块所有涉及到 Seed&Key 算法的时候公用的 dll。因此，用户需要把各种 level 等级的 Key 计算方法都放到此函数库中。



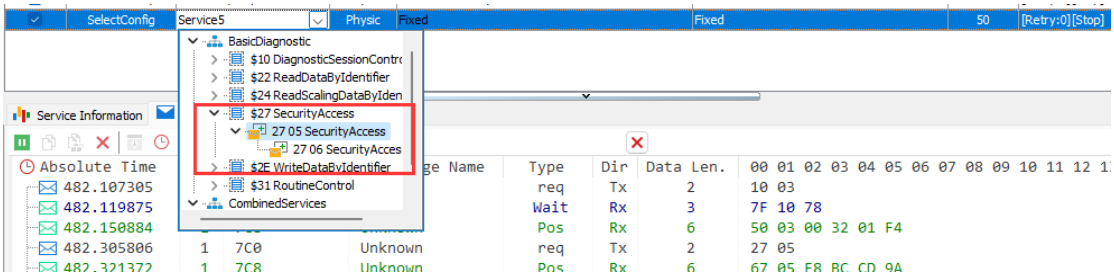
【2】 方式 1：在自动流程步骤中，添加 Seed&Key 类型的运行步骤，然后选择 GetSeed 的 Level，如下所示：



【3】 方式 2：在 BasicConfig 中，添加 0x27 GetSeed 和 SendKey 服务（注意，这两个服务必须要成对配置），如下所示：



在自动测试流程中，添加 SelectConfig 步骤，并从中选择前面配置的 0x27 服务。如下所示：



1.19.7.3. 为啥 Dll 拷贝到其他电脑过后无法使用了

这种情况，简而言之：**缺乏该 DLL 运行的依赖库文件**。造成这种情况的主要包含以下几种情况：

- 该 DLL 的运行依赖了其他库函数，拷贝的时候只拷贝了该 DLL，没有拷贝依赖的几个库函数，因此在新的电脑上无法运行。这种情况是比较显而易见的，大部分情况都比较容易解决。
- 该 DLL 发布的时候，不是采用 Release 模式，而是采用的 DEBUG 模式。**这种问题就非常隐蔽了**。比如用 C++开发了一个 Test.dll。所有的算法都写在这个 dll 里面了，没有其他外部依赖项，但是拷贝到其他电脑上依然无法工作。这是因为，C++开发的 DLL 库，运行的过程还需要依赖 C++运行时仓库，比如 vcruntime140.dll，这个 DLL 在一般的电脑上都是存在的。但是如果采用 Debug 模式发布的 DLL，他的运行时库是

vcruntime140d.dll。注意标红部分的 d，只有安装了 Visual Studio 等开发环境的电脑上才会有此运行时库。因此就会出现，在自己的开发电脑上运行的好好的仓库，拷贝到其他电脑无法运行的情况。

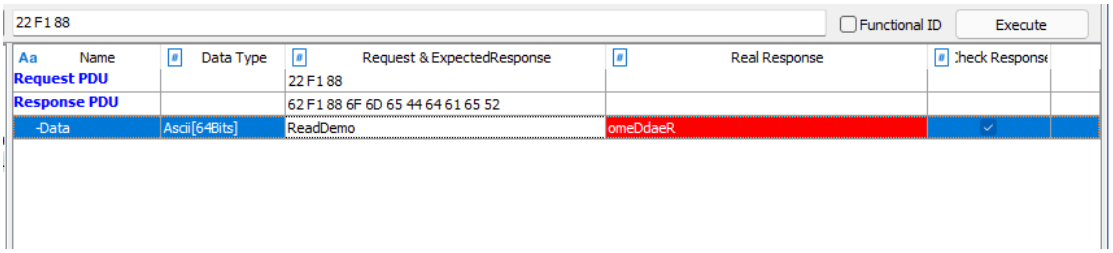
总结：1. 确保依赖项 DLL 都拷贝了。2. 确保以 Release 模式发布 DLL。

1.19.7.4. DownloadFile 调试以及文件处理

//Next

1.19.7.5. 为什么读取上来的字符串是反的

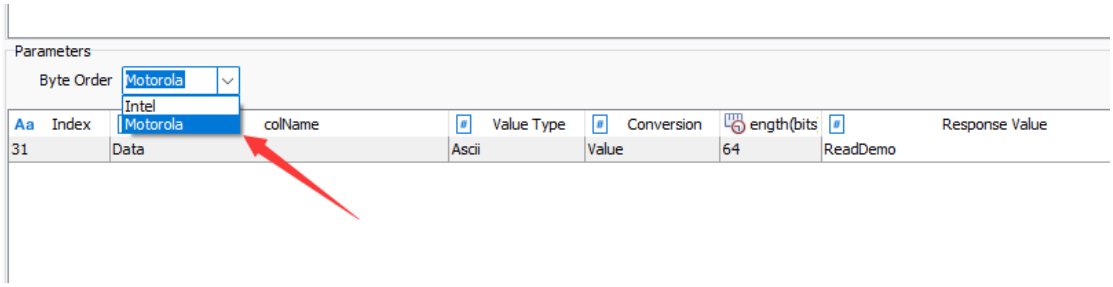
用户期望读取的字符串是 ReadDemo，但是读取上来的字符串是 omeDdeaR，完全是反的，如下所示：



这是因为配置的字符串解析顺序跟实际的字符存储顺序不匹配，所以解析出来的字符串也跟着是反的。

解决方法：

调整字符串的解析顺序，比如之前为 Motorola，现在修改为 Intel，修改过后，读取的字符串就和期望的字符串匹配了。



1.19.7.6. 为什么明明看到了总线上面有报文，报文长度也是足够的，但是显示的字符串是空的？

该情况如下图所示：

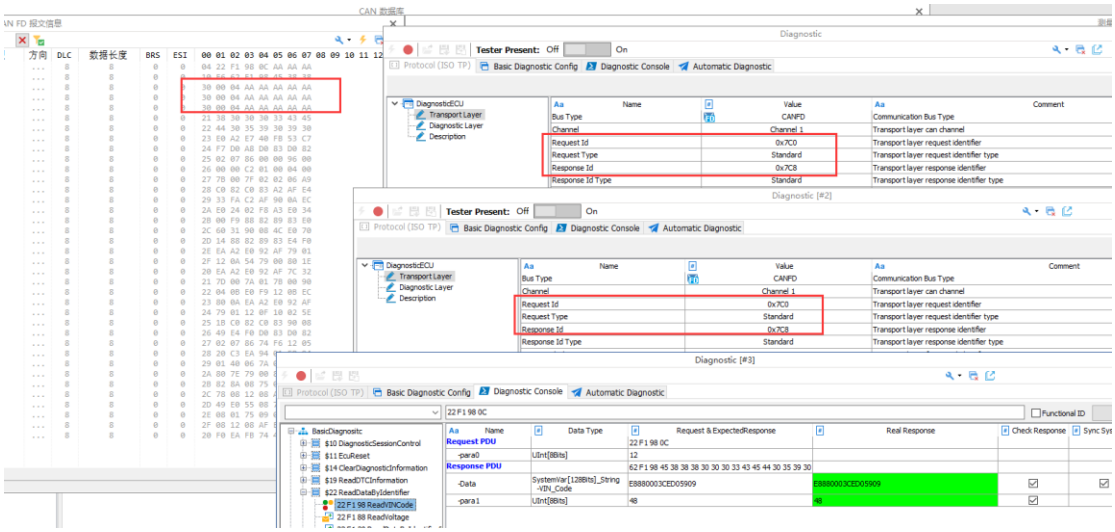
1.19.7.7. 流控帧重复应答

使用诊断模块功能的时候，查看 Trace 流程发现出现了重复比如流控帧。如下所示，出现了重复的 2 帧甚至更多帧的流控帧。

造成上述症状的原因是有多个诊断模块针对 ECU 进行了应答。因为 TSMaster 是支持多个诊断模块同时运行的，但是如果管理不善，就会造成这种诊断模块同时应答的情况。常见造成这种情况的原因主要有下面这些：

1.19.7.7.1. 创建了多个诊断模块，使用了同样的 ID 和通道

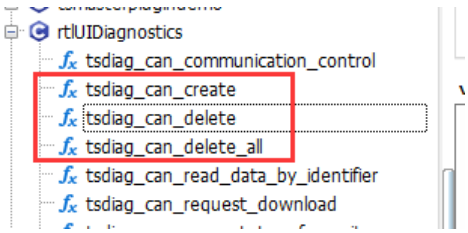
如果创建了多个诊断模块，使用了同样的 ID（请求，应答，功能 ID）以及同样的通道，那么这些模块对来自 ECU 的诊断报文的处理是同时进行的。比如进行诊断传输的时候，ECU 请求诊断应答的情况下，多个模块就会同时做出应答。从而在 Trace 中会看到多个功能帧同时发出去的情况。如下所示：



虽然诊断请求是从诊断模块 3 中发出来的，但是因为这三个诊断模块是完全一样的参数，因此，可以看到从 PC 端发出了三帧一模一样的流控帧。

1.19.7.7.2. 脚本中创建了诊断模块，但是没有释放

为了支持多个诊断模块同时运行，TSMaster 的脚本系统中，提供了如下 API 函数：



tsdiag_can_create 用于创建一个诊断模块，该函数执行成功过后，用户将获得该诊断模块的唯一句柄，如下所示：

```
if(rtlUIDiagnostics.tsdiag_can_create(&udsHandle,CH1,0,8,0x7C0,1,0x7C8,1,1,1) == 0x00)
{
  printf("Create Diagnostic Success, module handle = %d",udsHandle);
}
```

后面使用该诊断模块的诊断行为全部要基于此唯一句柄进行。比如要实现读取 0xF198 的数据，如下所示：

```
u8 reqDataArray[] = {0x22,0xF1,0x98,0x0C,0x00};
u8 responseArray[600];
int responseArraySize = 600;
if(rtlUIDiagnostics.tstp_can_request_and_get_response(udsHandle, reqDataArray,5, responseArray, &responseArraySize) == 0x00)
{
 log("send diagnostic payload and get response success! %d datas received", responseArraySize);
}
else
{
 log("send diagnostic payload and get response failed! %d datas received", responseArraySize);
}
```

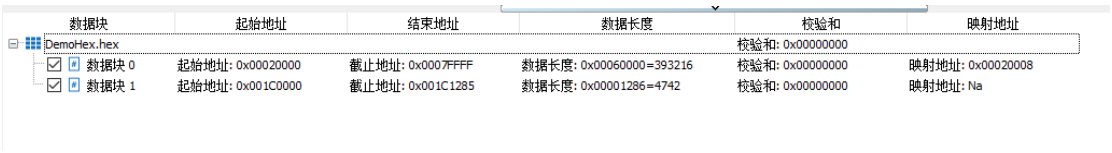
但是在使用结束过后，如果用户忘了调用 tsdiag_can_delete 函数删掉该诊断模块。那么该诊断模块就会一直驻留在系统中，并且对诊断行为进行应答。因此，在使用诊断模块的时候，tsdiag_can_create 和 tsdiag_can_delete 一定是配对使用的。比如，在脚本系统中，在脚本启动模块中创建了诊断模块，则在脚本退出事件中，一定要执行删除该诊断模块的函数，如下所示：

在程序启动时间中添加诊断模块，获得唯一句柄 udsHandle，如下：

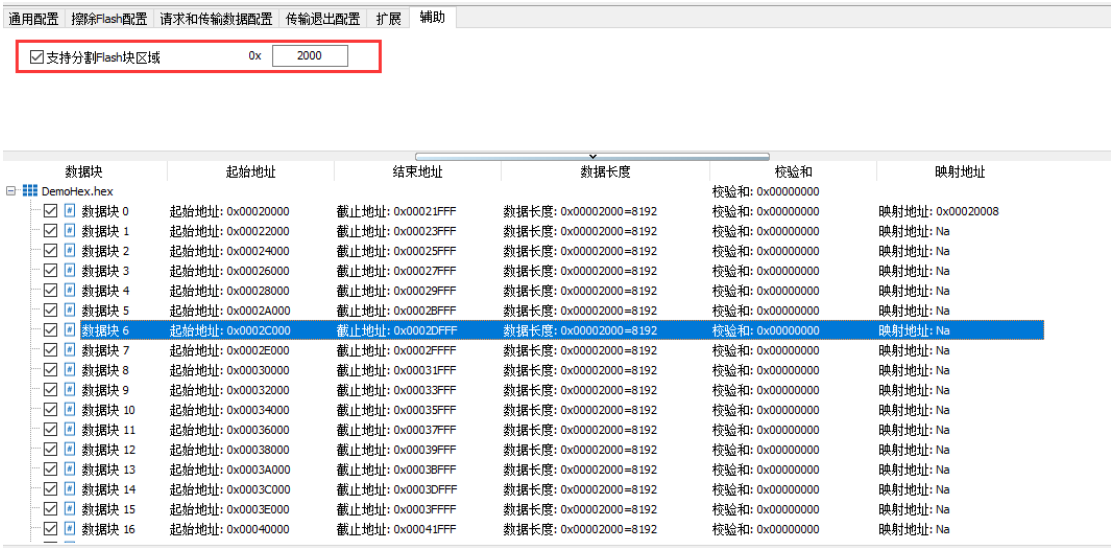
在程序退出事件中，根据该唯一句柄 udsHandle，删除该诊断模块，如下：

1.19.7.8. 下载的文件段落太大，ECU 内部来不及处理？

在配置下载文件的时候，可能存在下载文件的段落数据长度太长，但是 ECU 处理不过来的情况。比如如果一个 hex 数据文件中存在两个地址段：一段长度为 0x60000，一段长度为 0x1286，如下图所示：



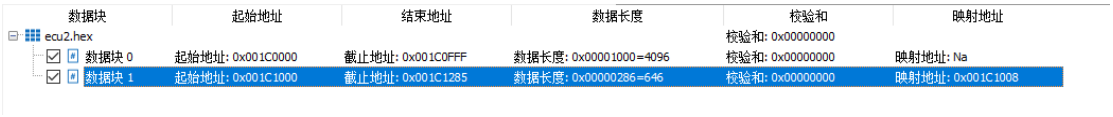
对于一些 ECU 来说，第一段的长度太长，没办法一下子处理掉（实际上这种情况并不普遍，跟一些客户 ECU 内部设计有关系，正常来说，通过服务层的传输块大小[BS 参数]限制，也能规避这种情况）。这种情况下，就需要把数据段进一部分细分，因此 TSMaster 的诊断模块提供了一个配置参数：是否支持分割 Flash 区域。用户选择支持此特性，然后设置了最大允许的 Flash 区域大小过后，软件会自动把完整的 Flash 块再进一步细分为多个逻辑 Flash 子块，操作如下图所示：



如上所示，选择支持分割 Flash 区域，并且设置每一个 Flash 块大小为 0x2000. 软件自动把之前的 0x60000 切割成了 48 个 0x2000 大小的 Flash 块。这样处理过后，在执行刷写的时候，软件就会按照 48 个独立的 Flash 块来进行处理。

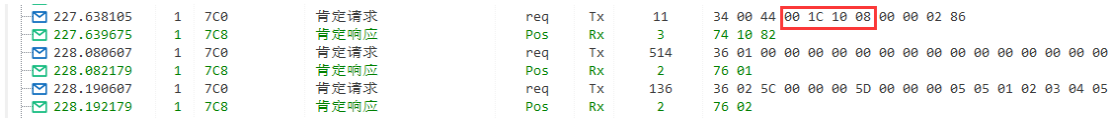
1.19.7.9. 下载的文件段落地址存在偏移的情况？

在程序执行刷写过程中，可能会存在刷写的数据地址需要偏移的情况。比如 Flash 数据文件中地址为 0x800000-0x860000 的地址段数据，实际上是需要存储在 0x900000-0x960000 地址段内。常规处理办法是对数据源文件进行处理或者刷写软件中对这部分地址段数据进行特殊处理，这些方法在实际操作中都挺麻烦。TSMaster 中提供了一种地址偏移的方法，允许用户选择数据块地址段进行灵活的地址映射。如下图所示：



段落 0 的原始起始地址为 0x001C0000,映射地址为 Na，表示没有偏移；段落 1 的原始起始地址为 0x001C1000，映射地址编程 0x001C1008。执行刷写操作，传输数据如下：

可见，对于段落 0，其传输层地址为 0x001C0000，地址没有偏移。



可见，对于段落 1，其传输层地址为 0x001C1008，而不是 0x001C1000，地址发生了偏移。

1.19.7.9.1. 映射操作

在下载配置界面的 Flash 数据段界面，右键弹出映射菜单。主要包含三类操作：增加/编辑映射，删除映射，清除所有映射，如下图所示：

- 新增/编辑映射信息：选择指定的 Flash，右键弹出地址编辑窗口，如下图所示：

新增映射信息后，映射地址栏会显示相应的映射地址，如果没有映射信息，则显示为"Na".

- 清除映射信息：清除指定 Flash 块的映射信息。
- 清除所有映射信息：清除 Flash 文件中所有数据块的映射信息。

1.19.8. 在 EOL 和非标项目中的应用

1.19.8.1. 导出配置文件

在 TSMaster 中完成诊断流程的开发过后，支持导出配置文件(*.tflash)。

1.19.8.2. TFlash 载入配置文件

1.19.8.3. 外部程序调用 TFlash 自动化服务器

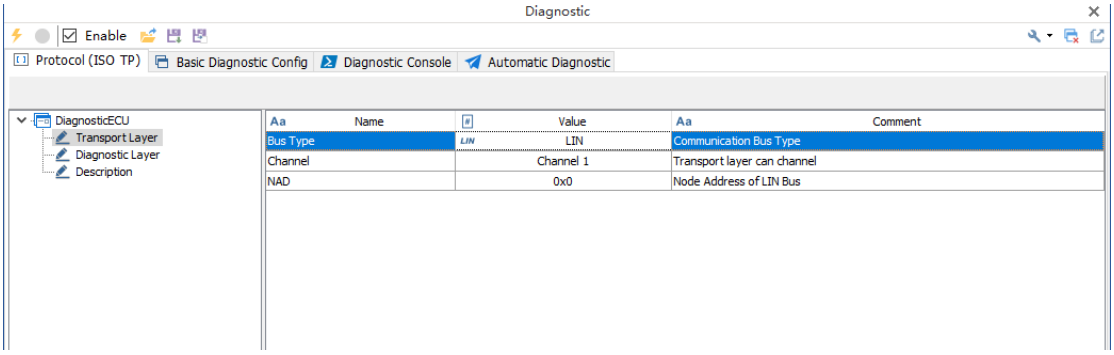
1.20. LIN 诊断（Diagnostic_LIN）

1.20.1. Diagnostic TP 参数配置

TSMaster 提供了诊断控制台基础功能，用户可以根据需求配置自己的发送和应答请求。按照如下步骤操作即可。其中很多重复的步骤本章节不再讲解，用户可以查看 CAN 诊断章节。

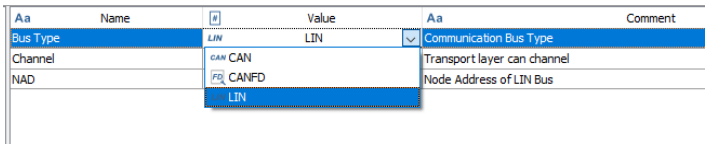
1.20.1.1. 传输层参数：

选择总线类型为 LIN 总线，点击确定后，界面如下所示：

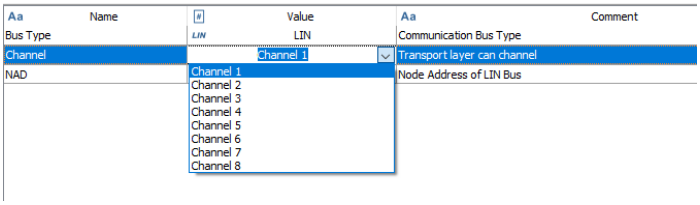


其中，各个参数解释如下：

- Bus Type: 诊断传输层类型，目前已经支持 CAN/CANFD/LIN，接下来支持以太网和 Flexray 等。通过下拉列表可以选择，如下图所示：



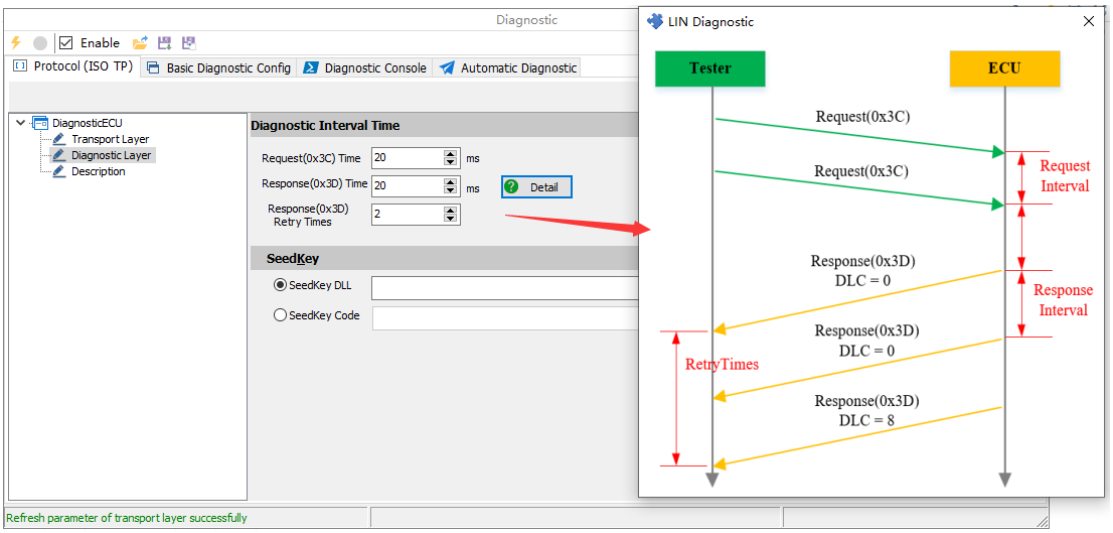
- Channel：诊断模块用到的逻辑通道。TSMaster 支持多个诊断模块同时在线工作，这里用于选择当前诊断模块使用系统的哪一个逻辑通道。通过下拉列表进行选择，如下图所示：



- NAD：节点地址。LIN 总线中，所有的节点都是通过 0x3C 发送诊断请求报文，0x3D 发送诊断应答报文。为了区分该诊断的目标节点，引入了参数 NAD，通过 NAD，才知道当前的诊断报文具体是用于那个 LIN 节点的。

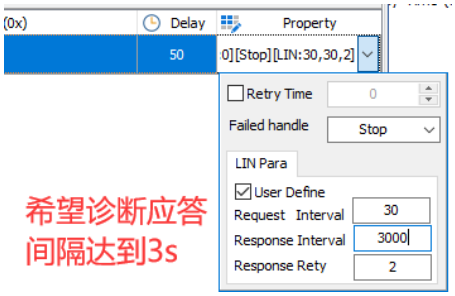
1.20.1.2. 服务层参数：

服务层参数主要包含请求（0x3C）和应答（0x3D）的间隔时间参数，应答重复次数，以及加载 SeedKey 的 dll。如下图所示：



- 间隔时间（IntervalTime）：如上图详情中示意图所示，间隔时间是指诊断和应答之间的报文时间间隔。为了便于用户使用，请求间隔和应答间隔可以单独配置。
- 应答重复次数：当主节点发送 0x3D 的时候，ECU 如果还没有准备好应答数据，将不予回复。这种情况下，通过设置此参数，ECU 将多次尝试去获取应答，具体尝试的次数就是此参数。

上面配置的时间是诊断请求，应答间隔时间以及重复尝试次数都是诊断模块的默认参数。在实际使用过程中，会出现比如需要诊断应答间隔时间较长的情况，比如 PC 软件通过诊断命令发送擦除 Flash 指令给 ECU，但是 ECU 需要较长时间才能应答，此时需要 0x3D 的间隔时间较长。因此，则 TestFlow 模块中，也提供了配置窗口，让用户可以配置该步骤希望的诊断请求应答间隔以及重复次数，如下图所示，如果没有配置，则使用默认参数：



1.20.1.3. Seed&Key

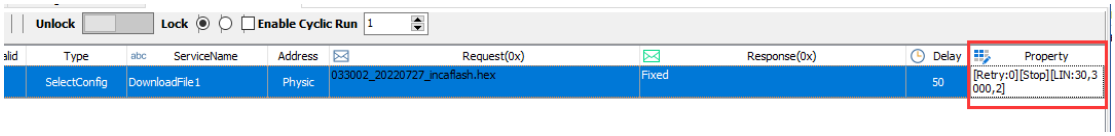
参考 CAN 部分 Seed&Key 的配置。

1.20.2. 基础诊断配置

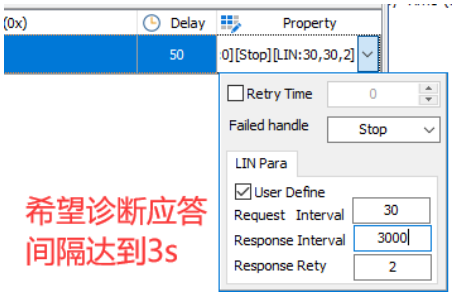
诊断配置部分，思想和配置方法等同于 CAN 部分配置。跟 CAN 总线配置不一样的地方有以下环节：

1.20.2.1. TestFlow 中增加了控制参数配置

在 TestFlow 配置窗口中，选择一个配置流程，选中一个配置项，点开该配置项的 Property 属性窗口，如下所示：



如下窗口所示，属性中增加了 LIN Para 选项。如果用户想使用默认参数，则不勾选 UserDefine。如果本步骤中想使用自定义参数，则勾选 UserDefine，并且填入参数。

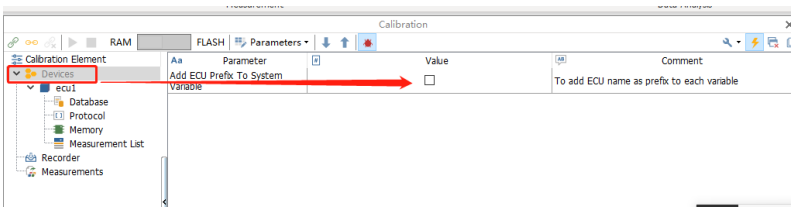


上述配置的参数只会作用于本步骤，进入新的步骤过后还会继续使用默认参数。这样做最大的好处是，让用户可以灵活选择诊断报文的间隔时间。比如，Session Conrol 命令 ECU 的处理比较简单，则直接使用默认参数，然后 Erase Flash 命令，ECU 的处理比较好时，则使用自定义参数，紧跟着 Transmit Data 命令，ECU 处理时间较长，也可以使用自定义参数，填入较长的应答间隔时间以及较多的应答尝试次数。

1.21. 标定（Calibration）

1.21.1. 多个 ECU 同时标定，如何区分

当多个 ECU 同时进行标定，使用同样的 A2L 文件或者文件中变量有重复的时候，需要区分来自不同 ECU 的标定和测量变量。TSMaster 提供的解决办法是，在标定配置工具中勾选增加 ECU 名字的前缀：



勾选标定 ECU 名称

勾选此选项过后，重启软件。系统变量中相关的标定量会自动添加相应的前缀，如下所示：

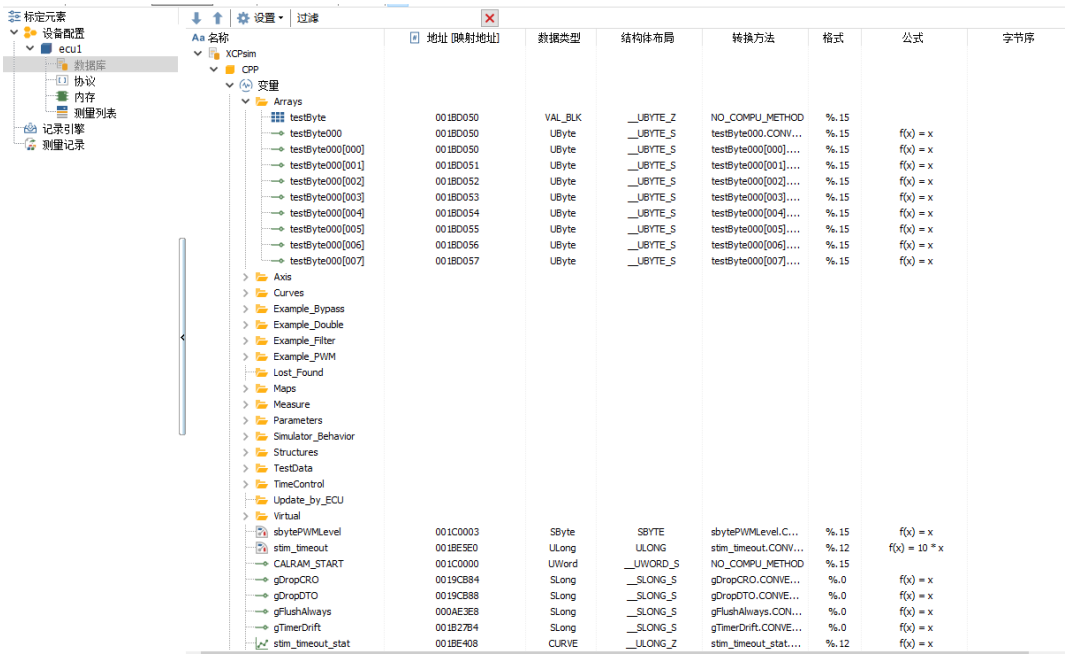
Internal Variables | User Variables

Internal Variable	Type	Value	Last Written	Owner
ecu1.CALRAM_START	Int64	0	n.a.	Calibration
ecu1.Simulator_concept	Double	0.000000000000000	n.a.	Calibration
ecu1.TestStruct1.s0	Int64	0	n.a.	Calibration
ecu1.TestStruct1.s1	Double	0.000000000000000	n.a.	Calibration
ecu1.TestStruct2[000].s0	Int64	0	n.a.	Calibration
ecu1.TestStruct2[000].s1	Double	0.000000000000000	n.a.	Calibration
ecu1.TestStruct2[001].s0	Int64	0	n.a.	Calibration
ecu1.TestStruct2[001].s1	Double	0.000000000000000	n.a.	Calibration
ecu1.TestStructStruct1.Tes...	Int64	0	n.a.	Calibration
ecu1.TestStructStruct1.Tes...	Double	0.000000000000000	n.a.	Calibration
ecu1.TestStructStruct1.do...	Double	0.000000000000000	n.a.	Calibration
ecu1.TestStructStruct1.ints	Int64	0	n.a.	Calibration
ecu1.TestStructTypes.cTest	Int64	0	n.a.	Calibration
ecu1.TestStructTypes.dTest	Double	0.000000000000000	n.a.	Calibration
ecu1.TestStructTypes.fTest	Double	0.000000000000000	n.a.	Calibration

新增了 ECU 前缀的系统变量

1.21.2. 数据库视图

TSMaster 中，A2L 数据库的视图有两种模式：树形模式和列表模式。顾名思义：树形模式就是以树形结构展开 A2L 数据库内部的变量，从中可以看到各个变量的层次关系等，如下图所示：



列表模式，就是平铺显示 A2L 里面的所有的观测量和标定量，如下图所示：

用树形模式的优点是方便查看变量的层次关系，但是缺点是不方便同时选择多个变量。比如想一次性添加上百个或者所有变量到测量列表中，如果在树形视图下，需要一个一个展开并进行添加。这种情况下就需要切换到列表模式，在列表模式下，直接全选中，并添加即可。

1.21.3. CCP DAQ

当无法从 A2L 中直接获取 DAQ 相关配置的时候，可以参考已有工程，手动添加 DAQ 的描述配置，其主要包含两部分： Event Channel + DAQ。过程如下所示：

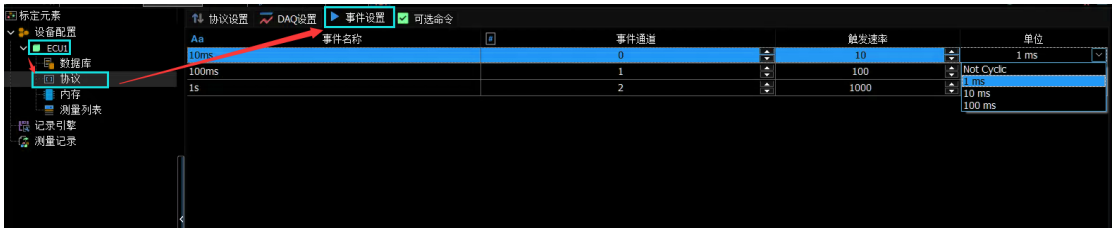
1.21.3.1. 事件通道(Event Channel)

事件通道是开发 ECU CCP 模块的过程中就定义好的事件触发机制。因此，可以根据描述文件，手动添加事件通道。事件通道主要包含以下几个属性：

- 通道名称：这个名称是用户任意命名的，主要不要重复，具有明确的提示性即可。
- 事件通道：该通道编号是 ECU 设计的时候确定的，跟触发速度是匹配的。比如 ECU 中定义了通道为 0 的事件对应的是 10ms 触发速率，他们俩的对应关系不能有误。
- 触发速率：选择跟通道编号对应的速度。
- 单位：选择 1ms 即可。

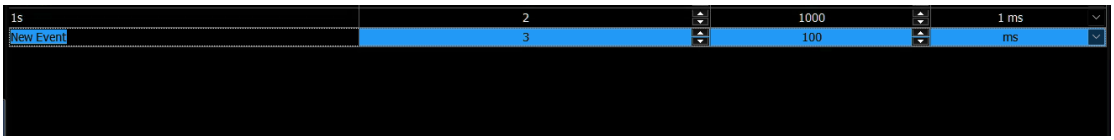
添加步骤：

1. 通过 ECU->协议->事件设置



2. 右键添加一个事件

3. 输入名称，选择事件通道，触发速率和单位即可。



1.21.3.2. DAQ 配置

当无法从 ECU 配置中自动读取 DAQ 列表的时候，用户需要手动添加 DAQ 列表描述，如下所示：

1. 通过路径 ECU->协议->DAQ 设置，进入设置界面：

2. 去掉自动检测的勾，如下：

如果不去掉勾选，则 ODT 列表是 ECU 在下载过程中，通过命令动态从 ECU 中获取 ODT 列表的描述，并且动态分配 ODT 列表，这里的属性就不需要手动配置了，因此也就

无法配置属性。

3. 空白位置处右键，可以增加和删除 ODT 列表描述

4. 设置 ODT 列表属性

如果不清楚 ODT 列表详细含义，参照现有工程，把上述描述表配成一样即可。

5. 注意事项：

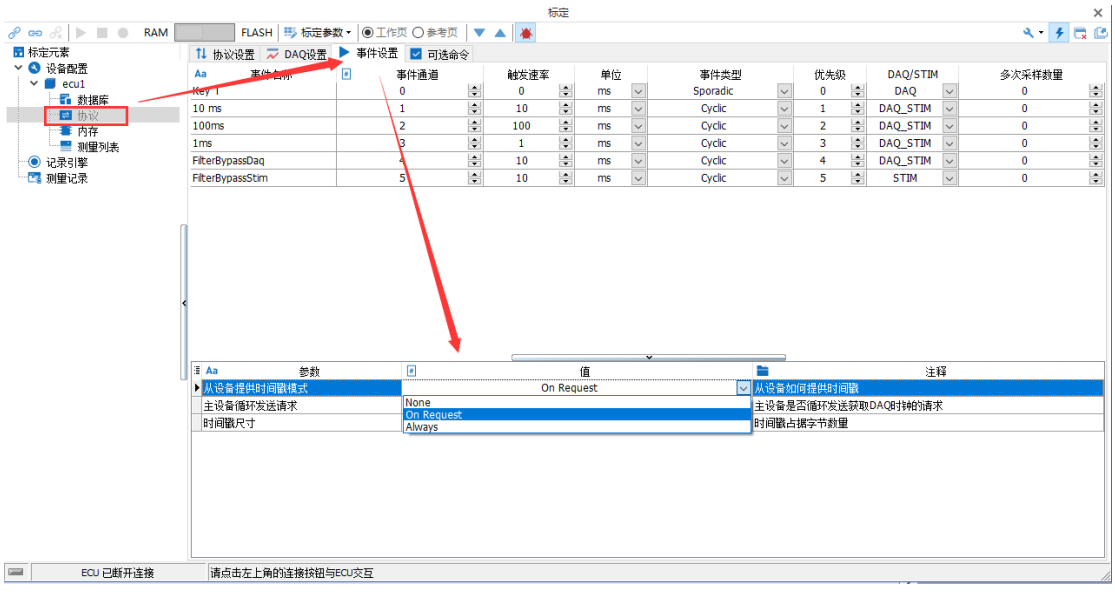
事件设置中设置了通道过后，需要保存工程，重启一下软件，在 DAQ 设置的下拉框中才能看到相应的事件通道属性。

1.21.4. XCP DAQ

1.21.4.1. DAQTimeStamp

在 XCP 标定协议中，有一个 DAQ 时间戳特性。当启动本功能后，标定设备从被测 ECU 节点读取当前的时间戳用于矫正标定设备端的时间戳。在 TSMaster 中，相关的配置如下：

1. 首先，相关配置的路径在：协议->事件设置->时间戳设置。可见主要包含三个配置：从设备提供时间戳的模式，主设备是否循环发送请求以及时间戳尺寸。

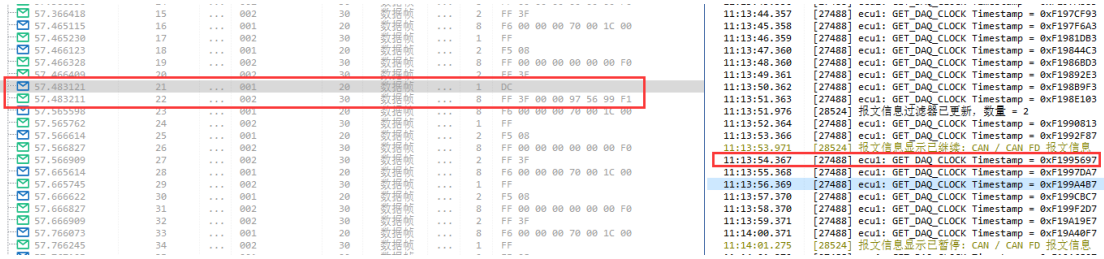


2. 从设备提供时间戳模式：

- None: 无论从设备是否有能力提供时间戳，都关闭其 DAQ TimeStamp 的功能。
- OnRequest/Always: 如果从设备具备提供时间戳的能力，那么启动其 DAQ DAQTimestamp 的功能。

3. 主设备循环发送请求：

如果本配置设置为 True，则标定主设备每隔 1s 发送一个 GetDAQClock 命令给从设备，从设备返回当前对应的 DAQ 时间戳值。反映到报文上，如下图所示：

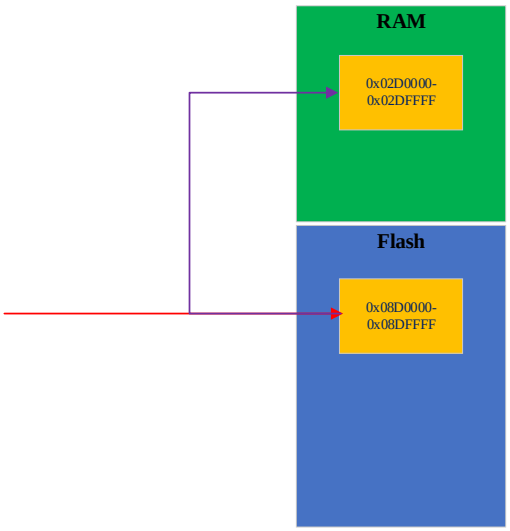


如上图所示，标定设备每隔 1s 钟向从设备请求一次 DAQ 时间戳值。

1.21.5. RAM 和 ROM 切换

1.21.5.1. 原因分析：

有些 ECU 中，标定时候需要切换内存地址映射，比如需要切换到 RAM 中进行标定（跟 ECU 内部 CCP 架构有关系，并不是所有都需要此机制）。如下所示：

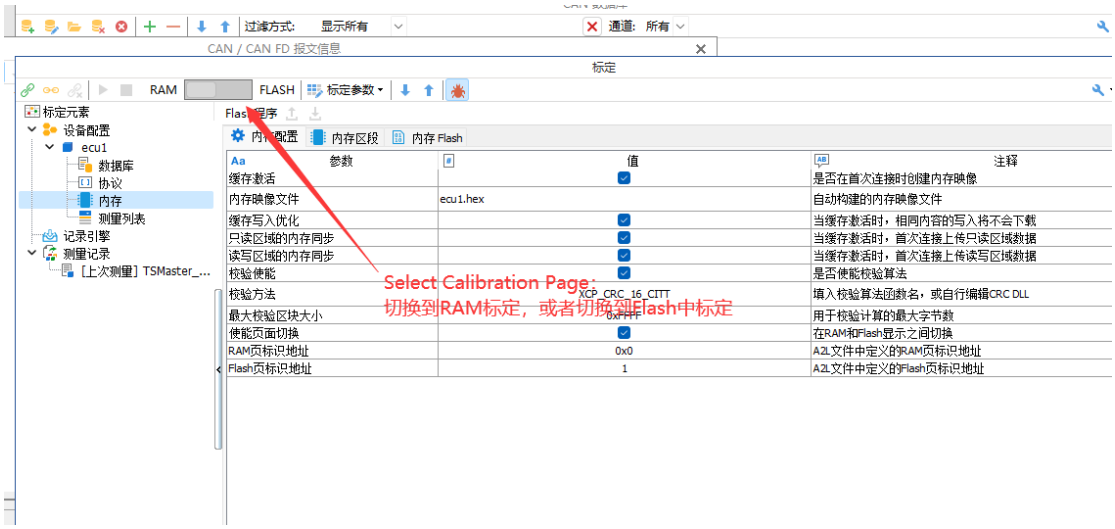


如上图所示，ECU 中设置了内存映射，那么标定 0x08D0000-0x08DFFFF 所在的 Flash 地址的时候，实际直接标定的是内存地址为 0x02D0000-0x02DFFFF 的 RAM 地址。在标定完成后，PC 端会形成一个 Hex 文件，然后通过 Bootloader 把此 Hex 文件烧录到 Flash 地址处即可。

如果不切换，ECU 的 CCP 模块收到标定地址过后，就直接去该 Flash 地址处进行标定。因为 Flash 不同于 RAM，不能随机写入，会触发 Flash 写入硬件错误，造成 ECU 标定模块直接异常退出。

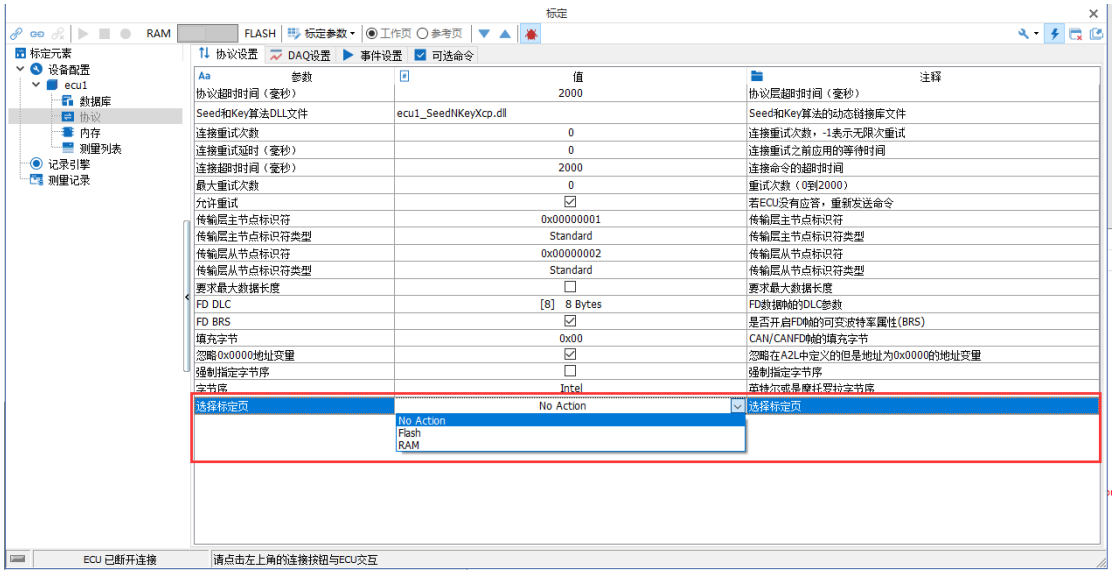
1.21.5.2. 切换标定页：

ECU 第一次上电的时候，在完成 TSMaster 连接过后，如果 ECU 当前还处于 Flash 标定状态，则只需要点击标定管理器上面的切换按钮，通过 CCP 命令让 ECU 切换到 RAM 标定即可。如下图所示：



1.21.5.3. 连接过程中自动切换

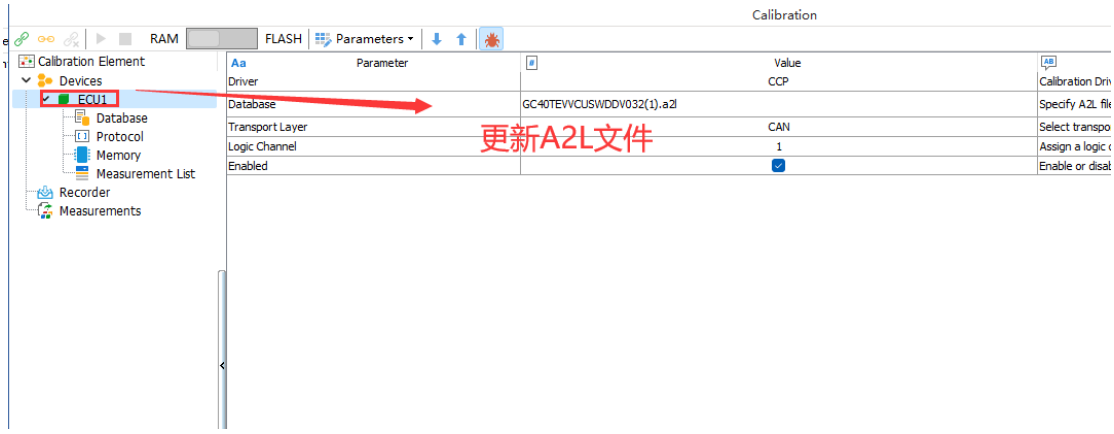
被标定 ECU 需要切换到对应的页面才能正常进行数据的标定。这个切换过程用户可以在完成 ECU 连接过后手动切换，也可以选择在连接 ECU 的过程中就完成该切换动作。具体采用那种方式，用户需要到协议->协议设置->选择标定页配置中去选择，如下图所示：



1.21.6. A2L 文件

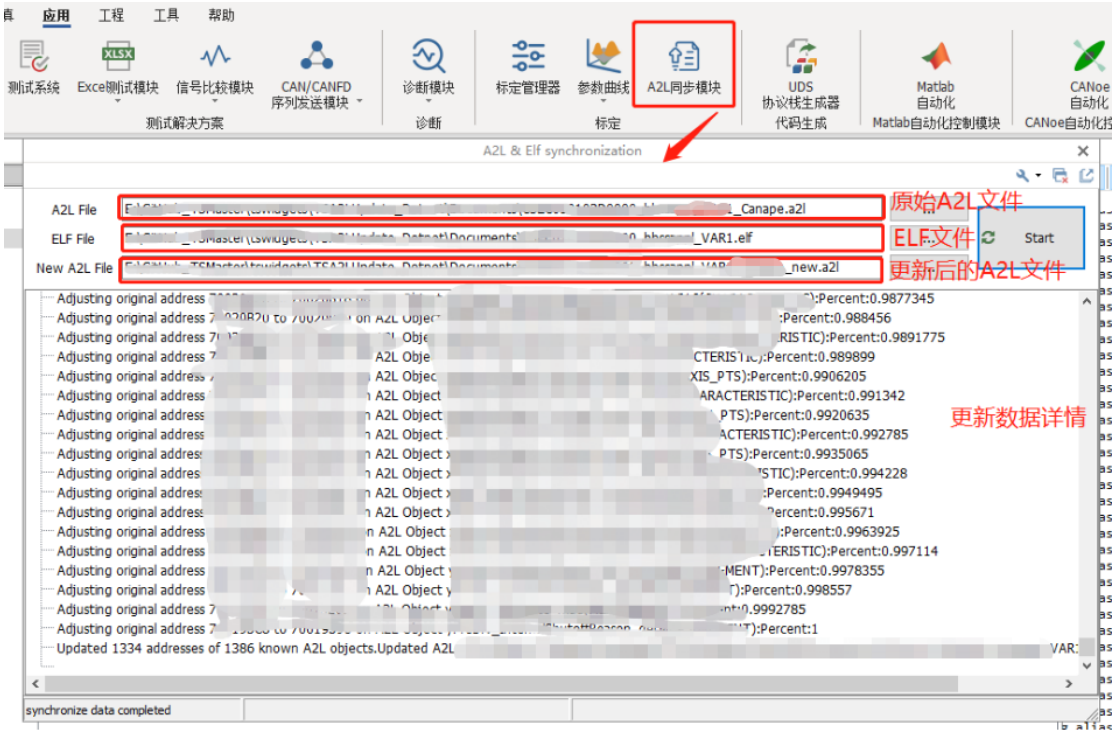
1.21.6.1. 载入新的 A2L 文件

完成上述配置过后，保存一个基础模板工程。当后续 A2L 发生变化的时候，只需要在此基础工程上更新 A2L 文件即可。如下图所示：



1.21.6.2. 更新 A2L 文件

更新 A2L 文件内部地址。TSMaster 中提供了 A2L 同步模块，允许用户在更新 Elf 数据过后，把新的数据地址同步到新的 A2L 文件中，如下所示：



1.21.7. 使用 CANFD 通道

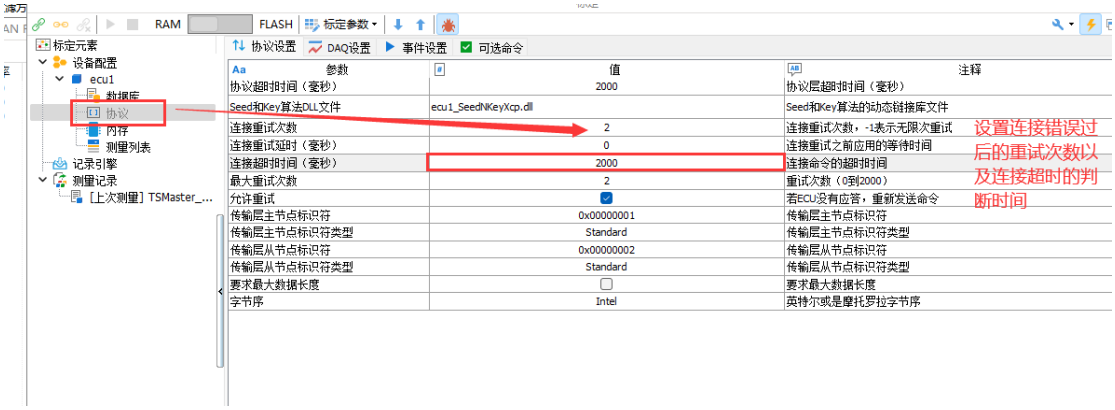
TSMaster 支持 CANFD 的 XCP 标定。要设置支持 CANFD 的标定特性，需要进入标定模块->标定 ECU->协议里面，选择传输层主/从节点标识符类型，如下图所示，选择 standard FD/Extended FD：

1.21.8. ECU 连接失败(Connect Fail)

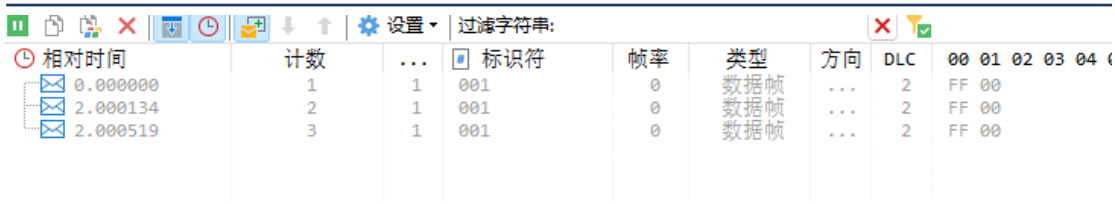
1.21.8.1. 设置重连 ECU

在连接 ECU 的过程中，可能出现这种情况。ECU 节点收到第一帧 Connect 命令过后，不回复，收到第二帧连接命令才回复，如下图所示：

如果标定没有配置重连机制，就会造成软件只尝试一次连接过后，即返回连接失败的错误。解决办法，就是配置 ECU 的重连机制，从 ECU->协议->协议设置，如下图所示：



经过如上图所示的设置过后，软件会再尝试 2 次连接，如下图所示：



1.21.9. 标定数据管理

1.21.9.1. 简介

标定模块中，标定数据的管理也是其核心功能。主要包括以下方面的内容：

- 标定数据的载入。把现有的标定数据载入到标定模块中。
- 标定数据导出。把已经标定好的数据从标定模块中导出为数据文件(s19, hex, mot)进行管理。用于下一次标定或者刷写到目标 ECU 中。
- 标定数据的刷写。通过 XCP Program/UDS 协议把标定好的数据文件下载并固化到目标 ECU 中，让标定数据生效。
- 配套应用程序的刷写。在标定现场，也支持用户通过标定模块直接下载配套的应用程序。

下面对详细介绍这些功能。

1.21.9.2. 标定数据的载入

标定数据的载入路径如下：选择目标 ECU->内存->内存配置->内存映像文件->加载按键。

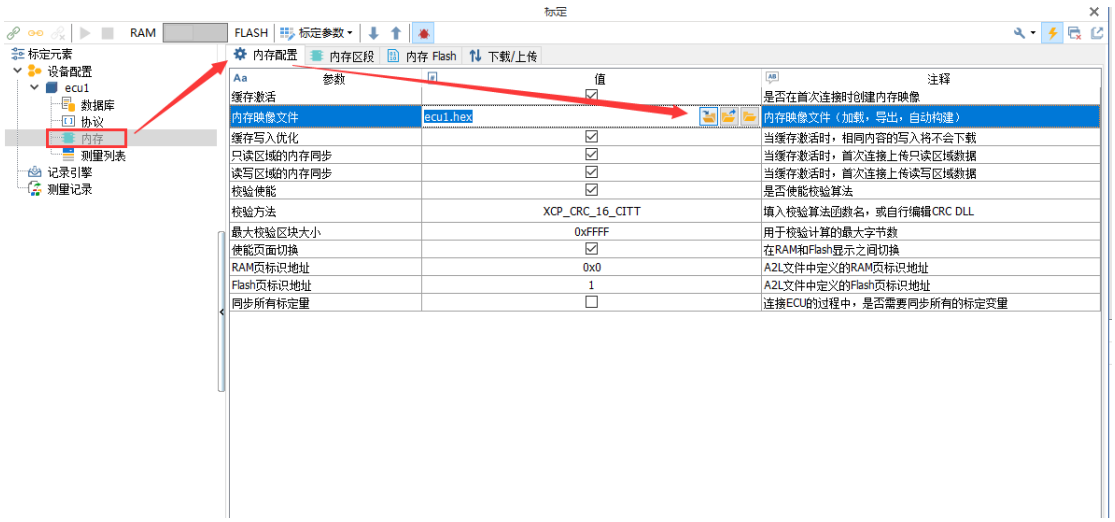


图1. 加载标定文件的路径

此操作等同于对标软件加载(hex,s19)文件的操作。加载的时候是直接多种格式的，加载过后该文件会自动被转存为(ECU 名字+.hex)，加载文件框如图所示：

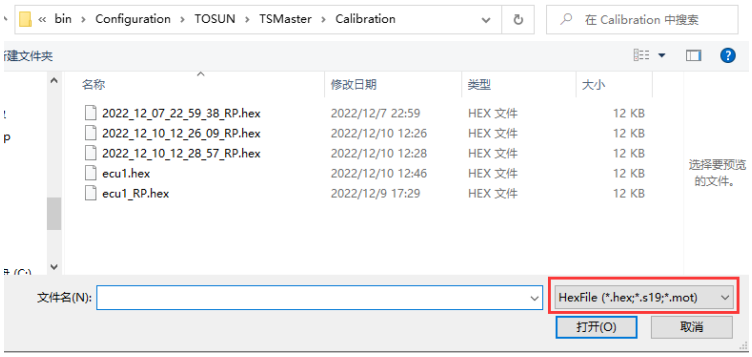


图2. 支持多种数据格式的加载

1.21.9.3. 连接速度慢的问题

在使用标定模块的时候，有用户反馈会有连接速度较慢的问题，比如完成一次 ECU 连接过程需要长达 1 分多钟。这是因为，标定模块在连接 ECU 的时候，会首先检验 ECU 内部的标定数据是否和标定软件中的标定数据匹配。如果不匹配，则需要把标定软件中的数据同步到 ECU 中，或者把 ECU 中的数据同步到标定软件中，如下图所示：

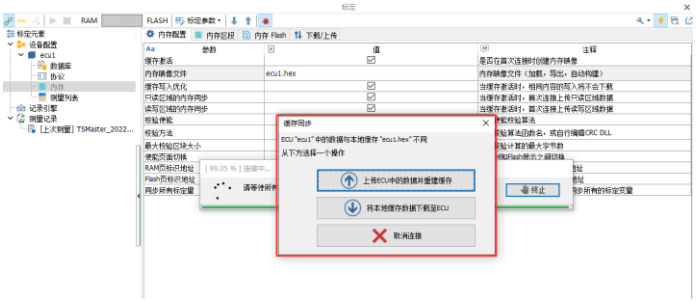


图3. 需要同步 ECU 和标定软件之间的数据

更极端的情况是，标定软件中压根就没有加载标定文件，所以连接的时候，就需要把 ECU 中的标定数据读取到标定软件中。当标定数据比较多的时候，**该读取过程就会很消耗时间**，这就是为什么连接过程很慢的原因。

因此，解决办法就是在连接之前，把标定数据文件加载到内存印象中。当 ECU 连接的时候，当监测到 ECU 中的标定数据和软件中标定数据一致的时候，不会有同步数据的过程，就能很快完成 ECU 的连接过程（实测以 s 为单位）。

1.21.9.4. 标定数据的导出

1.21.9.4.1. 直接导出现有的标定文件

从现有的标定文件中导出数据文件，路径如下：选择目标 ECU->内存->内存配置->内存映像文件->导出按键。

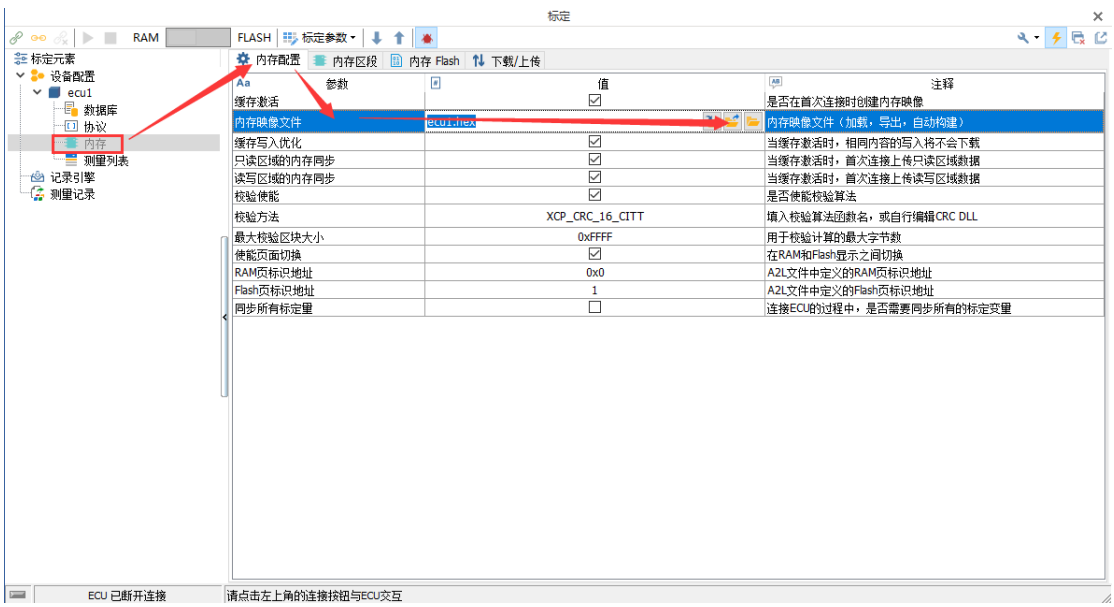


图4. 从现有的标定文件中导出

标定数据支持存储为 s19，hex，bin 等数据格式，点击数据导出按钮过后，导出数据格式选择如下所示：

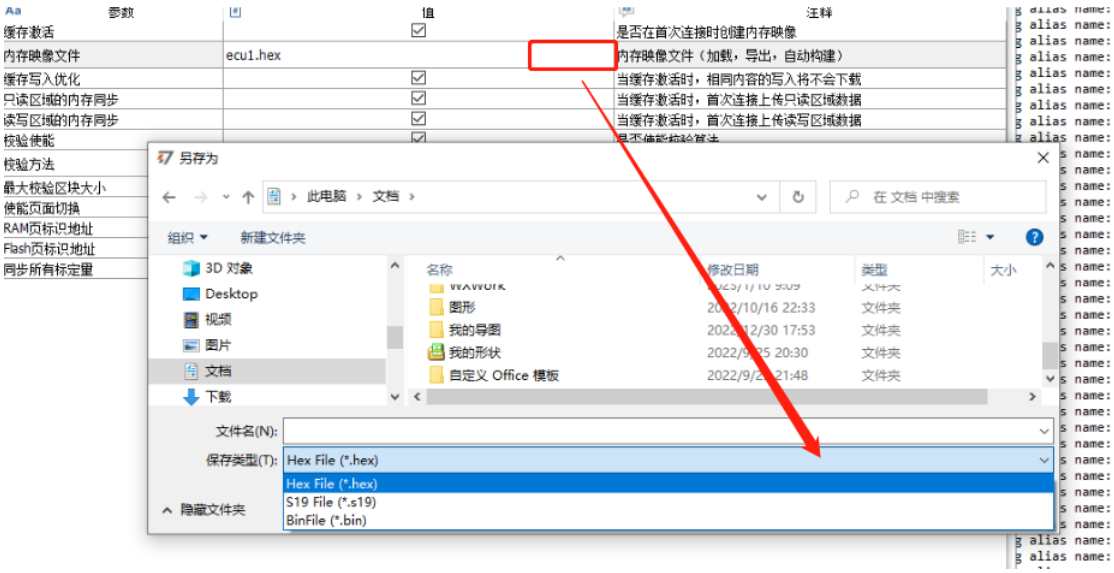


图5. 支持导出多种数据格式

1.21.9.4.2. 从 ECU 中读取并导出

从 ECU 中读取并导出数据文件之前，需要先完成 ECU 的连接。导出路径为：连接 ECU->内存->下载/上传->上传。

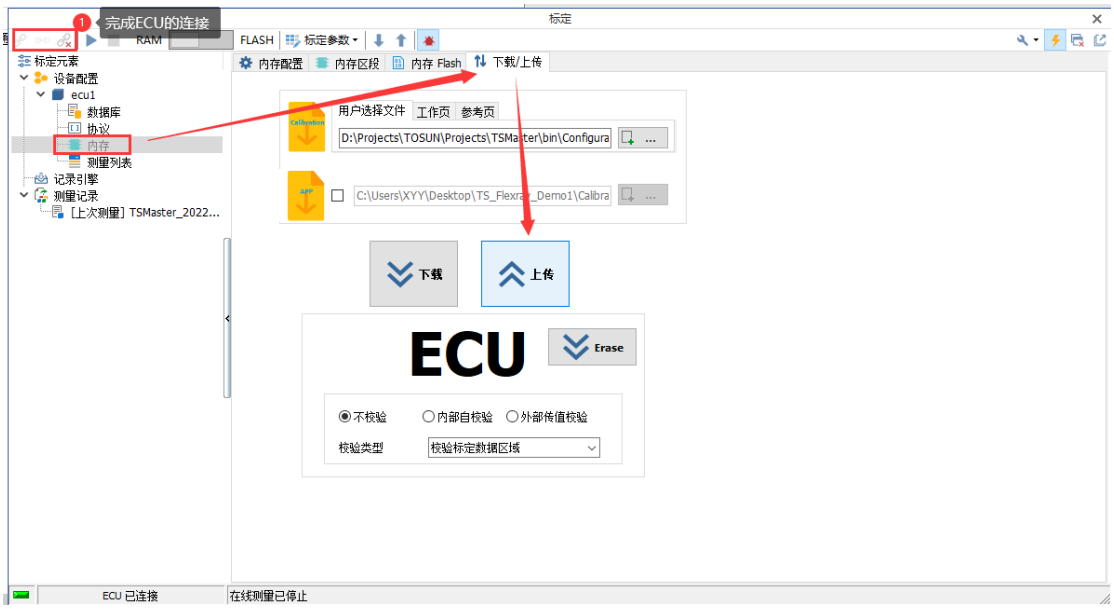


图6. 在线导出标定数据文件

1.21.9.5. 标定数据/应用程序的刷写

通过 XCP Program 协议（UDS 协议另行讲解）把标定好的数据文件下载并固化到目标 ECU 中，让标定数据生效。其操作路径如下：选中 ECU->内存->下载/上传->下载。

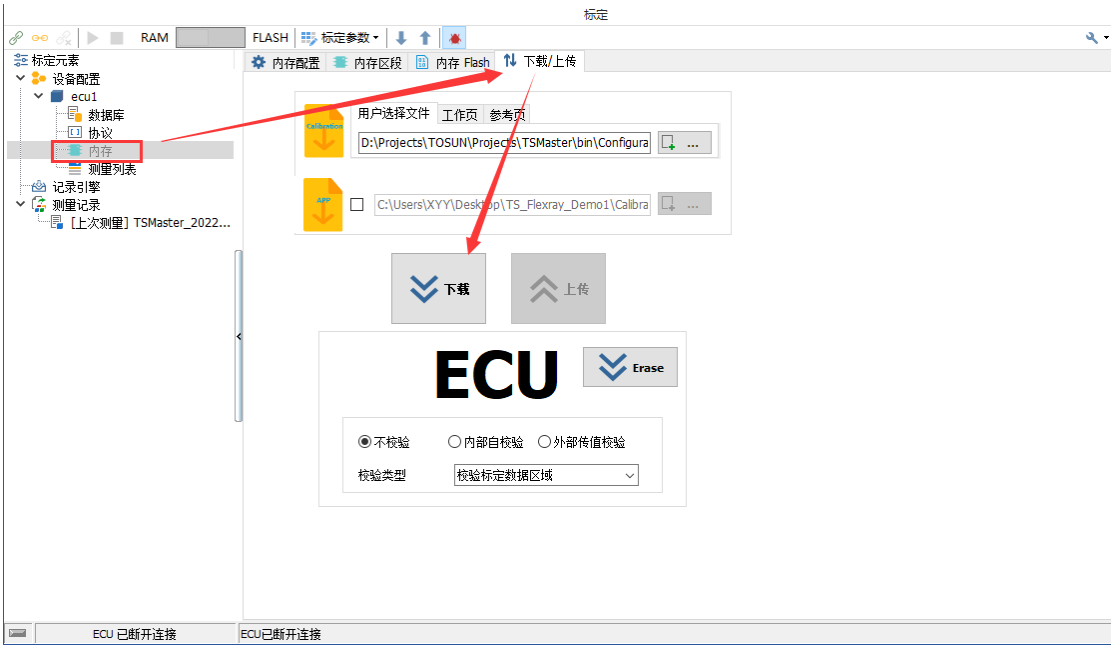


图7. XCP Program 下载路径

1.21.9.5.1. 基本配置

相关的配置主要包括选择标定文件，使能/选择应用程序文件，选择校验类型，如下所示：

图8. 下载配置

1.21.9.5.2. 是否选择应用程序数据

其中配置 2（使能应用程序文件），允许用户选择是否同时下载应用程序数据。正常情况下，ECU 的应用程序数据只有在发布新版本过后才需要重新载入，用户在完成标定数据的修改过后，只需要单独下载标定数据就可以，此时配置 2 选择不要载入应用程序数据，这样的话可以极大的减少重复刷写的数据量，节省刷写时间。

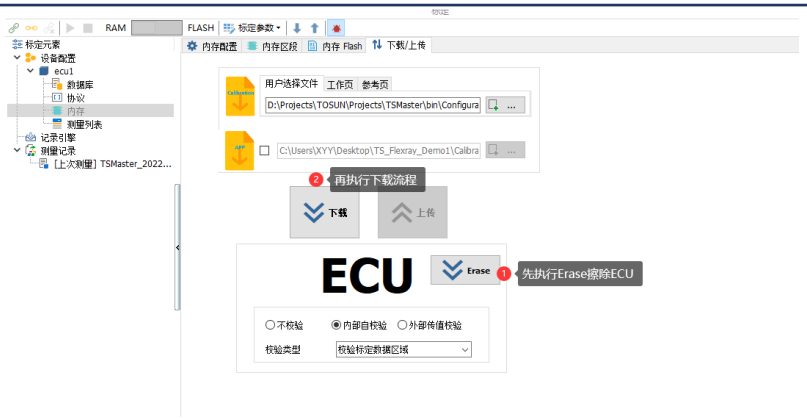
当开发人员发布了新版本的应用程序数据后，需要勾选上配置 2，并把应用程序数据加载到配置中，这样在下载的时候，通过 XCP 协议，可以把应用程序数据和标定数据同时下载到 ECU 中。

1.21.9.5.3. 下载速度区别

实测某华东主机厂的控制器，发现有如下区别：

- 当 ECU 中的 App 应用程序就绪的时候，此时的下载模式是非 Block 下载模式，也就是一问一答的下载方式；
- 当 ECU 中的 APP 被擦除掉的时候，此时的下载模式是 Block 下载模式。

这两种下载模式的速度差别可以达到 5-10 倍的差距。因此，TSMaster 专门提供了一个擦除模式，用于清除 ECU 中的内部 APP 程序。如下所示：



也就是说，在完成了下载参数的配置过后，推荐的下载方式是：

1. 先擦除 ECU 内部数据。
2. 再执行下载流程。

采用这种方式，刷写速度会远远快于直接执行下载流程，0x3C0000（2359296）个字节数据可以在 1 分钟之内完成下载。相关同事可以直接实测。

1.21.9.6. 常见错误情况

1.21.9.6.1. 下载过后 ECU 无法正常运行

- **现象描述：**

客户现场发现标定数据和应用数据下载到 ECU 过后，ECU 无法正常运行。通过比对报文，确认所有的数据都正确下载到了 ECU 正确的地址位置，但是 ECU 启动过后还是工作异常。

- **原因分析：**

经过排查，发现下载模块中没有勾选校验类型。该 ECU 的下载流程中，规定了下载数据过后，必须要进行内部校验，确认数据文件是正确的。如果没有内部校验，ECU 不敢贸然启动，相当于 ECU 内部的数据都是无效的。

- **解决办法：**

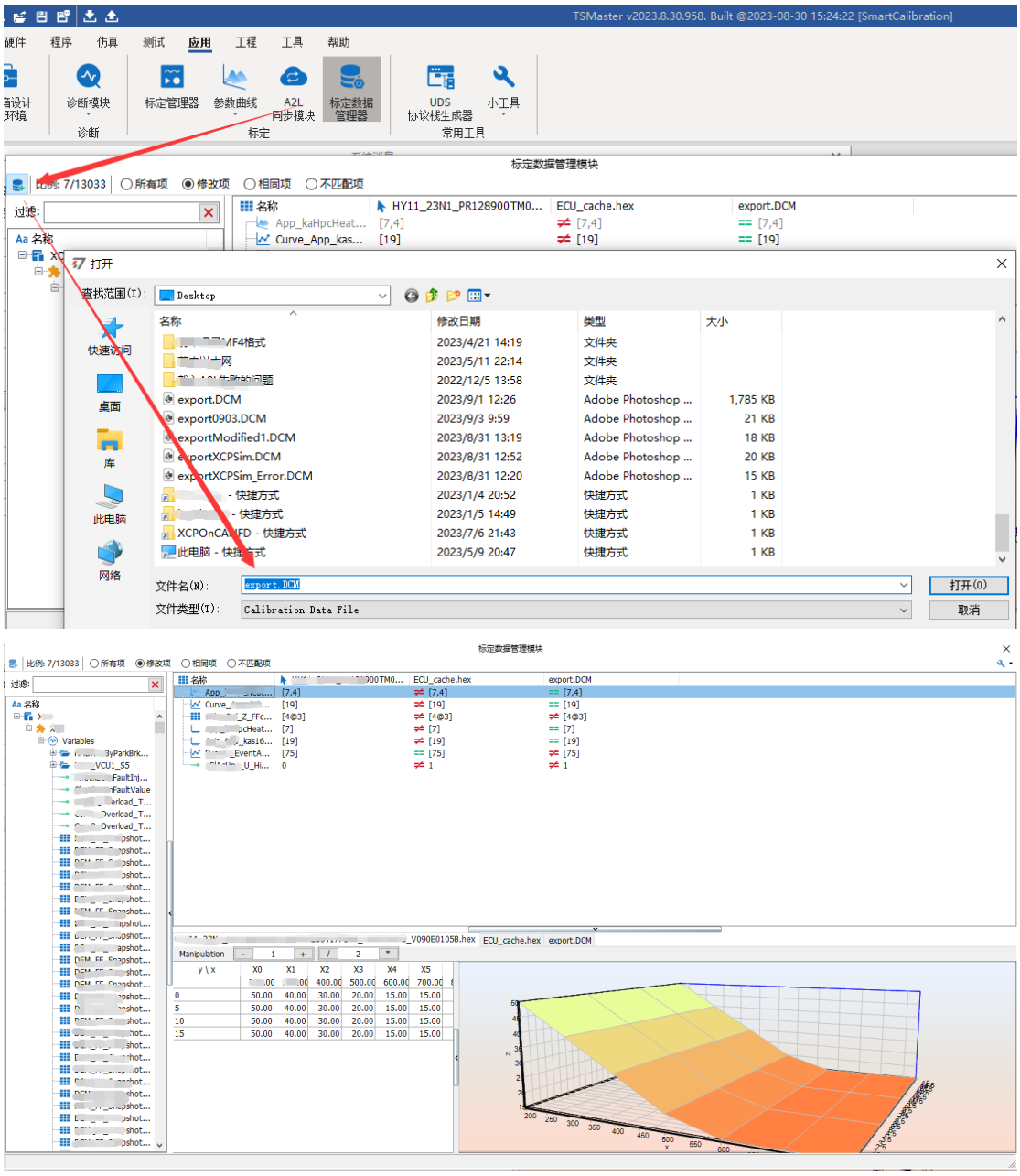
勾选 ECU 的校验选项。如下：

1.21.10. 标定数据管理器

TSMaster 标定模块中，为了管理标定过程数据，提供了专用的标定数据管理器（Calibration Data Manager Studio），用于标定人员完成标定数据管理，标定数据比对，快速修改，裁剪等功能。从大类上，主要包含以下的操作类型：

1.21.10.1. 支持加载多种数据类型。

包括 Hex，S19，bin 等数据文件，也包括 Par，DCM 等标定数据管理文件。导入窗口如下图所示：

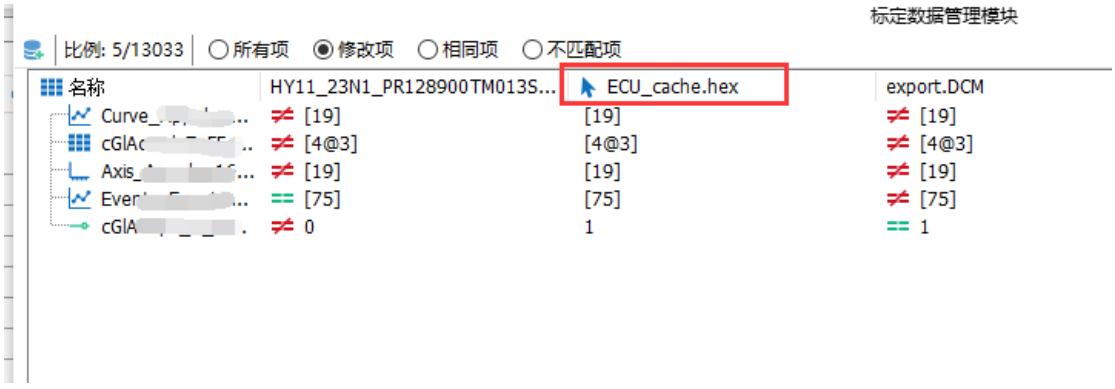


1.21.10.2. 多标定数据的分析比对。

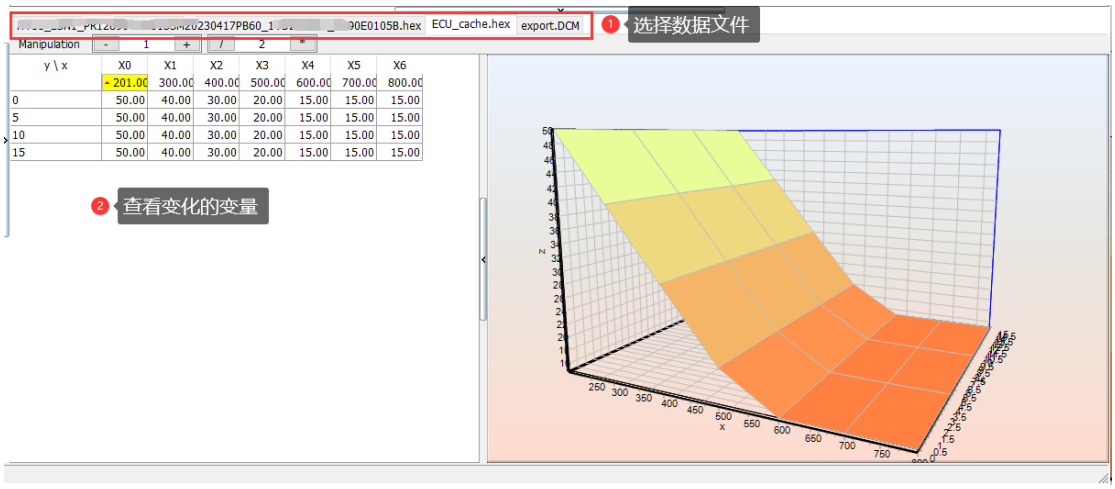
TSMaster 的 CDM 模块提供了过滤选择项目，主要包含：

- 所有项： 显示所有变量。
- 修改项： 只显示修改过后的变量
- 相同项： 只显示没有变化的变量
- 不匹配项：加载数据文件中出现了 A2L 中未定义的变量

用户可以选择制定的文件作为参考文件，切换参考文件过后，相应比较分析结果会及时刷新，如下图所示：

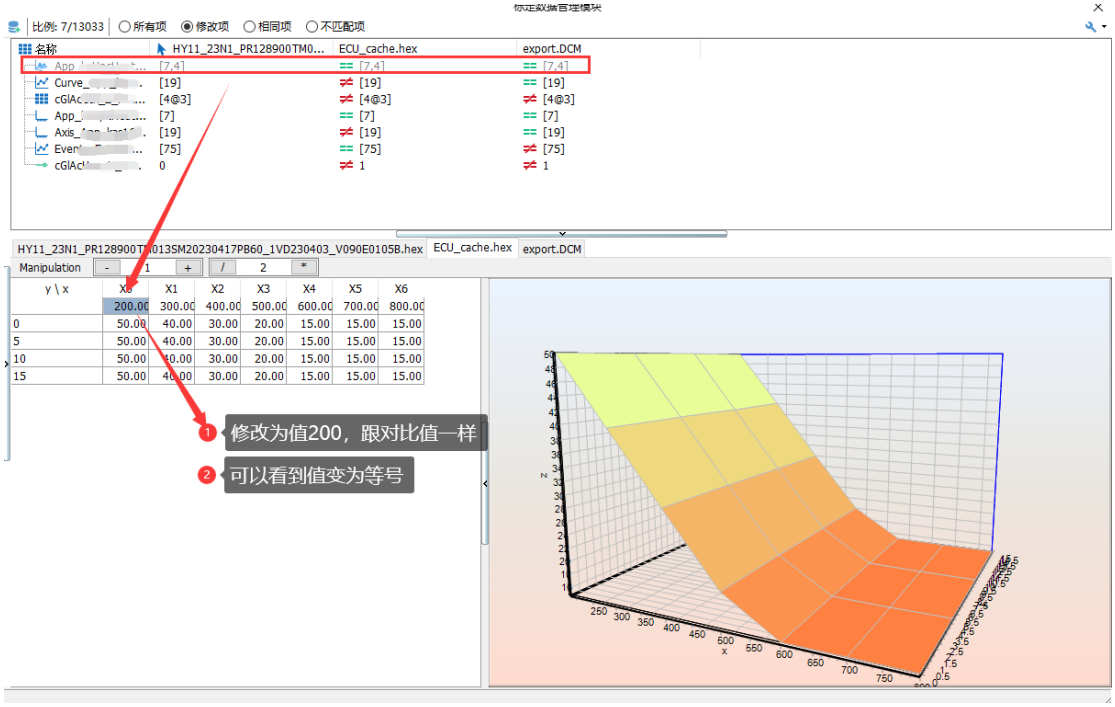


选中制定的数据量，可以查看该数据在不同数据源文件中的详细的变化情况。如下图所示：黄色背景的表示被修改的数据。



1.21.10.3. 离线修改标定参数

在 CDM 模块中，用户可以直接修改标定数据文件，其操作工程跟静态标定一样。在操作完成后，支持用户导出 hex/s19 等数据文件或者 cdm/par 等标定数据管理文件。



1.22. J2534

1.22.1. 临时安装说明

->以管理员身份运行 startup.bat

这会按照 J2534-1 的要求注册 WIN 系统注册表。dll 会被拷贝到系统文件夹，如果您不需要检测注册表，可以省略这个步骤。只要把文件夹中的所有 dll 文件都放到可以调用的位置（因编程语言而不同）即可。

1.22.2. 简介

提供的 dll 都是 32 位 dll，其中 J2534 的 dll 是用户调用的。dll 需要按照 WINAPI 形式进行调用。

支持的协议如下：ISO J2534-1 2004，ISO J2534-2 2019，GMW 17753。

支持的 J2534 版本为 **04.04**。

1.22.3. 函数说明

1.22.3.1. PassThruOpen

J2534 04.04 版本只支持同时使用一个 CAN 通道，默认 open 会连接随机同星设备的通道 1。调用 Open 函数时，软件即会与硬件进行连接，并有可能直接产生通讯。这是因为 TOSUN 的 API 连接之后才能拿到硬件的 handle。

如果您想要指定通道，可以像这样进行调用，如下调用会连接通道 2（但依然不能指定硬件）：

```
unsigned long deviceID;
PassThruOpen("J2534-2:TOSUN CH2", &deviceID);
```

1.22.3.2. PassThruClose

断开和硬件的连接。释放 dll 中的资源。

1.22.3.3. PassThruConnect

选择使用的协议，如果指定非 CAN FD 协议，会直接修改硬件的 CAN 波特率设置。如果指定 CAN FD 协议，CAN 波特率会被缓存。在调用 PassThruIoctl 设置数据场波特率时，才会真正被应用。

输入变量 Flags 不会被检测，硬件一定会同时支持 11 位 CAN ID 标准帧和 29 位 CAN ID 扩展帧。

可以支持的 protocol 如下：

CAN ， CAN_PS ， FD_CAN_PS ， CAN_FD_PS ， ISO15765 ， ISO15765_PS ， FD_ISO15765_PS，ISO15765_FD_PS。

需要说明的是，因为不支持 pin selection，所以带 PS 和不带 PS 的协议没有实际上的区别。

1.22.3.4. PassThruDisconnect

删除 connect 之后进行的所有设置。但不会断开或停止硬件（例如硬件依然会回复 ACK）。再次进行 connect 时，J2534 API 会对之间收到的报文进行清理，不会再读到。

1.22.3.5. PassThruReadMsgs

读取 J2534 报文。输入的 Timeout 参数不会被处理。

J2534 API 会维护一个接收缓存，报文在接收时会被放进缓存中，Read 从缓存中拿取数据。

支持的 RxStatus：

CAN_29BIT_ID：是否使用扩展帧。

TX_MSG_TYPE：是否是回读到的报文。

FD_CAN_BRS 和 CAN_FD_BRS：是否使用数据场比特率切换。

FD_CAN_FORMAT 和 CAN_FD_FORMAT：是否是 CAN FD 帧。

FD_CAN_ESI 和 CAN_FD_ESI：报文的 ESI 位是否置 1。

1.22.3.6. PassThruWriteMsgs

发送 J2534 报文。支持的类型都可以发送，不一定需要和 Connect 设置的 protocol 一致。Timeout 参数不会被处理。

如果您选择发送 ISO15765 的报文，但是 CAN ID 没有提前使用 PassThruStartMsgFilter 注册，函数会自动注册一个请求 ID 是本 ID，但是没有响应 CAN ID 的 TP 层地址映射。这用于触发功能寻址请求，但是如果不对过滤器额外进行配置，您将收不到任何响应报文。

支持的 TxFlag：

CAN_29BIT_ID：是否使用扩展帧。

FD_CAN_BRS 和 CAN_FD_BRS：是否使用数据场比特率切换。

FD_CAN_FORMAT 和 CAN_FD_FORMAT：是否是 CAN FD 帧。

1.22.3.7. PassThruStartPeriodicMsg

周期发送报文。最多支持 10 条周期报文。

CAN 和 CAN FD 报文，以及 ISO156765 报文中确定是单帧的报文，可以被周期发送。其它的报文无法周期发送。

1.22.3.8. PassThruStopPeriodicMsg

停止周期发送报文。

1.22.3.9. PassThruStartMsgFilter

设置过滤器。注意：按照 J2534 协议，**如果您不设置过滤器**，则无法收到任何一条报文。**API 将丢弃所有收到的报文**。

PASS_FILTER 和 BLOCK_FILTER 在任何情况下都可以设置；FLOW_CONTROL_FILTER 只能在 Connect 时选择 ISO15765 相关协议时才能设置。

PASS_FILTER 和 BLOCK_FILTER 加起来支持 10 个，FLOW_CONTROL_FILTER 支持 64 个。

对于 FLOW_CONTROL_FILTER，函数不会验证输入的 PASSTHRU_MSG，只要 data 大于 4，前 4 位都会按照 CAN ID 处理，后面的内容都不会处理。

GMW 17753 提供的，用于 29bit ID 的批量映射，API 没有支持。

1.22.3.10. PassThruStopMsgFilter

删除指定的过滤器。

1.22.3.11. PassThruSetProgrammingVoltage

不支持，函数一定会返回 ERR_NOT_SUPPORTED。

1.22.3.12. PassThruReadVersion

会拿到正确的 passthru 版本号。但是其它两个都无法获取到正确的答案。

1.22.3.13. PassThruGetLastError

获取最后一个错误的详细描述。如果属于 J2534 的错误，则会按照 J2534 进行说明；如果原本是 TOSUN API 返回的错误，则会拿到 TOSUN API 的错误描述。协议规定字符串长度为 80，超过的错误说明会被强行截断。

1.22.3.14. PassThruIoctl

支持的参数：
GET_CONFIG 和 SET_CONFIG：将在最后进行说明。
CLEAR_TX_BUFFER：不支持，但会直接返回成功。
CLEAR_RX_BUFFER：清空接收的缓存。
CLEAR_PERIODIC_MSGS：停止全部的周期报文发送。
CLEAR_MSG_FILTERS：删除所有的过滤器。
GET_DEVICE_INFO：获取 API 支持的协议。
GET_PROTOCOL_INFO：获取协议支持的具体内容。
WRITE_MSG_EXTENSION 和 READ_MSG_EXTENSION：是 GMW 17753 中规定的超过 PASSTHRU_MSG 字节的 CAN FD ISO15765 报文时的处理方案，都被支持。如果不是 GWM 17753 中规定的 CAN FD ISO15765，超过长度的报文会被丢弃。

支持的 CONFIG 参数：
DATA_RATE：修改仲裁段比特率。
LOOPBACK：是否支持回读。注意回读的报文不受过滤器的影响。
BIT_SAMPLE_POINT：不支持，但会返回成功。
SYNC_JUMP_WIDTH：不支持，但会返回成功。
ISO15765_BS：设置同星硬件回复的流控帧内容。
ISO15765_STMIN：设置同星硬件回复的流控帧内容。
ISO15765_BS_TX：不支持，但会返回成功。
ISO15765_STMIN_TX：不支持，但会返回成功。
ISO15765_WFT_MAX：支持。
CAN_MIXED_FORMAT：支持。API 支持同时收发 ISO15765 和对应协议的普通报文。
J1962_PINS：这个参数不会有任何动作，只会直接返回成功。即使没有设置 pin 脚，其它函数也不会因此报错。
FD_CAN_DATA_PHASE_RATE 和 CAN_FD_DATA_PHASE_RATE：J2534-2 和 GMW 17753 规定中都定义了这个参数，但值不同。因此只有对应的协议才能设置这些参数。用于更改 CAN FD 的数据场比特率。对于 CAN FD 的协议，只有在设置了这个参数之后，才

可以进入 CAN FD 的模式。

FD_ISO15765_TX_DATA_LENGTH 和 CAN_FD_TC_DATA_LENGTH：J2534-2 和 GMW 17753 规定中都定义了这个参数，但值不同。因此只有对应的协议才能设置这些参数。设置 CAN FD 模式的 ISO 15765 报文，单帧的最大 DLC。

HS_CAN_TERMINATION 和 CAN_FD_TERMINATION：J2534-2 和 GMW 17753 规定中都定义了这个参数，但值不同。因此只有对应的协议才能设置这些参数。设置是否使用同星硬件上的终端电阻。J2534-1 中规定的协议使用 J2534-2 中定义的参数值。

N_CR_MAX：J2534-2 和 GMW 17753 规定中都定义了这个参数，名称相同但值不同。目前不支持这个参数，但会直接返回成功。

ISO15765_PAD_VALUE：ISO15765 使用的填充位，都可以使用。

CAN_FD_TYPE：支持使用非 ISO 的 CAN FD。

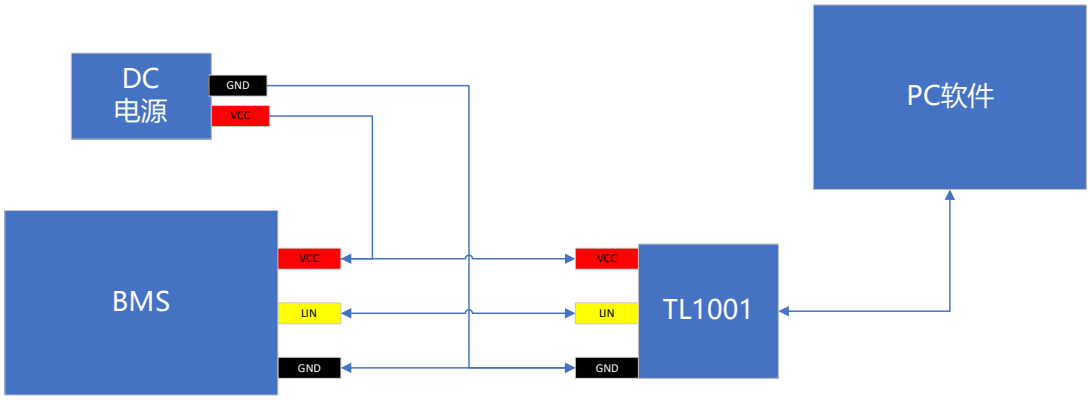
其余的参数函数都会返回不支持。

2. TOSUN Hardware

2.1. LIN 总线设备

2.1.1. TSLIN 接线图

2.1.1.1. 外部供电（开漏设计）：



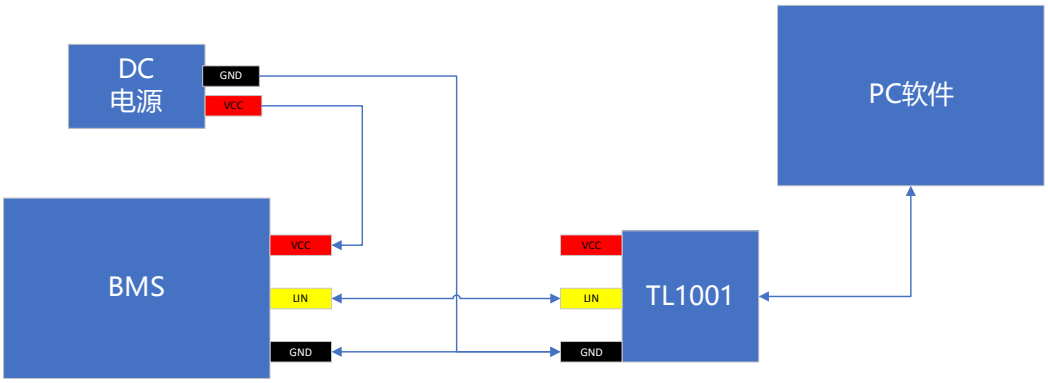
主节点模式下：
TL1001需要给LIN总线供电。从严谨的角度，为了保持LIN总线电平信号跟被测件一致，TL1001的供电采用开漏设计，VCC使用被测件一样的电源，确保LIN电平信号统一。

TL1001，TC1016，TC1012 等设备的电平信号采用外部供电，具有以下的优缺点：

- 优点：因为采用外部供电，所以 LIN 信号可以使用更宽的电平信号范围。比如乘用车的 LIN 信号电平是 12V，商用车的 LIN 信号电平是 24V。通过 VCC 外接不同的电压电平，可以支持不通过工作电压的 LIN 节点。
- 缺点：1. 需要外接电源，在实车等场合使用会不方便。2. 电源是外部供给，如果外部电源同时供给电机等类似负载，负载工作起来过后会对电压造成很大的干扰，从而也影响到 LIN 总线信号质量。

2.1.1.2. 内部供电，P 系列：

TL1001P作为主节点



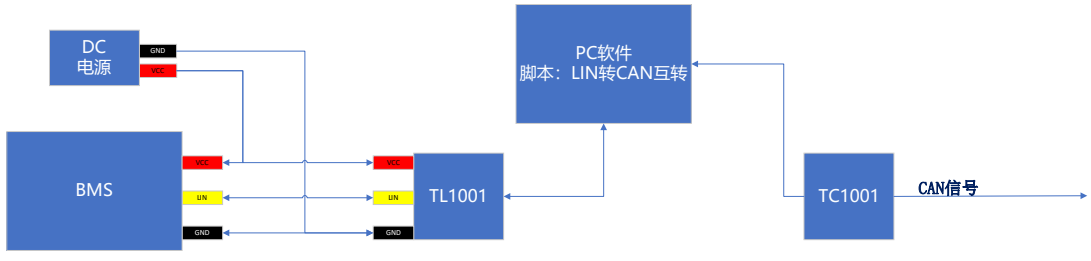
主节点模式下：
TL1001无需给LIN总线提供信号电平，因此可以选择悬空。

TL1001P，TC1016P，TC1012P 等设备的电平信号采用内部供电，固定提供 12V 的电平信号，和采用外部供电的设备对应的，具有以下的优缺点：

- 优点：1. 采用内部供电，只需要通过 USB 接上电脑就能工作，在实车等场合很方便。
  2. 电压又内部产生，不受外部测试 ECU 工作电压影响。
- 缺点：因为采用内部固定 12V 供电，如果碰到商用车 24V 工作电压和信号的场合，这类 LIN 设备就不能使用了。

2.1.1.3. 网关节点：

TL1001(LIN)和TC1001（CAN）转发



从节点模式下：
TL1001无需给LIN总线提供信号电平，因此可以选择悬空。

2.2. 常见错误

2.2.1. Send Break Failed

某些情况下，Trace 界面提示发生 Send Break Failed。如下所示：

这是因为 USB 设备供电不足造成 LIN 设备的信号隔离器，收发器等无法正常工作造成的。解决方案是：检查 USB 是否直接连接主板，如果通过 USBHub 等级联，很容易造成供电不足的情况。