诊断模块接口说明

一、意义

为了指导开发工程师，正确的使用诊断模块，快速开发出满足车厂要求的诊断功能。

二、诊断模块介绍

此诊断模块根据 ISO-14229-1 文档，并结合部分车厂的文档进行开发，使用面向对象的思路进行设计，将模块需要处理的所有事情封装在模块内部，留出模块处理过程接口和配置接口供调用接口的工程师使用。通过调用配置接口，可以配置我们想要的功能。通过调用处理过程接口，诊断模块便能提供诊断服务，无需其他操作，便能实现诊断功能，开发起来方便快捷。

使用过程中的复杂之处在于配置，需要根据具体项目的诊断需求进行具体配置。以下详细介绍。

三、快速入门(保证 CAN 驱动 OK)

1.CAN 报文接收完成，调用接口传入 UDS 模块，如下图：

```
}
Diagnostic_RxFrame(canId , canData , canIde , canDlc , canRtr);
```

2.UDS 配置参考 DiagnosticDemo.c-> Diagnostic_Init_Config()函数，如下图：

```
void Diagnostic_Init_Config(void)
{
    Diagnostic_Init(0x721, 0x728 , 0x7DF, 0xA00 , 1024 , SendFrame,0x0032,0x00C8);
    //********************************* service 10*************************************/
    //Diagnostic_Set2ndReqAndResID(0x18DA19F9, 0x18DAF919 , 0x18DBFFF9);

    InitSetSessionSupportAndSecurityAccess(TRUE,0x10,LEVEL_ZERO,LEVEL_ZERO,LEVEL_ZERO,LEVEL_Z
    InitSetSessionControlParam(TRUE , TRUE , TRUE , FALSE , FALSE , TRUE);
```

3.主循环调用 Diagnostic_Proc 函数，如下图：

```
while(1)
{
    Diagnostic_Proc();
}
```

4.一毫秒定时中断调动 Diagnostic_1msTimer，为 UDS 模块提供定时基准

```
    CPMUFLG_RTIF = 1; /*clear r
    Diagnostic_1msTimer();
```

5.确认实现 CAN 报文发送接口，注意次函数指针在步骤 2 初始化时候传入

```
byte SendFrame(VUINT32 ID, byte *array, byte length, byte priority , uint8_t rtr ,uint8_t ide)
{
```

6.实例配置代码

UDSDemo \ CODE \DiagnosticDemo.c

四、接口与配置说明

1、类型定义

1)、安全等级定义

```
typedef enum{
    LEVEL_ZERO = 7,//安全等级 0，当一个服务不需要安全解锁时，使用此安全等级。
    LEVEL_ONE = 1,//安全等级 1，当一个服务可以在安全等级 1 时，使用此安全等级。
    LEVEL_TWO = 2,//安全等级 2，当一个服务可以在安全等级 2 时，使用此安全等级。
    LEVEL_THREE = 4,//安全等级 3，当一个服务可以在安全等级 3 时，使用此安全等级。
    LEVEL_FOUR = 8,//安全等级 4，工厂模式会话使用此安全等级，用户零部件商下线配置。
    LEVEL_UNSUPPORT = 0,//不支持，当一个服务在某个会话模式不支持时，使用此等级。
}SecurityLevel;
```

2)、复位类型，参考 ISO-14229-1 中 11 服务复位类型的定义。

```
typedef enum{
    HARD_RESET = 1,//硬件复位
    KEY_OFF_ON_RESET = 2,//关开钥匙复位
    SOFT_RESET = 3,//软件复位
    ENABLE_RAPID_POWER_SHUTDOWN = 4,//预留，一般不使用
    DISABLE_RAPID_POWER_SHUTDOWN = 5,//预留，一般不使用
}EcuResetType;
```

3)、DTC 类型定义。

```
typedef enum{
    ISO15031_6DTCFORMAT= 1,
    ISO14229_1DTCFORMAT = 2,
    SAEJ1939_73DTCFORMAT = 3,
}DTCFormatIdentifier;
```

4)、诊断故障状态定义

```
typedef enum{
    PASSED,//测试通过
    IN_TESTING,//测试未完成
    FAILED,//测试失败
}DTCTestResult;
```

5)、DID 类型定义

```
typedef enum{
    EEPROM_DID,//静态存储器 DID，存储在 EEPROM 中的 DID 使用此类型
    REALTIME_DID,//实时 DID，存储在 RAM 中，会实时改变的数据使用此类型
    IO_DID,//输入输出控制 DID，需要通过 2F 服务控制的 DID 使用此类型
}DIDType;
```

6)、DID 的读写属性

```
typedef enum{
```

```
    READONLY = 1,//只读
    WRITEONLY = 2,//只写
    READWRITE = 3,//可读写
}ReadWriteAttr;
```

7)、通信控制参数

```
typedef enum{
    ERXTX,//enableRxAndTx
    ERXDTX,//enableRxAndDisableTx
    DRXETX,//disableRxAndEnableTx
    DRXTX,//disableRxAndTx
    //ERXDTXWEAI,//enableRxAndDisableTxWithEnhancedAddressInformation
    //ERXTXWEAI,//enableRxAndTxWithEnhancedAddressInformation
}CommulicationType;
```

8)、通信控制的控制对象参数

```
typedef enum{
    NCM = 1,//application message
    NWMCM,//network manage message
    NWMCM_NCM,//application and netwrok manage message
}communicationParam;
```

9)、子功能在会话的支持情况

```
typedef enum{
    SUB_DEFAULT = 1,//sub function supported in default session
    SUB_PROGRAM = 2,//sub function supported in program session
    SUB_EXTENDED = 4,////sub function supported in extedned session
    SUB_FACTORY = 8,//sub funcion supported in factory session,
    SUB_ALL = 7,//sub function supported in both of three session
}SubFunSuppInSession;
```

10)、诊断故障码的等级

```
typedef enum{
    LEVEL_A,
    LEVEL_B,
    LEVEL_C,
}DTCLevel;
```

11)、输入输出控制接口函数原型

```
typedef uint8_t (*IoControl)(uint8_t ctrl, uint8_t param);
```

Ctrl 表示控制类型:

0：归还控制权到 ECU

1：恢复默认状态

2：冻结当前

3：短时调整

只有当 ctrl 为 3 时存在 param，param 参数根据具体的 DID 而不同，如当控制开关时，可以表示为：

0：关

1：开，

如控制档位时
1:1 档
2:2 档
....

12）、安全解锁算法接口函数原型

typedef uint32_t (*SecurityFun)(uint32_t);
此原型表示一个函数指针，有一个 uint32_t 型的参数，表示种子。返回 uint32_t 型值，表示根据算法算出的秘钥

13）、DTC 的检测接口函数原型

typedef DTCTestResult (*DetectFun)(void);
无参数，需要返回 DTCTestResult 类型的值，2 表示测试失败，0 表示测试通过，1 表示正在测试（测试未完成）。

14）、复位接口函数原型。

typedef void (*ResetCallBack)(EcuResetType);
参数
EcuResetType：复位类型，取值范围，1-5，分别表示硬件复位，key-off-on 复位，软件复位。通常用 1 和 3。

15）、通信控制接口函数原型

typedef void (*CommCallBack)(CommulicationType , communicationParam);
参数：
CommulicationType ：参考 1.7 定义
communicationParam：参考 1.8 定义

16）、CAN 发送接口函数原型

typedef uint8_t (*SendCANFun)(uint32_t ID, uint8_t *array, uint8_t length, uint8_t priority);
注：以上所有接口函数原型供诊断开发时使用，开发时必须提供以上接口，否则诊断模块将无法正常工作。

```
#define        USE_MALLOC            0//1：使用动态内存分配，0：不使用动态内存分配。
#define        USE_J1939_DTC         0//建议不修改
```

当时不使用动态内存分配时候，存在以下参数，可调节。

```
/*======================== buf size config ================================*/
#define MAX_DTC_NUMBER                  35//最大 DTC 个数
#define MAX_DID_NUMBER                 70//最大 DID 个数
#define MAX_SNAPSHOT_NUMBER             10//最大快照信息个数
#define MAX_GROUP_NUMBER                5//最大 DTC 组个数
/*======================== buf size config ================================*/
```

2、接口函数

1）、诊断基本配置函数

void Diagnostic_Init(uint32_t requestId, uint32_t responseId, uint32_t funRequestId, uint16_t EEPromStartAddr, uint16_t EEpromSize,SendCANFun sendFun,uint16_t p2CanServerMax, uint16_t p2ECanServerMax);

requestId:诊断仪请求 ID（物理寻址）
responseId：ECU 响应 ID（物理寻址）
funRequestId：功能寻址请求 ID
EEPromStartAddr：诊断模块可使用的 EEPROM 起始
EEpromSize：诊断模块可使用的 EEPROM 的大小
sendFun：诊断模块发送 CAN 报文使用的函数指针
p2CanServerMax：诊断的响应时间参数限制（未发送 78 响应时，具体可参数项目诊断规范）
p2ECanServerMax：诊断的响应时间参数限制（发送了 78 响应后，具体可参数项目诊断规范）

2）、诊断额外支持的请求和响应 ID

void Diagnostic_Set2ndReqAndResID(uint32_t requestId1, uint32_t responseId1,uint32_t funRequestId1);
requestId1：诊断仪第二请求 ID（物理寻址）
responseId1：ECU 第二响应 ID（物理寻址）
funRequestId1：第二功能寻址请求 ID

3）、诊断模块释放接口

void Diagnostic_DelInit(void);
此接口会处理释放内存，保存故障码的操作，一定要在休眠之前调用。

4）、诊断模块报文接收函数

void Diagnostic_RxFrame(uint32_t ID,uint8_t* data,uint8_t IDE,uint8_t DLC,uint8_t RTR);
此函数需要在接收中断中调用，如果不调用,诊断模块将无法收到任何报文，无法提供任何服务。参数：
ID：报文 ID，可以是 11 位和 29 位 ID
Data：报文数据指针
IDE：参考 S12G 手册
DLC：报文长度
RTR：参考 S12G 手册

5）、诊断模块时间基数函数

void Diagnostic_1msTimer(void);
此函数需要在 1 毫秒的 RTI 中断中调用，如不调用，诊断模块所有与超时相关的功能将不能工作（包括多帧响应，S3 超时等）。

6）、添加安全算法的函数

bool InitAddSecurityAlgorithm(SecurityLevel level, SecurityFun AlgoritthmFun,byte SeedSubFunctionNum,byte KeySubFunctionNum , uint8_t* FaultCounter,uint8_t FaultLimitCounter , uint32_t UnlockFailedDelayTimeMS,SubFunSuppInSession SubFuntioncSupportedInSession,uint8_t KeySize);
此函数的功能是为诊断模块的添加安全算法，最多支持三个等级的安全算法，如果不添加安全算法，27 服务将没有正响应。参数：
Level：能使用 LEVEL_ONE，LEVEL_TWO，LEVEL_THREE，不能使用 LEVEL_ZERO 和 LEVEL_UNSUPPORT
AlgoritthmFun：安全解锁算法函数，参考三.1.12。
SeedSubFunctionNum：此算法支持的请求种子的子功能，如“27 01”中的“01”
KeySubFunctionNum ：此算法支持的发送秘钥的子功能，如“27 02”中的“02”
FaultCounter：预留参数，设置为 NULL
FaultLimitCounter：解锁失败次数限制，超时此次数时，启用延时

UnlockFailedDelayTimeMS：解锁失败后延时时间，单位为毫秒，如 3000 表示 3 秒
SubFuntioncSupportedInSession：子功能在会话模式的支持情况，可以是 SUB_PROGRAM ，SUB_EXTENDED，也可以都支持，使用按位或的方式 SUB_PROGRAM | SUB_EXTENDED。
KeySize：seed 和可以的长度，可以设置为 2 或者 4。设置为 2 时只使用高生成种子的高两个字节，解锁算法生成的秘钥也需要放到高两个字节。设置为 4 时将使用所有字节。

7)、初始化工厂模式安全算法函数

void InitFactorySecuriyAlgorithm(void);
无参数，此函数内部会调用 InitAddSecurityAlgorithm 函数，添加安全算法，算法包含于内部，如不进行此初始化，工厂模式将无法解锁。

8)、配置服务的函数

bool InitSetSessionSupportAndSecurityAccess(bool support ,uint8_t service,uint8_t PHYDefaultSession_Security, uint8_t PHYProgramSeesion_Security, uint8_t PHYExtendedSession_Security, uint8_t FUNDefaultSession_Security, uint8_t FUNProgramSeesion_Security, uint8_t FUNExtendedSession_Security);
Support：只能为 TRUE，如果为 FALSE 和未配置一样会有 11 否定响应。
Service：服务名称，如 0x10，0x11，0x27 等（一次只能使用一个）
PHYDefaultSession_Security：服务在物理寻址默认会话支持的安全访问等级。
PHYProgramSeesion_Security：服务在物理寻址编程会话支持的安全访问等级。
PHYExtendedSession_Security：服务在物理寻址扩展会话支持的安全访问等级。
FUNDefaultSession_Security,：服务在功能寻址默认会话支持的安全访问等级。
FUNProgramSeesion_Security：服务在功能寻址编程会话支持的安全访问等级。
FUNExtendedSession_Security：服务在功能寻址扩展会话支持的安全访问等级。
注意：以上 6 个参数，
如果支持，不需要安全解锁，则使用 LEVEL_ZERO,
如果不支持，则使用 LEVEL_UNSUPPORT，
如果需要安全解锁等级 1 才能支持，则使用LEVEL_ONE，
如果需要安全解锁等级 2 才能支持，则使用 LEVEL_TWO，
如果需要安全解锁等级 3 才能支持，则使用 LEVEL_THREE，
如果同时支持多个安全等级，则只用按位或的方式，如 LEVEL_TWO|LEVEL_THREE

9)、添加 DID 的接口函数

void InitAddDID(uint16_t DID , uint8_t DataLength , uint8_t* DataPointer , DIDType DidType , IoControl ControlFun , ReadWriteAttr RWAttr,uint16_t EEaddr, bool SupportWriteInFactoryMode);
DID：DID 数字，如：0xF190
DataLength：DID 的数据长度，如 F190 为 17。
DataPointer：DID 数据指针，此指针由应用程序提供，当类型为 EEPROM_DID 时，此参数设为 NULL，类型为 IO_DID 并且不需要读时，也可设置为 NULL。
DidType:DID 的类型可以是 EEPROM_DID,REALTIME_DID,IO_DID。
ControlFun：输入输出控制的函数指针，当类型不为 IO_DID 时，此参数设置为 NULL。
RWAttr：读写属性
EEaddr：DID 的 eeprom 地址只有 DidType 为 EEPROM_DID 时有效，当此参数为 0 时，诊断模块将自动分配 eeprom 地址，因此如果不需要手动指定地址，将此值设置为 0 即可。
SupportWriteInFactoryMode:是否支持在工厂模式可写。
注意：工厂模式的会话模式为 0x71，需要先切换到 10 03 扩展会话，才能切换到工厂模式会

话，工厂模式写 DID 数据需要先 27 解锁，分别是 27　71 请求种子，27　72 发送秘钥。工厂模式解锁算法包含在诊断模块内部，对客户不可见。

10）、添加故障码的接口函数

```
#if USE_J1939_DTC
void Diagnostic_DM1MsgEnable(bool dm1en);
bool InitAddDTC(uint32_t DTCCode,DetectFun MonitorFun,byte DectecPeroid, byte ValidTimes,DTCLevel dtcLevel,uint32_t spn, uint8_t fmi);
#else
bool InitAddDTC(uint32_t DTCCode,DetectFun MonitorFun,byte DectecPeroid, byte ValidTimes,DTCLevel dtcLevel);
#endif
```

灰色部分可以不作关注

DTCCode：诊断故障代码，如 0x910223，诊断模块只使用低 24 位，高 8 位设置为零。

MonitorFun：故障检测函数指针。

DectecPeroid：故障检测周期，此参数暂未使用，可以设置为 0.

ValidTimes：故障有效次数，记录历史故障码的故障出现次数，当在历史故障和当前故障码同时置位时，设置为 1，当历史故障码需要多个点火循环才能置位时，可设置为大于等于 2 的数。2 表示需要两个点火周期，3 表示 3 个，类推。

dtcLevel：故障等级，可以设置为 LEVEL_C

11）、添加快照信息接口函数

```
void InitAddDTCSnapShot(uint8_t recordNumber , uint16_t ID , uint8_t* datap , uint8_t size);
```

recordNumber：快照信息记录号，如 1，表示全局快照，2，表示局部快照。

ID：此快照的 ID，如 0x9102 表示快照车速信息。

Datap：此快照记录的内存指针，需要是能表示实时状态（如实时车速）的内存指针。

Size：此快照的大小，字节数。

12）、设置故障扩展信息-老化计数器的扩展信息号的接口函数

```
void InitSetAgingCounterRecordNumber(uint8_t RecordNumer);
```

RecordNumer：老化计数器信息的序号（需要参考诊断规范中 19 06 的响应信息，一般范围 1-4）

13）、设置故障扩展信息-已老去计数器的扩展信息号的接口函数

```
void InitSetAgedCounterRecordNumber(uint8_t RecordNumer);
```

RecordNumer：已老去计数器信息的序号（需要参考诊断规范中 19 06 的响应信息，一般范围 1-4）

14）、设置故障扩展信息-故障发生次数计数器的扩展信息号的接口函数

```
void InitSetOccurrenceCounterRecordNumber(uint8_t RecordNumer);
```

RecordNumer：故障发生次数计数器信息的序号（需要参考诊断规范中 19 06 的响应信息，一般范围 1-4）

15）、设置故障扩展信息-故障待定计数器的扩展信息号的接口函数

```
void InitSetPendingCounterRecordNumber(uint8_t RecordNumer);
```

RecordNumer：故障待定计数器信息的序号（需要参考诊断规范中 19 06 的响应信息，一般范围 1-4）

16）、设置支持的故障位的接口函数

```
void InitSetDTCAvailiableMask(uint8_t AvailiableMask);
```

AvailiableMask：故障位，如 0x09 表示支持当前位和历史位

17）、设置 DTCgroup 的接口函数

void InitAddDTCGroup(uint32_t Group);

Group：14 服务的 group，目前支持支 0xFFFFFF（仅低 24 位有效），清除所有故障码。

18）、配置 11 服务的接口函数

void InitSetSysResetParam(bool support01 , bool support02 , bool support03 , bool support04 , bool support05 , ResetCallBack callback, bool supressPosResponse);

support01:11 服务是否支持 01 子功能，TRUE:支持，FALSE：不支持

support02:11 服务是否支持 02 子功能，TRUE:支持，FALSE：不支持

support03:11 服务是否支持 03 子功能，TRUE:支持，FALSE：不支持

support04:11 服务是否支持 04 子功能，TRUE:支持，FALSE：不支持

support05:11 服务是否支持 05 子功能，TRUE:支持，FALSE：不支持

Callback:复位接口函数指针，由应用提供，诊断模块只调用，具体的复位动作需要应用根据参数执行。

supressPosResponse:是否支持抑制响应，TRUE：支持，FALSE：不支持

19）、配置 28 服务的接口函数

void InitSetCommControlParam(bool supportSubFun00, bool supportSubFun01 , bool supportSubFun02 , bool supportSubFun03 , bool supportType01, bool supportType02, bool supportType03, CommCallBack callback, bool supressPosResponse);

supportSubFun00：是否支持 00 子功能-使能接收和发送，TRUE：支持，FALSE：不支持

supportSubFun01：是否支持 01 子功能-使能接收关闭发送，TRUE：支持，FALSE：不支持

supportSubFun02：是否支持 02 子功能-关闭接收使能发送，TRUE：支持，FALSE：不支持

supportSubFun03：是否支持 03 子功能-关闭接收和发送，TRUE：支持，FALSE：不支持

supportType01：是否支持控制参数 01-一般通信报文，TRUE：支持，FALSE：不支持

supportType02：是否支持控制参数 02-网络管理报文，TRUE：支持，FALSE：不支持

supportType03：是否支持控制参数 03-通信报文和网络管理报文，TRUE：支持，FALSE：不支持

Callback：通信控制接口函数指针，由应用提供，诊断模式只负责调用，控制逻辑由应用实现。

supressPosResponse：是否支持抑制响应，TRUE：支持，FALSE：不支持

20）、配置 10 服务的接口函数

void InitSetSessionControlParam(bool supportSub01, bool supportSub02,bool supportSub03, bool sub02SupportedInDefaultSession, bool sub03SupportedInProgramSession, bool supressPosResponse);

supportSub01：是否支持 01 子功能-默认会话，TRUE：支持，FALSE：不支持

supportSub02：是否支持 02 子功能-编程会话，TRUE：支持，FALSE：不支持

supportSub03：是否支持 03 子功能-拓展会话，TRUE：支持，FALSE：不支持

sub02SupportedInDefaultSession：在默认会话是否支持 02 子功能，TRUE：支持，FALSE：不支持

sub03SupportedInProgramSession：在编程会话是否支持 03 子功能，TRUE：支持，FALSE：不支持

supressPosResponse：是否支持抑制响应，TRUE：支持，FALSE：不支持

21）、配置 3E 服务的接口函数

void InitSetTesterPresentSupress(bool supressPosResponse);

supressPosResponse：是否支持抑制响应，TRUE：支持，FALSE：不支持

22）、配置 85 服务的接口函数

void InitSetDTCControlSupress(bool supressPosResponse);

supressPosResponse：是否支持抑制响应，TRUE：支持，FALSE：不支持

23）、配置当前会话模式 DID 的接口函数

void InitSetCurrentSessionDID(uint16_t m_DID);

由于此数据在诊断模块，应用无法得到，所以使用此接口即可。此函数内部会添加 DID。

24）、配置 CAN 数据库 DID 的接口函数

void InitSetCanDataBaseVersionDID(uint16_t m_DID);

由于此数据在诊断模块，应用无法得到，所以使用此接口即可。此函数内部会添加 DID。

25）、配置 CAN 诊断版本 DID 的接口函数

void InitSetCanDiagnosticVersionDID(uint16_t m_DID);

由于此数据在诊断模块，应用无法得到，所以使用此接口即可。此函数内部会添加 DID。

26）、配置网络管理版本 DID 的接口函数

void InitSetCanNMVersionDID(uint16_t m_DID);

由于此数据在诊断模块，应用无法得到，所以使用此接口即可。此函数内部会添加 DID。

27）、配置 CAN 驱动版本 DID 的接口函数

void InitSetCanDriverVersionDID(uint16_t m_DID);

由于此数据在诊断模块，应用无法得到，所以使用此接口即可。此函数内部会添加 DID。

28）、加载所有诊断模块数据的接口函数

void Diagnostic_LoadAllData(void);

需要先 配置好 DID，安全算法，DTC 后才能调用此接口函数，此接口函数回从 EEPROM 中读取所有需要的数据。

29）、配置车架号的接口函数

void Diagnostic_ConfigVIN(uint8_t length, uint8_t* data);

/************set netwrok layer parameters********/

30）、设置网络层参数的接口函数

void Diagnostic_SetNLParam(uint8_t TimeAs, uint8_t TimeBs, uint8_t TimeCr, uint8_t TimeAr, uint8_t TimeBr, uint8_t TimeCs, uint8_t BlockSize, uint8_t m_STmin, uint8_t FillData);

TimeAs：网络层定时参数 AS

TimeBs：网络层定时参数 BS

TimeCr：网络层定时参数 CR

TimeAr：网络层定时参数 AR

TimeBr：网络层定时参数 BR

TimeCs：网络层定时参数 CS

BlockSize：网络层参数 BloskSieze（BS）

STmin：网络层定时参数 STmin

FillData：未使用字节的填充数据

31）、诊断处理过程的接口函数

void Diagnostic_Proc(void);

此函数时最终实现诊断功能的函数，需要放到主循环不停的调用，如有需要，可以设置定时调用，最大定时为 1MS。