## Muc\_uds-master 协议栈使用说明

**文件说明：**

1. Network\_layer.c , network\_layer.h, network\_layer\_private.h 三个文件是根据IS015765-2网络层的协议实现
2. Usd\_util.c,usd\_util.h, usd\_type.h,uds\_support.c,uds\_support.h,uds\_status.c,uds\_status.h,uds\_service.c,usd\_service.h 等这些文件是根据ISO14229网络诊断协议的实现
3. Obd\_dtc.c,obd\_dtc.h,obd\_dtc\_private.h等是关于obd，及UDS中与obd故障诊断相关服务的描述及功能支持，当然具体的故障判定结果还需设备自检测，然后将结果写回到诊断码中，供诊断仪或其他设备进行读取及使用。

**需要的硬件资源支持：**

* 1. Can 收发器、控制器
  2. 一个能配置成1ms的中断定时器

**接口使用：**

* 1. Can 相关的硬件接口函数

发送函数：ZTai\_UDS\_Send (send\_buf, UDS\_VALID\_FRAME\_LEN);

接收函数(中断中)： uds\_get\_frame (uint8\_t func\_addr, uint8\_t frame\_buf[], uint8\_t frame\_dlc)

使用此软件进需对这第一个函数进行硬件实现，将第二个函数在can接收中断中调用。

* 1. 软件栈使用接口

初始化函数：uds\_init (void)

1ms定时中断中运行函数：uds\_main (void)

其中uds\_init在主函数中运行一次即可，而uds\_main需在定时器中运行，定时器的中断优先及要小于can接收中断的优先级。同时MCU应当支持中断嵌套。

3. DTC使用接口

DTC 结构存储static obd\_dtc\_data\_t obd\_dtc\_data[OBD\_DTC\_CNT];

DTC 标志数值 static obd\_dtc\_para\_t obd\_dtc\_para[OBD\_DTC\_CNT]=

{

0x04621A,

0x04631B,

0xF00316,

0xF00317,

0xC07388,

0xC10001,

0xC10101,

0xC12101,

0xC12701,

0xC12801,

0xC12901,

0xC13101,

0xC14100,

0xC15101,

0xC16300,

0xC16900,

0xC18101,

0xC23601,

0xC16400,

0xC16700,

0x953D01,

0x953E01,

0x953F01,

0x954001,

0x954101,

0x954401,

0x954501,

0x954601,

0x954701

};

DTC 所有诊断 typedef enum \_\_OBD\_DTC\_NAME\_T\_\_

{

DTC\_FUEL\_SENSOR\_BELOW = 0,

DTC\_FUEL\_SENSOR\_ABOVE,

DTC\_BATT\_VOLTAG\_BELOW,

DTC\_BATT\_VOLTAG\_ABOVE,

DTC\_CAN\_BUS\_OFF,

DTC\_COM\_LOST\_EMS,

DTC\_COM\_LOST\_TCU,

DTC\_COM\_LOST\_ABS,

DTC\_COM\_LOST\_TPMS,

DTC\_COM\_LOST\_EPB,

DTC\_COM\_LOST\_ESP,

DTC\_COM\_LOST\_EPS,

DTC\_COM\_LOST\_BCM,

DTC\_COM\_LOST\_SRS,

DTC\_COM\_LOST\_DVD,

DTC\_COM\_LOST\_PDC,

DTC\_COM\_LOST\_ALS,

DTC\_COM\_LOST\_PEPS,

DTC\_COM\_LOST\_AC,

DTC\_COM\_LOST\_AVM,

DTC\_LIGHT\_ERROR\_BREAK,

DTC\_LIGHT\_ERROR\_EPC,

DTC\_LIGHT\_ERROR\_TCU,

DTC\_LIGHT\_ERROR\_AIR\_BAG,

DTC\_LIGHT\_ERROR\_ESC,

DTC\_LIGHT\_ERROR\_ABS,

DTC\_LIGHT\_ERROR\_MIL,

DTC\_LIGHT\_ERROR\_EPB\_FAULT,

DTC\_LIGHT\_ERROR\_ESC\_OFF,

OBD\_DTC\_CNT

}obd\_dtc\_name\_t;

所有的DTC码都应先在obd\_dtc\_name\_t 中写上其测试名称，然后再obd\_dtc\_para中添加其标记值。

DTC诊断开始：obd\_dtc\_start\_opcycle (uint16\_t dtc\_n)

系统如果要启用这些故障诊断功能，则应在初始化时将所有的项均先打开。

DTC 诊断结果更新：uds\_update\_obddtc (uint16\_t dtc\_n, obd\_dtc\_test\_t test\_result)

若某时刻对某个故障进行了诊断，其结果通过此接口写入到故障码变量中，供诊断仪测试及读取。

**程序运行过程**

1. 初始化后，程序配置好回调函数

usdata.ffindication = uds\_dataff\_indication;

usdata.indication = uds\_data\_indication;

usdata.confirm = uds\_data\_confirm;

return network\_reg\_usdata (usdata);

1. 在1ms中断函数中，对网络层，uds层通信进行监测，主要是是否超时及超时后的处理策略。

uds\_main (void)

{

    network\_main ();

    if (uds\_timer\_run (UDS\_TIMER\_S3server) < 0)

   {

        uds\_session = UDS\_SESSION\_STD;

        curr\_sa = UDS\_SA\_NON;

        uds\_ioctrl\_allstop ();

    }

   if (uds\_timer\_run (UDS\_TIMER\_FSA) < 0)

    {

uds\_fsa\_cnt = 0;

   }

}

1. 在can接收中断中进行数据的处理服务的支持

uds\_get\_frame (uint8\_t func\_addr, uint8\_t frame\_buf[], uint8\_t frame\_dlc)

{

    network\_recv\_frame (func\_addr, frame\_buf, frame\_dlc);

}

network\_recv\_frame (uint8\_t func\_addr, uint8\_t frame\_buf[], uint8\_t frame\_dlc)

{

uint8\_t err;

uint8\_t pci\_type; /\* protocol control information type \*/

/\*\*

\* The reception of a CAN frame with a DLC value

\* smaller than expected shall be ignored by the

\* network layer without any further action

\*/

if(frame\_dlc != UDS\_VALID\_FRAME\_LEN) return;

if (func\_addr == 0)

g\_tatype = N\_TATYPE\_PHYSICAL;

else

g\_tatype = N\_TATYPE\_FUNCTIONAL;

// OSMutexPend(UdsMutex,0,&err);

pci\_type = NT\_GET\_PCI\_TYPE (frame\_buf[0]);

switch(pci\_type)

{

case PCI\_SF:

if (nwl\_st == NWL\_RECV || nwl\_st == NWL\_IDLE)

{

clear\_network ();

if (nwl\_st == NWL\_RECV)

N\_USData.indication (recv\_buf, recv\_len, N\_UNEXP\_PDU);

recv\_singleframe (frame\_buf, frame\_dlc);

}

break;

case PCI\_FF:

if (nwl\_st == NWL\_RECV || nwl\_st == NWL\_IDLE)

{

clear\_network ();

if (nwl\_st == NWL\_RECV)

N\_USData.indication (recv\_buf, recv\_len, N\_UNEXP\_PDU);

if (recv\_firstframe (frame\_buf, frame\_dlc) > 0)

nwl\_st = NWL\_RECV;

else

nwl\_st = NWL\_IDLE;

}

break;

case PCI\_CF:

if (nwl\_st == NWL\_RECV && g\_wait\_cf == TRUE)

{

if (recv\_consecutiveframe (frame\_buf, frame\_dlc) <= 0)

{

clear\_network ();

nwl\_st = NWL\_IDLE;

}

}

break;

case PCI\_FC:

if (nwl\_st == NWL\_XMIT && g\_wait\_fc == TRUE)

if (recv\_flowcontrolframe (frame\_buf, frame\_dlc) < 0)

{

clear\_network ();

nwl\_st = NWL\_IDLE;

}

break;

default:

break;

}

}

**其他说明：**

1. Uds服务中如果涉及到相应硬件相关的部分，需要在相应的回调函数中进行相关功能的添加。
2. Uds服务中并不包括网络管理功能，如果要支持autoSAR的网络管理功能，需在中断中添加相关代码支持网络三种状态的变化功能（包括唤醒与周期性发送数据及休眠）。这部分与UDS唯一共用的就是can接收的数据。
3. 第三方需要添加的数据发用及查询写入需根据14229协议分析是否属于某个服务，如果是则在相应的回调函数中进行相应功能的开发，如果不是则需在中断中添加相关代码，此时这部分代码与诊断是分离的，唯一共用的就是can接收的数据。
4. 文件中与操作系统相关的代码在应用是需去掉。

总结： 整个UDS协议对CAN网络及其诊断的各个方面进行了定义，具体分为以下五个方面的服务。

1. 对会话和诊断管理服务
2. 对数据传输管理服务
3. 对故障码存取管理服务
4. 对IO控制服务
5. 对程序更新下载管理务

总的来的一款车载产品只要具备了UDS + 网络管理功能，便能做为一个车载设备接入到CAN网络系统中。

其中UDS + 网络管理功能 是与具体产品所要实现的功能不相关的，只不过需要其提供一些网络诊断需要的数据及状态标识，具体的则由车厂出的诊断矩阵来决定，这些都是协议上的东西。

注意： 禁止在中断中使用延时函数，会影响系统的稳定性。