Technical Note

CANFD 网络设备通讯协议

CANFD 网络设备系列产品

TN01010101 1.13 Data:2022/06/30

类别	内容
关键词	CANFDNET通讯协议
摘要	CANFD报文传输通讯协议描述

修订历史

版本	日期	原因
V1.00	2019/07/24	创建文档
V1.01	2019/08/23	1.认证数据包分为请求和应答包 2.增加总线错误说明
V1.02	2019/08/26	1.定时发送实时更新/启动支持
V1.03	2019/11/22	1.更新报文格式，时间戳单位修改为 us，大小为 8 字节 2.更新报文格式，增加发送回显、发送标志 3.增加总线负载上报格式定义
V1.04	2020/04/02	1.增加报文压缩传输定义 2.增加扩展包格式定义。
V1.06	2020/07/06	1.增加 GPS 数据包 2.更新 CAN 报文错误帧定义 3.增加报文发送后延时操作
V1.07	2020/07/09	1.更新 GPS 数据包格式 2.增加请求应答包定义 3.更新文档格式 4.增加设备状态包
V1.08	2020/07/29	1.增加清空发送延时队列操作 2.更新获取定时发送列表操作
V1.09	2020/9/10	1.队列定时发送时间单位改为 100us.
V1.10	2020/11/5	1.获取设备信息增加协议版本字段 2.定时发送报文最大周期由 60s 调整到 600s，支持 100us 定时 3.获取定时发送列表增加周期单位字段
V1.11	2021/01/28	1.增加 LIN 数据包格式 2.增加动态配置请求 3.增加清空 LIN 应答数据请求 4.增加获取 LIN 发送 FIFO 状态请求
V1.12	2021/08/09	1.增加配置数据包
V1.13	2021/8/27	1.更新文档说明
V1.14	2021/12/20	1.请求应答包添加动态配置和持久化配置(json/数据流)请求
V1.15	2022/1/21	1.修正参考代码描述错误
V1.16	2022/6/30	1.添加发送报文应答返回 JSON 信息增加剩余队列空间大小参数
V1.17	2022/10/26	1.添加 uds 诊断请求应答包
V1.18	2022/11/22	1.uds 诊断状态/控制返回值添加“参数错误”错误代码
V1.19	2023/2/3	1.增加包类型“请求应答包(ex)”
V1.20	2023/3/20	1.请求应答包设备信息新增 CAN 和 LIN 通道数字段
V1.21	2023/7/27	1.修改 LIN 传输报文结构
V1.22	2023/11/16	1.增加 SN 码字段
V1.23	2024/07/23	1.更新 logo 图片

目 录

1. 网络传输包定义

1.1 格式说明

表 1.1 网络传输包格式

包头					数据区	校验码
起始标识	包类型	类型参数	标记	数据长度		

表 1.2 包参数说明

包参数	大小（Byte）	说明
起始标识	1	固定为 0x55；
包类型	1	指示该包类型，见表 1.3；
类型参数	1	包类型对应参数，见表 1.3；
标记	1	Bit0：是否包含 SN，0-不包含,1-包含；[2]
数据长度	2	指示数据区长度；
数据区	不定	包类型不同，数据不同；[3]
校验码	1	采用 BCC（异或校验法），校验范围从起始标识开始直到校验码前一字节为止。

注：

[1] 包格式中若无特殊说明，均采用大端格式传输；

[2] 仅设备上行有效，设备下行时该字段固定为 0；

[3] 当选择包含 SN 时，数据区的前 20 字节为设备 SN 码，其余部分根据不同数据类型，数据内容也不同。

表 1.3 包类型说明

包类型	类型值	说明
CAN 数据包	0x00	指示该包为 CAN 数据包，数据区为 CAN 格式报文（见表 1.4）， 数据中最大报文个数分上传和下发：上传由配置设定，下发每次最多 50 个 CAN 报文； 包类型参数定义如下： Bit0：是否压缩，1 为压缩，采用 zlib 压缩，仅压缩数据区； 数据长度为 n * CAN 报文长度（n 为报文个数）[1]
CAN FD 数据包	0x01	指示该包为 CAN FD 数据包，数据区为 CAN FD 格式报文（见表 1.4）， 数据中最大报文个数分上传和下发：上传由配置设定，下发每次最多 18 个 CANFD 报文

		包类型参数定义与 CAN 数据包一致。 数据长度为 n * CAN FD 报文长度（n 为报文个数）[1]
定时发送数据包	0x02	指示该包为定时发送数据包，用于更新/启动定时发送报文，该定时发送报文掉电不保存。 数据为定时发送报文格式（见表 1.10），每次最多发送 10 个定时发送报文。 包类型参数定义与 CAN 数据包一致。 数据长度为 n * 定时报文长度（n 为定时发送报文个数）[1] 该包仅由客户端下发，设备不主动发送该数据包。
总线利用率指示包	0x03	指示该包为 CAN 总线利用率指示包； 设备配置周期上报总线利用率后，将周期上报该包。 该包只由设备上传，设备接收该包无效。 包类型参数保留； 数据为总线利用率信息，定义见表 1.11； 数据长度为总线利用率信息长度
GPS 数据包	0x05	指示该包为 GPS 数据包； 该包只由设备上传，设备接收该包无效。 包类型参数保留； 数据为 n*GPS 信息(GPS 信息定义见表 1.12)； 数据长度为 n*GPS 信息长度(n 为 GPS 信息个数)。
LIN 数据包	0x06	指示该包为 LIN 数据包 设备收到 LIN 总线数据后， 将主动上传。 客户端可下发 LIN 数据， 设备收到后将数据发送至 LIN 总线。 当设备 LIN 通道作为主机时， 下发 LIN 数据将直接发送 LIN 总线。 当设备 LIN 通道作为从机时，下发 LIN 数据将更新设备 LIN 应答数据表。 该包帧类型参数保留； 数据为 n*LIN 帧（LIN 帧定义见表 1.13） 数据长度为 n*LIN 帧长度（n 为 LIN 帧个数）
请求应答包	0x50	由客户端发起请求包，设备端回应应答包； 包类型参数定义如下： Bit0：0-请求包，1-应答包 Bit1~3：包序号，每次请求递增，应答与请求一致 Bit4~7:保留 请求包数据区格式见表 1.14

		应答包数据区格式见表 1.15 注：该数据包类型可用“请求应答包(ex)(0x52),”替代，建议使用“请求应答包(ex)”开发请求应答相关功能；
请求应答包(ex)	0x52	由客户端发起请求包,设备端回应应答包; 包类型参数定义如下: Bit0: 0-请求包,1-应答包 Bit1~3: 包序号,每次请求递增,应答与请求一致 Bit4~7:保留 请求包数据区格式见表 1.16 应答包数据区格式见表 1.18 注:协议版本 V1.19 开始支持该包,可通过“请求应答包(0x50)->获取设备信息请求”获取协议版本号
配置数据包	0x51	包内容为《CANFDNET 配置通讯协议》数据,即对配置协议的封装。设备或上位机收到该类型的数据包后，可对数据包内容按照《CANFDNET 配置通讯协议》解析; 包类型参数保留;

注：[1] 若为压缩，则数据长度为压缩后长度，解压后为 n 个 CAN/CANFD 报文；

[2] 超时时间与重发次数用应用协商定义。

1.2 CAN（FD）报文格式定义

表 1.4 CAN/CAN FD 报文格式

参数	大小（Byte）	说明
时间戳	8	当前报文接收/发送时间，单位 us; 当发送报文时,报文信息中 SndDelay=1,则该参数最低 4 字节表示发送后延时时间,单位 100us.
报文 ID	4	Bit0~28:报文 ID，标准帧为 11 位，扩展帧为 29 位; Bit29~31:保留，为 0;
报文信息	2	报文标识: [bit15:11]:保留，为 0; [bit11] : BusMirror,1-BusMirror CAN 报文,0-设备 CAN 通道报文 [bit10] : SndDelay[6],1-发送后延时,0-普通发送 [bit9] : ESI[1]，1-被动错误，0-主动错误; [bit8] : BRS[1]，1-CANFD 加速，0-不加速（CANFD 有效）; [bit7] : ERR，1-错误报文[2]，0-正常报文（接收有效）; [bit6] : EXT，1-扩展帧，0-标准帧; [bit5] : RTR[3]，1-远程帧，0-数据帧;

		[bit4] : FD[1], 1-CANFD, 0-CAN; [bit3] : ECHO[4],1-发送回显，0-发送不回显 [bit2] : EchoFlag[4],发送时该位为 0；接收时该位表示回显报文标识，1-发送回显报文，0-普通报文； [bit1:0] : 发送类型（仅发送有效，接收为 0）； 0: 正常发送；1: 单次发送；2: 自发自收
报文通道	1	CAN（FD）通道，取值:0~设备通道数
数据长度	1	报文数据长度；取值如下： CAN 报文： 0~8； CANFD 报文： 0~8，12，16，20，24，32，48，64
数据	8/64[5]	报文数据； CAN : 报文数据长度为 8 字节； CAN FD : 报文数据长度 64 字节；

注：[1] FD 位在控制器类型为 CANFD 时置 1 有效，ESI 仅 CANFD 接收有效，BRS 位在 FD 为 1 时有效；

[2] ERR 位为 1 时，指示该帧为错误帧，帧 ID 无效，数据长度为 8 字节。数据域定义见表 1.5；

[3] RTR 位在 FD 位为 1 时，不允许设置为 1；

[4] ECHO 位发送时有效;当 ECHO 位为 1 且报文发送成功时，设备返回该报文，此时 EchoFlag 位为 1，时间戳为报文发送完成时间戳。

[5] CAN 和 CAN FD 报文格式仅报文数据长度不一致。

[6] 发送后延迟表示 CAN/CANFD 报文开始发送后,延时报文中设定的时间后,从发送延时队列中取出下一帧发送。

表 1.5 错误帧数据域格式说明

数据区	说明
Byte0	总线状态，定义见表 1.6
Byte1	总线错误类型，总线状态为总线错误时有效，定义见表 1.7
Byte2	总线状态对应信息码
Byte3	接收错误计数
Byte4	发送错误计数
Byte5~7	保留，当前为 0x00

表 1.6 总线状态定义

类型值	类型说明
0x00	总线积极
0xE1	总线错误[1]

0xE2	总线告警
0xE3	总线消极
0xE4	总线关闭
0xE5	总线超载[2]
0xE6	控制器错误[3]
0xE7	其他错误状态[4]

注:[1]总线错误非总线状态，仅提示当前总线上的错误。

[2]保留，不再使用；

[3]CAN 控制器错误，该错误时，接收、发送错误计数无效；具体错误值由 BYTE2 字节定义，具体见表 1.8；

[4]其他错误，用于定于终端设备的错误，由终端应用定义。该错误时，接收、发送错误计数无效；具体错误值由 BYTE2 字节定义，具体见表 1.9。

表 1.7 总线错误值定义

错误值	错误说明
0x01	位错误
0x02	应答错误
0x04	CRC 错误
0x08	格式错误
0x10	填充错误
0x20	超载错误
0x40	接收缓冲区满[1](保留)
0x80	发送缓冲区满[1](保留)

注:[1]保留，新设备不再使用。

表 1.8 控制器错误定义

错误值	错误说明
0x01	控制器接收 FIFO 溢出
0x02	驱动接收缓存溢出
0x03	发送缓冲区溢出
0x04	发送报文无效
0x05	控制器内部错误

表 1.9 其他错误定义

错误值	错误说明
0x01	终端应用接收缓存溢出
0x02	终端应用发送缓存溢出

1.3 定时发送报文格式定义

表 1.10 定时发送报文格式

参数	大小(Byte)	参数说明
编号	1	定时发送编号，取值为 0~31
使能	1	定时发送使能，1：使能，0：失能
周期	2	发送周期，单位毫秒，取值 1~60000ms
次数	2	发送次数，取值 0~65535，0 为一直发送
标志	2	Bit0: 保留，默认为 0 Bit1: 为 1 表示发送周期使用报文中 timestamp 字段低 4 字节，单位 100us Bit2~15:保留，默认为 0
报文	80	报文固定为 CANFD 格式报文，格式见表 1.4

1.4 总线利用率信息格式定义

表 1.11 总线利用率信息定义

参数	大小	数据类型	参数说明
起始时间戳	8	UINT64	测量起始时间戳，单位 us
结束时间戳	8	UINT64	测量结束时间戳，单位 us
通道号	1	UINT8	当前上报总线利用率 CAN 通道
保留	1	UINT8	保留
总线利用率	2	UINT16	总线利用率（%），总线利用率*100 展示，取值 0~10000，如 8050 时为 80.50%
收发报文数	4	UINT32	收发报文数

1.5 GPS 信息格式定义

表 1.12 GPS 信息参数定义

参数	大小	数据类型	说明
时间戳	8	UINT64	Bit0~55:采集 GPS 数据时间戳(UTC),单位 us; Bit56~62:保留,后续扩展表示使用,当前为 0 Bit63:为 1 表示有效定位;为 0 表示无效定位，沿用上次有效定位

			数据
经度	4	UINT	Bit0~27:以度为单位的纬度值乘以 10^6，精确到百万分之一度； Bit28~29:保留 Bit30:为 0 表示东经;为 1 表示西经 Bit31:为 1 表示有效定位;为 0 表示无效定位，沿用上次有效定位数据
纬度	4	UINT	Bit0~27:以度为单位的纬度值乘以 10^6，精确到百万分之一度; Bit28~29:保留 Bit30: 为 0 表示北纬;为 1 表示南纬 Bit31:为 1 表示有效定位;为 0 表示无效定位，用上次有效定位数据
速度	4	UINT	Bit0~15:单位 0.1km/h,取值 0~10000(表示 0km/h~1000.0km/h), Bit31：有效位，1 为有效
航向角	4	UINT	Bit0~30:以度为单位，乘以十的六次方。 Bit31:有效位，1 为有效
海拔高度	4	UINT	Bit0~15:海拔高度，单位米,偏移 1000m。 Bit16~30:保留 Bit31:有效位，1 为有效
保留	4	UINT	保留

1.6 LIN 帧格式定义

表 1.13 LIN 帧格式定义

参数	大小	参数说明
Timestamp	8	接收报文时间戳，单位 us。 主机发送时保留为 0.
Chn	1	通道号，从 0 开始
PID	1	PID; 作为发送时，设备默认将自动填充 PID 后发送，主机发送 ID 即可。 作为接收时，为设备接收 PID
Flags	2	报文标识： 主机（PC）发送报文时，定义如下： Bit0~1: 发送方式， 0 - 订阅数据（此时根据报文长度 Length 接收数据） 1 - 从机应答数据 2- 发送主机请求，仅发送报头

		3 - 主机发送请求及数据（报头+从机数据） Bit2~3：校验方式，仅发送方式为 0/1/3 时有效； 0 - 默认，启动时配置 1- 经典校验， 2- 增强校验， 3 - 自动，设备识别自动识别校验方式，仅发送方式为 0 有效。 Bit4~15：保留，为 0 设备上传报文时，该域定义如下： Bit0：DIR，传输方向，1-发送，0-接收；传输方向为从机任务数据方向 Bit1：ERR，错误信息标识； 0-正常报文，1-错误信息 Bit2~5：ErrStage,错误阶段，ERR 为 1 时有效； 0：总线空闲 1：Break 段 2：同步段 3：ID 段 4~12：数据错误（包括校验），表示接收哪个字节时出错，4 表示字节 0； 15：未知 Bit6~9：ErrReason，错误原因，ERR 为 1 有效； 0：接收超时 1：接收到不合法（不符合协议）数据，如 ID 校验错误、同步段错误、校验错误。此时报文中对应字段表示该错误实际值。 2：接收到字节错误（停止位为显性） 3：Break 段错误 15：未知错误 Bit10~15：保留，为 0
Res	2	保留，为 0
Chksum	1	校验和，接收校验和 发送时无需设置该字段 接收时表示设备接收报文实际 Chksum
Length	1	接收时，取值 0~8；表示实际接收数据 发送时，因发送方式不同而定义不同： 发送方式=0，表示订阅该报文 ID 的数据长度；若设置为 0Xff,则设备自动识别报文长度； 发送方式=1/3（从机发送数据），表示从机发送数据长度； 发送方式=2（发送报头），该字段保留为 0

Data	8	报文数据 接收时，表示实际接收数据。 发送时，仅当发送方式=1/3 时有效

1.7 请求应答包定义

当客户端发送请求包时，设备将对支持的请求回应应答包，其请求应答格式定义如下。

1.7.1 请求包格式定义

表 1.14 请求包数据区定义

参数	大小	数据类型	说明
请求事务 ID	1	UINT8	请求 ID，由客户端定义
请求类型	1	UINT8	请求类型值，定义见表 1.17
请求数据	/	/	由具体请求定义，长度不定

1.7.2 应答包格式定义

表 1.15 应答包数据区定义

参数	大小	数据类型	说明
事务 ID	1	UINT32	与请求 ID 一致
应答结果	1	UINT8	1-应答正确，0-应答错误
应答数据	/	/	由具体请求定义，长度不定

1.7.3 请求类型说明

同 1.81.8.3

1.8 请求应答包(ex)定义

当客户端发送请求包时，设备将对支持的请求回应应答包，其请求应答格式定义如下。

1.8.1 请求包格式定义

表 1.16 请求包数据区定义

参数	大小	数据类型	说明
请求事务 ID	4	UINT32	请求 ID，由客户端定义
请求类型	1	UINT8	请求值，定义见表 1.17
请求数据	/	/	由具体请求定义，长度不定

表 1.17 请求定义

请求	请求值（Hex）

获取设备信息	1
获取设备状态	2
获取定时发送列表	3
获取发送延时队列状态	4
清空发送延时队列	5
设置动态配置(json)	6
清空 LIN 应答数据	7
获取 LIN 发送队列状态	8
发送带确认数据包（CAN）	9
发送带确认的数据包（CANFD）	0a
发送带确认的数据包（定时发送）	0b
设置持久配置(json)	0c
设置动态配置（数据流）	0d
设置持久配置（数据流）	0e
UDS 诊断请求	0f
UDS 诊断状态获取	10
UDS 诊断控制	11
同步设备时间	12

各个请求定义见 1.8.3 说明。

1.8.2 应答包格式定义

表 1.18 应答包数据区定义

参数	大小	数据类型	说明
事务 ID	4	UINT32	与请求 ID 一致
应答类型	1	UINT8	与请求包类型一致
应答结果	1	UINT8	1-应答正确，0-应答错误
应答数据	/	/	由具体请求定义，长度不定

1.8.3 请求类型说明

1. 获取设备信息

该请求用于获取设备信息，如设备版本等信息。

该请求无数据区；应答包数据区为 JSON 字符串表示设备信息。JSON 中各参数定义见下表。

表 1.19 设备信息参数定义

参数	参数类型	参数说明
DevType	UINT	设备类型
Model	STRING	设备型号，如 CANFDNET-400U
Firmware	STRING	固件版本号，格式为 Vx.xx.xx
Hardware	STRING	硬件版本号，格式为 Vx.xx.xx
SN	STRING	设备序列号，为 20 字节字符串
Uboot[1]	STRING	Uboot 版本号
Kernel[1]	STRING	内核版本号
Rootfs[1]	STRING	文件系统版本号
Fpga[1]	STRING	FPGA 版本号
CloudSN[1]	STRING	云序列号
ProtoVer[2]	STRING	通讯协议版本，格式为 Vx.xx
CanChnNum[3]	UINT	设备 CAN 通道数
LinChnNum[3]	UINT	设备 LIN 通道数

注：[1]若设备不支持，则无此字段。

[2] V1.10 版本协议开始存在该字段。

[3] V1.20 版本协议开始存在该字段。

2. 获取设备状态

获取设备状态暂未支持。

3. 获取 CAN(FD)定时发送列表

该请求用于获取当前定时发送列表信息。

该请求数据区为 JSON 格式字符串。其请求参数定义如下表所示。

表 1.21 获取定时发送列表请求参数

参数	参数类型	参数说明
Chn	UINT	CAN 通道号
Offset	UINT	从获取定时发送列表的 Offset 开始获取报文，取值为 0~定时发送列表大小-1
Num	UINT	获取定时发送列表的报文个数，取值为 1~定时发送列表大小

应答包数据区为 JSON 字符串，表示定时发送列表信息。该 JSON 为包含 Num 个子对象，子对象键值为 Msgx（x 为 Offset~Offset+Num-1）,子对象为定时发送消息，子对象定义如下表所示。

表 1.22 定时发送报文参数

参数定义	参数类型	参数说明
Enable	UINT	消息使能，0：失能，1：使能
Period	UINT	报文发送周期（单位:ms）， 取值范围 1~600000
PeriodUnit	int	发送周期单位（无该参数时为 ms） 0：单位 ms，1：单位 100us
Cnt	UINT	报文发送次数， 取值范围 0~65535，0 为一直发送
MsgID	UINT	帧 ID，标准帧：0x0~0x7FF；扩展帧 0x0~0x1FFFFFFF
MsgFlag	UINT	帧标志， [bit15:10] ：保留 [bit9] ：ESI，1-被动错误，0-主动错误(CANFD 接收有效) [bit8] ：BRS，1-CANFD 加速，0-不加速（CANFD 有效） [bit7] ：ERR，1-错误报文，0-正常报文（接收有效） [bit6] ：EXT，1-扩展帧，0-标准帧 [bit5] ：RTR，1-远程帧，0-数据帧 [bit4] ：FD， 1-CANFD 报文，0-CAN 报文 [bit3:2] ：保留 [bit1:0] ：发送类型（发送有效），0-正常发送，1-单次发送
MsgData	ARRAY-UINT8	帧数据，最长为 64 字节

4. 获取发送延时队列状态

该请求用于获取发送延时队列状态。

请求包数据区为 JSON 字符串，参数为获取通道号。其参数定义如下表所示。

表 1.23 请求队列状态参数

参数	参数类型	参数说明
Chn	UINT	通道号

该请无数据区；应答包数据区为 JSON 字符串表示队列状态。其参数定义如下表所示。

表 1.24 应答队列状态参数

参数	参数类型	参数说明
Chn	UINT	通道号
Size	UINT	发送队列帧数
Remain	UINT	队列剩余可填充帧数

5. 清空发送延时队列

该请求用于获取发送延时队列状态。

请求包数据区为 JSON 字符串，参数为获取通道号。其参数定义如下表所示。

表 1.25 清空队列状态参数

参数	参数类型	参数说明
Chn	UINT	通道号

应答包无数据区，客户端应按应答结果判断该命令是否执行成功。

6. 设置动态配置（json）

该请求用于设置实时配置，用于设备在线时动态配置设备参数，动态配置参数掉线不保存。

请求包数据为配置项的 JSON 字符串， 与配置协议一致。 仅支持动态配置的模块可设置该项。

应答包数据区无效，客户端应按应答结果判断该命令是否执行成功。

7. 设置持久配置（json）

该请求用于设置持久化配置，其请求和响应内容除请求值外与【6 设置动态配置 】一致，区别是使用该请求配置的设备参数掉电保存。

8. 设置动态配置（数据流）

该请求用于设备在线时动态配置设备某一个参数，旨在精简，一次只配置一个参数项。动态配置参数掉线不保存。

请求包数据为字节流形式，其结构定义参考表 1.26。 仅支持动态配置的模块可设置该项。

注：以下配置多字节参数均采用小端字节序。

表 1.26 设置动态配置数据结构定义

参数	大小	数据类型	说明
配置类型	1	UINT8	配置类型，定义见表 1.27
配置数据	n	/	由具体参数类型定义，具体见各模块类型参数定义

表 1.27 配置类型

模块类型	类型值(Dec)	说明
CAN 配置	0	CAN 配置，对应的参数类型及数据区定义见表 1.28
LIN 配置	1	LIN 配置，详细参考表 1.29

表 1.28 CAN 配置

配置项	大小	说明
Type	1	参数类型，定义见表 1.29
Channel	1	通道号
Value	n	参数值，内容视参数类型而定，具体参考表 1.29

表 1.29 CAN 参数定义

配置类型	类型值(Dec)	Value 大小	Value 说明
Mode	0	1	工作模式，0：正常模式；1：只听模式
NominalBaud	1	4	仲裁域波特率(bps)，支持值：50000、100000、125000、250000、500000、800000、1000000，其他值请使用 6 小节的方式配置
DataBaud	2	4	数据域波特率(bps)，支持值：100000、125000、250000、500000、800000、1000000、2000000、4000000、5000000，其他值请使用 6 小节的方式配置

表 1.30 LIN 配置

配置项	大小	说明
Type	1	参数类型，定义见表 1.31
Channel	1	通道号
Value	n	参数值，内容视参数类型而定，具体参考表 1.31

表 1.31 LIN 参数定义

配置项	类型值(Dec)	大小	说明
IsMaster	0	1	是否作为主机，1-主机，0-从机
Baudrate	1	2	波特率(bps)，取值 1000~20000
Feature	2	1	LIN 特性位 Bit0: 增强型校验，1-增强型校验和，0-经典型校验和 Bit1: 关闭可变消息长度，1-关闭可变 DLC，0-可变 DLC，关闭可变消息长度后，消息长度将不再依赖消息 ID Bit2~31: 保留，为 0 *注：对于 LIN2.X，Bit0 和 Bit1 均设为 1；对于 LIN1.X，Bit0 和 Bit1 均设为 0*

应答包数据区无效，客户端应按应答结果判断该命令是否执行成功。

(1) 配置示例

配置 CAN 通道 0 的仲裁域波特率为 500kbps 的数据流如下所示：

```
55 50 00 00 00 09 00 0d 00 01 00 20 a1 07 00 86
```

其中，各字节按照协议解析如表 1.32 所示：

表 1.32 设置 CAN 仲裁域波特率请求解析

包头					数据区						校验码
					请求包头		请求包数据区				
起始标识	包类型	类型参数	保留	数据长度	请求 ID	请求值	配置类型	参数类型	通道号	参数值	
55	50	00	00	00 09	00	0d	00	01	00	20 a1 07 00	86

当设置其他类型参数时，可参照表 1.32 的组织方式，根据不同配置参数，修改“请求包数据区”的内容，然后根据情况填写“数据长度”字段并重新计算校验码即可。设置设备动态参数 C 参考代码见【附录 2.2.1 设置动态/持久配置（数据流）参考代码】

注：[1] “数据长度”字段为大端字节序，其余均为小端字节序，编程时需注意字节序转换；

[2] 如连续发起多次配置请求，每次请求需递增“请求 ID”字段，以保证请求有序进行。

设置成功时设备的响应数据流如下所示：

```
55 50 01 00 05 ad 00 01 00 … ad
```

其中，各字节按照协议解析如表 1.33 所示：

表 1.33 设置 CAN 仲裁域波特率响应

包头					数据区			校验码
					应答包头		数据区	
起始标识	包类型	类型参数	保留	数据长度	事务 ID	应答结果	应答数据	
55	50	01	00	05 ad	00	01	00 …	ad

其中，“应答结果”字段表示设置是否成功，设置成功为 1，设置失败为 0；

9. 设置持久配置（数据流）

该请求用于设置持久化配置，其请求和响应内容除请求值外与【31.81.8.38 设置动态配置（数据流）】一致，区别是使用该请求配置的设备参数掉电保存。

10. 清空 LIN 应答数据

该请求用于 LIN 通道作为从机时，清空用户设置的应答数据。

请求包数据区为 JSON 字符串，参数为待清空应答的 LIN 通道。其参数定义如下表示。

表 1.34 清空 LIN 应答数据请求参数

参数	参数类型	参数说明
Chn	UINT	通道号

应答包无数据区，客户端应按应答结果判断该命令是否执行成功。

11. 获取 LIN 发送队列状态

该请求用于获取 LIN 发送队列状态。

请求包数据区为 JSON 字符串，参数为获取通道号。其参数定义如下表所示。

表 1.35 请求 LIN 发送队列状态参数

参数	参数类型	参数说明
Chn	UINT	通道号

应答包数据区为 JSON 字符串表示 LIN 发送队列状态。其参数定义如下表所示。

表 1.36 应答 LIN 发送队列状态参数

参数	参数类型	参数说明
Chn	UINT	通道号
Size	UINT	发送队列总大小
Remain	UINT	队列剩余可填充帧数

12. 发送带确认的 CAN（FD）数据包

该请求用于发送带确认的数据包（CAN(FD)），设备收到该包后，返回处理结果。应答数据区为 JSON 字符串，其参数定义如下表所示。

参数	参数类型	参数说明
Cnt	UINT	设备收到的报文个数
FreeNum	UINT ARRAY	设备 CAN(FD)通道发送队列剩余空间，参数类型为无符号整型数组，数组成员为 CAN0 通道开始，每通道剩余可填充队列空间

13. UDS 诊断请求

该请求用于进行 UDS 诊断服务。设备接收到该请求后，立即按照请求配置和数据进行 UDS 请求服务。

该请求的请求包数据区定义如下：

表 1.37 UDS 诊断请求包

参数	大小	数据类型	说明
UDS 请求配置	n	string	UDS 请求配置，json 字符串，详见表 1.38
分割符	1	UINT8	分割符，固定 0x00
UDS 请求数据	n	UINT8	UDS 请求数据，数据内容视具体服务而定（如读取 VIN 的请求数据通常为 0xF1，0x90 两个字节），数据长度在表 1.38 中 DateLen 字段定义。

表 1.38 UDS 服务配置 json 定义

Key	Value 类型	Value 说明
ReqID	UINT	请求事务 ID，值由客户端定义，范围 0~65535，标志本次 UDS 请求，14 中通过该 ID 查询本次 UDS 请求的运行状态。
Chn	UINT	CAN 通道号，取值由具体设备定义，范围 0~255
FrameType	UINT	帧类型，0：CAN，1：CANFD，2：CANFD 加速
SrcAddr	UINT	请求地址，取值 0~0x1FFFFFFF（与是否为扩展帧相关），为功能地址或物理地址
DstAddr	UINT	响应地址，取值 0~0x1FFFFFFF（与是否为扩展帧相关）
SuppressResp	UINT	是否抑制响应，0 为不抑制，1 为抑制
SID	UINT	请求服务 ID
SessionParam	OBJ	会话层参数，obj 类型，表 1.39

ISO15765Param	OBJ	传输层参数，obj 类型，详见表 1.40
DateLen	UINT	请求数据长度，其值视具体服务而定，范围 0~60000

表 1.39 会话层参数 json 定义

Key	Value 类型	Value 说明
Timeout	UINT	响应超时时间，单位 ms
EnhancedTimeout	UINT	收到消极响应错误码为 0x78 后的超时时间(ms)
ChkAnyNegResp	UINT	接收到非本次请求服务的消极响应时是否需要判定为响应错误
WaitIfSuppressResp	UINT	抑制响应时是否需要等待消极响应，等待时长为响应超时时间

表 1.40 ISO15765 传输层参数定义

Key	Value 类型	Value 说明
Version	UINT	传输协议版本， 0：ISO15765-2(2004)，1：ISO15765-2(2016)
MaxDataLen	UINT	单帧最大数据长度， CAN：8，CANFD：64
LocalStMin	UINT	连续帧之间的最小间隔，设备发送流控时用，0x00-0x7F(0ms~127ms)， 0xF1-0xF9(100us~900us)
BlockSize	UINT	流控帧的块大小
FillByte	UINT	无效字节的填充数据
Ext	UINT	帧类型，0：标准帧；1：扩展帧
IsModifyEcuStMin	UINT	是否忽略 ECU 返回流控的 STmin，强制使用本次设置的 RemoteStMin。 0：使用 ECU 返回流控的 STmin 1：使用 RemoteStMin
RemoteStMin	UINT	STmin，设备发送多帧时用，IsModifyEcuStMin 为 1 时有效，范围 0ms~127ms
FCTimeout	UINT	接收流控超时时间(ms)， 如发送首帧后需要等待回应流控帧

该请求的响应包数据区定义如下：

表 1.41 UDS 诊断响应包

参数	大小	数据类型	说明
UDS 响应结果	n	string	UDS 响应结果，json 字符串，详见表 1.42
分割符	1	UINT8	分割符，固定 0x00
UDS 响应数据	n	UINT8	UDS 响应数据（原始数据），数据内容视具体服务而定，数据长度在表 1.42 中 DateLen 字段定义。

表 1.42 UDS 响应结果 json 定义

Key	Value 类型	Value 说明
Status	UINT	错误码，定义如下（十进制）： 0：无错误 1：响应超时 2：发送数据失败 3：取消请求 4：抑制响应 5：忙碌中 6：参数错误
RespType	UINT	响应类型，0：消极响应；1：积极响应
SID	UINT	服务 ID
NegCode	UINT	消极响应 ID，固定为 0x7F 消极响应时有效
NegErr	UINT	消极响应错误码 消极响应时有效
DateLen	UINT	响应数据长度，其值视具体服务而定

14. UDS 诊断状态

该请求用来查询 UDS 诊断请求的状态，设备接收到该请求后，立即按照参数查询正在进行的 UDS 请求的运行状态并返回。

该请求的请求包数据区为 json 字符串，json 对象定义如下：

表 1.43 UDS 诊断控制请求 json 对象定义

Key	Value 类型	Value 说明
ReqID	UINT	UDS 请求事务 id，对应表 1.38 中 ReqID 的值

注：

该请求的应答包数据区为 json 字符串，json 对象定义如下：

Key	Value 类型	Value 说明
Status	UINT	状态码，定义如下（十进制）： 0：未请求 1：请求中 2：参数错误

15. UDS 诊断控制

UDS 诊断控制用来控制正在进行的 UDS 请求，设备接收到该请求后，立即按照控制参数控制正在进行的 UDS 请求。

该请求的请求包数据区为 json 字符串，json 对象定义如下：

表 1.44 UDS 诊断控制请求 json 对象定义

Key	Value 类型	Value 说明
ReqID	UINT	UDS 请求事务 id，对应表 1.38 中 ReqID 的值
CtrlType	UINT	控制类型：0：停止正在进行的 UDS 请求

该请求的应答包数据区为 json 字符串，json 对象定义如下：

Key	Value 类型	Value 说明
Status	UINT	错误码： 0：成功 1：失败 2：参数错误

16. 同步设备时间

该请求用来同步设备系统时间，请求包数据区为时间字符串，字符串格式为 2023-04-28T10:43:36.125。

1.8.4 系统状态信息

系统状态信息用于上报系统运行时间、产品序列号、设备名等信息。各个参数定义如下表所示。

表 1.45 系统信息参数定义

字段名	字段值类型	说明
SysTime	UINT	设备当前时间（UTC），单位 s
StartupTime	UINT	系统启动时间（UTC）,单位 s
Sn	STRING	设备 SN 序列号
Buzzer	UINT	蜂鸣器是否使能：0：未使能，1：使能
DevName	String	设备名
Ntp	UINT	Ntp 校时状态。 0：未使能 NTP 1：使能，未校时 2：使能，已校时

系统状态信息数据示例：

程序清单 1.1 系统状态信息数据示例

```
{
    "SysTime": 1586752675,
    "StartupTime": 1586752670,
    "Sn": "123456abc",
```

```
    "Buzzer": 1,
    "DevName": "ZLG-001",
    "Ntp": 1
}
```

1.8.5 CAN 通道状态信息

CAN 通道信息使用对象数组进行描述，对象名为：“Channel”，每个通道对应一个对象。

CAN 通道状态信息数据中 JSON 字段定义：

表 1.46 CAN 模块状态信息格式

字段名	字段值类型	说明
Enable	UINT	通道使能；0：未使能，1：使能
Chn	UINT	通道号
BaudRate	UINT	通道波特率，单位 bps
ResEnable	UINT	通道终端电阻使能；0：未使能，1：使能
Recv	UINT	通道接收计数
Send	UINT	通道发送计数
RErr	UINT	通道接收错误计数
SErr	UINT	通道发送错误计数
Link	UINT	状态上报周期内数据活动；0：无活动，1：有活动
RatioPeriod	UINT	总线利用率上报周期，单位 ms，未上报时为 0

CAN 模块状态信息示例如下：

程序清单 1.2 CAN 模块状态信息示例

```
{
    "Channel": [
        {
            "Enable": 1,
            "Chn": 0,
            "BaudRate": 500000,
            "ResEnable": 1,
            "Recv": 1234,
            "Send": 1234,
            "RErr": 0,
            "SErr": 0,
            "Link": 1,
```

```
            "RatioPeriod": 0
        },
        {
            "Enable": 1,
            "Chn": 1,
            "BaudRate": 500000,
            "ResEnable": 1,
            "Recv": 1234,
            "Send": 1234,
            "RErr": 0,
            "SErr": 0,
            "Link": 1,
            "RatioPeriod": 0
        },
        {
            "Enable": 1,
            "Chn": 2,
            "BaudRate": 500000,
            "ResEnable": 1,
            "Recv": 1234,
            "Send": 1234,
            "RErr": 0,
            "SErr": 0,
            "Link": 1,
            "RatioPeriod": 0
        },
        {
            "Enable": 1,
            "Chn": 3,
            "BaudRate": 500000,
            "ResEnable": 1,
            "Recv": 1234,
            "Send": 1234,
            "RErr": 0,
            "SErr": 0,
            "Link": 1,
            "RatioPeriod": 0
```

```
        }
    ]
}
```

1.8.6 记录模块状态信息

记录模块状态信息使用对象进行描述，状态信息格式如下：

表 1.47 记录模块状态信息格式

字段名	字段值类型	说明
RecordState	UINT	记录模块记录状态： 1：正在记录 0：未记录
RecordMode	UINT	记录模式： 0：长时间记录； 1：条件记录； 2：预触发记录； 3：定时记录； 4：不记录。
FileMode	UINT	文件操作模块 0：循环记录 1：记满停止
FileMax	UINT	记录文件大小；单位：M
SdDetect	UINT	SD 卡探测，1 有 SD 卡，0 无 SD 卡
SdErr	UINT	SD 卡错误代码，SdDetect 状态为 true 时有效： 0：无错误； 1：文件系统异常； 2：IO 操作异常。

程序清单 1.3 记录模块状态数据示例

```
{
    "RecordState": 1,
    "RecordMode": 0,
    "FileMode": 0,
    "SdDetect": true,
    "SdErr": 0
}
```

1.8.7 网络连接状态信息

设备网络连接模块是管理设备的网络接口，不同的产品提供的网络接口数量可能不相同，设备网络连接模块使用对象进行描述，具体的网络接口使用数组进行描述，设备的网络接口数等于数组的元素个数。

设备网络连接模块状态数据格式如下：

表 1.48 设备网络连接状态格式

字段名	字段值类型	说明
DefIface	STRING	默认网卡接口名称
Iface	Arrary	设备网络接口状态，格式定义见表 1.50

设备网络接口状态使用对象进行描述，状态数据格式如下：

表 1.49 设备网络连接模块状态格式

字段名	字段值类型	说明
Name	STRING	网络接口名称
Dhcp	UINT	DHCP 服务状态，1-启动，0-关闭
Connect	UINT	连接状态：1-已连接；未连接
IP	STRING	网络接口 IP 地址
Mask	STRING	网络接口子网掩码
GateWay	STRING	网络接口网关地址

程序清单 1.4 设备网络连接模块数据示例

```
{
    "DefIface": "Lan0",
    "Iface": [
        {
            "Name": "Lan0",
            "Dhcp": 0,
            "Connect": 1,
            "IP": "192.168.1.100",
            "Mask": "255.255.255.0",
            "GateWay": "192.168.1.1"
        }
    ]
}
```

2. 附录

2.1 设置动态/持久配置（数据流）参考代码

程序清单 2.1 设置动态/持久配置（数据流）参考代码

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/shm.h>

#define BUFFER_SIZE 1460

#define PRINT_BUF(info, pbuf, len)                              \
    {                                                           \
        int i;                                                  \
        printf(info);                                           \
        for (i = 0; i < len; i++)                               \
        {                                                       \
            printf("%02x ", ((uint8_t *)pbuf)[i]); \
        }                                                       \
        printf("\n");                                           \
    }

// 请求类型
enum req_type
{
    RT_SET_CFG_DYNAMIC_BY_JSON = 6,
    RT_SET_CFG_PERSIST_BY_JSON = 12,
    RT_SET_CFG_DYNAMIC_BY_BIN = 13,
    RT_SET_CFG_PERSIST_BY_BIN = 14,
};

// 配置类型
enum cfg_type
{
    CT_CAN = 0,
    CT_LIN = 1,
```

```
};

// CAN 配置参数类型
enum can_cfg_param_type
{
    CCPT_MODE = 0,
    CCPT_NOMINAL_BAUD,
    CCPT_DATA_BAUD
};

// LIN 配置参数类型
enum lin_cfg_param_type
{
    LCPT_MODE = 0,
    LCPT_BAUD,
    LCPT_FEATURE
};

#pragma pack(1) //  设定为 1 字节对其

//  请求配置 CAN 模式结构
typedef struct req_cfg_can_mode
{
    uint8_t start;
    uint8_t type;
    uint8_t param;
    uint8_t res;
    uint16_t length;
    uint8_t id;
    uint8_t req_val;
    uint8_t cfg_type;
    uint8_t param_type;
    uint8_t channel;
    uint8_t mode;
    uint8_t bcc;
} req_cfg_can_mode_t;

//  请求配置 CAN 波特率结构
typedef struct req_cfg_can_baud
{
    uint8_t start;
    uint8_t type;
    uint8_t param;
    uint8_t res;
```

```
    uint16_t length;
    uint8_t id;
    uint8_t req_val;
    uint8_t cfg_type;
    uint8_t param_type;
    uint8_t channel;
    uint32_t baud;
    uint8_t bcc;
} req_cfg_can_baud_t;

// 请求 LIN 模式结构
typedef struct req_cfg_lin_mode
{
    uint8_t start;
    uint8_t type;
    uint8_t param;
    uint8_t res;
    uint16_t length;
    uint8_t id;
    uint8_t req_val;
    uint8_t cfg_type;
    uint8_t param_type;
    uint8_t channel;
    uint8_t mode;
    uint8_t bcc;
} req_cfg_lin_mode_t;

// 请求 LIN 波特率结构
typedef struct req_cfg_lin_baud
{
    uint8_t start;
    uint8_t type;
    uint8_t param;
    uint8_t res;
    uint16_t length;
    uint8_t id;
    uint8_t req_val;
    uint8_t cfg_type;
    uint8_t param_type;
    uint8_t channel;
    uint16_t baud;
    uint8_t bcc;
} req_cfg_lin_baud_t;
```

```
// 请求配置 LIN 特性结构
typedef struct req_cfg_lin_feature
{
    uint8_t start;
    uint8_t type;
    uint8_t param;
    uint8_t res;
    uint16_t length;
    uint8_t id;
    uint8_t req_val;
    uint8_t cfg_type;
    uint8_t param_type;
    uint8_t channel;
    uint8_t feature;
    uint8_t bcc;
} req_cfg_lin_feature_t;

#pragma pack() // 恢复对齐状态

// 大小端转换
void change_endian(uint8_t *buf, int len)
{
    int index = 0;
    while (index < len - index - 1)
    {
        const uint8_t t = *(buf + index);
        *(buf + index) = *(buf + len - index - 1);
        *(buf + len - index - 1) = t;
        ++index;
    }
}

// BCC（异或校验法）校验码计算
uint8_t calc_xor(const uint8_t *data, int size, uint8_t init)
{
    uint8_t ret = init;
    for (int i = 0; i < size; ++i)
    {
        ret ^= (*data);
        ++data;
    }
    return ret;
}
```

```
int main()
{
    // 定义 sockfd
    int sock_cli = socket(AF_INET, SOCK_STREAM, 0);

    // 定义 sockaddr_in
    struct sockaddr_in servaddr;
    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(8000);                    // 服务器端口
    servaddr.sin_addr.s_addr = inet_addr("172.16.9.220"); // 服务器 ip

    // 连接服务器，成功返回 0，错误返回-1
    if (connect(sock_cli, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0)
    {
        perror("connect");
        exit(1);
    }
    printf("服务器连接成功\n");
    char sendbuf[BUFFER_SIZE];
    char recvbuf[BUFFER_SIZE];
    while (fgets(sendbuf, sizeof(sendbuf), stdin) != NULL)
    {
        switch (sendbuf[0])
        {
        case '1':
        {
            printf("设置 CAN 模式...\n");
            memset(sendbuf, 0, BUFFER_SIZE);
            req_cfg_can_mode_t *p_struct = (req_cfg_can_mode_t *)sendbuf;
            p_struct->start = 0x55;
            p_struct->type = 0x50;
            p_struct->param = 0;
            p_struct->res = 0;
            p_struct->id = 0;
            p_struct->req_val = RT_SET_CFG_PERSIST_BY_BIN;
            p_struct->cfg_type = CT_CAN;
            p_struct->param_type = CCPT_MODE;
            p_struct->channel = 0;
            p_struct->mode = 1;
            p_struct->length = sizeof(req_cfg_can_mode_t) - 7;
            change_endian((uint8_t *)&p_struct->length, sizeof(p_struct->length));
            p_struct->bcc = calc_xor((const uint8_t *)p_struct, sizeof(req_cfg_can_mode_t) - 1, 0);
            PRINT_BUF("向服务器发送数据：", sendbuf, sizeof(req_cfg_can_mode_t));
```

```
            send(sock_cli, sendbuf, sizeof(req_cfg_can_mode_t), 0);
        }
        break;
        case '2':
        {
            printf("设置 CAN 仲裁域波特率...\n");
            memset(sendbuf, 0, BUFFER_SIZE);
            req_cfg_can_baud_t *p_struct = (req_cfg_can_baud_t *)sendbuf;
            p_struct->start = 0x55;
            p_struct->type = 0x50;
            p_struct->param = 0;
            p_struct->res = 0;
            p_struct->id = 0;
            p_struct->req_val = RT_SET_CFG_PERSIST_BY_BIN;
            p_struct->cfg_type = CT_CAN;
            p_struct->param_type = CCPT_NOMINAL_BAUD;
            p_struct->channel = 0;
            p_struct->baud = 800000;
            p_struct->length = sizeof(req_cfg_can_baud_t) - 7;
            change_endian((uint8_t *)&p_struct->length, sizeof(p_struct->length));
            p_struct->bcc = calc_xor((const uint8_t *)p_struct, sizeof(req_cfg_can_baud_t) - 1, 0);
            PRINT_BUF("向服务器发送数据：", sendbuf, sizeof(req_cfg_can_baud_t));
            send(sock_cli, sendbuf, sizeof(req_cfg_can_baud_t), 0);
        }
        break;
        case '3':
        {
            printf("设置 CAN 数据域波特率...\n");
            memset(sendbuf, 0, BUFFER_SIZE);
            req_cfg_can_baud_t *p_struct = (req_cfg_can_baud_t *)sendbuf;
            p_struct->start = 0x55;
            p_struct->type = 0x50;
            p_struct->param = 0;
            p_struct->res = 0;
            p_struct->id = 0;
            p_struct->req_val = RT_SET_CFG_PERSIST_BY_BIN;
            p_struct->cfg_type = CT_CAN;
            p_struct->param_type = CCPT_DATA_BAUD;
            p_struct->channel = 0;
            p_struct->baud = 4000000;
            p_struct->length = sizeof(req_cfg_can_baud_t) - 7;
            change_endian((uint8_t *)&p_struct->length, sizeof(p_struct->length));
            p_struct->bcc = calc_xor((const uint8_t *)p_struct, sizeof(req_cfg_can_baud_t) - 1, 0);
            PRINT_BUF("向服务器发送数据：", sendbuf, sizeof(req_cfg_can_baud_t));
```

```
            send(sock_cli, sendbuf, sizeof(req_cfg_can_baud_t), 0);
        }
        break;
        case '4':
        {
            printf("设置 LIN 模式...\n");
            memset(sendbuf, 0, BUFFER_SIZE);
            req_cfg_lin_mode_t *p_struct = (req_cfg_lin_mode_t *)sendbuf;
            p_struct->start = 0x55;
            p_struct->type = 0x50;
            p_struct->param = 0;
            p_struct->res = 0;
            p_struct->id = 0;
            p_struct->req_val = RT_SET_CFG_PERSIST_BY_BIN;
            p_struct->cfg_type = CT_LIN;
            p_struct->param_type = LCPT_MODE;
            p_struct->channel = 0;
            p_struct->mode = 1;
            p_struct->length = sizeof(req_cfg_lin_mode_t) - 7;
            change_endian((uint8_t *)&p_struct->length, sizeof(p_struct->length));
            p_struct->bcc = calc_xor((const uint8_t *)p_struct, sizeof(req_cfg_lin_mode_t) - 1, 0);
            PRINT_BUF("向服务器发送数据：", sendbuf, sizeof(req_cfg_lin_mode_t));
            send(sock_cli, sendbuf, sizeof(req_cfg_lin_mode_t), 0);
        }
        break;
        case '5':
        {
            printf("设置 LIN 波特率...\n");
            memset(sendbuf, 0, BUFFER_SIZE);
            req_cfg_lin_baud_t *p_struct = (req_cfg_lin_baud_t *)sendbuf;
            p_struct->start = 0x55;
            p_struct->type = 0x50;
            p_struct->param = 0;
            p_struct->res = 0;
            p_struct->id = 0;
            p_struct->req_val = RT_SET_CFG_PERSIST_BY_BIN;
            p_struct->cfg_type = CT_LIN;
            p_struct->param_type = LCPT_BAUD;
            p_struct->channel = 0;
            p_struct->baud = 1200;
            p_struct->length = sizeof(req_cfg_lin_baud_t) - 7;
            change_endian((uint8_t *)&p_struct->length, sizeof(p_struct->length));
            p_struct->bcc = calc_xor((const uint8_t *)p_struct, sizeof(req_cfg_lin_baud_t) - 1, 0);
            PRINT_BUF("向服务器发送数据：", sendbuf, sizeof(req_cfg_lin_baud_t));
```

```
            send(sock_cli, sendbuf, sizeof(req_cfg_lin_baud_t), 0);
        }
        break;
        case '6':
        {
            printf("设置 LIN 特性...\n");
            memset(sendbuf, 0, BUFFER_SIZE);
            req_cfg_lin_feature_t *p_struct = (req_cfg_lin_feature_t *)sendbuf;
            p_struct->start = 0x55;
            p_struct->type = 0x50;
            p_struct->param = 0;
            p_struct->res = 0;
            p_struct->id = 0;
            p_struct->req_val = RT_SET_CFG_PERSIST_BY_BIN;
            p_struct->cfg_type = CT_LIN;
            p_struct->param_type = LCPT_FEATURE;
            p_struct->channel = 0;
            p_struct->feature = 3;
            p_struct->length = sizeof(req_cfg_lin_feature_t) - 7;
            change_endian((uint8_t *)&p_struct->length, sizeof(p_struct->length));
            p_struct->bcc = calc_xor((const uint8_t *)p_struct, sizeof(req_cfg_lin_feature_t) - 1, 0);
            PRINT_BUF("向服务器发送数据：", sendbuf, sizeof(req_cfg_lin_feature_t));
            send(sock_cli, sendbuf, sizeof(req_cfg_lin_feature_t), 0);
        }
        break;
        case 'q':
            close(sock_cli);
            return 0;
        default:
            break;
        }
        memset(recvbuf, 0, sizeof(recvbuf));
        int len = recv(sock_cli, recvbuf, sizeof(recvbuf), 0); //  接收
        if (len < 0)
        {
            printf("从服务器接收数据出错，关闭连接！\n");
        }
        if (len > 0)
        {
            PRINT_BUF("从服务器接收数据：", recvbuf, len);
        }
    }
    close(sock_cli);
    return 0;
```

```
}
```

3. 免责声明

本着为用户提供更好服务的原则，广州致远电子股份有限公司（下称“致远电子”）在本手册中将尽可能地为用户呈现详实、准确的产品信息。但介于本手册的内容具有一定的时效性，致远电子不能完全保证该文档在任何时段的时效性与适用性。致远电子有权在没有通知的情况下对本手册上的内容进行更新，恕不另行通知。为了得到最新版本的信息，请尊敬的用户定时访问致远电子官方网站或者与致远电子工作人员联系。感谢您的包容与支持！

诚信共赢，持续学习，客户为先，专业专注，只做第一

广州致远电子股份有限公司
更多详情请访问 www.zlg.cn
欢迎拨打全国服务热线 400-888-4005