色谱仪板卡通讯协议 Local-CAN V1.0

介绍

本文档描述了基于 CAN 的 Local-CAN 通信协议，主要的应用场景是气相色谱仪等对实时性要求较高的场景。协议通信的实现使用了生产者消费者模型，数据不需要确认帧，大大减小了网络的负载，并且提高了系统的实时性；采用基于指针的数据字典，并且为了加快查找速度，我们在数据字典的基础上，加入了段式管理，能够实现即使在用户数据字典庞大的情况下，也能实现高效率查找。实现了设备快速查找和修改数据字典数据，优于传统的 CANOpen 协议；采用基于优先级的发送队列，使对实时性要求高的数据更快的发出；使用段管理方式，结合 CAN-ID 的特点，可以实现 CAN 通信的组播，广播，单播，以及系统内部通信和系统间通信，极大丰富了 CAN 通信方式。以上优点使 Local-CAN 通信协议优于其他基于 CAN 的通信协议。

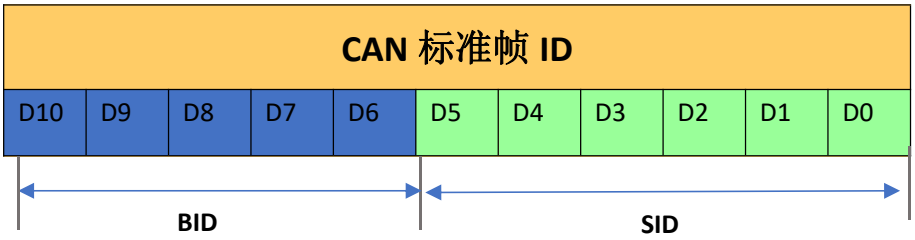
1、Local-CAN-ID 标识符定义

Local-CAN-ID 与 CAN-ID 类似，采用标准格式与扩展格式。具体结构如下：

1.1Local-CAN-ID 标准帧格式

标准格式将传统的 CAN-ID 标准帧格式分为两部分，一部分为 BID（系统板号），一部分位 SID(段 ID)。

图 1：Local-CAN 标准格式结构



BID：即用来确定该帧的发出者属于 CAN 总线中的哪一系统， SID 即用来说明该帧来自字典的哪一段。

1.2Local-CAN-ID 扩展帧格式

扩展帧格式将传统的 CAN-ID 扩展帧格式分为三部分，一部分为 BID（系统板号），一部分为 SID(段 ID)，一部分为 S_Index(段内索引)

图 2：Local-CAN 扩展帧格式结构

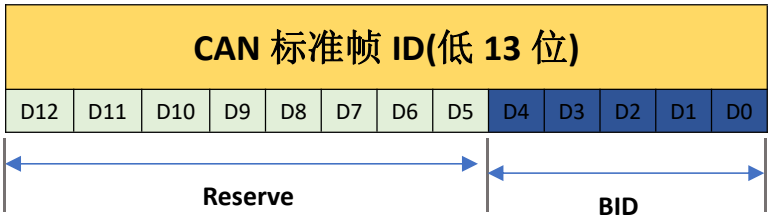
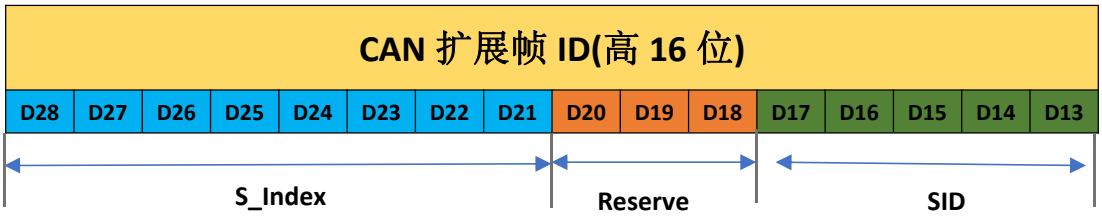


图 3：Local-CAN 扩展帧格式结构



S_Index：段内索引，用于快速查找数据字典。

Reserve：保留未使用（默认 0）。

BID：用来确定该帧的发出者属于 CAN 总线中的哪一系统。

SID：用来说明该帧来自字典的哪一段。

CAN 协议虽规定了 CAN 的标识符，但并不能在物理上区分板块，在 Locla-CAN 中，我们加入 BID 和 SID 概念，从而在逻辑上，实现了系统中组播，广播，单播。

2、Local-CAN 帧中数据域定义

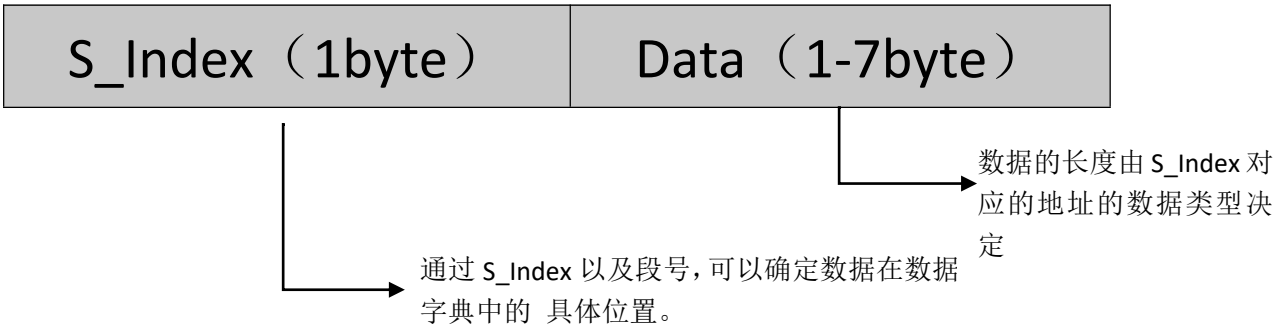
CAN 帧中数据域的长度为 8 个字节，为了兼顾传输速度以及传输效率，尽可能的节省时间及空间，根据 CAN 帧的特点，我们将数据域分为标准数据域和扩展数据域。

2.1 标准数据域

标准数据域，指的是当传输的 CAN 帧为标准帧时，数据域的定义如下：在

此情况下，Local-CAN 数据域数据长度最短为 2 个字节，最长为 8 个字节。数据长度由传输的数据类型决定。

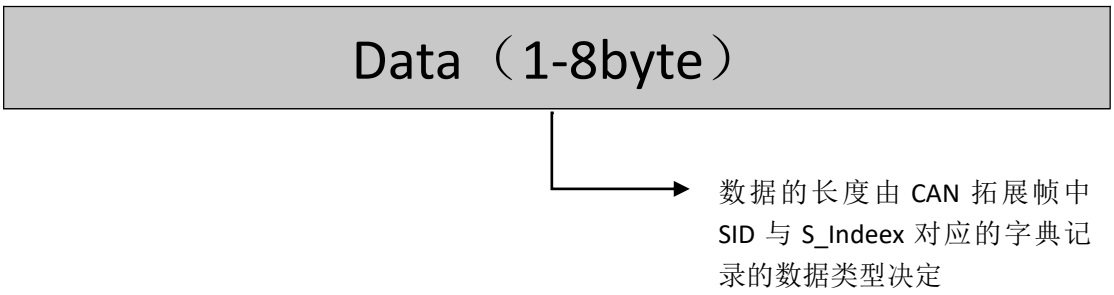
图 4：Local-CAN 的标准数据域结构



2.2 扩展数据域

扩展数据域，指的是当传输的 CAN 帧为扩展帧时，数据域的定义如下：在此情况下，Local-CAN 数据域数据长度最短为 1 个字节，最长为 8 个字节。数据长度由传输的数据类型决定。

图 5：Local-CAN 的扩展数据域结构



Data 可以是以下类型的数据：uint8，uint16，uint32，int8，int16，int32，real（IEEE754），Char arry[3]，Char arry[5]，Char arry[6]，Char arry[7],Double(仅在扩展帧格式中)

表 1：数据类型编码表

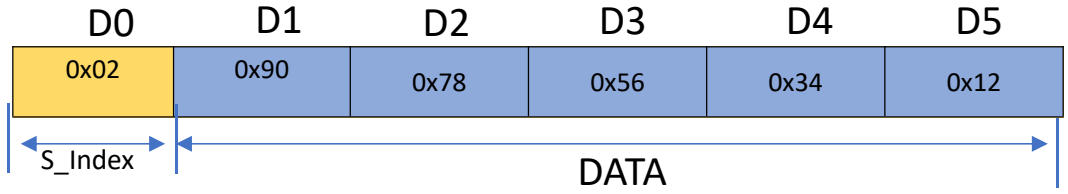
数据类型	编码
uint8	00h
uint16	01h
uint32	02h
int8	03h
int16	04h
int32	05h
real（IEEE754）	06h

Char arry[3]	07h
Char arry[5]	08h
Char arry[6]	09h
Char arry[7]	0Ah
Double	0Bh

注: Double 只在扩展数据域中存在，具体 Index 对应的数据类型请参看“数据字典索引表”。

STM32 中的数据是按照小端字节序存储的，我们规定 Data 中的数据也要按小端字节序发送，即数据的低字节存储在低地址，比如传输的数据为 0x1234567890,S_index 为 0x02，在数据域中的结构如图 3。接收端应按照小端字节序解析 Data，因为发送端和接收端都使用 STM32，发送和接受数据都不用进行字节序的转换，节省了时间，提高了数据的实时性。

图 6：数据在 Local-CAN 帧数据域的结构



3、数据字典索引表

表 2：数据字典索引表

目标板	索引值	优先级	类型	生产者内容	消费者内容	段号	段内索引
TCD/FID	00h	高	Char arry[7]	检测器注册 SN 信息	NULL	00h	00h
	01h	高	Char arry[6]	NULL	检测器分配 ID 信息		01h
	02h	高	int32	检测器 0 的 AD 值	NULL	01h	00h
	03h	低	uint8	检测器 0 类型	NULL		01h
	04h	低	int16	NULL	检测器 0 档位/桥电流		02h
	05h	高	int32	检测器 1 的 AD	NULL	02h	00h

				值			
	06h	低	uint8	检测器1类型	NULL		01h
	07h	低	int16	NULL	检测器1档位/桥电流		02h
	08h	高	int32	检测器2的AD值	NULL	03h	00h
	09h	低	uint8	检测器2类型	NULL		01h
	0Ah	低	int16	NULL	检测器2档位/桥电流		02h
	0Bh	高	int32	检测器3的AD值	NULL	04h	00h
	0Ch	低	uint8	检测器3类型	NULL		01h
	0Dh	高	int16	NULL	检测器3档位/桥电流		02h
加热板	0Eh	低	real（IEEE754）	通道0温度	NULL	05h	00h
	0Fh	低	real（IEEE754）	通道1温度	NULL		01h
	10h	低	real（IEEE754）	通道2温度	NULL		02h
	11h	低	real（IEEE754）	通道3温度	NULL		03h
	12h	低	real（IEEE754）	通道4温度	NULL		04h
	13h	低	real（IEEE754）	通道5温度	NULL		05h
	14h	低	real（IEEE754）	通道6温度	NULL		06h

	15h	低	real（IEEE754）	通道 7 温度	NULL		07h
	16h	低	real（IEEE754）	NULL	通道 0 设定温度	06h	00h
	17h	低	real（IEEE754）	NULL	通道 1 设定温度		01h
	18h	低	real（IEEE754）	NULL	通道 2 设定温度		02h
	19h	低	real（IEEE754）	NULL	通道 3 设定温度		03h
	1Ah	低	real（IEEE754）	NULL	通道 4 设定温度		04h
	1Bh	低	real（IEEE754）	NULL	通道 5 设定温度		05h
	1Ch	低	real（IEEE754）	NULL	通道 6 设定温度		06h
	1Dh	低	real（IEEE754）	NULL	通道 7 设定温度		07h
	1Eh	低	real（IEEE754）	NULL	通道 0 温度保护值	07h	00h
	1Fh	低	real（IEEE754）	NULL	通道 1 温度保护值		01h
	20h	低	real（IEEE754）	NULL	通道 2 温度保护值		02h
	21h	低	real（IEEE754）	NULL	通道 3 温度保护值		03h
	22h	低	real（IEEE754）	NULL	通道 4 温度保护值		04h

	23h	低	real（IEEE754）	NULL	通道 5 温度保护值		05h
	24h	低	real（IEEE754）	NULL	通道 6 温度保护值		06h
	25h	低	real（IEEE754）	NULL	通道 7 温度保护值		07h
	26h	低	real（IEEE754）	NULL	通道0P设定	08h	00h
	27h	低	real（IEEE754）	NULL	通道1P设定		01h
	28h	低	real（IEEE754）	NULL	通道2P设定		02h
	29h	低	real（IEEE754）	NULL	通道3P设定		03h
	2Ah	低	real（IEEE754）	NULL	通道4P设定		04h
	2Bh	低	real（IEEE754）	NULL	通道5P设定		05h
	2Ch	低	real（IEEE754）	NULL	通道6P设定		06h
	2Dh	低	real（IEEE754）	NULL	通道7P设定		07h
	2Eh	低	real（IEEE754）	NULL	后开门P设定		08h
	2Fh	低	real（IEEE754）	NULL	通道0I设定		09h
	30h	低	real（IEEE754）	NULL	通道1I设定		0Ah
	31h	低	real（IEEE754）	NULL	通道2I设定		0Bh

	32h	低	real（IEEE754）	NULL	通道3I设定		0Ch
	33h	低	real（IEEE754）	NULL	通道4I设定		0Dh
	34h	低	real（IEEE754）	NULL	通道5I设定		0Eh
	35h	低	real（IEEE754）	NULL	通道6I设定		0Fh
	36h	低	real（IEEE754）	NULL	通道7I设定		10h
	37h	低	real（IEEE754）	NULL	后开门I设定		11h
	38h	低	real（IEEE754）	NULL	通道0D设定		12h
	39h	低	real（IEEE754）	NULL	通道1D设定		13h
	3Ah	低	real（IEEE754）	NULL	通道2D设定		14h
	3Bh	低	real（IEEE754）	NULL	通道3D设定		15h
	3Ch	低	real（IEEE754）	NULL	通道4D设定		16h
	3Dh	低	real（IEEE754）	NULL	通道5D设定		17h
	3Eh	低	real（IEEE754）	NULL	通道6D设定		18h
	3Fh	低	real（IEEE754）	NULL	通道7D设定		19h
	40h	低	real（IEEE754）	NULL	后开门D设定		1Ah

	41h	低	uint8	通道 0 工作状态信息	NULL	09h	00h
	42h	低	uint8	通道 1 工作状态信息	NULL		01h
	43h	低	uint8	通道 2 工作状态信息	NULL		02h
	44h	低	uint8	通道 3 工作状态信息	NULL		03h
	45h	低	uint8	通道 4 工作状态信息	NULL		04h
	46h	低	uint8	通道 5 工作状态信息	NULL		05h
	47h	低	uint8	通道 6 工作状态信息	NULL		06h
	48h	低	uint8	通道 7 工作状态信息	NULL		07h
	49h	低	uint8	载气保护状态	NULL	0Ah	00h
	4Ah	低	uint8	NULL	各通道工作开/关设定		01h
	4Bh	低	uint8	NULL	0 通道工作模式设定		02h
I/O 板	4Ch	低	uint8	NULL	点火开关	0Bh	00h
	4Dh	低	uint8	电火状态	NULL		01h
	4Eh	低	uint16	NULL	0 通道点火电流/功率大小		02h

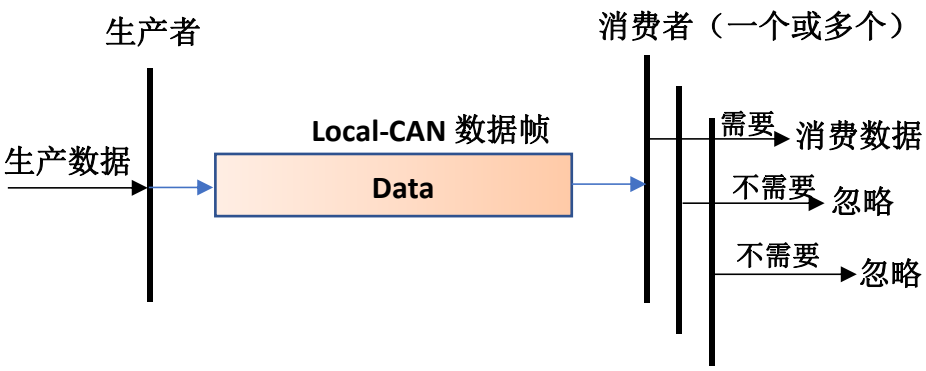
	4Fh	低	uint16	NULL	1 通道点火电流/功率大小		03h
	50h	低	uint16	NULL	2 通道点火电流/功率大小		04h
	51h	低	uint16	NULL	0 通道点火时间		05h
	52h	低	uint16	NULL	1 通道点火时间		06h
	53h	低	uint16	NULL	2 通道点火时间		07h
	54h	低	uint8	I/O 输出值	NULL		08h

注：表中不同底色在数据字典中为不同段。

4、基于生产者-消费者的通信模式

Local-CAN 传输数据是单向传输的，无需接收节点回应确认报文来确认，从通讯术语上来说是属于“生产消费”模型，接收端只有对数据有需求才会修改数据字典，没有需求则忽略，就像食品销售柜台，生产者摆出“食品”，但只有“需要”的消费者才会来买，没有指向性。这样做会大大减轻网络负载，提高通信的实时性。

图 6：生产者消费者通信模型



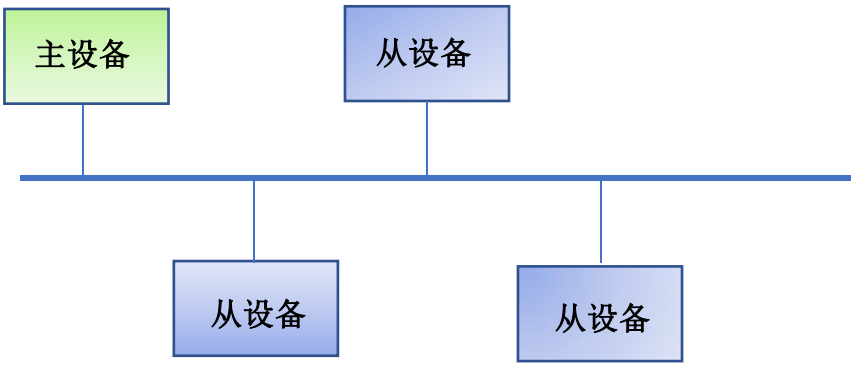
1、Local-CAN 主设备

在整个 Local-CAN 网络中，为了管理网络的负载调整、节点状态监控、ID 分配，需要在网络中确定一个主设备。这个主设备需要维护一个 ID 分配表，来记录每个设备的通信 ID 及各个设备的属性。并且维护一个完整的数据字典，来实时监控每个设备的状态。所以，主设备是所有从设备的消费者。

2、Local-CAN 从设备

在 Local-CAN 网络中，有多个从设备向主设备提供数据或接收数据，Local-CAN 的网络结构如下：

图 7：Local-CAN 网络结构图



3、Local-CAN 通信流程

Local-CAN 实现了主设备和从设备的数据字典相关联，从设备的数据字典的数据指针所指的数据区发生改变，Local-CAN 通信协议会把变化发送给其他设备，由于我们将 SID 与 BID 作为 CAN 帧 ID 的设计依据（具体参照章节 2），因此只有具有相同段的设备会收到该信息，并修改本设备的数据字典生产者数据指针所指的数据区。从用户角度看，好像主设备和从设备在维护同一个数据字典，设备只需要检测自己数据字典生产者数据指针所指的数据区的变化，不需要关注通信的过程，数据就好像是在自己的数据字典生产者数据指针所指的数据区中产生的。由于我们在字典的基础上引进了字典目录这一概念，这样可使用户按需查找自己的数据字典，方便数据的查找，从而实现用户数据字典总量为系统总字典的子集，可以降低用户数据区数据存储，同时，我们将字典目录动态设置，可实现用户的动态更改。

图 8：数据字典关联模型

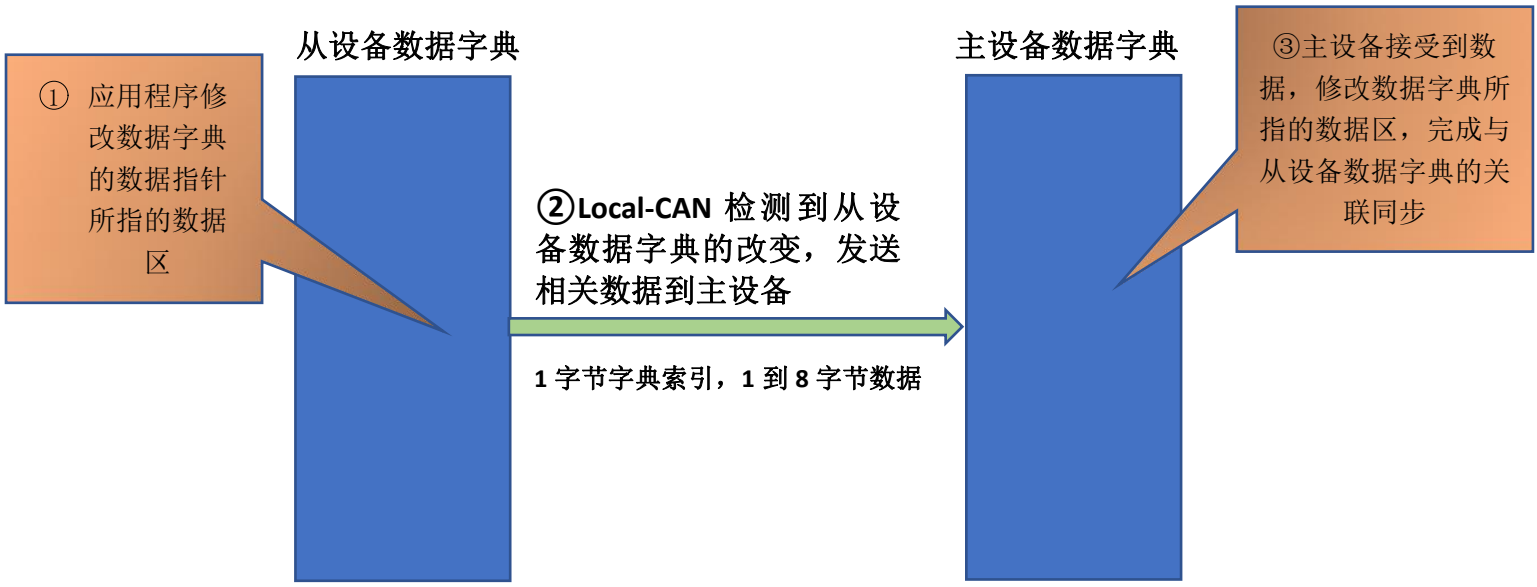
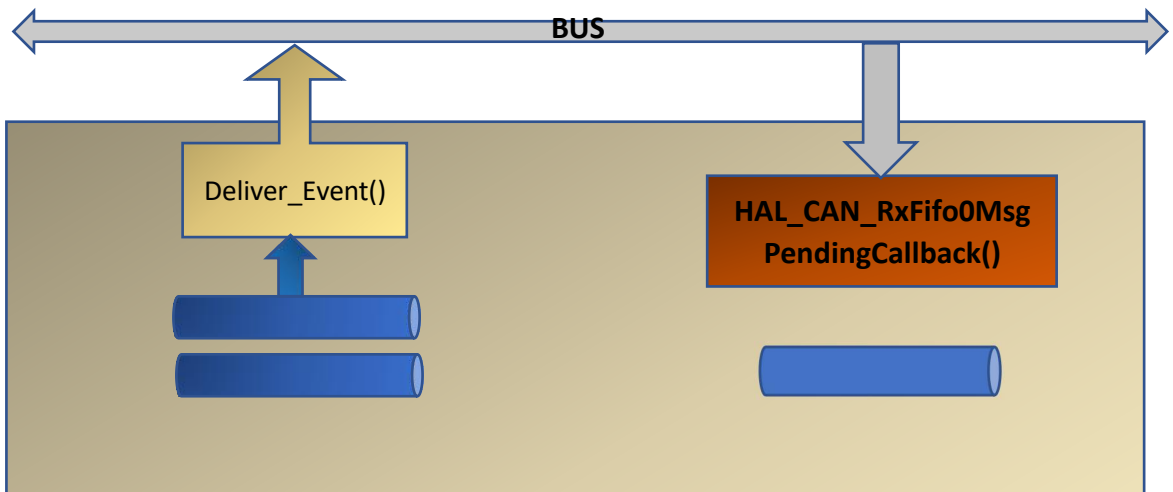
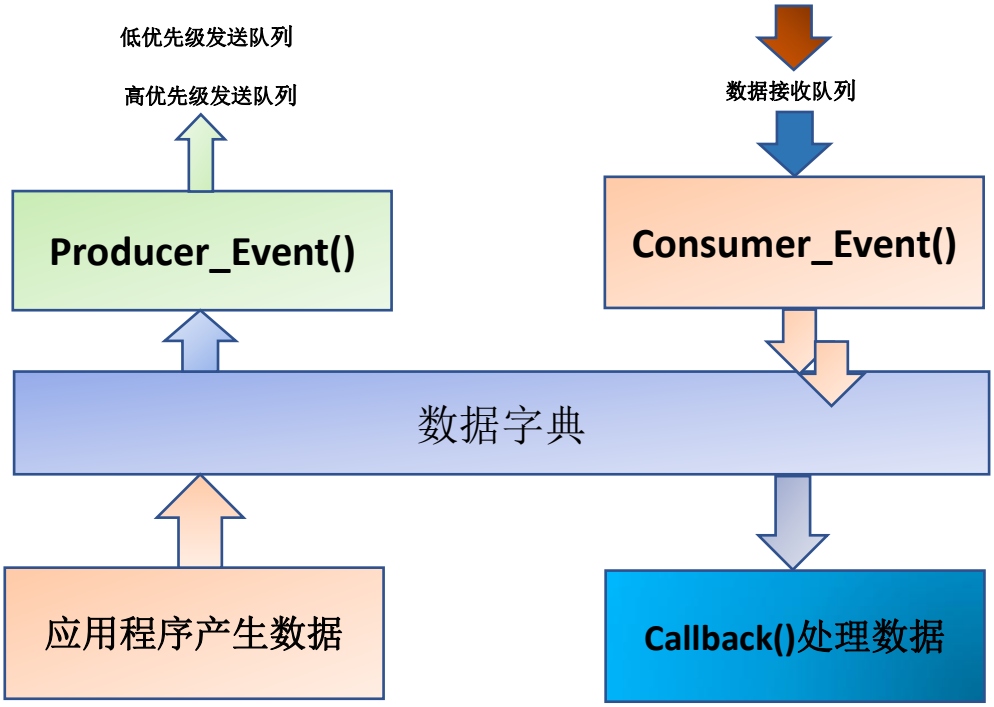


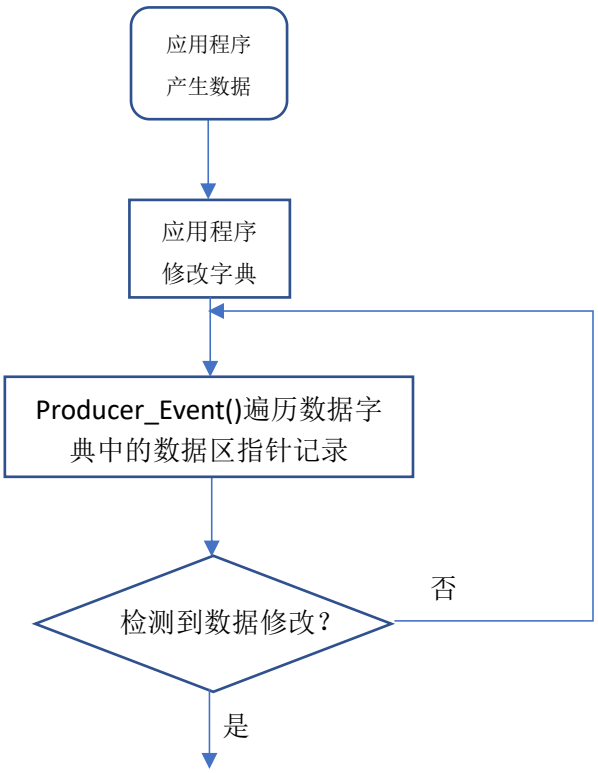
图 9：Loca-CAN 通信模型

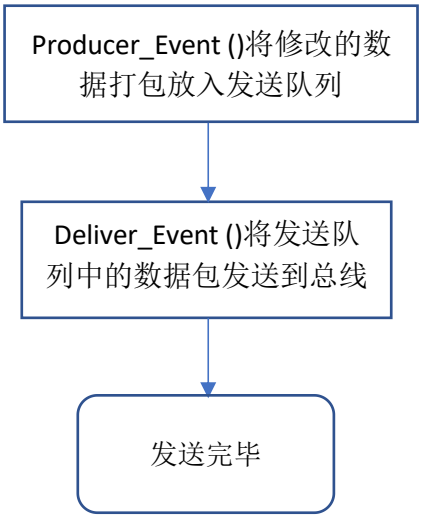




发送处理流程

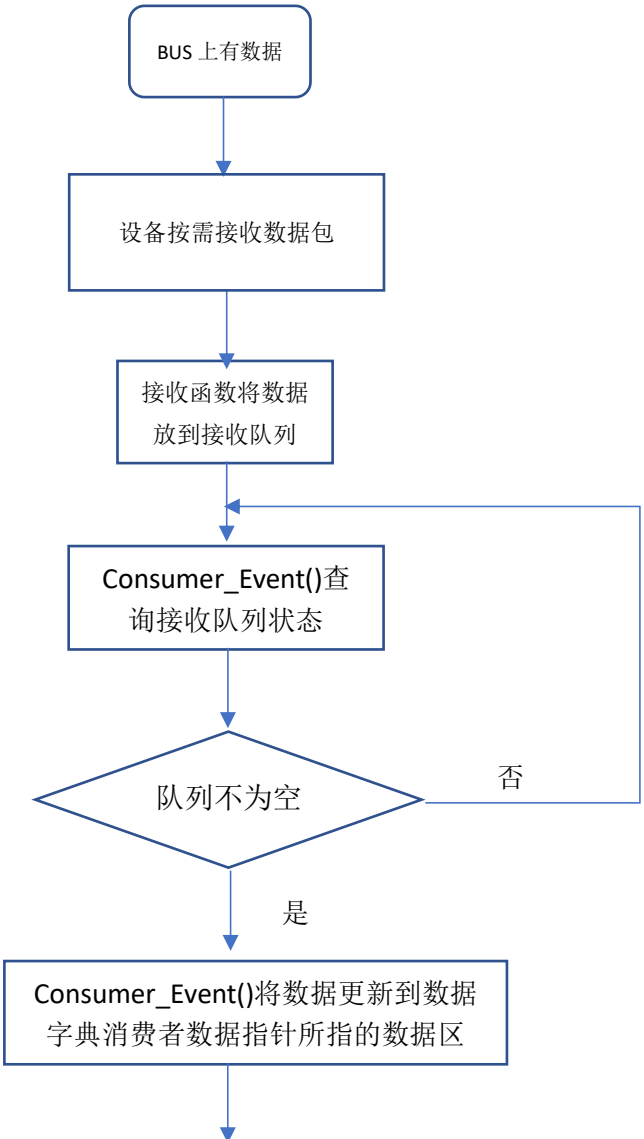
生产者：首先应用程序产生数据，修改数据字典记录，然后 Producer_Event()检测到数据字典的数据指针所指的数据区发生变化，将变化的数据字典记录打包成数据包，根据记录中的优先级选择放在合适的发送队列里，再由 Deliver_ Event ()根据优先级将数据包添加指定段内索引 S_Index、段号 SID 以及 BID 系统板号发出，完成数据发送。





接收处理流程

消费者：设备通过 HAL_CAN_RxFifo0MsgPendingCallback()函数将 BUS 上跟自己相关的的数据放到接收队列中（具体实现参照章节 5），Consumer_Event()函数不断地查看接收队列的状态信息，如果队列中有数据，Consumer_ Event ()将数据取出，根据数据的段号以及段内索引修改指定的数据字典记录，并调用索引对应的数据字典记录中的回调函数来完成相应的功能。



5、Local-CAN 中 CAN 过滤器

在本协议中，根据 Local-CAN 本身 CAN 帧的特点，根据标准数据域的标准 CAN 帧域扩展数据域中的 CAN 扩展帧，我们分别设置两种过滤器模式，以尽可能提高 CAN 总线效率。

5.1 标准 CAN 过滤器

当用户在标准帧模式下，本协议会将 CAN 过滤器模式设置为列表标识符模式，根据章节 2 中标准 CAN 帧格式，将 CAN 过滤器组做如下初始化配置。

首先扫描字典目录表，字典目录表（表格实例见表 3）的下标为段号，之后查看该字段段长是否为 0，为 0 说明该段在本设备屏蔽，不为 0 便根据当前段号以及用户所设置的 BID 设置 CAN 过滤器的 ID，采用 16 位过滤器模式，具体规则为：

CAN 过滤寄存器值：(SID+ (BID<<6))<<5;

由于采用了列表标识符模式，所以所设置的 ID 数量有限，在 STM32F1 系列，共有 14 个 CAN 过滤器组（其他系列可有 28 组），每组两个 32 位寄存器，因为我们设置的是 16 位模式，一共可以设置 56 个 ID，对于 Local-CAN 协议，这些 ID 已经足够。

5.2 扩展 CAN 过滤器

当用户在扩展帧模式下，本协议会将 CAN 过滤器模式设置为掩码模式，根据章节 2 中扩展 CAN 帧格式，将 CAN 过滤器组做如下初始化配置。

首先扫描字典目录表，字典目录表（表格实例见表 3）的下标为段号，之后查看该字段段长是否为 0，为 0 说明该段在本设备屏蔽，不为 0 便根据当前段号以及用户所设置的 BID 设置 CAN 过滤器的 ID，以及段内索引 S_Index，采用 32 位过滤器模式，具体规则为：

ID	CAN_FxR1[31:24]	CAN_FxR1[23:16]		CAN_FxR1[15:8]	CAN_FxR1[7:0]			
掩码	CAN_FxR1[31:24]	CAN_FxR1[23:16]		CAN_FxR1[15:8]	CAN_FxR1[7:0]			
映射	STID[10:3]	STID[2:0]	EXID[17:13]	STID[12:5]	STID[4:0]	IDE	RTR	0

不同颜色代表了不同的过滤器寄存器，为方便配置，将 32 位过滤器拆分成两个 16 位寄存器，根据第二章的 CAN 扩展帧格式，配置规则如下：

CAN 过滤寄存器 ID 值（高 16 位）： SID;

CAN 过滤寄存器 ID 值（低 16 位）： BID<<3;

CAN 过滤寄存器掩码值（高 16 位）： 0x1f;

CAN 过滤寄存器掩码值（低 16 位）： 0x00f8;

经此配置，CAN 设备可以选择接收本设备中设备字典用到的段的数据包，并且由于段内索引存放在 EXID 中，所以，可以将数据部分全部用于数据传递，极大的提高了传输数据的效率，满足了用户高实时性的需求。

由于采用了掩码模式，所设置的掩码数量有限，在 STM32F1 系列，共有 14 个 CAN 过滤器组（其他系列可有 28 组），每组两个 32 位寄存器，因为我们设置的是 32 位模式，一共可以设置 14 个 mask，对于 Local-CAN 协议，这些掩码已经足够。

表 3：字典目录表（例子）

起始 Index	段长
0	2
2	3

……	……
5001	200
5201	23

6、Local-CAN 中函数

在 Local-CAN 中，我们将部分函数封装完毕，用户在使用时，需按照一下步骤进行初始配置。

6.1 数据字典及目录

用户初次使用，首先根据需求填写用户字典，数据字典具体数据如下：

表 4：字典定义

字典内容	代表含义	数据类型
data_type	数据优先级	uint8_t
priority	本设备记录的属性	uint8_t
*consumer_object	消费者数据区指针	Void
* producer_object	生产者数据区指针	Void
*data_buffer	生产者仓库指针	Char
*Callback_func)(void *)	回调函数指针	Void

填写完成后，Local-CAN 将会进行维护此数据字典，同时要求用户根据需求填写字典目录表，系统运行后，用户可进行动态调整目录表，以此来设置 CAN 的 ID，实现动态调整消息数据包接收过滤段。

6.2 协议初始化

根据用户配置的数据字典目录，以及用户所选的模式（标准或扩展），协议

提供初始化函数，进行 CAN 过滤器的初始化配置。函数名如下：

void Set_Can_Filter(CAN_HandleTypeDef *hcan)

传递参数为：CAN 处理结构体

调用此函数后，协议会根据用户设定进行过滤器组的配置，具体为：

1. 若用户在标准帧模式下工作，此函数会将过滤器设置为列表标识符模式，本函数会扫描用户设置的字典目录，将字典目录中段长非空的段号记录，作为设置过滤 ID 的依据。扫描结束后，根据扫描结果，进行 CAN 过滤 ID 的设置，每 4 个 ID 为一组，对每过滤器组分别进行初始化。
2. 若用户在扩展帧模式下工作，此函数会将过滤器设置为掩码模式，跟标准模式一样，会继续扫描用户设置的字典目录，将字典目录中段长非空的段号记录，作为设置过滤 ID 的依据，在扩展模式下，用户根据段索引（S_Index），段号（SID），版号(BID)，进行过滤器 ID 设置，（具体见章节 2），同时，设置掩码，掩码的设置，我们只需接收 SID，BID 与我们需要的一样即可，而暂时忽略 S_Index，S_Index 交由用户进一步处理，掩码具体设置见章节五。

6.3 供用户使用的函数

初始化完成后，用户可以正常使用该协议，用户在变动完一个或者多个函数后，可以调用生产者函数 Producer_Event()，进行 CAN 包生产。当用户调用发布者函数 Deliver_Event()，该函数会检测两个发送队列是否有数据发送，函数会将 CAN 包封装成帧，发送到 BUS，以供其他消费者消费，并且将发送消息队列更新。两个函数具体过程如下：

Producer_Event()生产者函数，负责生产 CAN 数据包，将 CAN 数据包根据优先级放入不同的消息队列。调用此函数后，会根据用户字典目录遍历字典，当字典生产者数据指针指向的数据区发生变化（本函数会将此数据区数据指针与生产者仓库中的数据进行对比），会将此数据以及段号（SID），段内索引号（S_Index）打包，再根据此条记录的优先级放入不同的队列。重复上述过程，直至遍历完成，并返回给用户共产生多少 CAN 数据包。

Deliver_Event()发布者函数，负责将不同优先级发送队列的数据封装成帧，发送到总线。运行该函数后，当发送队列有数据发送，会将此数据包取出，根据数据包里的 SID,S_Index 进行发送 CAN ID 的设置，根据数据包里的数据类型，设置发送数据包的长度，最终将数据封装成 CAN 帧，交由 CAN 发送邮箱，发送数据。

当 BUS 总线上有数据包时，设备会根据设置的 CAN 过滤规则，接收特定的 CAN 数据包，其中,CAN 的接收中断处理函数会将数据包放到接收消息队列，等待下一步处理。

对于接收消息，用户可以调用消费者函数 Consumer_Event()，进行接收数据包的处理，具体过程如下：

本函数会检测接收消息队列，若接收消息队列中有数据包，则会根据数据帧，取出 SID 与 S_Index，首先判断 SID 与 S_Index 是否合法，若合法，则查找字典目录表，找到相应的的数据字典记录，更改数据字典生产者数据指针指向的数据区，同时，调用相应的回调函数，继续完成接收到本消息之后的其他事务。

用户可以方便的调用生产者函数 Producer_Event()，发布者函数 Deliver_Event()，消费者函数 Consumer_Event()，完成数据传递。实现不同设备的变量关联。

7、Local-CAN 特点

本协议 Local-CAN 设置的初衷就是为了实现小系统内不同设备中的变量关联，实现较小数据（小于等于 8 字节）的快速传输。为实现上述目的，我们采取了多种方法。

7.1 变量关联

我们在此协议中，参照 CANOpen 协议，引入了数据字典，因此我们传送数据的目的，就成了维护数据字典在不同设备上相关数据的一致性，即相关变量关联。CANOpen 数据字典庞大繁琐，在此基础上，我们简化了字典设计，字典仅存储数据类型，数据优先级，生产者数据指针，消费者数据指针，生产者仓库，

回调函数指针。这样得到的数据字典简介容易理解，用户可以快速熟悉并应用。

在数据字典的基础上，我们参照操作系统的段表结构，引入字典目录表这一概念，我们建议用户将字典数据按照用途进行分类，这样，可以快速找到字典记录，从而加快数据访问速度，实现不同设备之间的变量关联。

7.2 数据传送方式

CAN 协议虽规定了 CAN 的标识符，但并不能在物理上区分板块，但 CAN 的标识符可以决定发送的优先级，原理如下：

CAN 标准数据帧：

帧起始	仲裁段	控制段	数据段	CRC 段	ACK 段	帧结束

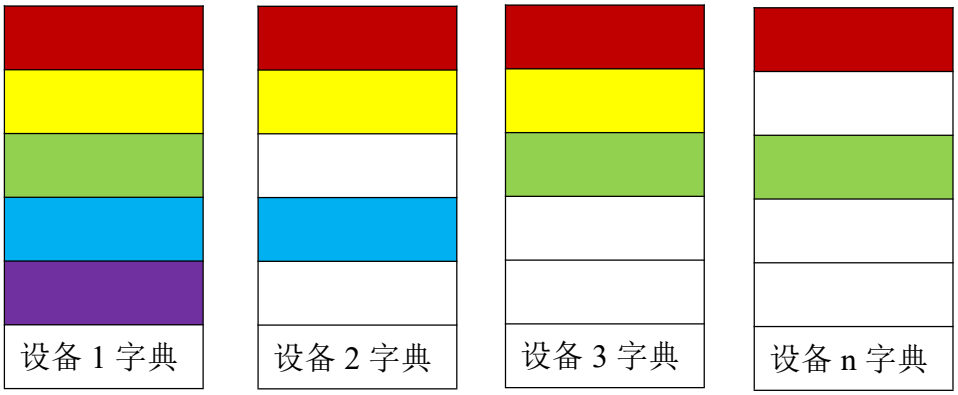
数据帧以一个显性位(逻辑 0)开始，以 7 个连续的隐性位(逻辑 1)结束，在它们之间，分别有仲裁段、控制段、数据段、CRC 段和 ACK 段。

当同时有两个报文被发送时，总线会根据仲裁段的内容决定哪个数据包能被传输，这也是它名称的由来。

仲裁段是由本数据帧的 ID 信息(标识符)作为优先级仲裁依据，其中 CAN 数据帧是具有标准格式以及扩展格式两种，他们的区别就在于标识符的长度，标准格式的 CAN ID 为 11 位，扩展帧格式的 CAN ID 为 29 位，它是在标准 ID 的基础上多出 18 位。在 CAN 协议中，ID 信息起着重要的作用，它决定着数据帧发送的优先级，也决定着其它节点是否会接收这个数据帧。CAN 协议不对挂载在它之上的节点分配优先级和地址，对总线的占有权是由信息的重要性决定的，即对于重要的信息，我们会给它打包上一个优先级高的 ID，使它能够及时地发送出去。也正因为它这样的优先级分配原则，使得 CAN 的扩展性大大加强，在总线上增加或减少节点并不影响其它设备。

CAN 总线发送的数据帧实际是广播的形式，这是 CAN 的特点，为了提高数据的效率，对于本设备用不到的帧，是不需要接收的，基于此，我们利用 CAN 过滤器组，可以方便的设置多种数据发送形式。

下图代表了一个系统内不同设备的字典使用情况，不同颜色代表不同的数据段，白色代表在本设备中，该数据段未启用，即该数据段对本设备无效。



1. 广播的实现

可以看到，该系统每个字典都启用了红色数据段，即当某一设备发出红色数据段上的数据时，该系统内所有的设备都会接受该数据，从而实现广播

2. 组播的实现

上图中，设备 1，2，3 均启用了黄色数据段，当他们之中有一个发出数据后，其他两个设备会接收到此数据，而设备 n 并不会接收此数据，从而实现了组播。

3. 单播的实现

上图中，设备 1，2，均启用了蓝色数据段，当他们之中有一个发出数据后，另外一个备会接收到此数据，而其他设备并不会接收此数据，从而实现了单播。

此外，根据 CAN 协议发送消息优先级的设定，我们可以将紧急数据安排到 SID 较低的段，这样，在发送数据时会获得较高优先级。由于两种模式下 CAN 过滤器组是根据用户字典目录表进行配置的，所以用户可以动态的更改设备对于每一段的接收方式。

7.3 高效率

数据字典是基于指针的，能够实现对数据字典的快速检索，同时，为了加快字典的遍历，我们采用段式结构，引入字典目录表，这样便大大提高了数据字典的遍历速度，能够高效的处理数据。同时，我们根据用户设置的数据优先级，将

不同优先级的数据放入不同的数据发送队列，这样保证了优先级高的数据能够快速发送，尽最大努力提高数据的实时性。在发送数据方面，我们将要发送的数据与仓库数据作比较，采用指针的方式进行比较，如果两次数据没有发生变化，便不进行数据的发送，从而减少 CAN 总线的数据量，提高数据的实时性。