打印单元管理系统说明

|  |  |  |  |
| --- | --- | --- | --- |
| 版本号 | 更新日期 | 更新内容 | 操作者 |
| V1 | 2016/8/13 | 创建本文档 | 李培锋 |
|  |  |  |  |

目录

[1 数据结构定义 1](#_Toc458858348)

[1.1 订单表结构的定义 1](#_Toc458858349)

[1.1.1 概要 1](#_Toc458858350)

[1.1.2 代码文件位置 1](#_Toc458858351)

[1.2 批次数据表的定义 1](#_Toc458858352)

[1.2.1 概要 1](#_Toc458858353)

[1.2.2 代码文件位置 2](#_Toc458858354)

[1.3 状态报文类型 2](#_Toc458858355)

[1.3.1 概要 2](#_Toc458858356)

[1.3.2 代码文件位置 3](#_Toc458858357)

[1.4 状态报文底层封装 4](#_Toc458858358)

[1.4.1 概要 4](#_Toc458858359)

[1.4.2 代码文件位置 5](#_Toc458858360)

# 数据结构定义

## 订单表结构的定义

### 概要

根据数据交互表所定义的订单格式，创建了如下结构以记录订单的相关信息，其定义如下所示：

typedef struct order\_information {

u32\_t mcu\_id; //主控板id

u32\_t sever\_send\_time; //服务器发送时间

u32\_t serial\_number; //订单序号

u16\_t size; //订单长度

u16\_t batch\_number; //所属批次

u16\_t batch\_within\_number; //批次内序号

u16\_t check\_sum; //校验和

u16\_t preservation; //保留

u16\_t data\_source; //数据来源

u8\_t \*data; //数据域

u8\_t priority; //优先级，判断是否为加急订单

u8\_t status; //状态

u8\_t next\_print\_node; //下一打印结点

}order\_info; /\* 订单数据结构 \*/

### 代码文件位置

代码位于[USER/data\_form.h](../../USER/data_form.h)。

## 批次数据表的定义

### 概要

服务器是以批次的形式去发送订单给打印机的，因此需要一个记录批次信息的数据表来保存这些信息。

初始时，会创建容量为10的批次表，之后对每个接收的批次的序号去其哈希值，将其放置在对应的表中，尔后若订单需要批次的信息时，可访问该批次表。

批次表数据结构定义见下：

typedef struct batch\_information {

#define MAX\_BATCH\_NUM 10

#define MAX\_BATCH\_HEAD\_LENGTH 20

u32\_t sever\_send\_time; //服务器发送时间

u16\_t order\_number; //订单数量

u16\_t batch\_length; //批次长度

u16\_t batch\_number; //所属批次

u16\_t check\_sum; //校验和

u16\_t preservation; //保留字段，最后一位是紧急指针

u8\_t num\_printed\_order; //已打印完成订单数量

u8\_t num\_order\_que;

}batch\_info; /\* 批次数据表结构 \*/

批次表变量定义如下:

batch\_info batch\_info\_table[MAX\_BATCH\_NUM]; //批次表

//变量定义于[USER/print\_queue.c](../../USER/print_queue.c)

### 代码文件位置

代码位于[USER/data\_form.h](../../USER/data_form.h)。

## 状态报文类型

### 概要

为实现实时状态反馈机制，故需要建立起一个状态反馈系统。并且，对状态反馈类型还要进行分类。

为统一操作，所有的状态反馈报文一致，但其中有几个字节用于定义报文类型及各类型下的状态，详情见**数据交互表**。

下列定义给出各种状态反馈报文的类型：

typedef enum Req\_Type {

first\_req, //初次请求数据链接

order\_req, //达到阈值请求订单

batch\_status, //批次状态

order\_status, //订单状态

printer\_status //打印机状态

}req\_type; /\* 状态反馈类型枚举 \*/

以上枚举仅仅是为了在发送报文时定义发送报文为何种类型，实际的状态报文标志为以下宏定义：

#define BATCH\_STATUS 0x00 //批次状态

#define ORDER\_STATUS 0x20 //订单状态

#define PRINTER\_STATUS 0x40 //打印机状态

#define FIRST\_REQ 0x60 //初次请求链接

#define ORDER\_REQ 0x80 //达到阈值请求订单

#define ACK\_STATUS 0xa0 //本地状态应答

### 代码文件位置

更多详细的定义位于[USER/data\_form.h](../../USER/data_form.h)及[USER/status.mesg.h](../../USER/status_mesg.h)。

## 状态报文底层封装

### 概要

当确定好要发送的状态反馈类型及标志之后，便可以对其进行封装以发送到服务器或本地数据接收端（如PC或手机APP）。而发送部分在发送之前需调用该接口才能发送正确的数据。但由于被实现了若干层封装，故反馈状态线程无需考虑此部分。

底层实现API如下：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*@Name................: Pack\_Req\_Or\_Status\_Message

\*@Description......: 封装数据报

\*@Parameters......: message :用于存放即将打包的消息

\* type :报文类型

\* symbol :标志位

\* id :可以是主控板id(32位),也可以批次序号(高16位)

\* UNIX\_time :服务器或主控板发送时间；在本地状态应答时，为

区分打印机或订单应答

\* preservation :保留字段，在订单中，高16位为批次序号，低16

位为批次内序号；在本地状态应答时为应答序号

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Pack\_Req\_Or\_Status\_Message(char \*message, req\_type type, u8\_t symbol, u32\_t id, u32\_t UNIX\_time, u32\_t preservation)

{

u16\_t check\_sum;

/\*起始符\*/

message[0] = '\xCF';

message[1] = '\xFC';

/\*设置类型和标志\*/

message[2] = type;

message[3] = symbol;

SET\_DATA\_4B(&message[4], id);//设置id

SET\_DATA\_4B(&message[8], UNIX\_time);//设置Unix时间戳

SET\_DATA\_4B(&message[12], preservation);//填充段

/\*终止符\*/

message[18] = '\xFC';

message[19] = '\xCF';

/\*获取校验和\*/

check\_sum = Check\_Sum((u16\_t\*)message, SEND\_DATA\_SIZE);

SET\_DATA\_2B(&message[16], ((check\_sum << 8) + (check\_sum >> 8)));

}

上述代码中的**SET\_DATA\_4B**是一个宏，用于将4字节的数据填充进以某一地址开始的4个字节中，**SET\_DATA\_2B**同理。

宏定义于[USER/pack\_data.h](../../USER/pack_data.h)，此外还有**ANALYZE\_DATA\_4B**及**ANALYZE\_DATA\_2B**，其行为与**SET**相反，用于解析接收的数据，定义于[USER/analyze\_data.h](../../USER/analyze_data.h)。

### 代码文件位置

代码位于[USER/pack\_data.c](../../USER/pack_data.c)。