神思小额支付终端通信协议标准规范

V0.0.1

版本历史记录

序号	版本号	作者	修改时间	修改内容
1	V0.0.1		2016.08.11	创建
2	V0.0.2		2016.09.19	1、修改文件导出协议，详见 1.2 章节和 3.12 章节。 2、修改数字按键操作以及返回键功能。详见 3.2 章节和 3.8 章节

文档说明

本文档依据广东香山衡器集团股份有限公司（香山衡器）与神思电子技术股份有限公司（神思电子）签署的《战略合作伙伴框架协议》，对小额支付电子秤分体机项目配套商用秤、水产秤、挂钩秤通信协议进行详细说明。

本文档使用对象包括香山衡器开发人员、商务人员、品管人员，神思电子开发人员、采供人员、品管人员。用于指导产品开发、生产、品质验收、供货及维修服务等工作。

本文档仅用于香山衡器和神思电子内部交流使用，严禁泄露。

1、指令集定义

1.1、结果状态值定义

所有命令均由计价秤发起，刷卡终端接收到命令后，进行数据校验，校验正确后刷卡终端发响应码给计价秤。计价秤在 1 秒钟未收到刷卡终端返回的响应信号，将重新发送该数据，至多重复发送三次，如三次都未收到刷卡终端的响应信号，终止本次数据传送。

结果状态值由刷卡终端发送给电子计价秤，提示本次指令的操作结果。暂定义三种状态：成功，失败，不支持，可参考下面表格。

序号	状态	结果状态值	备注
1	成功	0x90	
2	失败	0x41	除不支持以外的所有失败
3	不支持	0x6D	

备注：在失败的状态值之后的数据区可以跟随失败的原因，也可以没有数据。

1.2、终端指令定义

目前电子秤提供指令如下表所示：

序号	指令名称	指令描述	指令值（16 进制格式）
1	消费	直接按下“挥卡/确认”键	0X3000
2	返回	直接按下“返回”键	0X3001
3	手输金额消费	手输金额后按“挥卡/确认”键	0X3002
4	菜单/下翻	直接按下“菜单/下翻”键	0X3003
5	累计	直接按下“累计”键	0X3004
6	累计清除	直接按下“累清”键	0X3005
7	累计消费	累计消费后按下“挥卡/确认”键	0X3006
8	数字键	按下菜单键后进入菜单状态，该状态下按数字键发送本次键值，按取消键后推出菜单状态，按数字键不反应	0X3007
9	获取终端版本号	请求获取刷卡终端的型号、软件和硬件版本号	0X7F01
~~10~~	~~获取交易~~	~~请求获取刷卡终端的交易流~~	~~0X7E00~~

	~~流水数目~~	~~水总数~~	
~~**11**~~	~~获取 N 条交易流水~~	~~请求获取刷卡终端指定的 N 条流水~~	~~0X7E01~~
12	文件上传下载指令	用于终端流水导出	0X3008

2、协议组包格式

2.1、请求数据包格式

协议请求数据包格式定义如下

HEAD	LEN	CMDR	DATA	CRC

格式说明：

代码	描述	长度	备注
HEAD	数据头 5 个字节固定值+1 字节地址字节，按顺序排列	6 字节	固定值如下： 0x53 0x44 0x73 0x45 0x73(字符串 SDsEs 的 ASCII 码)。 定义地址字节 0xFF 为广播地址，0x00 表示对地址值不进行处理。其他可分配
LEN	长度值 是从 CMDR 开始（包括 CMDR）到 CRC（包括 CRC）之间字节数的值。 4 字节，组包时长度值高位字节在前，低位字节在后	4 字节	举例： 假设长度值为 512，即 0x00000200，则 LEN 部分应该表示为：0x00,0x00,0x02,0x00
CMDR	请求命令值 命令值参考 1.2 部分具体定义。 2 字节。组包时命令值高位字节在前，低位字节在后	2 字节	举例： 假设命令值为 0x3000，则 CMDR 部分应该为 0x30,0x00
DATA	数据 包含电子秤要发送的数据内容 N 字节，可以为 0 字节	N 字节	
CRC	校验字节 采用 CRC-16 校验	2 字节	CRC-16 具体算法实现请参考附录

	从 LEN 开始（包括 LEN）到 DATA（包括 DATA）之间数据按字节进行计算的值。 2 字节		A 部分

2.2、应答数据包格式

协议应答数据包格式定义如下：

HEAD	LEN	CMDA	STATE	DATA	CRC

格式说明：

代码	描述	长度	备注
HEAD	数据头 5 个字节固定值+1 字节地址字节，按顺序排列	6 字节	固定值如下： 0x53 0x44 0x73 0x45 0x73(字符串 SDsEs 的 ASCII 码)。 定义地址字节 0xFF 为广播地址，0x00 表示对地址值不进行处理。其他可分配
LEN	长度值 是从 CMDA 开始（包括 CMDA）到 CRC（包括 CRC）之间字节数的值。 4 字节，组包时长度值高位字节在前，低位字节在后	4 字节	举例： 假设长度值为 512，即 0x00000200，则 LEN 部分应该表示为：0x00,0x00,0x02,0x00
CMDA	应答命令值 2 字节，是命令值 CMDR 异或上 0x8000 得到的值	2 字节	举例 假设请求命令 CMDR=0x3000，则应答命令 CMDA 的值为 0xB000
STATE	状态值 1 字节，具体参考结果状态值定义	1 字节	举例： 0x90 表示 CMDR 指令执行成功 0x41 表示 CMDR 指令执行失败
DATA	数据 要发送给电子秤的数据内容 N 字节，可以是 0 字节	N 字节	
CRC	校验字节 2 字节，采用 CRC-16 校验 是从 LEN 开始（包括 LEN）到 DATA	2 字节	CRC-16 具体算法实现请参考附录 A 部分

	（包含 DATA）之间数据按字节进行计算的值		

3、终端指令详解

3.1、“挥卡/确认”键命令

本指令用于提供给刷卡终端交易的详细信息，供刷卡终端进行挥卡消费以及交易数据保存等操作。

数据请求包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	4+N
CMDR	命令值	2 字节	0X3000
DATA	数据	N 字节	**数据格式详见 3.1.1**
CRC	校验字节	2 字节	计算值

成功应答数据包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	5
CMDA	命令值	2 字节	0XB000
STATE	状态值	1 字节	0x90
DATA	数据	0 字节	
CRC	校验字节	2 字节	计算值

失败或者不支持应答包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	5+N
CMDA	命令值	2 字节	0XB000
STATE	状态值	1 字节	0x41 或 0x6D

DATA	数据	0 字节	错误码、错误信息或不填（0 字节）
CRC	校验字节	2 字节	计算值

3.1.1、数据格式详解

“挥卡/确认”键命令：数据域应包含本次操作的**重量、单价、总价、PLU 码、追源码**等信息。电子秤把该数据发送给刷卡终端，终端对该部分数据进行解析，执行相应操作。

注：重量、单价、金额某项为非正常值时该字段全填 0，PLU 码以及溯源码不存在或不支持时，可不填。

数据格式采用 TLV 格式，数据域各个单项的标签值定义见下表：

名称	标签值（T）	长度（L）	值（V）
重量	0xA1	5 字节	单位:克，最大 99999 克，不够 5 位左补“0”，每个字节取值 0x30 到 0x39
单价	0x A2	5 字节	单位:分，最大 99999 分，不够 5 位左补“0”，每个字节取值 0x30 到 0x39
总价	0x A3	10 字节	单位:分，最大 9999999999 分，不够 10 位左补“0”，每个字节取值 0x30 到 0x39
PLU 码	0x A4	VAL(变长)	ASCII 码，4 字节或 5 字节
追溯码	0x A5	VAL(变长)	ASCII 码，N 字节追溯码
RFU	——	——	——
RFU	——	——	——

举例：重量为 9876 克，单价为 500 分/Kg，总价为 4938 分，PLU 码为 4133
追溯码为 06900000545331，则数据组包格式如下：
A1 05 30 39 38 37 36 A2 05 30 30 35 30 30 A3 0A 30 30 30 30 30 30 34 39 33 38 A4 04 34 31 33 33 A5 0E 30 36 39 30 30 30 30 30 35 34 35 33 33 31

3.2、“返回”键命令

本指令用于刷卡终端取消本次操作以及返回待机状态的功能。

注：返回上一层功能取消，可以从界面上通过数字键来实现该功能，【返回】按键作为退出菜单状态专用，同时【返回】键执行取消本次操作的功能（该功能通过软件实现，取消操作后即退出菜单状态）。

数据请求包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	4
CMDR	命令值	2 字节	0X3001
DATA	数据	0 字节	
CRC	校验字节	2 字节	计算值

举例：该命令的数据为：

53 44 73 45 73 00　00 00 00 04　30 01　C9 74

HEAD　LEN　CMDR　CRC

成功应答数据包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	5
CMDA	命令值	2 字节	0XB001
STATE	状态值	1 字节	0x90
DATA	数据	0 字节	
CRC	校验字节	2 字节	计算值

53 44 73 45 73 00　00 00 00 05　B0 01　90　F2 32

失败或者不支持应答包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	5+N
CMDA	命令值	2 字节	0XB001
STATE	状态值	1 字节	0x41 或 0x6D
DATA	数据	0 字节	错误码、错误信息或不填（0 字节）

CRC	校验字节	2 字节	计算值

53 44 73 45 73 00　00 00 00 05　B0 01　41　29 6E

3.3、手输金额消费命令

本指令提供在不称重的情况下直接输入固定金额进行消费的功能。具体操作是，首先按电子秤“金额”键，然后按数字键进行金额输入，最后按“挥卡/确认”键，发送消费指令。

数据请求包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	4+N
CMDR	命令值	2 字节	0X3002
DATA	数据	N 字节	**数据格式详见 3.3.1**
CRC	校验字节	2 字节	计算值

成功应答数据包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	5
CMDA	命令值	2 字节	0XB002
STATE	状态值	1 字节	0x90
DATA	数据	0 字节	
CRC	校验字节	2 字节	计算值

失败或者不支持应答包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	5+N
CMDA	命令值	2 字节	0XB002
STATE	状态值	1 字节	0x41 或 0x6D
DATA	数据	0 字节	错误码、错误信息或不填（0 字节）
CRC	校验字节	2 字节	计算值

3.3.1、数据格式详解

该字段只包括消费的总金额，格式采用 TLV，总价标签值为 0xA3，长度为 10 字节，详细参考 3.1.1。

举例：手输金额为 234567 分，该数据域数据如下：

A3 0A 30 30 30 30 32 33 34 35 36 37

3.4、“菜单/下翻”键命令

本指令提供刷卡终端进行菜单选择操作，通过该指令终端可以完成菜单的翻页，选择等操作。

数据请求包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	4
CMDR	命令值	2 字节	0X3003
DATA	数据	0 字节	
CRC	校验字节	2 字节	计算值

成功应答数据包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	5
CMDA	命令值	2 字节	0XB003
STATE	状态值	1 字节	0x90
DATA	数据	0 字节	
CRC	校验字节	2 字节	计算值

失败或者不支持应答包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	5+N
CMDA	命令值	2 字节	0XB003

STATE	状态值	1 字节	0x41 或 0x6D
DATA	数据	0 字节	错误码、错误信息或不填（0 字节）
CRC	校验字节	2 字节	计算值

3.5、“累计”键命令

本指令用于提供给刷卡终端单次累计的详细信息，供刷卡终端进行数据存储操作。

数据请求包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	4+N
CMDR	命令值	2 字节	0X3004
DATA	数据	N 字节	**数据格式详见 3.5.1**
CRC	校验字节	2 字节	计算值

成功应答数据包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	5
CMDA	命令值	2 字节	0XB004
STATE	状态值	1 字节	0x90
DATA	数据	0 字节	
CRC	校验字节	2 字节	计算值

失败或者不支持应答包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	5+N
CMDA	命令值	2 字节	0XB004
STATE	状态值	1 字节	0x41 或 0x6D
DATA	数据	0 字节	错误码、错误信息或不填（0 字节）
CRC	校验字节	2 字节	计算值

3.5.1、数据格式详解

“累计”键命令：数据域应包含本次累计的**重量、单价、总价、PLU 码、追源码、次数**等信息。电子秤把该数据发送给刷卡终端，终端对该部分数据进行解析存储。

数据格式采用 TLV 格式，数据域各个单项的标签值定义见下表：

名称	标签值（T）	长度（L）	值（V）
重量	0xA1	5 字节	单位:克，最大 99999 克，不够 5 位左补“0”，每个字节取值 0x30 到 0x39
单价	0x A2	5 字节	单位:分，最大 99999 分，不够 5 位左补“0”，每个字节取值 0x30 到 0x39
本次累计总价	0x A3	10 字节	单位:分，最大 9999999999 分，不够 10 位左补“0”，每个字节取值 0x30 到 0x39
PLU 码	0x A4	VAL(变长)	ASCII 码，4 字节或 5 字节
追溯码	0x A5	VAL(变长)	ASCII 码，N 字节追溯码
累计次数	0xB0	1 个字节	该字段表示第几次累计
RFU	——	——	——

举例：重量为 9876 克，单价为 500 分/Kg，总价为 4938 分，PLU 码为 4133
追溯码为 06900000545331，第 2 次按累计，则数据组包格式如下：
A1 05 30 39 38 37 36 A2 05 30 30 35 30 30 A3 0A 30 30 30 30 30 30 34 39 33 38 A4 04 34 31 33 33 A5 0E 30 36 39 30 30 30 30 30 35 34 35 33 33 31 B0 01 02

3.6、“清除/累清”键命令

只有“累计”键被按下后，按“清除/累计”键才会发送该命令。本指令用于刷卡终端进行累计的清除操作。避免终端和秤底座统计不同步。

数据请求包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	4
CMDR	命令值	2 字节	0X3005
DATA	数据	0 字节	
CRC	校验字节	2 字节	计算值

成功应答数据包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	5
CMDA	命令值	2 字节	0XB005
STATE	状态值	1 字节	0x90
DATA	数据	0 字节	
CRC	校验字节	2 字节	计算值

失败或者不支持应答包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	5+N
CMDA	命令值	2 字节	0XB005
STATE	状态值	1 字节	0x41 或 0x6D
DATA	数据	0 字节	错误码、错误信息或不填（0 字节）
CRC	校验字节	2 字节	计算值

3.7、累计消费命令

本指令用于累计结束进行刷卡消费。具体操作，累计结束后，按“挥卡/确认”键，把累计的总金额以及累计总次数发送给刷卡终端。该命令需要“累计”键和“挥卡/确认”配合使用。

数据请求包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	4+N
CMDR	命令值	2 字节	0X3006
DATA	数据	N 字节	**数据格式详见 3.7.1**
CRC	校验字节	2 字节	计算值

成功应答数据包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	5
CMDA	命令值	2 字节	0XB006
STATE	状态值	1 字节	0x90
DATA	数据	0 字节	
CRC	校验字节	2 字节	计算值

失败或者不支持应答包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	5+N
CMDA	命令值	2 字节	0XB006
STATE	状态值	1 字节	0x41 或 0x6D
DATA	数据	0 字节	错误码、错误信息或不填（0 字节）
CRC	校验字节	2 字节	计算值

3.7.1、数据格式详解

累计消费命令：累计结束后按“挥卡/确认”键，发送的数据域应包含所有累计的**总价、总次数**。

数据格式采用 TLV 格式，数据域各个单项的标签值定义见下表：

名称	标签值（T）	长度（L）	值（V）
累计总价	0xA6	10 字节	单位:分，最大 9999999999 分，不够 10 位左补“0”，每个字节取值 0x30 到 0x39
累计总次数	0x B1	1 字节	本次累计总的累计次数

举例：累计总价为 4938 分，累计总次数为 9 次，则数据组包格式如下：

A6 0A 30 30 30 30 30 30 34 39 33 38 B1 01 09

3.8、数字键命令

本指令传递秤底座上的数字按键（包括“.”）的键值。

注：数字键只有在菜单状态下被按下才会发送键值，其他状态下无反应。按【菜单】键进入菜单状态，按【返回】键退出菜单状态。

数据请求包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	5
CMDR	命令值	2 字节	0X3007
DATA	数据	1 字节	键值 例如：键值是“3”，该处应填 0x33
CRC	校验字节	2 字节	计算值

成功应答数据包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	5
CMDA	命令值	2 字节	0XB007
STATE	状态值	1 字节	0x90
DATA	数据	0 字节	
CRC	校验字节	2 字节	计算值

失败或者不支持应答包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	5+N
CMDA	命令值	2 字节	0XB007
STATE	状态值	1 字节	0x41 或 0x6D
DATA	数据	0 字节	错误码、错误信息或不填（0 字节）
CRC	校验字节	2 字节	计算值

3.9、获取终端版本号

本指令用于获取终端的具体型号和软件、硬件的版本号。终端型号的格式定义为ASCII字符的样式，（使用‘$’字符定义起始和结束），参考格式：,$TYPE=SS728Z03P$$SOFTVER=V1.00$$HARDVER=V1.00$'。

备注：本指令必须支持，且返回成功的应答。

数据请求包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	4
CMDR	命令值	2 字节	0X7F01
DATA	数据	0 字节	无
CRC	校验字节	2 字节	计算值

成功应答数据包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	5+N
CMDA	命令值	2 字节	0XFF01
STATE	状态值	1 字节	0x90
DATA	数据	N 字节	
CRC	校验字节	2 字节	计算值

~~3.10、获取交易流水数目~~

~~本指令用于获取终端存储交易流水的总条数，终端收到该请求后返回存储的流水的总条数。~~

~~数据请求包格式：~~

~~代码~~	~~描述~~	~~长度~~	~~值~~
~~**HEAD**~~	~~数据头，固定值~~	~~6 字节~~	~~“SDsEs”+地址值~~
~~**LEN**~~	~~长度值~~	~~4 字节~~	~~4~~
~~**CMDR**~~	~~命令值~~	~~2 字节~~	~~0X7E00~~

DATA	数据	0 字节	无
CRC	校验字节	2 字节	计算值

成功应答数据包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	5+2
CMDA	命令值	2 字节	0XFE00
STATE	状态值	1 字节	0x90
DATA	数据	2 字节	流水数目：最大值 0xFFFF， 例如 512 条，该值为 0x02 0x00
CRC	校验字节	2 字节	计算值

失败或者不支持应答包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	5+N
CMDA	命令值	2 字节	0XFE00
STATE	状态值	1 字节	0x41 或 0x6D
DATA	数据	0 字节	错误码、错误信息或不填（0 字节）
CRC	校验字节	2 字节	计算值

3.11、获取 N 条交易流水

该指令用于获取终端存储的交易流水，终端根据请求获取流水的数目来打包数据。

数据请求包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	4+4
CMDR	命令值	2 字节	0X7E01
DATA	数据	4 字节	请求获取流水的地址及数目 前两个字节为获取流水的地址，后

			两个字节为获取流水的条数。 例如从第 512 条开始取 10 条，该值应填 0x02 0x00 0x00 0x0A
CRC	校验字节	2 字节	计算值

成功应答数据包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	5+N
CMDA	命令值	2 字节	0XFE01
STATE	状态值	1 字节	0x90
DATA	数据	N 字节	请求的固定条数交易流水信息
CRC	校验字节	2 字节	计算值

失败或者不支持应答包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	5+N
CMDA	命令值	2 字节	0XFE01
STATE	状态值	1 字节	0x41 或 0x6D
DATA	数据	0 字节	错误码、错误信息或不填（0 字节）
CRC	校验字节	2 字节	计算值

3.12、文件上传下载指令

本指令用于配置终端应用参数或者对用户公开的相关参数，同时用于终端流水导出操作。

数据请求包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	4+N
CMDR	命令值	2 字节	0X3008
DATA	数据	N 字节	**请求数据格式详见 3.12.2**
CRC	校验字节	2 字节	计算值

成功应答数据包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	5+N
CMDA	命令值	2 字节	0XB008
STATE	状态值	1 字节	0x90
DATA	数据	N 字节	**应答数据，格式详见 3.12.3**
CRC	校验字节	2 字节	计算值

失败或者不支持应答包格式：

代码	描述	长度	值
HEAD	数据头，固定值	6 字节	“SDsEs”+地址值
LEN	长度值	4 字节	5+N
CMDA	命令值	2 字节	0XB008
STATE	状态值	1 字节	0x41 或 0x6D
DATA	数据	0 字节	错误码、错误信息或不填（0 字节）
CRC	校验字节	2 字节	计算值

3.12.1、数据格式定义：

数据本身按照固定格式进行封装，数据协议中的各个参数之间使用‘$’字符进行分割，每个参数由参数名、等号、参数值组成。

举例：“$Param1=value1$$Param2=value2$”。

协议中的参数定义如下表所示

参数名	参数说明	备注
CMD	命令字	‘0’-下发文件信息 ‘1’-下载文件内容 ‘2’-上传文件信息 ‘3’-上传文件内容 ‘4’-上传下载结束
FILENAME	文件名	用于表示上送或下载文件的文件名，最大长度 20 字节

FILETYPE	文件类型	‘0’-二进制文件； ‘1’-‘9’由终端和下载工具的配置参数协定； 传输二进制文件时，文件需要拆分。例如：\x30\x1F\x00\x12 传送为：ASC 格式的 301F0012
FILELEN	文件的总长度	用于表示上送或下载文件的实际总长度。值使用 ASC 表示，最大 10 字节。
SUMPACK	文件的总包个数	1 个文件可以分多个包，使用 ASC 表示，最大 4 字节。
NOWPACK	当前包序号	使用 ASC 表示，最大 4 字节
PACKLEN	当前包内容长度	使用 ASC 表示，最大 4 字节，二进制文件传送因内容拆分，长度要增加一倍，例：\x30\x1F\x00\x12，实际文件长度 4，本处长度为 8
PACKBUF	当前包内容	表示当前包的数据

3.12.2、参数下载流程

执行参数下载时操作时终端回复报文的 DATA 域为空。

1、 下载文件信息

下载工具应先打包文件信息（CMD=’0’,FILENAME、FILETYPE、FILELEN、SUMPACK），发送至终端，终端解析成功后进行下一步，否则提示“错误信息，请重新下载”。

例：PC 发送的 DATA 为：

$CMD=0$$FILENAME=ZGYL.INI$$FILETYPE=1$$FILELEN=1000$$SUMPACK=2$

POS 终端返回 DATA 为空

2、 分包下载文件内容

只需传输(CMD=’1’,NOWPACK、PACKLEN、PACKBUF) 终端解析成功后进行回复， 注意 PACKBUF 一定要放到最后一个参数。

例：PC 发送 DATA：

$CMD=’1’$$NOWPACK=1$PACKLEN=500$$PACKBUF=500 字节的内容$

POS 终端返回 DATA 为空

3、 传输完成

参数文件下载完成，下载工具发送 CMD=’4’,终端解析成功，代表下载文件完成，然后回复解包结果，终端根据结果提示。

例：PC 发送 DATA：

$CMD=’4’$

POS 终端返回 DATA 为空。

3.12.3、数据导出流程如下:

1、获取文件信息

流水导出工具应先打包要获取的文件信息（CMD='2',FILENAME、FILETYPE）至终端，终端收到请求后回复（CMD 、FILELEN、SUMPACK），交互成功后进行下一步，否则提示错误信息，重新导出。

例：PC 发送的 DATA 为：$CMD=2$$FILENAME=ZGYL.INI$$FILETYPE=1$

POS 终端返回 DATA 为：$CMD=2$$FILELEN=1000$$SUMPACK=2$

2、分包获取文件内容

流水导出工具打包(CMD='3',NOWPACK)，终端回复（CMD='3',NOWPACK 、PACKLEN、PACKBUF），PACKBUF 一定要放到最后一个参数。

例：PC 发送的 DATA 为：$CMD=3$$NOWPACK=1$

POS 终端返回 DATA 为：

$CMD=3$$NOWPACK=1$$PACKLEN=500$$PACKBUF=500 字节的文件内容$

3、传输完成

传输完成后，下载工具发送 CMD='4',终端解析下载文件完成后，回复解包结果，终端根据结果提示。

例：PC 发送 DATA：$CMD='4'$

POS 终端返回 DATA 为空。

4、数据传输方式

数据传输采用加密传输方式。由于采用蓝牙无线通信的方式进行通信，通信数据容易被截取，因此采用数据加密的方式进行数据传输。加密方式采用 DES 加密。消息加密后双方无法正常显示，可能存在编码方式不同导致显示的消息也不同，因此统一采用 Base64 编码的形式进行对数据编码，这样可以做到标准化，规范化。

消息加密实现需要对缓冲区 buffer 做如下表所示的划分：

缓冲区 buffer	Encrypt Flag	Key Length	Message Length	Key	Message
字节数	1	1	4	……	……

缓冲区 buffer 主要分为五个部分，分别为：

1、 加密标志位：占 1 个字节，用 0 和 1 标示：0 标示不使用 DES 加密，1 标示使用 DES 加密。

2、 密钥长度：DES 加密算法中密钥的长度，占 1 个字节，如果没有使用加密算法，则该值为 0。

3、 消息长度：默认使用 4 个字节存放（这里需要注意，对于密钥长度和消息长度都是经过 Base64 编码后获得的长度，而非实际数据的长度）

4、 密钥：DES 加密使用的密钥，（经过 Base64 编码以后的数据）

5、 加密后的消息：需要传递的数据（经过 Base64 编码以后的数据）。

对方收到缓冲区数据之后，首先需要对加密标志位进行判断，如果加密，则需要获得加密密钥的长度，然后再获得密钥（此时获得的密钥是经过 Base64 编码的，需要先对其解码才能获得用于 DES 解密的 KEY 值）。接着获取消息长度，再获取消息，和密钥一样，获取的消息也是经过 Base64 编码的数据，需要进行 Base64 解码才能获得有效数据。

举例：以“返回”键命令为例。报文53 44 73 45 73 00 00 00 00 04 30 01 C9 74

采用DES方式加密，加密标志为：0x01；

密钥明文为 “FF FF FF FF FF FF FF FF”，长度为0x08；

base64编码后的字符串“//////////8=”，对应的十六进制为“2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 38 3D”，长度为0x0C。

数据明文为：“53 44 73 45 73 00 00 00 00 04 30 01 C9 74”，长度是14（0x0E）；DES加密后的数据为：“1C EC 9F AF 00 C5 F5 C1 A7 F7 FD FB A3 C3 34 3C”，长度为16（0x10）；

base64编码后得到字符串为“HOyfrwDF9cGn9/37o8M0P===”，对应十六进制为“48 4F 79 66 72 77 44 46 39 63 47 6E 39 2F 33 37 6F 38 4D 30 50 3D 3D 3D”，长度为24（0x18）。

因此缓存区数据格式为：01 0C 00 00 00 18 2F 2F 2F 2F 2F 2F 2F 2F 2F 2F 38 3D 48 4F 79 66 72 77 44 46 39 63 47 6E 39 2F 33 37 6F 38 4D 30 50 3D 3D 3D

返回键举例：

发送：

数 据 ： 53 44 73 45 73 00 00 00 00 04 30 01 C9 74 ==> U0RzRXMAAAAABDAByXQ=

数据传输格式：00 00 00 00 00 14 55 30 52 7A 52 58 4D 41 41 41 41 41 42 44 41 42 79 58 51 3D

正确返回

数 据 ： 53 44 73 45 73 00 00 00 00 05 B0 01 90 F2 32 ==> U0RzRXMAAAAABbABkPIy

数据传输格式：00 00 00 00 00 14 55 30 52 7A 52 58 4D 41 41 41 41 41 42 62 41 42 6B 50 49 79

错误返回

数 据 ： 53 44 73 45 73 00 00 00 00 05 B0 01 41 29 6E ==> U0RzRXMAAAAABbABQSlu

数据传输格式：00　　00　　00 00 00 14　　55 30 52 7A 52 58 4D 41 41 41 41 41 42 62 41 42 51 53 6C 75

附录 A　CRC 校验及算法实现

CRC 码有多种校验位数，8 位、16 位、32 位。为了便于处理器实现，尽可能压缩代码并保证可靠性。本文档采用 16 位的 CRC 校验码。采用 CRC-CCITT 多项式 $X^{16}+X^{12}+X^5+1$。参考算法如下：

```
//头 文 件 部 分

#ifndef __TYPE_DEFINE_H__
#define __TYPE_DEFINE_H__


//本 部 分 的 定 义 需 要 根 据 不 同 的 编 译 器 进 行 修 改
//U8 U16 U32分 别 表 示 8位 、 16位 、 32位 无 符 号 数 值
#define U8  unsigned char
#define U16 unsigned short
#define U32 unsigned int



#endif


//C文 件 部 分

U16 Crc16CCITT_Table[16]={ /* CRC 16bit余 式 表 */
    0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5, 0x60c6, 0x70e7,
    0x8108, 0x9129, 0xa14a, 0xb16b, 0xc18c, 0xd1ad, 0xe1ce, 0xf1ef
};

/*****************************************************************************
 函 数 名     : void Crc16CCITT(U8 *pDataIn, U32 DataLen, U8 CrcOut[2])
 描 述        : 用 移 位 的 方 法 计 算 一 组 数 字 的 16位 CRC-CCITT校 验 值
 输 入 参 数  : 1、 U8 *pDataIn : 要 进 行 16位 CRC-CCITT计 算 的 数
                2、 U32 DataLen : DataIn数 组 的 长 度
 输 出 参 数  : 1、 U8 CrcOut[2] : 16位 CRC-CCITT计 算 的 结 果
 返 回 值     : 无
*****************************************************************************/
void Crc16CCITT(U8 *pDataIn, U32 DataLen, U8 CrcOut[2])
{
    U16 Crc = 0;
    U8 Temp;

    while (DataLen-- != 0)
    {
        Temp = ((U8)(Crc>>8))>>4;
        Crc <<= 4;
        Crc ^= Crc16CCITT_Table[Temp^(*pDataIn/16)];
        Temp = ((BYTE)(Crc>>8))>>4;
        Crc <<= 4;
        Crc ^= Crc16CCITT_Table[Temp^(*pDataIn&0x0f)];
        pDataIn++;
    }
    CrcOut[0] = Crc/256;
    CrcOut[1] = Crc%256;
}
```

附录 B　BASE64 编码原理及实现

【Base64】

- base64 的编码都是按字符串长度，以每 3 个 8bit 的字符为一组，
- 然后针对每组，首先获取每个字符的 ASCII 编码，
- 然后将 ASCII 编码转换成 8bit 的二进制，得到一组 3*8=24bit 的字节
- 然后再将这 24bit 划分为 4 个 6bit 的字节，并在每个 6bit 的字节前面都填两个高位 0，得到 4 个 8bit 的字节
- 然后将这 4 个 8bit 的字节转换成 10 进制，对照 Base64 编码表 （下表），得到对应编码后的字符。

注：1. 要求被编码字符是 8bit 的，所以须在 ASCII 编码范围内（\u0000-\u00ff），中文就不行。

2. 如果被编码字符长度不是 3 的倍数的时候，则都用 0 代替，对应的输出字符为“=”

Base64 编码表							
Value	**Char**	**Value**	**Char**	**Value**	**Char**	**Value**	**Char**
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

举下面 2 个例子：

a) 字符长度为能被 3 整除时：比如“Tom”：

```
           T          o          m
ASCII:     84         111        109
8bit字节:  01010100   01101111   01101101
6bit字节:    010101     000110     111101     101101
十进制:    21         6          61         45
对应编码:  V          G          9          t
```

所以，btoa('Tom') = VG9t

b) 字符串长度不能被 3 整除时，比如“Lucy”：

```
           L          u          c          y
ASCII:     76         117        99         121
8bit字节:  01001100   01110101   01100011   01111001   00000000   00000000
6bit字节:    010011     000111     010101     100011   011110   010000   000000   000000
十进制:    19         7          21         35         30       16       (异常)   (异常)
对应编码:  T          H          V          j          e        Q        =        =
```

由于 Lucy 只有 4 个字母，所以按 3 个一组的话，第二组还有两个空位，所以需要用 0 来补齐。这里就需要注意，因为是需要补齐而出现的 0，所以转化成十进制的时候就不能按常规用 base64 编码表来对应，所以不是 a， 可以理解成为一种特殊的“异常”，编码应该对应“=”。

代码实现：

```
/**
 * base64 encoding & decoding
 * for fixing browsers which don't support Base64 | btoa |atob
 */

(function (win, undefined) {

    var Base64 = function () {
       var base64hash = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/';

       // btoa method
       function _btoa (s) {
           if (/([^\u0000-\u00ff])/.test(s)) {
               throw new Error('INVALID_CHARACTER_ERR');
           }
           var i = 0,
               prev,
               ascii,
               mod,
               result = [];

           while (i < s.length) {
               ascii = s.charCodeAt(i);
               mod = i % 3;
```

```
            mod = i % 3;

            switch(mod) {
                // 第一个6位只需要让8位二进制右移两位
                case 0:
                    result.push(base64hash.charAt(ascii >> 2));
                    break;
                //第二个6位 = 第一个8位的后两位 + 第二个8位的前4位
                case 1:
                    result.push(base64hash.charAt((prev & 3) << 4 | (ascii >> 4)));
                    break;
                //第三个6位 = 第二个8位的后4位 + 第三个8位的前2位
                //第4个6位 = 第三个8位的后6位
                case 2:
                    result.push(base64hash.charAt((prev & 0x0f) << 2 | (ascii >> 6)));
                    result.push(base64hash.charAt(ascii & 0x3f));
                    break;
            }

            prev = ascii;
            i ++;
        }

        // 循环结束后看mod, 为0 证明需补3个6位，第一个为最后一个8位的最后两位后面补4个0。另外两个6位对应的是异常的"=";
        // 循环结束后看mod, 为0 证明需补3个6位，第一个为最后一个8位的最后两位后面补4个0。另外两个6位对应的是异常的"=";
        // mod为1，证明还需补两个6位，一个是最后一个8位的后4位补两个0，另一个对应异常的"="
        if(mod == 0) {
            result.push(base64hash.charAt((prev & 3) << 4));
            result.push('==');
        } else if (mod == 1) {
            result.push(base64hash.charAt((prev & 0x0f) << 2));
            result.push('=');
        }

        return result.join('');
    }

    // atob method
    // 逆转encode的思路即可
    function _atob (s) {
        s = s.replace(/\s|=/g, '');
        var cur,
            prev,
            mod,
            i = 0,
            result = [];

        while (i < s.length) {
```

```
            while (i < s.length) {
                cur = base64hash.indexOf(s.charAt(i));
                mod = i % 4;

                switch (mod) {
                    case 0:
                        //TODO
                        break;
                    case 1:
                        result.push(String.fromCharCode(prev << 2 | cur >> 4));
                        break;
                    case 2:
                        result.push(String.fromCharCode((prev & 0x0f) << 4 | cur >> 2));
                        break;
                    case 3:
                        result.push(String.fromCharCode((prev & 3) << 6 | cur));
                        break;

                }

                prev = cur;
                i ++;
            }
```

```
                i ++;
            }

            return result.join('');
        }

        return {
            btoa: _btoa,
            atob: _atob,
            encode: _btoa,
            decode: _atob
        };
    }();

    if (!win.Base64) { win.Base64 = Base64 }
    if (!win.btoa) { win.btoa = Base64.btoa }
    if (!win.atob) { win.atob = Base64.atob }

})(window)
```

详细请参考下面的博文：
http://www.cnblogs.com/hongru/archive/2012/01/14/2321397.html