目 录

1 概述..... 2
2 接口说明 2
2.1 系统结构..... 2
2.2 消息格式..... 2
2.3 消息类型..... 3
2.4 连接建立..... 6
3 DCC 演示程序代码说明..... 7
3.1 文件列表..... 7
3.2 接口说明..... 7
3.3 Windows 环境下编译说明 10
3.4 Linux 环境下编译说明 10

1 概述

本手册描述了用户的数据服务器（DCC）与 mServer 之间的接口，通过该接口客户程序可以对无线数据终端（DTU）进行管理和数据通信。最后用一个示例程序（dc_client.exe）说明了该接口。
mServer 提供了对 DTU 的管理、测试、数据收发的功能，并且提供基于 TCP/IP 的 SOCKET 接口，方便用户开发与数据中心相接的数据中心客户端。DCC 是由用户实现的程序，它通过与 mServer 相接，实现对 DTU 进行数据收发，并对收到的数据进行分析和处理。
本手册对应的 mServer 的版本是 V2.7，数据中心客户端演示程序的版本是 1.0.9。

2 接口说明

在本节中描述了 DCC 与 mServer 之间的接口。

1. 系统结构
2. 消息格式
3. 消息类型
4. 连接建立

2.1 系统结构

1. 整个系统的结构如下图所示。DTU 通过 UDP、TCP、ETCP 三种方式和 mServer 相连，DCC 则通过 TCP 方式和 mServer 相连。
2. mServer 是无线数据服务器端，在指定的端口上监听，等待 DCC 的连接。同一时刻可以同时接入多个 DCC，同时可以接入的最多的 DCC 的个数由 mServer 设定。
3. 请参考《mServer 用户手册》5.1 节来配置 mServer，以使能 mServer 与 DCC 之间的接口。

2.2 消息格式

1. mServer 和 DCC 之间的消息格式如下图所示。

ID	IMEI	Name	Msg Type	Reserved	Msg Len	Msg Body
4 bytes	16 bytes	16 bytes	1 byte	1 byte	2 bytes	Indicate by Msg Len

2. ID：表示一个消息包的开始，为四个字节的 0x7e。

3. IMEI：是 DTU 的唯一标识，是最后一个字节为结束符“ \0 ”的，实际长度为 15 字节的字符串。IMEI 为“ 000000000000000 ”代表所有的 DTU，比如可以给所有的 DTU 广播发送数据。
4. Name：是 DTU 的别名，用来便于人工识别 DTU，是最后一个字节为结束符“ \0 ”的，实际最大长度为 15 字节的字符串。
5. Msg Type：消息类型，一个字节长，详细定义见 2.3 消息类型。
6. Reserved：保留字段，一个字节长。
7. Msg Len：指明了 Msg Body 的实际长度，两个字节长，以网络字节序存放。
8. Msg Body：消息体，长度由 Msg Len 字段指明，不同的消息类型有不同的消息体。最大消息体长度为 1492。

2.3 消息类型

1. DC_MSG_DATA

这个消息是双向的，mServer 通过这个消息把从 DTU 来的数据发到 DCC，DCC 也可以通过这个消息把数据交给 mServer，然后发送到 DTU。本消息的类型是 0x00，数据长度由 Msg Len 指示。
在从 mServer 到 DCC 的这种消息中，其中包含了 IMEI 和 Name，可以让 DCC 来区别是哪个 DTU 发送来的数据。
在从 DCC 到 mServer 的这种消息中，如果有 IMEI，则 mServer 通过 IMEI 来查找目标 DTU，如果 IMEI 为空，则通过 Name 来查找目标 DTU。mServer 发送完后会有一个 DC_MSG_SENDRESULT 消息返回给 DCC。
注意：如果想给所有 DTU 发送广播数据，可以在这个请求中把 IMEI 号置成“ 000000000000000 ”。发送广播数据功能需要 mServer 1.5 以上的版本支持。

2. DC_MSG_ONLINE

这个消息只能从 mServer 发到 DCC，表示一个 DTU 已上线，该 DTU 由 IMEI 和 Name 来标识。本消息的类型是 0x01，消息体长度为零。

3. DC_MSG_OFFLINE

这个消息只能从 mServer 发到 DCC，表示一个 DTU 已下线，该 DTU 由 IMEI 和 Name 来标识。本消息的类型是 0x02，消息体长度为零。

4. DC_MSG_GETSTATUS

这个消息只能从 DCC 发到 mServer，表示 DCC 想知道 DTU 的当前状态，这个消息会激发 mServer 返回一个 DC_MSG_STATUSRESULT 消息，该 DTU 由 IMEI 和 Name 来标识。本消息的类型是 0x03，消息体长度为零。
注意：如果想获得所有 DTU 的当前状态，可以在这个请求中把 IMEI 号置成“ 000000000000000 ”。这时候 mServer 会把所有 DTU 的当前状态挨个用 DC_MSG_STATUSRESULT 消息返回，每条 DC_MSG_STATUSRESULT 消息里面返回一台 DTU 的状态，直到最后一条 IMEI 为“ 000000000000000 ”的

DC_MSG_STATUSRESULT 消息结束。获取所有 DTU 当前状态的功能需要 mServer 1.5 及以上的版本支持。

5. DC_MSG_SENDRESULT

这个消息只能从 mServer 发到 DCC，表示 mServer 发送数据是否成功，该 DTU 由 IMEI 和 Name 来标识。本消息的类型是 0x04，消息体长度为 1，消息体指示发送是否成功，目前定义了四种结果：

SENDRESULT_SUCCESS	0x00	发送成功。
SENDRESULT_NOSUCHDTU	0x01	该 DTU 不存在。
SENDRESULT_OFFLINE	0x02	该 DTU 目前没有上线。
SENDRESULT_CONGESTED	0x03	该 DTU 数据链路拥塞，需要等候再发。

6. DC_MSG_STATUSRESULT

这个消息只能从 mServer 发到 DCC，是对 DC_MSG_GETSTATUS 消息的响应，该 DTU 由 IMEI 和 Name 来标识。本消息的类型是 0x05。消息体的第一个字节指示该 DTU 当前状态，目前定义了三种结果：

STATUSRESULT_NOSUCHDTU	0x00	该 DTU 不存在。
STATUSRESULT_ONLINE	0x01	该 DTU 已经上线。
STATUSRESULT_OFFLINE	0x02	该 DTU 目前没有上线。

如果 DTU 状态为 STATUSRESULT_NOSUCHDTU，则消息体长度为为 1。否则消息体结构如下（多字节字段都是网络字节序）：

Dtu ip：如果 DTU 是上线状态，Dtu ip 是当前 DTU 的 IP 地址。
Idx：该 DTU 在 mServer 里面的序号。
Logon time：如果 DTU 是上线状态，则 Logoff time 是最近一次 DTU 上线的时间，是 1970 年 1 月 1 日 0 点 0 分 0 秒以来的秒数。
Reserved：保留。
Data sent：向该 DTU 发送的用户数据的字节数。
Data rcvd：从该 DTU 接收的用户数据的字节数。
Logoff time：如果 DTU 是离线状态，则 Logoff time 是最近一次 DTU 离线的时间，是 1970 年 1 月 1 日 0 点 0 分 0 秒以来的秒数。
Version：该 DTU 的版本号。
All sent：向该 DTU 发送的所有数据的字节数，包括登录包、心跳包数据。
All rcvd：从该 DTU 接收的所有数据的字节数，包括登录包、心跳包数据。

status	Dtu ip	idx	Logon time	Reserved	Data sent	Data rcvd	Logoff time	Version	All sent	All rcvd
1 byte	4 bytes	2 bytes	4 bytes	1 byte	4 bytes	4 bytes	4 bytes	16 bytes	4 bytes	4 bytes

7. DC_MSG_LISTEN _DTU

这个消息只能从 DCC 发到 mServer，表示 DCC 想监听的某个或某些 DTU。在缺省情况下，mServer 会把所有的 DTU 的数据都转发到 DCC，如果 DCC 只想监听某个或某些 DTU，则可以通过 DC_MSG_LISTEN _DTU 消息来指定想监听的 DTU。

本消息的类型是 0x06，消息体长度为 0，要监听的 DTU 由 IMEI 字段指定。在 DCC 和 mServer 的一次连接里，DCC 可以发送多次 DC_MSG_LISTEN _DTU 消息给 mServer，总共可以设置的要监听的 DTU 的最大个数是 100 个。

8. DC_MSG_DISC_DTU

这个消息只能从 DCC 发到 mServer，表示 DCC 想命令一个 DTU 下线，该 DTU 由 IMEI 和 Name 来标识。DTU 真正下线之后，会有 DC_MSG_OFFLINE 消息发送给 DCC。
本消息的类型是 0x08，消息体长度为 0。

9. DC_MSG_ADD_DTU

这个消息只能 DCC 发到 mServer，表示 DCC 要增加一个 DTU 到 mServer 中。如果一个 DTU 没有在 mServer 中，只要它尝试连接 mServer，mServer 就会自动添加这个 DTU，不需要预先添加。
本消息的类型是 0x09，消息体长度为 0。

10. DC_MSG_AT_CMD

这个消息是双向的，由 DCC 用来通过 mServer 给 DTU 发送 AT 命令，并且 DTU 返回响应。该 DTU 由 IMEI 和 Name 来标识，AT 命令及 DTU 的响应放在 Msg Body 里面。DTU 所支持的 AT 命令请参考相应的操作手册。
本消息的类型是 0x0A，数据长度由 Msg Len 指示。mServer 发送完后会有一个 DC_MSG_SENDRESULT 消息返回给 DCC。

11. DC_MSG_RENAME_DTU

这个消息只能从 DCC 发到 mServer，表示 DCC 想给一个 DTU 改名，该 DTU 由 IMEI 来标识。
本消息的类型是 0x0E，消息体长度为 0。此消息需要 mServer 2.3 以上的版本支持。在 mServer 2.7 以上的版本，如果 DCC 通过本消息修改 DTU 名字成功，mServer 会返回一个 DC_MSG_DTU_MODIFIED 的消息。

12. DC_MSG_DEL_DTU

这个消息只能从 DCC 发到 mServer，表示 DCC 想删除一个 DTU，该 DTU 由 IMEI 来标识。
本消息的类型是 0x0F，消息体长度为 0。此消息需要 mServer 2.3 以上的版本支持。在 mServer 2.7 以上的版本，如果 DCC 通过本消息删除 DTU 成功，mServer 会返回一个 DC_MSG_DTU_DELETED 的消息。

13. DC_MSG_DTU_ADDED

这个消息只能从 mServer 发到 DCC，表示 mServer 中新加入了一个 DTU。
本消息的类型是 0x10，消息体长度为 0。此消息需要 mServer 2.7 以上的版本支持。在 mServer 2.7 以上的版本，如果 DCC 通过本消息添加 DTU 成功，mServer 会返回一个 DC_MSG_DTU_ADDED 的消息。

14. DC_MSG_DTU_DELETED

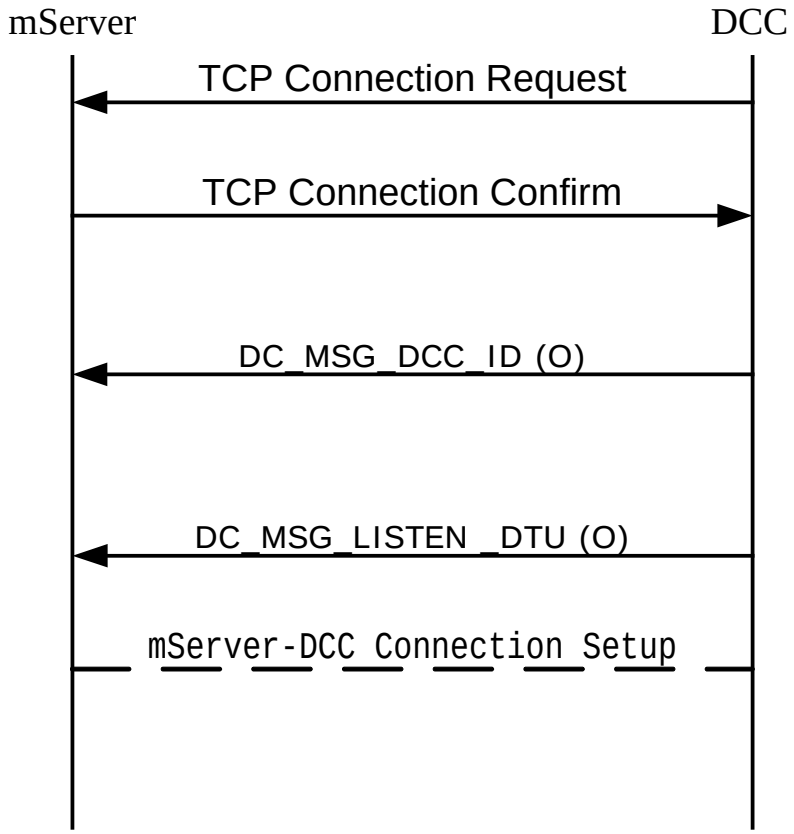
这个消息只能从 mServer 发到 DCC，表示 mServer 中删除了一个 DTU。
本消息的类型是 0x11，消息体长度为 0。此消息需要 mServer 2.7 以上的版本支持。

15. DC_MSG_DTU_MODIFIED

这个消息只能从 mServer 发到 DCC，表示 mServer 中修改了一个 DTU 的名字。
本消息的类型是 0x12，消息体长度为 0。此消息需要 mServer 2.7 以上的版本支持。

2.4 连接建立

下图是 DCC 和 mServer 的连接建立的流程。



1. DCC 向 mServer 发起一个 TCP 连接请求，mServer 接收这个连接请求。
2. 如果使用 DCC ID，DCC 向 mServer 发送 DC_MSG_DCC_ID 消息，这个步骤可以跳过。
3. 如果只需要监听特定的 DTU 的消息，DCC 可以给 mServer 发送 DC_MSG_LISTEN _DTU 的消息。这个步骤可以重复多次，也可以跳过。
4. mServer 和 DCC 的连接建立完成，接下来就可以进行正常的数据收发了。

3 DCC 演示程序代码说明

在本节中说明了 DCC 演示程序源代码。DCC 工程里实现了与 mServer 进行通信的功能，在 Windows 可以编译成 DLL，在 Linux 下编译成静态函数库。WINDOWS_DEMO 工程是 Windows 下的演示程序。LINUX_DEMO 工程是 Linux 下的演示程序。

1. 文件列表
2. 接口说明
3. Windows 环境下编译说明
4. Linux 环境下编译说明

3.1 文件列表

DCC 工程的主要文件有：

dcc.h	mServer 接口头文件。
dcc.c	mServer 接口源程序文件
dcc.def	DCC.DLL 输出接口描述文件
Makefile	Linux 环境下编译脚本文件

WINDOWS_DEMO 工程的主要文件有：

dcc.h	mServer 接口头文件。
SysConfDlg.h	系统设置对话框类头文件。
Resource.h	资源头文件，VC 自动生成。
StdAfx.h	标准 MFC 头文件，VC 自动生成。
dc_clientDlg.h	主对话框头文件，由 VC 自动生成。
dc_client.h	主程序头文件，由 VC 自动生成。
dc_client.cpp	主程序文件，由 VC 自动生成。
dc_clientDlg.cpp	主对话框文件，由 VC 自动生成。
StdAfx.cpp	标准 MFC 程序文件，由 VC 自动生成。
SysConfDlg.cpp	系统设置对话框文件。

LINUX_DEMO 工程的主要文件有：

main.c	主程序文件
Makefile	编译脚本文件

3.2 接口说明

头文件 dcc.h 提供了 mServer 接口结构和函数。

```
#define MODE_BLOCK      0
#define MODE_NONBLOCK   1

#define MODE_UDP  0
#define MODE_TCP  1
```

```
#define IMEI_LEN    15
#define DTU_NAME  15
#define MAX_MSG_LEN 1492

#define DC_MSG_DATA      0x00
#define DC_MSG_ONLINE   0x01
#define DC_MSG_OFFLINE  0x02
#define DC_MSG_GETSTATUS 0x03
#define DC_MSG_SENDRESULT 0x04
#define DC_MSG_STATUSRESULT  0x05
#define DC_MSG_LISTEN_DTU 0x06
#define DC_MSG_DISC_DTU        0x08
#define DC_MSG_ADD_DTU         0x09
#define DC_MSG_AT_CMD          0x0A
#define DC_MSG_RENAME_DTU    0x0E
#define DC_MSG_DEL_DTU         0x0F
#define DC_MSG_DTU_ADDED      0x10
#define DC_MSG_DTU_DELETED   0x11
#define DC_MSG_DTU_MODIFIED  0x12

#define SENDRESULT_SUCCESS           0x00
#define SENDRESULT_NOSUCHDTU        0x01
#define SENDRESULT_OFFLINE           0x02
#define SENDRESULT_CONGESTED        0x03

#define STATUSRESULT_NOSUCHDTU     0x00
#define STATUSRESULT_ONLINE          0x01
#define STATUSRESULT_OFFLINE         0x02

struct dc_msg {
        char imei[IMEI_LEN+1];              // 终端的 IMEI 号
        char name[DTU_NAME+1];          // 终端的别名
        unsigned char msg_type;             // 消息类型
        unsigned char reserved;             // 保留
        unsigned short msg_len;             // 消息长度
        unsigned char msg_body[MAX_MSG_LEN];        // 消息体
};
```

```
1.  int dcc_init(int mode,  unsigned short port,  const char *dc_ip,  unsigned short dc_port, int
    block);
```

dcc_init() 根据指定的模式和参数去初始化与 mServer 的连接。

mode：　　MODE_TCP。
port：　　忽略。
dc_ip：　　mServer 的 IP 地址，是一个形如 “ 127.0.0.1 ” 的字符串。
dc_port：　mServer 的监听端口。
block：　　MODE_BLOCK 或者 MODE_NONBLOCK，表明在发送或接收操作不能立刻完成时，是否等待。参见 dcc_msg_send 和 dcc_msg_receive 说明。
返回值：　　如果成功，返回正数，作为此连接的句柄，否则失败。

注意：用户需要保存这个返回值，作为 dcc_close()、dcc_msg_send()和 dcc_msg_receive()调用参数。

2. int dcc_close(int dcc_hdl);

dcc_socket_close() 关闭和 mServer 通信的连接。

dcc_hdl： 连接句柄，是 dcc_init()的返回值。
返回值： 总是返回零。

3. int dcc_msg_send(int dcc_hdl, struct dc_msg *msg);

dcc_msg_sender() 向 mServer 发送一个消息，在这个函数里对消息进行组包和通过 SOCKET 发送到 mServer。

dcc_hdl： 连接句柄，是 dcc_init()的返回值。
msg： 要向 mServer 发送的消息。
返回值： 大于 0，成功
等于 0，发送不能立刻完成，需再尝试。
小于 0，连接出错，需要把此连接 dcc_close()然后再重新用 dcc_init()建立连接。

4. int dcc_msg_receive(int dcc_hdl, struct dc_msg *msg);

dcc_msg_receive() 用来接收从 mServer 来的消息，如果 socket 是 MODE_BLOCK 方式，在没有数据的情况下，该函数会阻塞，直到有数据到达或者 socket 被关掉。如果 socket 是 MODE_NONBLOCK 方式，在没有数据的情况下，该函数也会立刻退出，返回值为 0，可以再次尝试。

dcc_hdl： 连接句柄，是 dcc_init()的返回值。
msg： 用来存放收到的消息
返回值： 大于 0，成功，收到的消息在 msg 里。
等于 0，没有数据，需再尝试。
小于 0，连接出错，需要把此连接 dcc_close()然后再重新用 dcc_init()建立连接。

注意：如果 socket 是 MODE_NONBLOCK 方式，在 dcc_msg_receive()返回表示收到消息时，需要再次调用 dcc_msg_receive()直到接收不到消息为止，否则可能会产生消息积压，导致消息丢失。

5. int dcc_msg_send_auth(int dcc_hdl, char *user, char *passwd);

如果在 mServer 选择了 DCC 需要认证，则在 dcc_init 成功之后需要调用 dcc_msg_send_auth() 来进行认证。如果 mServer 侧没有选择 DCC 需要认证，则调用此函数会失败。mServer 的认证功能需要 2.0.0 以上的版本支持。

dcc_hdl： 连接句柄，是 dcc_init()的返回值。
user： 用户名，需要和 mServer 侧的设置一致才能通过认证。

passwd： 密码，需要和 mServer 侧的设置一致才能通过认证。
返回值： 等于 0，成功。
小于 0，认证失败，需要把此连接 dcc_close()然后再重新用 dcc_init()建立连接。

3.3 Windows 环境下编译说明

编译环境是 VC6.0。
编译时需要先编译 dcc，然后编译 windows_demo 下的 dcc_client，要保证 dcc 和 dc_client 的都是 debug 或者 release 版本。

3.4 Linux 环境下编译说明

需要安装 gcc 编译环境。
编译时先进入 dcc 目录下，键入命令 make 将编译出来 dcc 静态函数库，然后进入 linux_demo 目录下，键入命令 make 将编译出来 dcc_demo 程序。

```
[user@jmx dcc_demo]$ cd dcc
[user@jmx dcc]$ make
cc -DLINUX -g -Wall   -c -o dcc.o dcc.c
ar rv libdcc.a dcc.o
a - dcc.o
[user@jmx dcc]$ cd ../linux_demo
[user@jmx linux_demo]$ make
cc -I ../dcc -g -Wall   -c -o main.o main.c
cc main.o -L../dcc -ldcc -o dcc_demo
```