巨龙软件
DRAGONSOFT

巨龙物联网应用终端/无线数据终端
应用开发手册

厦门市巨龙软件工程有限公司
XIAMEN DRAGON SOFTWARE ENGINEERING CO.,LTD.

修订记录

日期	修订版本	修订内容	作者
131015	0.1	初稿拟制	林文良

目录

前　言

欢迎您使用巨龙 DTU 系列产品 SDK，为易于您轻松快捷的掌握巨龙 DTU 系列产品 SDK 的使用，本手册为您详细介绍巨龙 DTU 系列产品 SDK 的使用。

希望用户阅读本帮助手册，能全面快速了解巨龙 DTU 系列产品 SDK，通过短时间的操作能很快的熟悉该 SDK。若您对本手册有任何意见、建议或问题，欢迎您来信或来电，我们将竭诚为您服务。

地址：厦门市火炬高新区创新二路1号

邮编：361006

电话：400-168-1993

传真：0592-5707888

E-mail：support@dragonsoft.com.cn

公司网址：http://www.dragonsoft.com.cn

1 DSTSDK 应用开发手册介绍

本章主要描述了本手册已经具备的功能和作用，同时也告诉了读者如何去阅读本手册，使其作为研发人员使用SDK开发的最佳指导文档。本章由以下三节组成：

1.目的

2.术语

3.约定

1.1 目的

本操作手册用于指导研发人员基于巨龙物联网数据终端实现快速开发数据中心应用。

1.2 术语

APN	接入点名称
DTU	数据终端单元
FLASH	存储芯片
GPRS	移动数据业务
GSM	全球移动通信系统
IP	互联网协议
M2M	终端统一管理平台
POS	销售终端
RTU	远方终端单元
RS232	异步传输标准接口
RX	通信接收单元
SIM	用户标识模块

SMS	短消息业务
SRAM	静态随机存储器
TCP	传输控制协议
TX	通信发送单元
UDP	用户自带寻址信息协议
UIM	用户标识模块

1.3 约定

为了方便使用，快速阅读本操作手册，在本手册中我们有一些简称、图示、说明、安全警示、使用技巧等约定如下：

约定项	释义说明	备注
DSTSDK	是指巨龙物联网终端系列产品的应用编程接口	出现在第二章节之后的过程中
	提示：开发需要注意的一些细节	出现在本手册指南第一章之外的说明过程中

2 DSTSDK 简介

DSTSDK是用于无线数据传输设备（DTU）与数据中心（DSC）之间进行数据分发的应用编程接口。动态库包括了数据中心应用开发所需要的API函数，包括服务的启动、服务的停止、数据的发送、数据的接收、参数的配置、参数的查询等，可满足快速开发数据中心应用的需求。

3 DSTSDK 开发使用环境

DSTSDK开发包是以动态链接库的形式发布，支持Windows和Linux操作系统。

- Windows 版本：支持 98/2000/2003/XP/WIN7/WIN8，无需 MFC 库的支

持；

- Linux 版本：支持 2.4、2.6 及 3.0 内核，需要 pthread 库支持。

4 DSTSDK 的常量定义

常量名	常量值	注释
MAX_DSTSDK_TERMINALID_LEN	128字节	最大终端长度
DEFAULT_DSTSDK_BUFFER_LEN	4096字节	默认缓冲区长度
DEFAULT_DSTSDK_MAXLINKIDLETIME	30秒	链路超时时间
DEFAULT_DSTSDK_ACTIVECYCLE	5秒	心跳周期数
DEFAULT_DSTSDK_MAXACTIVE	5秒	最大的心跳数
DEFAULT_DSTSDK_LOGINTIMEOUT	30秒	登录超时时间

5 DSTSDK 的类型声明

5.1 枚举类型

类型名称	类型值	注释
t_dstsdk_result 函数执行结果类型	DSTSDK_OK = 0	成功
	DSTSDK_INVALID	无效失败
	DSTSDK_BUSY	遇忙失败
	DSTSDK_EMPTY,	空失败

	DSTSDK_ERROR	失败
	DSTSDK_MAX	未知
e_dstsdk_listen SOCKET监听类型	DSTSDK_LISTEN_TCP	TCP监听
	DSTSDK_LISTEN_UDP	UDP监听
e_dstsdk_protocol 协议类型	DSTSDK_PROTOCOL_DTU	DTU 协议
	DSTSDK_PROTOCOL_M2M	M2M 协议
	DSTSDK_PROTOCOL_MAX	无效协议
e_dstsdk_event 事件类型	DSTSDK_EVENT_CONNECT	会话连接事件
	DSTSDK_EVENT_DISCONN	会话断开事件
	DSTSDK_EVENT_READ	会话接收到数据事件
	DSTSDK_EVENT_IDLE	会话空闲事件
	DSTSDK_EVENT_MAX	会话未知事件
e_dstsdk_idle 会话空闲类型	DSTSDK_IDLE_NONE,	不触发
	DSTSDK_IDLE_READ,	接收空闲触发
	DSTSDK_IDLE_WRITE,	发送空闲触发
	DSTSDK_IDLE_BOTH	收发空闲都触发

5.2 复合类型

类型名称	类型值	注释

应用实例	DSTSDK_HANDLER	DSTSDK句柄
会话类型	DSTSDK_SESSION	DSTSDK会话
事件处理函数原型	(*t_dstsdk_handler)(DSTSDK_SESSION session, e_dstsdk_event event, unsigned int param1, unsigned int param2)	事件处理函数原型， 参数： session：会话类型 event： 事件类型 param1：终端标识地址 param2：终端标识长度
事件描述	struct _dstsdk_event{ DSTSDK_SESSION session; e_dstsdk_event event; unsigned int param1; unsigned int param2; }t_dstsdk_event, *p_dstsdk_event;	事件描述结构体， 参数： session：会话类型 event： 事件类型 param1：终端标识地址 param2：终端标识长度

6 DSTSDK 事件说明

事件类型(event)	事件描述	
DSTSDK_EVENT_CONNECT	终端会话连接事件 DTU终端连接并登陆完成后触发本事件。	
	参数1	char *，终端设备号

	参数2	int，终端设备号长度
DSTSDK_EVENT_DISCONN	终端会话断开事件 DTU主动断开或调用dstsdk_closeSession主动断开连接触发该事件。 应用中需要调用dstsdk_cleanSession函数实现对会话的清除； 由于DSTSDK_EVENT_CONNECT事件在DTU登陆完成后才触发，所以该本事件触发之前可能未触发DSTSDK_EVENT_CONNECT事件。	
	参数1	NULL
	参数2	NULL
DSTSDK_EVENT_READ	终端会话接收到数据事件 SDK接收到DTU发送的数据后触发该事件。	
	参数1	char *，数据缓冲区指针
	参数2	int，接收到的数据长度
DSTSDK_EVENT_IDLE	终端会话空闲事件 会话创建时，默认不会触发该事件，可通过调用dstsdk_setSessionIdleMode函数配置空闲事件触发模型及触发周期。	
	参数1	NULL
	参数2	NULL

7 DSTSDK 的函数声明

7.1 函数列表

函数名	注释
dstsdk_init	初始化应用实例
dstsdk_deinit	销毁应用实例
dstsdk_getEvent	读取事件（非阻塞式）
dstsdk_setLoginTimeout	设置登录超时时间
dstsdk_setLinkIdleTime	设置链路超时时间
dstsdk_setActiveCycle	设置心跳周期
dstsdk_setMaxActive	设置最大心跳次数
dstsdk_setBufferLen	设置缓冲区大小
dstsdk_sessionSend	发送数据
dstsdk_getSessionProtocol	获取会话协议类型
dstsdk_closeSession	关闭会话
dstsdk_cleanSession	清除已关闭的会话
dstsdk_setSessionIdleMode	设置会话空闲触发模式
*dstsdk_getSessionTerminalId	获取会话的终端标识
dstsdk_getSessionTerminalLen	获取会话的终端标识

dstsdk_setSessionOpt	设置会话的选项
*dstsdk_getSessionOpt	获取会话的选项

7.2 函数详细功能

7.2.1 dstsdk_init

◆ 功能描述

初始化应用实例，该函数在一个应用中仅需调用一次，在应用退出时需要调用 dstsdk_deinit 进行销毁。

◆ 输入参数

参数名	注释
listenType	监听类型
ip	监听 IP 地址，值为 NULL 或"0.0.0.0"为本机所有 IP
port	监听端口
eventHandler	事件处理函数，非空时为事件处理模型；为空时为轮询模式（即需要通过 dstsdk_getEvent 获取事件）

◆ 返回值

失败返回 0，否则返回实例句柄

◆ 声明

```
extern    DSTSDK_HANDLER dstsdk_init(e_dstsdk_socket listenType,
const char *ip, int port, t_dstsdk_handler eventHandler);
```

返回函数列表

7.2.2 dstsdk_deinit

◆ 功能描述

销毁应用实例，该函数在应用退出时需要调用进行销毁。

- 输入参数

参数名	注释
handler	应用句柄

- 返回值

无

- 声明

```
extern    void dstsdk_deinit(DSTSDK_HANDLER handler);
```

返回函数列表

7.2.3 dstsdk_getEvent

- 功能描述

读取事件（非阻塞式）。

- 输入参数

参数名	注释
handler	应用句柄
event	事件结构体指针

- 返回值

成功返回 DSTSDK_OK

- 声明

```
extern    t_dstsdk_result dstsdk_getEvent(DSTSDK_HANDLER handler,
t_dstsdk_event *event);
```

返回函数列表

7.2.4 dstsdk_setLoginTimeout

- 功能描述

  设置登录超时时间

- 输入参数

参数名	注释
session	会话句柄
value	设置值，单位：秒

- 返回值

  无

- 声明

```
extern    void dstsdk_setLoginTimeout(DSTSDK_HANDLER handler, int value);
```

返回函数列表

7.2.5 dstsdk_setLinkIdleTime

- 功能描述

  设置链路超时时间

- 输入参数

参数名	注释
session	会话句柄
value	设置值，单位：秒

- 返回值

  无

- 声明

```
extern    void dstsdk_setLinkIdleTime(DSTSDK_HANDLER handler, int
value);
```

返回函数列表

7.2.6 dstsdk_setActiveCycle

◆ 功能描述

设置心跳周期。

◆ 输入参数

参数名	注释
session	会话句柄
value	设置值，单位：秒

◆ 返回值

无

◆ 声明

```
extern    void dstsdk_setActiveCycle(DSTSDK_HANDLER handler, int
value);
```

返回函数列表

7.2.7 dstsdk_setMaxActive

◆ 功能描述

设置最大心跳次数。

◆ 输入参数

参数名	注释
session	会话句柄
value	设置值，单位：秒

◆ 返回值

无

◆ 声明

```
extern    void dstsdk_setMaxActive(DSTSDK_HANDLER handler, int
value);
```

返回函数列表

7.2.8 dstsdk_setBufferLen

◆ 功能描述

设置缓冲区大小。

◆ 输入参数

参数名	注释
session	会话句柄
value	设置值，单位：秒

◆ 返回值

无

◆ 声明

```
extern    void dstsdk_setBufferLen (DSTSDK_HANDLER handler, int
value);
```

返回函数列表

7.2.9 dstsdk_sessionSend

◆ 功能描述

发送数据。

◆ 输入参数

参数名	注释

session	会话句柄
data	待发送数据
dataLen	数据长度

◆ 返回值

发送的数据长度

◆ 声明

```
extern    int dstsdk_sessionSend(DSTSDK_SESSION session, const char
*data, int dataLen);
```

返回函数列表

7.2.10 dstsdk_getSessionProtocol

◆ 功能描述

获取会话协议类型。

◆ 输入参数

参数名	注释
session	会话句柄

◆ 返回

e_dstsdk_protocol

◆ 声明

```
extern    e_dstsdk_protocol
dstsdk_getSessionProtocol(DSTSDK_SESSION session);
```

返回函数列表

7.2.11 dstsdk_closeSession

◆ 功能描述

关闭会话。

◆ 输入参数

参数名	注释
session	会话句柄

◆ 返回值

成功返回 DSTSDK_OK

◆ 声明

```
extern    t_dstsdk_result dstsdk_closeSession(DSTSDK_SESSION
session);
```

返回函数列表

7.2.12 dstsdk_cleanSession

◆ 功能描述

清除已关闭的会话，该函数仅在 DSTSDK_EVENT_DISCONN 事件中调用。

◆ 输入参数

参数名	注释
session	会话句柄

◆ 返回值

失败返回 0，否则返回实例句柄

◆ 声明

```
extern    t_dstsdk_result dstsdk_cleanSession(DSTSDK_SESSION
session);
```

返回函数列表

7.2.13 dstsdk_setSessionIdleMode

◆ 功能描述

设置会话空闲触发模式。

◆ 输入参数

参数名	注释
session	会话句柄
idle	空闲模式
second	空闲触发秒数，最小值为 1

◆ 返回值

无

◆ 声明

```
extern    void dstsdk_setSessionIdleMode(DSTSDK_SESSION session,
e_dstsdk_idle idle, int second);
```

返回函数列表

7.2.14 *dstsdk_getSessionTerminalId

◆ 功能描述

获取会话的终端标识。

◆ 输入参数

参数名	注释
session	会话句柄

◆ 返回值

终端标识指针，失败返回 NULL

◆ 声明

```
extern    const char *dstsdk_getSessionTerminalId(DSTSDK_SESSION
session);
```

返回函数列表

7.2.15 dstsdk_getSessionTerminalLen

◆ 功能描述

获取会话的终端标识。

◆ 输入参数

参数名	注释
session	会话句柄

◆ 返回值

终端标识长度，无效返回 0

◆ 声明

```
extern    int dstsdk_getSessionTerminalLen(DSTSDK_SESSION
session);
```

返回函数列表

7.2.16 dstsdk_setSessionOpt

◆ 功能描述

设置会话的选项。

◆ 输入参数

参数名	注释
session	会话句柄
opt	选项指针

◆ 返回值

无

◆ 声明

```
extern    void dstsdk_setSessionOpt(DSTSDK_SESSION session, void
*opt);
```

返回函数列表

7.2.17 *dstsdk_getSessionOpt

◆ 功能描述

获取会话的选项。

◆ 输入参数

参数名	注释
session	会话句柄
opt	选项指针

◆ 返回值

成功返回选项指针，否则返回 NULL

◆ 声明

```
extern    void *dstsdk_getSessionOpt(DSTSDK_SESSION session);
```

返回函数列表

8 DSTSDK 的示例源程序

8.1 事件模式

```
//DSTSDK 事件处理函数
void dstsdk_handler(DSTSDK_SESSION session, e_dstsdk_event event,
        unsigned int param1, unsigned int param2) {
switch (event) {
    case DSTSDK_EVENT_CONNECT:{ //会话连接事件
        char    *terminalId = (char *)param1; //终端编号
        int termianalLen =param2;       //终端编号长度
        //设置会话空闲触发模式
```

```
        dstsdk_setSessionIdleMode(session, DSTSDK_IDLE_BOTH, 5);
        //根据需要执行其它操作
        UserStruct *us = new UserStruct();
        ……
        dstsdk_setSessionOpt(session, us);
        break;
    }
    case DSTSDK_EVENT_DISCONN: //会话断开事件
        //清除已关闭的会话
        if (dstsdk_getSessionOpt(session) != NULL)
        {
            UserStruct *us = NULL;
            us = (UserStruct *) dstsdk_getSessionOpt(session);
            ….
            free(us);
        }
        dstsdk_cleanSession(session);
        break;
    case DSTSDK_EVENT_READ:{ //接收到数据事件
        char *recvData = (char *)param1;
        int dataLen =param2;
        UserStruct *us = (UserStruct *) dstsdk_getSessionOpt(session);
        //对业务数据进行操作
        ……
        break;
    }
    case DSTSDK_EVENT_IDLE: { //会话空闲事件
        UserStruct *us = (UserStruct *) dstsdk_getSessionOpt(session);
        //检测是否有需要下发的业务数据
```

```
            if (isDeliverData(us)){
                //下发业务数据
                dstsdk_sessionSend(session, "123456\n", strlen("123456\n"));
            }
            break;
        }
        default:
            break;
        }
    }

    int main(int args, char **argv) {
        DSTSDK_HANDLER handler;
        //初始化并采用事件模式
        handler = dstsdk_init(DSTSDK_SOCKET_TCP, "0.0.0.0", 1234,
dstsdk_handler);
        if (handler == NULL ) //初始化失败，退出应用
            return 0;
        dstsdk_setBufferLen(handler, 4096);// 设置 4K 缓冲区
        dstsdk_setLinkIdleTime(handler, 60);//设置链路超时时间 60 秒
        dstsdk_setMaxActive(handler,5 ); //设置最大心跳次数
        while (1){
            Sleep(1);
        }
    }
```

8.2 轮询模式

```
    int main(int args, char **argv) {
        DSTSDK_HANDLER handler;
```

```
t_dstsdk_event event;
//初始化并采用轮询模式
handler = dstsdk_init(DSTSDK_SOCKET_TCP, "0.0.0.0", 1234, NULL );
if (handler == NULL ) //初始化失败，退出应用
    return 0;
dstsdk_setBufferLen(handler, 4096);//  设置 4K 缓冲区
dstsdk_setLinkIdleTime(handler, 60);//设置链路超时时间 60 秒
dstsdk_setMaxActive(handler,5 );  //设置最大心跳次数
while (handler) {
    usleep(1);
    //读取事件(非阻塞模式)
    if (dstsdk_getEvent(handler, &event) == DSTSDK_OK) {
        switch (event.event) {
case DSTSDK_EVENT_CONNECT:{ //会话连接事件
    char    *terminalId = (char *) event.param1; //终端编号
    int termianalLen = event.param2;       //终端编号长度
    //设置会话空闲触发模式
    dstsdk_setSessionIdleMode(event.session, DSTSDK_IDLE_BOTH, 5);
    //根据需要执行其它操作
    UserStruct *us = new UserStruct();
    ……
    dstsdk_setSessionOpt(event.session, us);
    break;
}
case DSTSDK_EVENT_DISCONN: //会话断开事件
    //清除已关闭的会话
    if (dstsdk_getSessionOpt(event.session) != NULL)
    {
        UserStruct *us = NULL;
```

```
            us = (UserStruct *) dstsdk_getSessionOpt(event.session);
            ….
            free(us);
        }
        dstsdk_cleanSession(event.session);
        break;
    case DSTSDK_EVENT_READ:{ //接收到数据事件
        char *recvData = (char *) event.param1;
        int dataLen = event.param2;
        UserStruct *us = (UserStruct *) dstsdk_getSessionOpt(event.session);
        //对业务数据进行操作
        ……
        break;
    }
    case DSTSDK_EVENT_IDLE: { //会话空闲事件
        UserStruct *us = (UserStruct *) dstsdk_getSessionOpt(event.session);
        //检测是否有需要下发的业务数据
        if (isDeliverData(us)){
            //下发业务数据
            dstsdk_sessionSend(event.session, "123456\n", strlen("123456\n"));
        }
        break;
    }
    default:
        break;
    }


        }
    }
```

```
    //销毁应用实例
    dstsdk_deinit(handler);
    puts("!!!Hello World!!!");
    return EXIT_SUCCESS;
}
```