密级	A

六分科技
SIXENTS TECHNOLOGY

六分差分 SDK（嵌入式版）

集成开发指南

V1.1.1

北京六分科技有限公司

2021 年 3 月

法律申明

联系我们

北京六分科技有限公司

公司地址：北京市海淀区北清路与永丰路交汇口东南角四维图新大厦

官方网站：https://www.sixents.com/

技术支持：support@sixents.com

扫码关注公众号

文档修订记录

版本	修订日期	修订内容
V1.0.0	2020-08-07	首发
V1.0.1	2020-08-25	新增 sixents_sdkTick 接口
V1.0.2	2020-10-14	新增 sixents_sdkSetNwInfo、sixents_cbGetLocalIp、sixents_cbGetLocalPort 接口，删除 sixents_cbMutex 类接口
V1.1.0	2021-01-15	优化 sixents_sdkSetNwInfo()接口 删除 sixents_sdkConfEx 结构体及 sixents_sdkInitEx 接口 Sixents_sdkConf 结构体中删除 mtSupport 字段，并新增 sockIOFlag 字段 合并网络自定义接口到 Sixents_sdkConf
V1.1.1	2021-03-15	内部优化

目录

1. 概述

1.1. SDK 简介

六分差分 SDK（以下简称 SDK），负责封装差分数据服务及其配套组件，为嵌入式终端提供快捷获取差分数据的能力。

1.2. 适用设备

SDK 适用于嵌入式设备，对于嵌入式设备及其 OS 的要求是：能够提供网络服务以及线程、互斥锁服务接口。根据应用场景不同，通常嵌入式设备分为以下几类：

- Linux 系统：支持 POSIX 标准的 OS 和网络接口实现，本 SDK 完全适用，根据嵌入式设备使用的编译工具链完成编译后可以直接集成；
- RTOS 系统：比如 Free RTOS、RT-Thread 等，此类实时操作系统的底层不统一，提供的 OS 服务和网络层各不一样，本 SDK 提供相应接口，需要用户实现相关接口后再集成。参见附录接口开发说明。

1.3. 名词解释

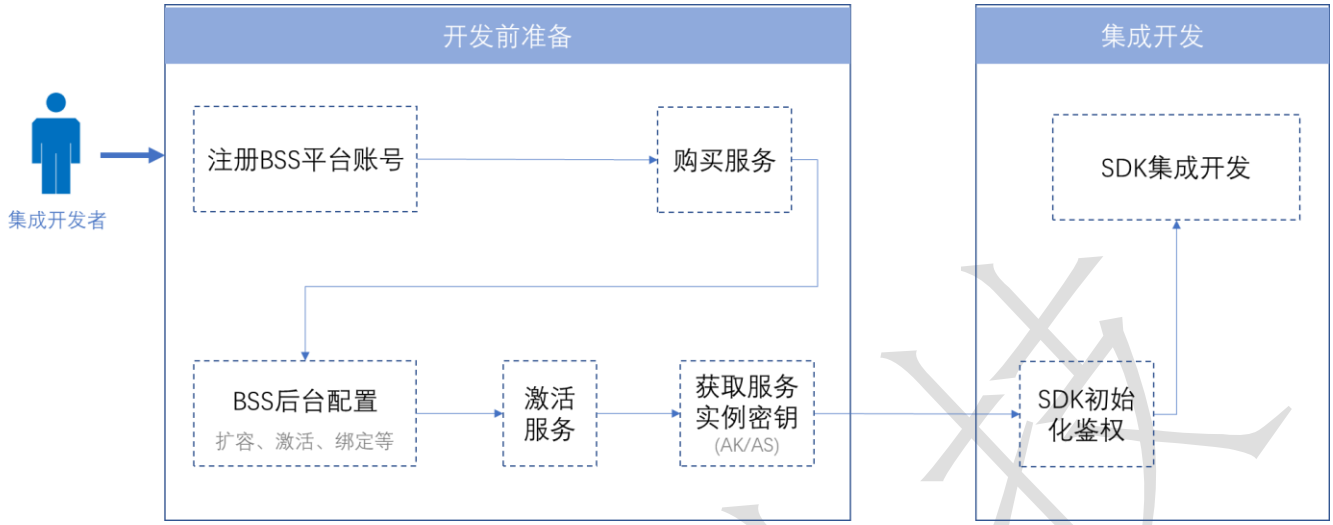
名词	解释
SDK	Software Development Kit 软件开发工具包
OS	Operating System 操作系统
AK	appKey 应用标识
AS	appSecret 应用密钥
GGA	NMEA-0183 数据协议中，表征一帧位置的数据，具体内容参照附录二 GGA 数据格式

2. 马上开始

2.1. 开发前准备

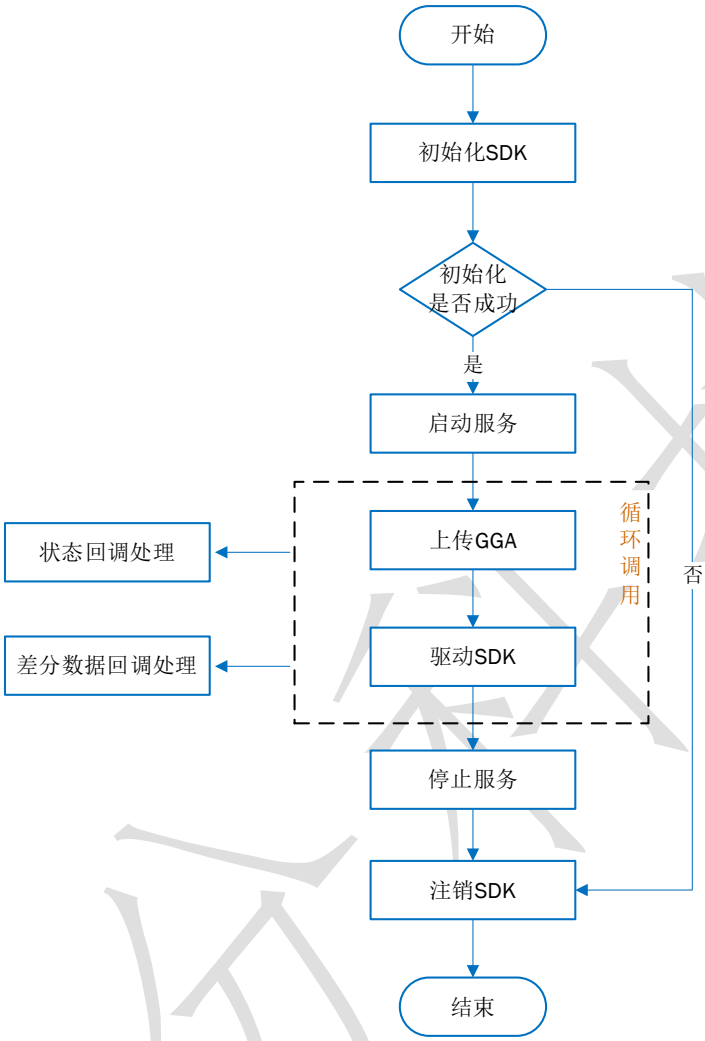
开发者需要先去公司官网(https://www.sixents.com)注册，通过个人/企业实名认证后，购买并激活服务，在控制中心获取 AK/AS，提供设备类型（DeviceType）和设备序列号（DeviceID）

完成设备绑定，即可对照本文档开始集成开发，您也可以选择自动激活服务或者自动绑定设备。开发前准备的基本流程见下图：



2.2. 接入流程

SDK 的接入流程图如下：



2.2.1. 初始化

首先配置初始化参数结构 sixents_sdkConf，然后调用接口 sixents_sdkInit 进行 SDK 初始化，初始化成功返回 0，否则返回错误值，只有初始化成功才能进行后续接口调用。

通过结构 sixents_sdkConf 可以配置 SDK 的初始参数,包括服务端的连接参数(包括:OpenAPI 的域名和端口号、数据服务器的域名和端口号等)，鉴权参数（鉴权类型、AK、AS、DeviceID、DeviceType）以及相关的回调函数等。**实际集成过程中，您只需要提供购买实例时获取的鉴权参数，其他默认参数即可满足绝大部分用的使用，请谨慎配置。**

示例代码如下：

```
sixents_sdkConfparam;
// step 0: 初始化参数配置
memset(&param, 0, sizeof(sixents_sdkConf));
param.keyType = SIXENTS_KEY_TYPE_AK;
```

```
memcpy(param.key,"Your AK",strlen("Your AK")); // 设置你购买服务实例的 AK
memcpy(param.secret, "Your AS",strlen("Your AS"));//设置你购买服务实例的 AS
memcpy(param.devID, "Your DevID",strlen("Your DevID")); // 设置你的设备 ID 或 SN
memcpy(param.devType,"Your DevType",strlen("Your DevType"));//设置你的设备类型
param.timeout = 10u; // 设置网络超时时间，单位：秒
param.cbGetDiffData = &GetDiffData; // 设置获取差分数据回调函数
param.cbGetStatus = &GetStatus;     // 设置获取状态回调函数
// step 1: 初始化 SDK 配置
sixents_sdkInit(&param);
```

2.2.2. 启动

初始化成功后，调用接口 sixents_sdkStart 启动服务线程，启动成功返回 0，否则返回错误值。启动成功后，需上传 GGA 数据以便获取差分数据，在差分数据回调接口 sixents_cbGetDiffData 及状态信息回调接口 sixents_cbGetStatus 中进行相关业务处理。

启动服务需要连接网络，根据业务需要可以使用循环方式调用启动接口。

示例代码如下：

```
// step 2: 启动服务
while (1)
{
    if(sixents_sdkStart()==SIXENTS_OK)
        break;
    usleep(1000*1000); // 1000ms
}
```

状态信息回调示例：

```
// 获取状态信息的回调函数
void GetStatus(sixents_uint32 status)
{
    // ToDo: 开发者可根据自己的需求进行处理
    printf("Cur Status:%d.\n", status);
}
```

2.2.3. 上传 GGA

启动成功后，调用接口 sixents_sdkSendGGAStr 或者 sixents_sdkSendGGA 上传 GGA 数据。开发者一般需要在死循环中调用上传 GGA 接口，每次上传 GGA 的间隔建议在 1 秒或以上。

示例代码如下：

```
// step 3: 定时发送 GGA 数据，时间间隔不小于 1s
while (1)
```

```
    {
        sixents_sdkSendGGA(lat,lon, height);
        usleep(2000*1000); // 2 秒
    }
```

2.2.4. 驱动 SDK 运行

上传 GGA 成功后，开发者需要在外部驱动 SDK 运行，建议设置调用时间间隔为 50-500ms，开发者根据平台情况可以进行适当调整。该接口一般在一个死循环内调用，示例代码如下：

```
// step 4: 定时驱动 SDK 运行
while (1)
{
    sixents_sdkTick();
    usleep(100*1000); // 100ms
}
```

实际开发过程中，一般应将上传 GGA 和驱动 SDK 运行放在一个死循环中控制。

成功调用上传 GGA 数据和驱动 SDK 运行后，将会触发差分数据回调接口，示例如下：

```
// 获取差分数据的回调函数
void GetDiffData(const sixents_char* buff, sixents_uint32len)
{
    sixents_uint32 i;
    // ToDo: 以下应该修改为客户自己对差分数据处理的逻辑
    for (i = 0; i<len; i++)
    {
        printf("%02X ", ( sixents_uint8 )buff[i]);
        if ((i + 1) % 80u == 0)
        {
            printf("\n");
        }
    }
    printf("\n");
}
```

2.2.5. 停止

调用接口 sixents_sdkStop 可停止 SDK 服务，停止服务后 SDK 不会再调用差分数据回调和状态信息回调接口，上传 GGA 接口、驱动 SDK 运行接口也将返回错误。停止服务后，可再调用接口 sixents_sdkStart 重新启动服务。

示例代码如下：

```
// step 5: 停止接收差分数据
sixents_sdkStop();
```

2.2.6. 注销

当无需继续使用 SDK 时，应调用 sixents_sdkFinal 注销 SDK，并释放 SDK 占用的资源。如果需要重新使用 SDK，需要重新按顺序调用初始化、启动等接口。

示例代码如下：

```
// step 6: 关闭 SDK
sixents_sdkFinal();
```

3. 接口详细说明

3.1. 接口概览

3.1.1. 初始化参数结构

序号	名称	说明
1	sixents_sdkConf	初始化 SDK 配置结构

3.1.2. 接口函数列表

序号	名称	说明
1	sixents_retCode sixents_sdkInit(const sixents_sdkConf* paramObj)	初始化 SDK
2	sixents_retCode sixents_sdkStart(void)	启动服务
3	sixents_retCode sixents_sdkTick(void)	驱动 SDK 运行
4	sixents_retCode sixents_sdkSendGGAStr(const sixents_char* gga, sixents_uint16 ggaLen)	上传 GGA（字符串）
5	sixents_retCode sixents_sdkSendGGA(sixents_float64 lat, sixents_float64 lon, sixents_float64 height)	上传 GGA（纬度、经度、高程）
6	sixents_retCode sixents_sdkStop(void)	停止服务
7	sixents_retCode sixents_sdkFinal(void)	注销 SDK
8	const sixents_char* sixents_sdkGetVer(void)	获取 SDK 版本
9	sixents_retCode sixents_sdkSetNwStatus(sixents_nwStatus curNwStatus)	设置网络状态

3.1.3. 回调接口列表

序号	名称	说明
1	typedef void (*sixents_cbGetDiffData)(sixents_char* buff, sixents_uint32len);	差分数据回调处理函数
2	typedef void (*sixents_cbGetStatus)(sixents_uint32 status);	状态回调处理函数

3.1.4. 自定义实现接口列表

考虑到不同嵌入式操作系统的实现区别，SDK 提供了网络和日志的回调接口，开发者可以根据情况自行实现。用户自定义接口见下表。

类别	接口	接口说明	备注
网络接口	sixents_cbConn	网络连接回调接口	开发者需实现此类别的所有接口
	sixents_cbSend	网络发送回调接口	
	sixents_cbRecv	网络接收回调接口	
	sixents_cbDisConn	网络断开回调接口	
	sixents_cbGetLocalIp	网络获取本地 ip 接口	开发者需实现此类别的所有接口
	sixents_cbGetLocalPort	网络获取本地端口接口	
日志接口	sixents_cbTrace	日志打印回调接口	

注意：开发者必须完整实现某一类接口的实现，否则会造成 SDK 不能正常工作。比如不能出现以下情况：开发者仅实现 sixents_cbConn,而不实现 sixents_cbSend 等接口。

3.2. 初始化参数结构 sixents_sdkConf

3.2.1. 结构定义

```
typedef struct
{
    sixents_keyType keyType;
    sixents_char key[SIXENTS_MAX_AK_LEN];
    sixents_char secret[SIXENTS_MAX_AS_LEN];
    sixents_char devID[SIXENTS_MAX_DEV_ID_LEN];
    sixents_char devType[SIXENTS_MAX_DEV_TYPE_LEN];
    sixents_char openApiHost[SIXENTS_MAX_SERVER_LEN];
    sixents_uint16 openApiPort;
    sixents_char serverHost[SIXENTS_MAX_SERVER_LEN];
    sixents_uint16 serverPort;
    sixents_char mountPoint[SIXENTS_MAX_MOUNT_POINT_LEN];
    sixents_uint32 timeout;
```

```
    sixent_sockIOFlag sockIOFlag;
    sixents_logLevel logPrintLevel;
    sixents_cbGetDiffData cbGetDiffData;
    sixents_cbGetStatus cbGetStatus;
    sixents_cbTrace cbTrace;
    sixents_cbConn cbConn;
    sixents_cbSend cbSend;
    sixents_cbRecv cbRecv;
    sixents_cbDisConn cbDisConn;
    sixents_cbGetLocalIp cbGetLocalIp;
    sixents_cbGetLocalPort cbGetLocalPort;
}sixents_sdkConf;
```

3.2.2. 字段说明

字段	类型	必填	说明
keyType	sixents_keyType	是	账号类型，取值： SIXENTS_KEY_TYPE_AK;
key	sixents_char[]	是	账号的 AK（账号类型 SIXENTS_KEY_TYPE_AK);
secret	sixents_char[]	是	账号的 AS（账号类型 SIXENTS_KEY_TYPE_AK);
devID	sixents_char[]	是	账号绑定的设备 ID;
devType	sixents_char[]	是	账号绑定的设备类型;
openApiHost	sixents_char[]	否	openApi 服务器域名; 默认值:openapi.sixents.com;
openApiPort	sixents_uint16	否	openApi 服务器端口，默认值:80;
serverHost	sixents_char[]	否	数据服务器域名; 默认值:vrs.sixents.com;
serverPort	sixents_uint16	否	数据服务器端口，默认值:8002;
mountPoint	sixents_char[]	否	挂载点名称，默认值:RTCM32_GNSS;
timeout	sixents_uint32	否	网络连接超时时间，单位秒，可设置范围[1，60]；默认值为 60s;
sockIOFlag	sixent_sockIOFlag	否	网络套接字阻塞模式标志位，可取值： SIXENTS_SOCK_IOFLAG_BLOCK，阻塞模式 SIXNETS_SOCK_IOFLAG_NOBLOCK，非阻塞模式; 默认值为 SIXENTS_SOCK_IOFLAG_BLOCK;
logPrintLevel	sixents_logLevel	否	为便于调试，SDK 内部采用 printf 方式打印日志信息，可通过此参数控制进行控制。 日志级别，取值：

			SIXENTS_LL_OFF, 关闭日志打印 SIXENTS_LL_ERROR,打印错误日志 SIXENTS_LL_WARN, 打印错误及警告的日志 SIXENTS_LL_INFO, 打印错误、警告、运行信息日志 SIXENTS_LL_DEBUG,打印调试等所有日志信息 默认值为 SIXENTS_LL_OFF;
cbGetDiffData	sixents_cbGetDiffData	是	差分数据回调函数，当 SDK 接收到差分数据后调用此接口；开发者必须实现该回调接口；
cbGetStatus	sixents_cbGetStatus	是	状态信息回调函数，当 SDK 状态发生变化后调用此接口；开发者必须实现该回调接口；
cbTrace	sixents_cbTrace	否	日志打印回调函数，默认值为 NULL。当开发者需要使用自己的日志打印功能时，可实现该回调接口；
cbConn	sixents_cbConn	否	网络连接回调函数，默认值为 NULL。当开发者需要使用自己的网络连接功能时，可实现该回调接口；
cbSend	sixents_cbSend	否	网络发送回调函数，同上；
cbRecv	sixents_cbRecv	否	网络接收回调函数，同上；
cbDisConn	sixents_cbDisConn	否	网络断开回调函数，同上；
cbGetLocalIp	sixents_cbGetLocalIp	否	获取本地 IP 回调函数，默认值为 NULL。当开发者的系统需要在网络连接时 bind 本地 IP/端口时，应实现该回调接口；
cbGetLocalPort	sixents_cbGetLocalPort	否	获取本地连接端口回调函数，默认值为 NULL。当开发者的系统需要在网络连接时 bind 本地 IP/端口时，应实现该回调接口；

3.3. 初始化接口 sixents_sdkInit

3.3.1. 接口定义

```
sixents_retCode sixents_sdkInit(const sixents_sdkConf* paramObj);
```

3.3.2. 参数说明

参数	类型	说明
paramObj	sixents_sdkConf	初始化参数配置项的指针

3.3.3. 返回值说明

返回值类型为 sixents_retCode。

返回值	说明
SIXENTS_RET_OK	初始化成功
SIXENTS_RET_INVALID_STATUS	不合法的状态，不是初始调用，或没有成功调用注销接口
SIXENTS_RET_NULL_PTR	传入空指针
SIXENTS_RET_INVALID_PARAM	传入参数不合法，建议检查 paramObj 配置项
SIXENTS_RET_FAILED	初始化失败，系统错误

3.4. 启动接口 sixents_sdkStart

3.4.1. 接口定义

```
sixents_retCode sixents_sdkStart(void);
```

3.4.2. 参数说明

无。

3.4.3. 返回值说明

返回值类型为 sixents_retCode。

返回值	说明
SIXENTS_RET_OK	启动服务成功
SIXENTS_RET_INVALID_STATUS	不合法的状态，未成功调用初始化接口
SIXENTS_RET_START_FAILED	启动服务失败，连接服务器或鉴权失败

3.5. 驱动 SDK 运行接口 sixents_sdkTick

驱动 SDK 运行。

3.5.1. 接口定义

```
sixents_retCode sixents_sdkTick(void);
```

3.5.2. 参数说明

无。

3.5.3. 返回值说明

返回值类型为 sixents_retCode。

返回值	说明
SIXENTS_RET_OK	成功
SIXENTS_RET_INVALID_STATUS	SDK 状态不正确

3.6. 上传 GGA 接口 sixents_sdkSendGGAStr

3.6.1. 接口定义

```
sixents_retCode sixents_sdkSendGgaStr(const sixents_char* ggaData, sixents_uint16 ggaLen);
```

3.6.2. 参数说明

参数	类型	说明
ggaData	const sixents_char*	gga 数据内容，参考附录一 GGA 数据格式
ggaLen	Sixents_uint16	gga 字符串长度

3.6.3. 返回值说明

返回值类型为 sixents_retCode。

返回值	说明
SIXENTS_RET_OK	上传 GGA 成功，SDK 会调用 sixents_cbGetStatus 状态回调函数
SIXENTS_RET_INVALID_STATUS	不合法的状态，未成功调用启动接口
SIXENTS_RET_NULL_PTR	传入空指针
SIXENTS_RET_INVALID_PARAM	参数不合法，gga 长度为 0、校验码不正确
SIXENTS_RET_SOCKET_SEND_ERROR	网络发送失败
SIXENTS_RET_SOCKET_TIMEOUT_ERROR	网络发送超时

3.7. 上传 GGA 接口 sixents_sdkSendGGA

3.7.1. 接口定义

```
sixents_retCode sixents_sdkSendGGA(sixents_float64 lat,sixents_float64 lon,sixents_float64
height);
```

3.7.2. 参数说明

参数	类型	说明
lat	sixents_float64	纬度
lon	sixents_float64	经度
height	sixents_float64	高程

3.7.3. 返回值说明

返回值类型为 sixents_retCode。

返回值	说明
SIXENTS_RET_OK	上传 GGA 成功，SDK 会调用 sixents_cbGetStatus 状态回调函数
SIXENTS_RET_INVALID_STATUS	不合法的状态，未成功调用启动接口
SIXENTS_RET_INVALID_PARAM	参数不合法
SIXENTS_RET_OUT_OF_BOUNDARY	系统错误，数组越界
SIXENTS_RET_SOCKET_SEND_ERROR	网络发送失败
SIXENTS_RET_SOCKET_TIMEOUT_ERROR	网络发送超时

3.8. 停止接口 sixents_sdkStop

在启动接口调用成功后,开发者可以调用停止接口,调用停止接口后可以重新调用启动接口。

3.8.1. 接口定义

```
sixents_retCode sixents_sdkStop(void);
```

3.8.2. 参数说明

无。

3.8.3. 返回值说明

返回值类型为 sixents_retCode。

返回值	说明
SIXENTS_RET_OK	停止成功
SIXENTS_RET_INVALID_STATUS	不合法的状态，未成功调用启动接口

3.9. 注销接口 sixents_sdkFinal

在初始化接口调用成功后，开发者可以调用注销接口，调用注销接口后还可以重新调用初始化接口。

3.9.1. 接口定义

```
sixents_retCode sixents_sdkFinal(void);
```

3.9.2. 参数说明

无。

3.9.3. 返回值说明

返回值类型为 sixents_retCode。

返回值	说明
SIXENTS_RET_OK	停止成功

3.10. 获取版本接口 sixents_sdkGetVer

该接口用于获取 SDK 的版本信息，调用方式如下：

```
const sixents_char* ver = sixents_sdkGetVer();
```

3.10.1. 接口定义

```
const sixents_char* sixents_sdkGetVer(void);
```

3.10.2. 参数说明

无。

3.10.3. 返回值说明

返回值类型为 const sixents_char*。

返回值	说明
const sixents_char*	当前版本信息字符串。

3.11. 设置网络信息接口 sixents_sdkSetNwStatus

该接口用于外部通知 SDK 当前的网络信息。

许多嵌入式终端使用移动网络，网络初始化或者重连时需要调用此接口通知 SDK 当前网络状况，以便 SDK 可以正常处理网络连接或重连。

3.11.1. 接口定义

```
sixents_retCode sixents_sdkSetNw(sixents_nwStatus curNwStatus);
```

3.11.2. 参数说明

参数	类型	说明
curNwStatus	sixents_nwStatus	网络状态标志位，取值如下： SIXENTS_NWSTATUS_ON,表示连接正常 SIXENTS_NWSTATUS_OFF,表示连接断开

3.11.3. 返回值说明

返回值类型为 sixents_retCode。

返回值	说明
SIXENTS_RET_OK	设置网络状态成功
SIXENTS_RET_INVALID_PARAM	不合法的参数，设置网络状态失败

3.12. 差分数据回调接口 sixents_cbGetDiffData

3.12.1. 接口定义

```
typedef void (*sixents_cbGetDiffData)(sixents_char* buff, sixents_uint32 len);
```

3.12.2. 参数说明

参数	类型	说明
buff	sixents_char*	指向接收到的差分数据 buffer，开发者需从 buffer 中取出数据然后进行处理
len	sixents_uint32	有效数据长度

3.12.3. 返回值说明

无。

3.13. 状态信息回调接口 sixents_cbGetStatus

3.13.1. 接口定义

```
typedef void (*sixents_cbGetStatus)(sixents_uint32 status)
```

3.13.2. 参数说明

参数	类型	说明
status	sixents_uint32	状态码

3.13.3. 返回值说明

无。

3.13.4. 状态码说明

状态码	值	说明
SIXENTS_STATE_NET_REQUEEST_OK	1001	网络请求成功
SIXENTS_STATE_NET_REQUEEST_FAIL	1002	网络请求失败
SIXENTS_STATE_NET_DISABLE	1003	网络不可用
SIXENTS_STATE_AUTHENTICATE_OK	1201	鉴权成功
SIXENTS_STATE_AUTHENTICATE_AKORAS_INVALID	1202	参数无效

SIXENTS_STATE_AUTHENTICATE_AS_INVALID	1203	AS 无效
SIXENTS_STATE_AUTHENTICATE_ACCOUNT_PASS_LIMIT	1204	账号已用完
SIXENTS_STATE_AUTHENTICATE_ACCOUNT_INVALID	1205	账号不存在
SIXENTS_STATE_AUTHENTICATE_FAIL	1206	鉴权失败
SIXENTS_STATE_AUTHENTICATE_ACCOUNT_NOT_ACTIVE	1207	账号未激活
SIXENTS_STATE_AUTHENTICATE_ACCOUNT_OVERDUE	1208	账号已过期
SIXENTS_STATE_AUTHENTICATE_SERVER_EXCEPTION	1209	服务器异常
SIXENTS_STATE_AUTHENTICATE_LOADING	1210	AK/AS 鉴权中
SIXENTS_STATE_A_AUTHENTICATE_ACCOUNT_EXCEPTION	1211	账号异常，为空等
SIXENTS_STATE_AUTHENTICATE_BINDING_FAILURE	1212	绑定失败
SIXENTS_STATE_AUTHENTICATE_UNSUPPORTED_PROTOCOL	1213	不支持的协议
SIXENTS_STATE_AUTHENTICATE_DEVICE_NO_BINDING	1214	设备未手动绑定
SIXENTS_STATE_AUTHENTICATE_ACTIVED_FAILURE	1215	激活失败
SIXENTS_STATE_HAVE_LOGIN_FAILURE	1216	已登录
SIXENTS_STATE_INSTANCE_UNUSED_FAILURE	1217	没有账号
SIXENTS_STATE_NW_CONNECT_SUCCESS	1301	网络连接成功
SIXENTS_STATE_NW_CONNECT_FAIL	1302	网络连接失败
SIXENTS_STATE_NW_CONNECT_LOADING	1303	网络连接中
SIXENTS_STATE_NW_CONNECT_TIMEOUT	1304	连接超时
SIXENTS_STATE_NW_ACCOUNT_OUT	1305	同一账号在另外设备登录，提示账号不可以用
SIXENTS_STATE_SERVER_AUTHENTICATE_FAIL	1306	数据服务认证失败
SIXENTS_STATE_SERVER_AUTHENTICATE_SUCCESS	1307	数据服务认证成功
SIXENTS_STATE_SERVER_DOMAIN_ERROR	1308	域名解析错误
SIXENTS_STATE_RTCM_GET_SUCCESS	1401	RTCM 数据获取成功
SIXENTS_STATE_RTCM_GRID_OUT_OF_RANGE	1402	地理网格码不在服务范围
SIXENTS_STATE_RTCM_GGA_OUT_OF_RANGE	1403	GGA 不在服务范围
SIXENTS_STATE_RTCM_GGA_GET_TIMEOUT	1404	60 秒未收到 GGA 数据
SIXENTS_STATE_RTCM_GGA_SEND_EXCEPTION	1405	发送 GGA 数据时异常
SIXENTS_STATE_RTCM_GET_RTCM_TIMEOUT	1406	60 秒未获取到 RTCM 数据
SIXENTS_STATE_RTCM_SERVER_ERR	1407	RTCM 服务器错误
SIXENTS_STATE_RTCM_UNKNOWN_ERR	1408	未知错误
SIXENTS_STATE_GGA_SUCCESS	1501	GGA 数据有效
SIXENTS_STATE_GGA_INVALID	1502	GGA 数据无效

3.14. 网络连接回调接口 sixents_cbConn

该接口用于和服务端建立网络连接，需要创建网络套接字。

3.14.1. 接口定义

```
typedef sixents_int32(*sixents_cbConn)(const sixents_char* serverIP, sixents_uint16 ServerPort,
```

```
const sixents_char* localIP, sixnets_uint16 localPort, sixent_sockIOFlag sockIOFlag);
```

3.14.2. 参数说明

参数	类型	说明
serverIP	const sixents_char*	服务端 ip 地址;
serverPort	sixents_uint16	服务端端口号;
localIP	const sixents_char*	本地 IP，可以为空; 不为空时，用于 socket 调用 bind 本地 IP 端口;
localPort	sixents_uint16	本地端口，当 localIP 不为空时有效; 用于 socket 调用 bind 本地 IP 端口;
sockIOFlag	sixent_sockIOFlag	网络套接字阻塞模式标志位，可取值: SIXENTS_SOCK_IOFLAG_BLOCK，阻塞模式; SIXNETS_SOCK_IOFLAG_NOBLOCK，非阻塞模式; 注意：阻塞模式下，也应设置接收/发送的超时时间，建议设置范围[50,200],单位 ms;

3.14.3. 返回值说明

返回值	说明
SIXENTS_RET_OK	连接成功
SIXENTS_RET_NULL_PTR	传入 ip 地址不合法
SIXENTS_RET_SOCKET_CREATE_ERROR	套接字创建失败
SIXENTS_RET_SOCKET_BIND_ERROR	绑定本地 ip 失败
SIXENTS_RET_SOCKET_CONNECT_ERROR	连接错误，请检查 ip、port 参数是否正确，网络是否正常

3.15. 网络发送回调接口 sixents_cbSend

该接口用于向服务端发送数据。

3.15.1. 接口定义

```
typedef sixents_int32(*sixents_cbSend)(const sixents_char* buff, sixents_uint16 len);
```

3.15.2. 参数说明

参数	类型	说明
buff	const sixents_char*	待发送的数据
len	sixents_uint16	待发送数据长度（字节）

3.15.3. 返回值说明

返回值	说明
>0	发送成功，返回发送字节数
0	发送超时或失败
SIXENTS_RET_INVALID_PARAM	传入参数不合法
SIXENTS_RET_SOCKET_SEND_ERROR	发送失败

3.16. 网络接收回调接口 sixents_cbRecv

该接口用于接收服务端数据。

3.16.1. 接口定义

```
typedef sixents_int32(*sixents_cbRecv)(sixents_char* buff, sixents_uint16 len);
```

3.16.2. 参数说明

参数	类型	说明
buff	sixents_char*	接收数据的 buffer
len	sixents_uint16	接收数据的字节长度

3.16.3. 返回值说明

返回值	说明
>0	接收成功，返回接收字节数
0	接收超时，接收到 0 字节
SIXENTS_RET_INVALID_PARAM	传入参数不合法
SIXENTS_RET_SOCKET_RECV_ERROR	接收失败
SIXENTS_RET_SOCKET_DISCONNECTED	服务端已断开连接

3.17. 网络断开回调接口 sixents_cbDisConn

该接口用于断开与服务端的连接。

3.17.1. 接口定义

```
typedef sixents_int32(*sixents_cbDisConn)(void);
```

3.17.2. 参数说明

无。

3.17.3. 返回值说明

返回值	说明
SIXENTS_RET_OK	断开连接成功

3.18. 网络获取本地 IP 函数 sixents_cbGetLocalIp

该接口用于获取本地 IP 地址，在系统初始化或网络重连时被 SDK 调用。该接口一般和 sixens_cbGetLocalPort 一起使用。

在某些嵌入式设备使用移动网络，在网络连接或重连时本地 IP 可能会改变，如果网络连接时需要 bind 本地 IP 端口，则 SDK 需要调用本接口获取当前的本地 IP 和端口。

3.18.1. 接口定义

```
typedef sixents_int32 (*sixents_cbGetLocalIp)(sixents_char* localIp, sixents_uint16 ipLen);
```

3.18.2. 参数说明

参数	类型	说明
localIp	sixents_char*	ip 地址的 buffer，需以‘\0’结束
ipLen	sixents_uint16	localIp 的最大长度，为 SIXENTS_MAX_HOST_LEN

3.18.3. 返回值说明

返回值	说明
SIXENTS_RET_OK	获取本地 ip 成功
SIXENTS_RET_FAILED	获取本地 ip 失败

3.19. 网络获取本地端口函数 sixents_cbGetLocalPort

该接口用于获取本地网络端口，在系统初始化或网络重连时被 SDK 调用。该接口一般和 sixens_cbGetLocalIP 一起使用。

3.19.1. 接口定义

```
typedef sixents_int32 (*sixents_cbGetLocalPort)(void);
```

3.19.2. 参数说明

无。

3.19.3. 返回值说明

返回值	说明
sixents_int32	本地端口号

3.20. 日志打印回调接口 sixents_cbTrace

SDK 内部默认使用 printf 方式进行日志打印。当开发者的系统不支持 printf 或者需要重新定义自己的日志打印功能时，可实现该接口。

3.20.1. 接口定义

```
typedef sixents_int32 (*sixents_cbTrace) (const sixents_char* buff, sixents_uint16 len)
```

3.20.2. 参数说明

参数	类型	说明
buff	const sixents_char*	待打印消息
Len	sixents_uint16	待打印消息的字节数

3.20.3. 返回值说明

返回值	说明
SIXENTS_RET_OK	日志打印成功
SIXENTS_RET_FAILED	日志打印失败

3.21. 返回值类型 sixents_retCode

返回值	值	说明
SIXENTS_RET_OK	0	执行成功
SIXENTS_RET_FAILED	-1	执行失败
SIXENTS_RET_NULL_PTR	-2	空指针
SIXENTS_RET_INVALID_PARAM	-3	无效参数
SIXENTS_RET_INVALID_STATUS	-11	SDK 状态不合法
SIXENTS_RET_SOCKET_DISCONNECTED	-101	网络被断开
SIXENTS_RET_SOCKET_CREATE_ERROR	-102	网络创建失败
SIXENTS_RET_SOCKET_BIND_ERROR	-103	绑定本地 IP 失败
SIXENTS_RET_SOCKET_CONNECT_ERROR	-104	网络连接失败
SIXENTS_RET_SOCKET_SEND_ERROR	-105	网络发送失败
SIXENTS_RET_SOCKET_RECV_ERROR	-106	网络接收失败
SIXENTS_RET_SOCKET_TIMEOUT_ERROR	-107	网络超时
SIXENTS_RET_DNS_ERROR	-108	域名解析错误
SIXENTS_RET_NW_STATUS_ERROR	-109	网络状态错误
SIXENTS_RET_SOCKET_INVALID_ERROR	-110	无效的 socket
SIXENTS_RET_GGA_OUT_OF_SERVICE_AREA	-201	GGA 不在服务范围内
SIXENTS_RET_INVALID_GGA	-202	无效的 GGA
SIXENTS_RET_AUTH_FAILED	-301	鉴权失败
SIXENTS_RET_START_FAILED	-401	启动服务失败
SIXENTS_RET_OUT_OF_MEMORY	-501	内存空间溢出
SIXENTS_RET_START_THREAD_ERROR	-502	启动线程失败
SIXENTS_RET_OUT_OF_BOUNDARY	-507	数组越界
SIXENTS_RET_UNKNOWN	-999	未知错误

4. 使用限制说明

（1）不要在回调函数中执行耗时操作，不要在回调函数中执行死循环代码，否则会造成 SDK 无法正常工作。

（2）不要在回调函数中调用本文描述的接口。

（3）不建议使用多线程调用本文描述的接口。

（4）本 SDK 使用了<math.h>，使用 Linux 版本 SDK 时，注意编译时需要加载相应的.so 或者.a 库。

5. 常见问题

5.1. 初始化失败怎么办

首先请关注初始化接口 sixents_sdkInit 的返回值，只有当返回值为 SIXENTS_RET_OK（0）表示初始化成功，如果返回值小于 0,则在 sixents_cbGetStatus 回调函数中可能会捕获到状态码，根据接口返回值和状态码，查找失败原因，然后进行相应处理。

5.2. GGA 上传间隔一般设置多久

每次上传 GGA 的间隔建议在 1 秒或以上。

5.3. 如何确保持续接收到差分数据

当服务启动成功后，您需要定时调用 sixents_sdkSendGGAStr 或者 sixents_sdkSendGGA 接口上传终端设备的位置,即可在回调函数 sixents_cbGetDiffData 中持续接收到差分数据。同时，您需要实时监听 sixents_cbGetStatus 回调接口输出的状态码，进行相应的业务处理。

5.4. 如何处理网络重连

用户无需处理网络重连，SDK 会自动进行网络重连，直到 SDK 重连成功或者开发者调用 sixents_sdkStop 停止服务或者程序退出。一般网络异常（比如网络连接断开）、账号过期等会导致重连失败。开发者可以通过监听 sixents_cbGetStatus 回调接口中输出的状态码，查找原因，进行相应处理。

5.5. 如何集成自定义接口

如果嵌入式设备为非标准 POSIX 接口的操作系统，开发者需要根据 SDK 提供的接口，自行实现网络、日志打印等接口函数。集成操作过程如下：

（1）根据设备情况，选择完成自定义接口（回调函数）的开发，参考本文第 3.1.4 节描述；

（2）配置 sixents_sdkConf 结构体中回调接口；

（3）参照本文第 2 章完成 SDK 接口调用即可。

注意：对于非标准 POSIX 接口的操作系统，建议联系六分科技的客服或业务人员进行定制化 SDK 开发。

5.6. SDK 与您的嵌入式设备不兼容怎么办

由于 Free RTOS、RT-Thread 等各类实时嵌入式操作系统的底层不统一，本 SDK 针对网络层、线程、日志等部分提供了相应接口，需要用户根据各自设备系统，实现这些接口后再进行集成操作。如果您根据此操作指南，仍然无法成功在您的设备上运行本 SDK，建议开发者及时联系六分科技的客服或业务人员，我们需要根据您的设备平台进行定制开发。

附录一 GGA 数据格式

GGA 数据，是 NMEA-0183 协议格式中使用最广泛的数据之一，用来表示一帧 GPS 的定位数据，常见的 GGA 数据包含 GPGGA 或者 GNGGA 等。

$GPGGA 语句包括 17 个字段：语句标识头、UTC 时间、纬度、纬度半球、经度、经度半球，定位质量指示、使用卫星数量、HDOP、海拔高、高度单位、高程异常、高度单位、差分零期、差分参考基站标号、校验和结束标记(用回车符<CR>和换行符<LF>)，分别用 14 个逗号进行分隔。

GGA 数据格式示例：

```
$GPGGA,014434.70,3817.13334637,N,12139.72994196,E,4,07,1.5,6.571,M,8.942,M,0.7,0016*7B
```

该数据帧的结构及各字段释义见下表：

```
$GPGGA,<1>,<2>,<3>,<4>,<5>,<6>,<7>,<8>,<9>,M,<10>,M,<11>,<12>*xx< CR><LF>
```

字段	类别	详细格式说明
$GPGGA	语句标识头	起始引导符及语句格式说明(本句为 GPS 定位数据)；
<1>	UTC 时间	格式为 hhmmss.sss
<2>	纬度	格式为 ddmm.mmmm(第一位是零也将传送)
<3>	纬度半球	N 或 S(北纬或南纬)
<4>	经度	格式为 dddmm.mmmm(第一位零也将传送)；
<5>	经度半球	E 或 W(东经或西经)
<6>	定位质量指示	0 初始化 1 单点定位 2 码差分 3 无效 4 固定解 5 浮点解 6 正在估算 7 人工输入固定值 8 模拟模式 9 WAAS 差分
<7>	使用卫星数量	参与解算的卫星数量
<8>	HDOP	水平精度因子，0.5 到 99.9，一般认为 HDOP 越小，质量越好
<9>	海拔高	-9999.9 到 9999.9 米
M	海拔高单位	M 指单位米
<10>	高程异常	-9999.9 到 9999.9 米
M	高程异常单位	M 指单位米
<11>	差分零期	最新接收到的差分数据与当前时间的差值，以秒为

		单位，不是差分模式则为空
<12>	差分参考基站标号	从 0000 到 1023(首位 0 也将传送)，不是差分模式则为空
*	\	正文与校验位分隔符
xx	校验位	从$开始到*之间的所有 ASCII 码的异或校验值，以十六进制表示
<CR>	回车符	结束标记
<LF>	换行符	结束标记