CApi.h Description document

1. All encoding for strings is UTF-8

2. The reading and writing of network sessions are multithreaded safe, and their security depends on the event core Note: Only reading and writing is multithreaded safe

Event Core Construction and Functional Description

```
///< Building is building an event core
HEventCore * CreateEventCore();

///< This releases the event core and automatically cancels the blocking state of the event loop
internally .
void FreeEventCore(const HEventCore *core);

///< Block the current process and convert it into an event process .
HBool Exec(const HEventCore *core);

///< No blocking, an internal thread will be opened as an event thread .
HBool Run(const HEventCore *core);

///< Notify Exit Event Loop .
void Quit(const HEventCore *core);
```

Network Session Construction and Function Description

```
///< Create a network session with the core as the event core and the protocol as the protocol
enumeration. Please refer to the header file enumeration for details. When the protocol enumeration
is a non-existent value, SDK 2.0 will be built by default
HSession * CreateNetSession(const HEventCore *core, eNetProtocol protocol);

///< Release the protocol session, which will internally release the memory of the protocol
void FreeNetSession(HSession *session);

///< Set protocol session function
HBool SetNetSession(HSession *session, int type, void *data);
```

CApi.h Description document

Description of Network Session Function Settings

```
///< Universal callback, triggered by all network session protocols

///< Network status callback, which will be triggered when the network is connected or disconnected
///< CurrSession is the current network session
///< Status is the current status of the network session, as detailed in the header
file enumeration
///< UserData is the data passed by the user themselves
typedef void (*NetStatusCallBack)(HSession *currSession, eNetStatus status, void
*userData);

///< Tcp callback, only those of type TCP will trigger this callback

///< Data read callback, which will be triggered after the protocol processes the data
///< Data is the data read
///< Len is the length of the read data
```

```
typedef void (*ReadyReadCallBack)(HSession *currSession, const char *data,
huint32 len, void *userData);

///< Upload file callback, which will be triggered during file upload
///< FileName is the file name of the uploaded file
///< SendSize is the size of the current sent data
///< FileSize is the size of the file
///< Status is the current status of the uploaded file. Please refer to the
header file enumeration for details
typedef void (*UploadFileProgressCallBack)(HSession *currSession, const char
*fileName, hint64 sendSize, hint64 fileSize, eUploadFileStatus status, void
*userData);

///< Error message callback, which will be triggered upon receiving the error code
///< Status is the error code value specified in the document
typedef void (*ErrorCodeCallBack)(HSession *currSession, int status, void
*userData);


///< Tcp server callback
///< CurrSession The current session of the service
///< The client session connected to newSession still needs to call the
release interface when it is not needed
typedef void (*NewConnect)(HSession *currSession, HSession *newSession, void
*userData);

///< Udp callback
///< After setting the detection device, this callback will be triggered. There will be
ID, IP, and raw read data
typedef void (*DeviceInfoCallBack)(HSession *currSession, const char *id,
huint32 idLen, const char *ip, huint32 ipLen, const char *readData, huint32
dataLen, void *userData);
```

Network operation interface

```
///< Determine whether the current session is in a connected state
HBool Isconnect(const HSession *session);

///< Connect Session
HBool Connect(HSession *session, const char *ip, int port);

///< Disconnect
void Disconnect(HSession *session);

///< Send data interface, SDK can directly send XML data Internal data processing will be carried out
HBool SendSDK(HSession *session, const char *data, huint32 len);

enum eFileType {
    kImageFile          = 0,    ///< Image
    kVideoFile          = 1,    ///< Video
    kFont               = 2,    ///< typeface
    kFireware           = 3,    ///< Firmware
    kFPGAConfig         = 4,    ///< Under normal circumstances, it
                                is not necessary to use
    kSettingCofnig      = 5,    ///< Under normal circumstances, it
                                is not necessary to use
    KProjectResources   = 6,    ///< resource file , Under normal
                                circumstances, it is not necessary to
                                use
    kData               = 7,    ///< Under normal circumstances, it is
                                not necessary to use
    kTemp               = 8,    ///< Under normal circumstances, it is
                                not necessary to use
    kTempImageFile      = 128,  ///< Temporary image file
    kTempVideoFile      = 129,  ///< Temporary video file
};
```

```
///< Send file interface, input file path and file type
HBool SendFile(HSession *session, const char *filePath, int type);
```