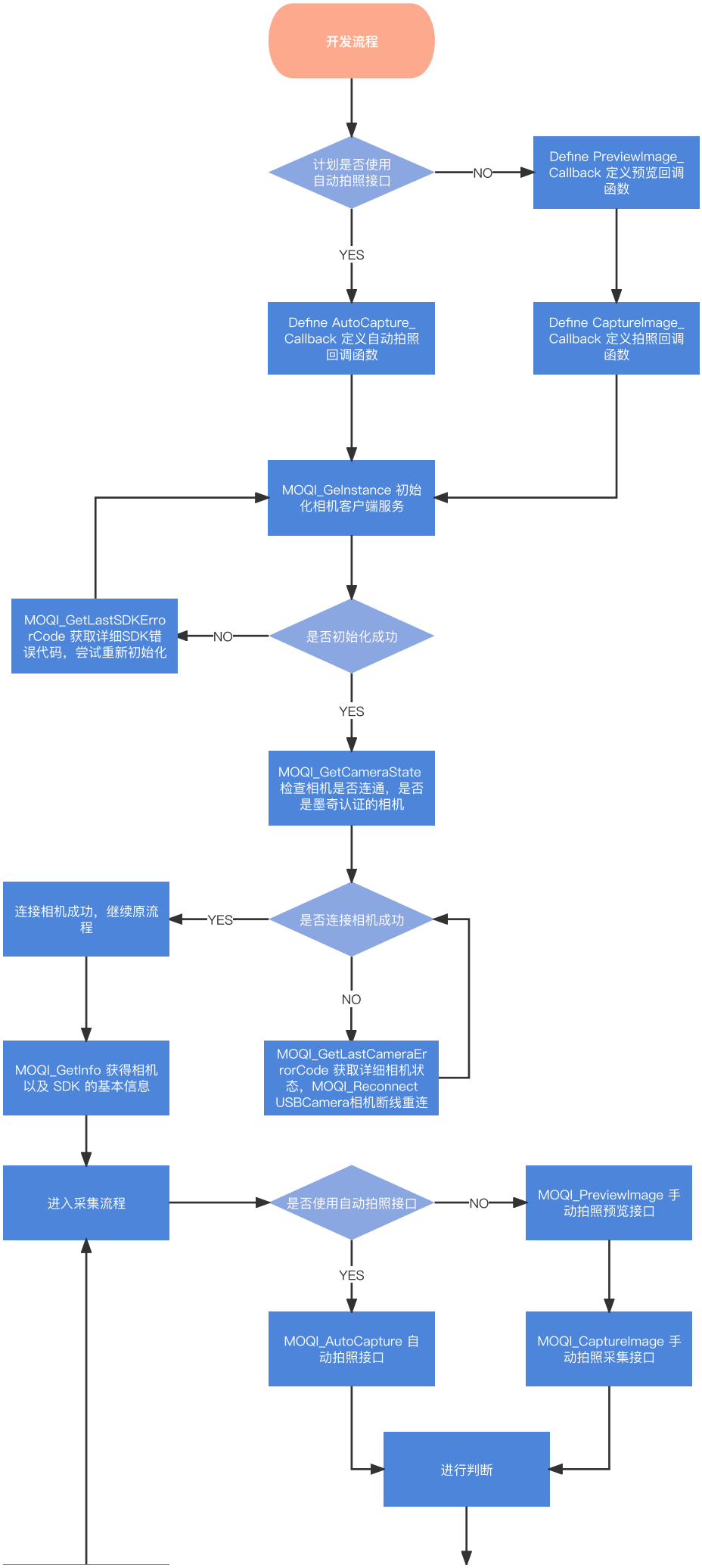
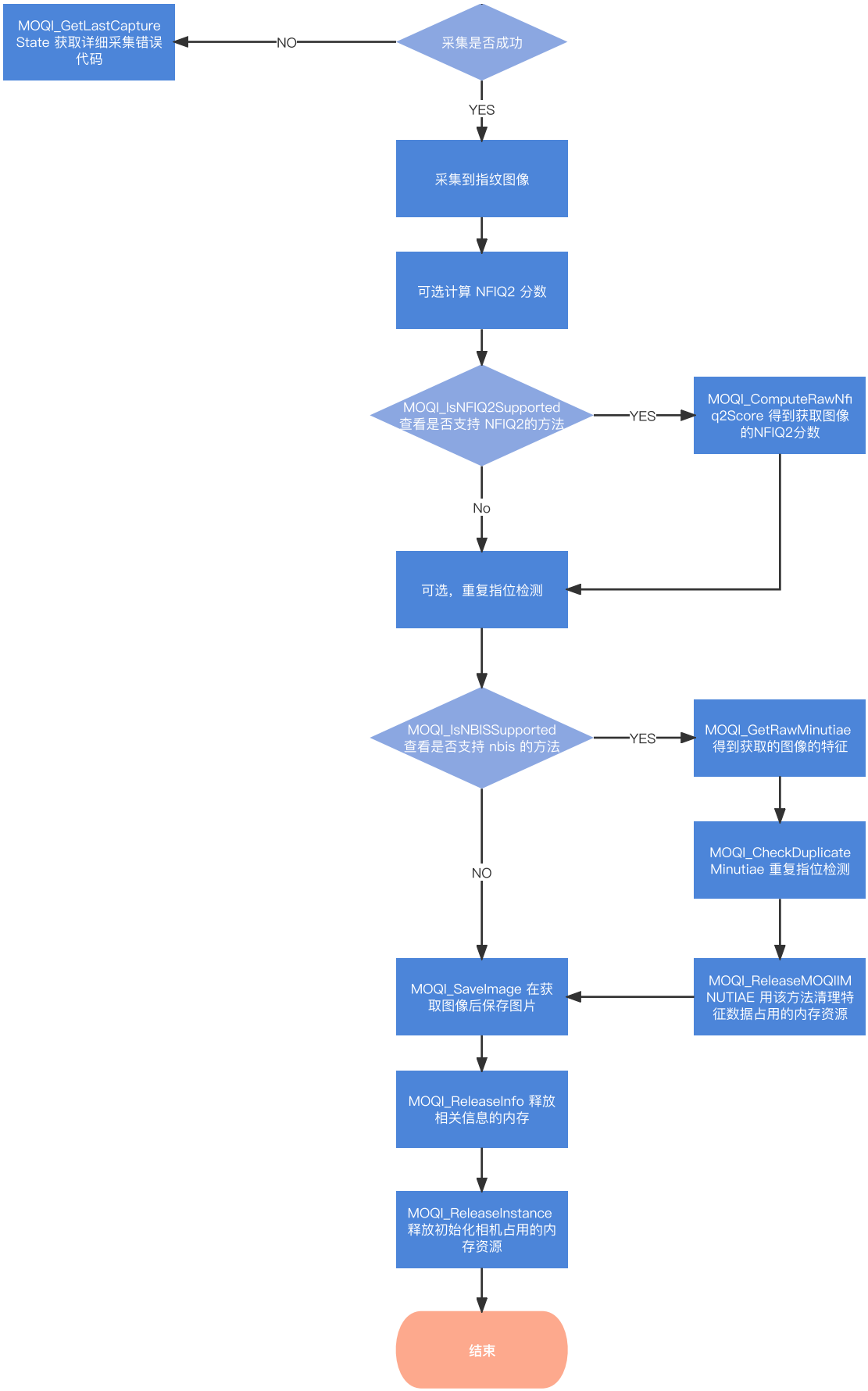
MQ-A1 SDK C/C++ API 使用手册

> 版本：v3.2.9

SDK 概述

在您开始看整个 SDK 的手册之前，您需要了解一个相对完整的 SDK 工作流程。 如下手动采集流程图显示了如何使用 SDK 的示例。当前设备分为标准版和 pro 版，通过 MOQI_GetInfo 的 is_pro 字段可以得知当前设备是否是 pro 版。对于标准版本，nfiq2（清晰度检测） 和 nbis（重复指位检测）相关功能是不支持的。


开发流程
计划是否使用自动拍照接口
NO
Define PreviewImage_Callback 定义预览回调函数
YES
Define AutoCapture_Callback 定义自动拍照回调函数
Define CaptureImage_Callback 定义拍照回调函数
MOQI_GeInstance 初始化相机客户端服务
MOQI_GetLastSDKErrorCode 获取详细SDK错误代码，尝试重新初始化
NO
是否初始化成功
YES
MOQI_GetCameraState 检查相机是否连通，是否是墨奇认证的相机
连接相机成功，继续原流程
YES
是否连接相机成功
NO
MOQI_GetInfo 获得相机以及 SDK 的基本信息
MOQI_GetLastCameraErrorCode 获取详细相机状态，MOQI_Reconnect USBCamera相机断线重连
进入采集流程
是否使用自动拍照接口
NO
MOQI_PreviewImage 手动拍照预览接口
YES
MOQI_AutoCapture 自动拍照接口
MOQI_CaptureImage 手动拍照采集接口
进行判断



如上显示了 SDK 基本的用法和流程：

1. 定义 PreviewImage_Callback
   此回调函数用于手动拍照接口，开发人员在预览图片的回调函数中定义，当相机处于预览状态下的时候，如何处理预览图片的逻辑和操作。
2. 定义 CaptureImage_Callback
   此回调函数用于手动拍照接口，相机获取完图片的时候，如何处理最终的图片。
3. 定义 AutoCapture_Callback
   此回调函数用于自动拍照接口，当相机处于预览状态，以及采集状态下的时候，如何处理预览图片的逻辑和操作。

4. MOQI_GetInstance()
   初始化相机客户端服务。
5. MOQI_GetCameraState()
   检查相机是否连通，是否是墨奇认证的相机。
6. MOQI_GetInfo()
   得到 SDK 的基本信息。相机的版本，SDK 的版本，设备的 SN 码，设备、固件对应的区域，以及每次预览图像的个数。
7. MOQI_ReconnectUSBCamera()
   当连接相机的 USB 线接触不良，或者被拔出重新接入时，可通过访问这个方法来重连相机。
8. MOQI_PreviewImage()
   开始预览相机的图片，当相机的预览图片视频流开始的时候，每获得一帧预览图片，上面定义的 PreviewImage_Callback 回调函数就会被触发。这里常用的回调函数处理预览图片的方法是在一个 UI 界面不停显示预览图片。以便用户调整手指的位置。
9. MOQI_CaptureImage()
   当用户通过预览图片调整好自己的手指位置的时候，这个时候，可以调用开始获取图片的函数。当相机成功抓取图片的时候，就会调用上面定义的获取图片的回调函数 CaptureImage_Callback。一般常用的图像处理都会在这个回调函数中进行。如保存图片，获取特征等。
10. MOQI_AutoCapture()
    进入自动采集状态， 在 callback 中，根据不同的阶段，分别得到预览和采集的图像。
11. MOQI_RawConvertBmp()

获取的图像都是 raw 格式。这个时候，可以通过这个方法，把 raw 转化成我们常用的 BMP 格式。

12. MOQI_IsNFIQ2Supported()
    查看是否支持 NFIQ2, 如果支持，继续进行下一步，如果不支持，跳过下一步。
13. MOQI_ComputeRawNfiq2Score()
    得到获取的图像的NFIQ2分数
14. MOQI_IsNBISSupported()
    查看是否支持重复职位检测，如果支持，继续进行下一步，如果不支持，直接跳到 18 步。
15. MOQI_GetRawMinutiae()
    得到获取的图像的特征。
16. MOQI_CheckDuplicateMinutiae()
    当我们通过如上 MOQI_CaptureImage() 获取到了两个手指的图像，并且分别提取了两张图片的特征。此时，用这个方法进行重复指位检测。
17. MOQI_ReleaseMOQIMINUTIAE()
    在获取特征的时候 MOQI_GetRawMinutiae() 会在 SDK 中有内存的占用。需要用该方法清理特征数据占用的内存资源。
18. MOQI_SaveImage()
    在获取图像后，保存图片。
19. MOQI_ReleaseInfo()
    清理之前保存相机基本信息的占用的内存资源。
20. MOQI_ReleaseInstance()
    应用退出之前，释放初始化相机客户端占用的内存资源。
21. MOQI_GetLastSdkErrorCode()
    获取上一次成功调用 MOQI_GetInstance 以及 MOQI_ReconnectUSBCamera 后保存的详细 SDK 错误代码。
22. MOQI_GetLastCameraErrorCode()
    获取上一次成功调用 MOQI_GetCameraState 后保存的详细相机状态。
23. MOQI_GetLastCaptureState()
    获取上一次进行指纹采集后保存的详细采集状态代码。

sdk.cfg 配置说明

当前 sdk.cfg 提供了 6 个参数，分别如下：

参数名	属性	说明
debug_folder	string	当该参数设置了合法的路径值，sdk 程序会创建文件夹，并将每次采集指纹图像产生的 debug 数据保存在该目录下，否则将跳过。注意， debug 数据较大，日常使用时需要注释掉该项。
need_adb	boolean	请填入**true**或者是**false**，该参数表示是否通过 sdk 初始化 ADB，默认是 true。
skip_process	boolean	请填入**true**或者是**false**，该参数表示是否通知采集仪跳过采集，直接返回空图，默认值是 false。
need_glog	boolean	请填入**true**或者是**false**，该参数表示是否在 sdk 内部初始化 glog（glog不能重复初始化），默认是 false。
capture_time_folder	string	当该参数设置了合法的路径值，sdk 程序会创建文件夹，并将每次采集指纹图像消耗的时间保存到 csv 文件中。
adb_server_port	string	该参数指定了 adb server 的监听端口，默认值为 5037。修改该数值可解决默认的端口被占用时 sdk 无法连接设备的问题。
adb_process_name_alias	list of string	该参数指定了 adb 进程的其他可能名称。名字与列表内字符串相同的进程将会被识别为 adb 进程。

sdk.cfg 文件内容：

```
// Sdk will save debug data in this folder.
// DELETE OR COMMENT OUT THIS LINE IN PRODUCTION ENVIROMENT!!!
// debug_folder=".\\debug"

// Do we need adb connection? The default value is true.
need_adb=true

// Do we need to skip the image process? The default value is false.
skip_process=false

// Do we need glog initialization from the sdk? The default value is false.
need_glog=false

// Adb server port, the default port is 5037
adb_server_port="5037"

// Save capture time in the given folder, or delete this item to close the function
capture_time_folder="./capture_time"

// If the image name of an running adb server is not "adb.exe", put the name in this list
adb_process_name_alias=("tadb.exe")
```

注意事项

端口占用

值得注意的是，在使用 sdk 连接后台相机的时候，使用了 adb 的连接方式， adb 程序会占用 `5037` 端口。所以，在特殊的情况下，会存在端口已经被占用的问题。这个时候，需要在配置文件 **sdk.cfg** 中指定一个可用端口作为 adb server 的监听端口。

当 `adb_server_port` 指定的端口被占用时，sdk 会检测占用该端口的进程名称，若被识别为 adb 进程则 sdk 会复用现有进程。但有些手机管家类的程序所使用的 adb 进程名称并非 “adb”，这会造成 sdk 连接设备失败，此时可以通过参数将指定名称的进程识别为 adb 进程。

重连的机制

通常来说, 除了研发人员最开始需要定义的一些基于采集预览的回调函数，第一个运行的函数就是，MOQI_GetInstance。比如自动采集流程如下:

- 定义 AutoCapture_Callback
- MOQI_GetInstance()

MOQI_GetInstance() 成功执行返回 MOQI_STATUS_OK，也会失败，此时会返回 MOQI_STATUS_FAIL。

- MOQI_STATUS_FAIL
  - 模块没有成功加载，无法继续进行下面的步骤，建议重新启动或者联系对接的研发人员。
  - 相机没有连通成功，可能是 USB 接口没有连接好，接触不良。也可能是 adb 的版本不兼容（可能性很小）。
    这个情况在连接的计算机不稳定的时候，比较容易出现。如，连接着 USB 线的笔记本被随便移动或者拉扯，造成接触不良。

对于后面的情况，只需要重新插紧松动的 USB 接口或者重新拔出再接入 USB 接口即可。然后调用 MOQI_ReconnectUSBCamera 如果这个函数返回 MOQI_STATUS_OK，那么就说明我们的设备重新连接上了我们的 SDK 。推荐使用一个 while 循环不停检查 MOQI_ReconnectUSBCamera 直至返回 MOQI_STATUS_OK。

关于 MOQI_ReconnectUSBCamera 的说明和用法

理想条件下，根据最上面的采集流程图即可顺利使用我们的服务。

但如果研发人员，希望在研发过程中，对于在调用任意其他函数的期间由于各种原因发生的设备连接断开的情况（如在如下情景，用户正在预览图像的过程中，工作人员整理文件碰到了连接设备的电脑，导致 USB 线松开，从而导致设备和电脑的连接断开），增加断线重连的处理并且重试这一步骤（如，此时工作人员又把 USB 线插紧，希望继续进行用户刚才的预览这一步骤），则应该频繁使用 MOQI_ReconnectUSBCamera 这个函数。

每个与设备交互的函数都会在设备连接断开的时返回一个相同的错误码 MOQI_ERR_CAMERA_IS_NOT_CONNECTED，在收到这个错误码的时候，研发人员可以修复连接线，同时持续地调用 MOQI_ReconnectUSBCamera，直至成功，然后重试之前的函数。

如下简要展示如何处理自动采集过程中，设备连接断开的问题：

```
int result = MOQI_AutoCapture(); // 自动采集
if (result != MOQI_STATUS_OK) {
    if (result == MOQI_ERR_CAMERA_IS_NOT_CONNECTED) {  // 采集过程中，连接断开了
        while(MOQI_ReconnectUSBCamera() != MOQI_STATUS_OK) {  // 不停地重试，直到重连成功
        }
        MOQI_AutoCapture();  // 重连成功之后，可以重复上面的步骤。
    }
}
```

sdk 相关的 dll 和 sdk.cfg 的存放问题

通常条件下，调用方，类似于提供的 demo 例子都是直接把所有的 sdk 相关的 dll 都放到与调用 sdk 的 exe 文件的同级目录下。如下，camera_server.exe 使用了相关的 contactlesssdk.dll 等 library，这些 library 被放到了与 camera_server.exe 相同的目录下。

```
 ~/win_x86-32  ll
total 182752
-rwxr-xr-x@ 1 moqi  staff    96K Oct 23 20:08 AdbWinApi.dll
-rwxr-xr-x@ 1 moqi  staff    62K Oct 23 20:08 AdbWinUsbApi.dll
-rwxr-xr-x@ 1 moqi  staff   903K Oct 21 18:54 abseil_dll.dll
-rwxr-xr-x@ 1 moqi  staff   2.5M Oct 23 20:08 adb.exe
-rwxr-xr-x@ 1 moqi  staff    65K Oct 21 18:59 cares.dll
-rwxr-xr-x@ 1 moqi  staff   1.5M Oct 23 20:08 contactlesssdk.dll
-rwxr-xr-x@ 1 moqi  staff   101K Oct 21 18:55 gflags.dll
-rwxr-xr-x@ 1 moqi  staff   118K Oct 21 18:57 glog.dll
-rwxr-xr-x@ 1 moqi  staff    70M Oct 23 20:08 libMOQI_NFIQ2.dll
-rwxr-xr-x@ 1 moqi  staff    51K Oct 21 18:56 libconfig++.dll
-rwxr-xr-x@ 1 moqi  staff    35K Oct 21 18:56 libconfig.dll
-rwxr-xr-x@ 1 moqi  staff   119K Oct 23 20:08 libgcc_s_dw2-1.dll
-rwxr-xr-x@ 1 moqi  staff   927K Oct 23 20:08 libnbis_sdk.dll
-rwxr-xr-x@ 1 moqi  staff   3.7M Oct 23 20:08 libopencv_core2413.dll
-rwxr-xr-x@ 1 moqi  staff   3.3M Oct 23 20:08 libopencv_imgproc2413.dll
-rwxr-xr-x@ 1 moqi  staff   1.3M Oct 23 20:08 libopencv_ml2413.dll
-rwxr-xr-x@ 1 moqi  staff   162K Oct 21 19:01 libpng16.dll
-rwxr-xr-x@ 1 moqi  staff   2.1M Oct 21 18:59 libprotobuf.dll
-rwxr-xr-x@ 1 moqi  staff   1.8M Oct 23 20:08 libstdc++-6.dll
-rwxr-xr-x@ 1 moqi  staff    62K Oct 23 20:08 libwinpthread-1.dll
-rwxr-xr-x@ 1 moqi  staff   859K Oct 22 12:12 re2.dll
-rwxr-xr-x@ 1 moqi  staff    73K Oct 21 19:01 zlib1.dll
```

在一些其他情况，有些研发人员则会把所有的 dll 都放到系统的 dll 目录下，比如 `C:\Windows\SysWOW64`， 这个时候由于 sdk 会通过 contactlesssdk.dll 本身所在的目录去追踪其配置 sdk.cfg，那么 sdk.cfg 也应该被复制到这个目录下面，以方便 contactlesssdk.dll 找到这个配置文件，同时包括 `adb.exe, adbWinApi.dll AdbWinUsbApi.dll` 在内的相关的都应该放在与 contactlesssdk.dll 相同的路径下

API 快速查看列表

所有 SDK 方法和回调方法的总结

NO	API Functions
1	int MOQI_WINAPI MOQI_GetInstance(int *handle_ptr, char *server_url)
2	int MOQI_WINAPI MOQI_ReleaseInstance(const int handle)
3	int MOQI_WINAPI MOQI_GetCameraState(int handle)
4	int MOQI_WINAPI MOQI_GetInfo(int handle, MOQI_INFO*& info)
5	int MOQI_WINAPI MOQI_ReleaseInfo(int handle, MOQI_INFO*& info)
6	int MOQI_WINAPI MOQI_ReconnectUSBCamera(int handle)
7	int MOQI_WINAPI MOQI_AudioPlay(int handle, const char *file_content, long size)
8	int MOQI_WINAPI MOQI_AudioSetMute(int handle, int mute)
9	int MOQI_WINAPI MOQI_AudioSetVolume(int handle, int volume)
10	int MOQI_WINAPI MOQI_PreviewImage(int handle, uint64_t request_id, int finger_index, MOQI_IMAGE_TYPE image_type, PreviewImage_Callback cb, void *context_ptr)

NO	API Functions
11	int MOQI_WINAPI MOQI_CancelProcess(int handle)
12	int MOQI_WINAPI MOQI_CaptureImage(int handle, uint64_t request_id, int finger_index, MOQI_IMAGE_TYPE image_type, CaptureImage_Callback callback, void *context_ptr)
13	int MOQI_WINAPI MOQI_AutoCapture(int handle, int finger_index, MOQI_IMAGE_TYPE preview_image_type, MOQI_IMAGE_TYPE capture_image_type, AutoCapture_Callback callback, void *context_ptr)
14	int MOQI_WINAPI MOQI_RawConvertBmp(int handle, MOQI_IMAGE raw_image, char *bmp_ptr)
15	int MOQI_WINAPI MOQI_IsNFIQ2Supported(int handle)
16	int MOQI_WINAPI MOQI_ComputeRawNfiq2Score(int handle, const char *image_ptr, int *score)
17	int MOQI_WINAPI MOQI_IsNBISSupported(int handle)
18	int MOQI_WINAPI MOQI_GetRawMinutiae(int handle, const char *raw_image, int raw_image_size, MOQI_MINUTIAE *&minutiae_ptr)
19	int MOQI_WINAPI MOQI_GetRawFileMinutiae(int handle, char *file_path, MOQI_MINUTIAE *&minutiae_ptr)
20	int MOQI_WINAPI MOQI_SaveImage(int handle, const char *file_path_ptr, int path_size, const char *image_ptr, int size)
21	int MOQI_WINAPI MOQI_CheckDuplicateMinutiae(int handle, MOQI_MINUTIAE *probe_minutiae_ptr, MOQI_MINUTIAE *target_minutiae_ptr, int *score_ptr)
22	int MOQI_WINAPI MOQI_ReleaseMOQIMINUTIAE(int handle, MOQI_MINUTIAE *target)
23	int MOQI_WINAPI MOQI_GetLastSdkErrorCode()
24	int MOQI_WINAPI MOQI_GetLastCameraErrorCode(int handle)
25	int MOQI_WINAPI MOQI_GetLastCaptureState(int handle)
26	int MOQI_WINAPI MOQI_GenerateIso19794File(const char** image_array, const int* image_size_array, const int* finger_index_array, const int num_fingers, char*& file_binary, int& file_size)
27	int MOQI_WINAPI MOQI_ReleaseCharArray(char* char_array)

NO	Callback Functions
1	typedef int (MOQI_CALLBACK *CaptureImage_Callback) (const int handle, uint64_t request_id, MOQI_CAPTURE_IMAGE image, int finger_index, void *context_ptr);
2	typedef int (MOQI_CALLBACK *PreviewImage_Callback)(const int handle, uint64_t request_id, MOQI_PREVIEW_IMAGE image, int* cancel_flag_ptr, void *context_ptr, const int currentNumber);
3	typedef int (MOQI_CALLBACK *AutoCapture_Callback) (const int handle, int finger_index, MOQI_AUTO_CAPTURE_STATE state, MOQI_PREVIEW_IMAGE preview_image, int* cancel_flag_ptr, const int current_preview_number, MOQI_CAPTURE_IMAGE capture_image, void *context_ptr);

API 方法

MOQI_GetInstance

- 原型
  - `int MOQI_WINAPI MOQI_GetInstance(int *handle_ptr, char *server_url)`
- 描述
  - 初始化设备服务对象，获得对象的句柄。这个句柄用来帮助索引服务对象方便使用后续的方法。方法访问过后，handle_ptr 指向对象的索引。此初始化的对象占用堆内存。在整个应用退出前，需要 MOQI_ReleaseInstance() 释放堆内存。
- 参数
  - handle_ptr
    - 指向服务对象的句柄，除了本方法，其他方法都需要用到这个句柄指向的值
  - server_url
    - SDK 基于 grpc 与设备通信。这个 URL 指向构建于设备上的服务。
- 返回
  - 成功
    - MOQI_STATUS_OK
  - 失败

        - MOQI_STATUS_FAIL
        - MOQI_ERR_CAMERA_IS_NOT_CONNECTED 相机没有联通

MOQI_ReleaseInstance

- 原型
    - `int MOQI_WINAPI MOQI_ReleaseInstance(const int handle)`
- 描述
    - 释放 MOQI_GetInstance() 占用的内存。在应用退出之前，必须释放该资源。
- 参数
    - handle
        - 设备服务对象的句柄
- 返回
    - 成功
        - MOQI_STATUS_OK
    - 失败
        - MOQI_STATUS_FAIL 失败

MOQI_GetCameraState

- 原型
    - `int MOQI_WINAPI MOQI_GetCameraState(int handle)`
- 描述
    - 检查相机是否连通，是否是墨奇认证的相机。
- 参数
    - handle
        - 设备服务对象的句柄
- 返回
    - 成功
        - MOQI_STATUS_OK
    - 失败
        - MOQI_ERR_CAMERA_NOT_FOUND 相机没有找到
        - MOQI_ERR_CAMERA_INVALID 非法的相机设备
        - MOQI_ERR_CAMERA_ERROR 相机发生了错误
        - MOQI_ERR_CAMERA_IS_NOT_CONNECTED 相机没有连通

MOQI_GetInfo

- 原型
    - `int MOQI_WINAPI MOQI_GetInfo(int handle, MOQI_INFO*& info)`
- 描述
    - 得到 SDK 的基本信息。相机的版本，SDK 的版本，设备的 SN 码，设备、固件对应的区域，以及每次预览图像的个数
- 参数
    - handle
        - 设备服务对象的句柄
- 返回
    - 成功
        - MOQI_STATUS_OK
    - 失败
        - MOQI_STATUS_FAIL 失败
        - MOQI_ERR_CAMERA_IS_NOT_CONNECTED 相机没有联通
        - MOQI_ERR_GRPC_REQUEST_FAIL 跟设备服务通信失败

MOQI_ReleaseInfo

- 原型
    - `int MOQI_WINAPI MOQI_ReleaseInfo(int handle, MOQI_INFO*& info)`
- 描述
    - 如上函数在 SDK 中占用了内存，需要用本方法将对应的内存释放
- 参数
    - handle
        - 设备服务对象的句柄

- 返回
  - 成功
    - MOQI_STATUS_OK
  - 失败
    - MOQI_STATUS_FAIL 失败

MOQI_ReconnectUSBCamera

- 原型
  - `int MOQI_WINAPI MOQI_ReconnectUSBCamera(int handle);`
- 描述
  - 当连接相机的 USB 线接触不良，或者被拔出重新接入时，可通过访问这个方法来重连相机。
    此方法推荐使用合适的时间间隔多次调用，因为相机重新连接后，会需要一段时间初始化连接。
- 参数
  - handle
    - 设备服务对象的句柄
- 返回
  - 成功
    - MOQI_STATUS_OK 重新连接成功
  - 失败
    - MOQI_STATUS_FAIL 重新连接失败

MOQI_AudioPlay

- 原型
  - `int MOQI_WINAPI MOQI_AudioPlay(int handle, const char *file_content, long size)`
- 描述
  - 播放声音文件。当前支持 WAV，MP3 和 OGG 格式。
- 参数
  - handle
    - 设备服务对象的句柄
  - file_content
    - 指向文件内容的指针
  - size
    - 文件大小
- 返回
  - 成功
    - MOQI_STATUS_OK
  - 失败
    - MOQI_ERR_GRPC_REQUEST_FAIL 跟设备服务通信失败
    - MOQI_ERR_CAMERA_IS_NOT_CONNECTED 相机没有联通

MOQI_AudioSetMute

- 原型
  - `int MOQI_WINAPI MOQI_AudioSetMute(int handle, int mute)`
- 描述
  - 设置设备是否静音，只对搭配 4.8.0 以及之前版本固件的设备有效，自 4.8.1 版本的固件开始静音功能被禁用。
- 参数
  - handle
    - 设备服务对象的句柄
  - mute
    - 静音开关
- 返回
  - 成功
    - MOQI_STATUS_OK
  - 失败
    - MOQI_ERR_GRPC_REQUEST_FAIL 跟设备服务通信失败
    - MOQI_ERR_CAMERA_IS_NOT_CONNECTED 相机没有联通

MOQI_AudioSetVolume

- 原型
  - `int MOQI_WINAPI MOQI_AudioSetVolume(int handle, int volume)`
- 描述
  - 设置设备音量，若固件版本大于等于 4.8.1 则最低设备音量被锁定为 20。
- 参数
  - handle
    - 设备服务对象的句柄
  - volume
    - 音量大小
- 返回
  - 成功
    - MOQI_STATUS_OK
  - 失败
    - MOQI_ERR_GRPC_REQUEST_FAIL 跟设备服务通信失败
    - MOQI_ERR_CAMERA_IS_NOT_CONNECTED 相机没有联通

MOQI_PreviewImage

- 原型
  - `int MOQI_WINAPI MOQI_PreviewImage(int handle, uint64_t request_id, int finger_index, MOQI_IMAGE_TYPE image_type, PreviewIma`
- 描述
  - 联通相机开始预览，预览图片可以在 PreviewImage_Callback 得到。这里仅仅给用户一个简单的图像，用来辅助用户调整手指的位置和方向。当用户的手指和方向处在合理的位置。这个时候，可以使用 MOQI_CaptureImage() 来得到最佳效果的图片，这将停止相机的预览模式，进入精确的图像采集模式。同时，用户也可以 MOQI_CancelProcess() 来取消当前的预览图片，断开视频流。
- 参数
  - handle
    - 设备服务对象的句柄
  - request_id
    - 请求 ID, 用户可在回调函数中，分辨是否属于同一个会话的请求
  - finger_index
    - 指位（指头的位置）， 1 到 5 分别表示右手拇指到小指，6 到 10 分别表示左手的拇指到小指
  - image_type
    - 预览图片的类型，当前只支持 JPEG （MOQI_IMAGE_FORMAT_JPEG）
  - cb
    - 预览图片的回调函数。这个函数的内部逻辑由用户来定义。当预览图片准备好的时候，会触发这个函数。这个函数每秒会被触发数次，用户在这个函数中对预览图片进行处理，比如在 web UI 不停的显示当前的预览图片，以调整自己手指的位置。
  - context_ptr
    - 用户自己的传入的上下文。这个对象将会作为回调函数里的参数。用户可以在回调函数中使用自己传入的上下文
- 返回
  - 成功
    - MOQI_STATUS_OK
  - Error
    - MOQI_ERR_INVALID_IMAGE_TYPE 非法图像类型
    - MOQI_ERR_INVALID_PREVIEW_IMAGE_STATE 非法的预览图像状态
    - MOQI_ERR_GRPC_REQUEST_FAIL 跟设备服务通信失败
    - MOQI_ERR_CAMERA_IS_NOT_CONNECTED 相机没有联通

MOQI_CancelProcess

- 原型
  - `int MOQI_WINAPI MOQI_CancelProcess(int handle)`
- 描述
  - 这个方法会停止当前的设备相机的预览状态。这个方法可以在 PreviewImage_Callback 中调用。也可以另起一个新的线程，在新的线程中调用，来停止当前线程的预览状态。
- 参数
  - handle
    - 设备服务对象的句柄
- 返回
  - 成功
    - MOQI_STATUS_OK

- 失败
  - MOQI_ERR_CONTEXT_NULL 预览的 grpc 上下文为空

MOQI_CaptureImage

- 原型
  - `int MOQI_WINAPI MOQI_CaptureImage(int handle, uint64_t request_id, int finger_index, MOQI_IMAGE_TYPE image_type, CaptureIma`
- 描述
  - 当设备在预览图片的时候，当用户认为当前手指位置和方向在一个正确的范围内，则调用本获取图像方法。来停止预览模式。开始进入获取图片的模式中。
- 参数
  - handle
    - 设备服务对象的句柄
  - request_id
    - 请求 ID, 用户可在回调函数中，分辨是否属于同一个会话的请求
  - finger_index
    - 指位（指头的位置）， 1 到 5 分别表示右手拇指到小指，6 到 10 分别表示左手的拇指到小指
  - image_type
    - 获取的图片格式，当前仅支持 RAW（推荐），BMP
  - cb
    - The capture image callback, the function is defined by the user. when the captured image is in process or ready. The current process and capture image could be fetched in the callback.
    - 获取图像的回调函数。方法由用户定义。当采集图片完成，或者采集图片过成功发生错误（MOQI_CAPTURE_IMAGE.state） 会触发这个回调函数的方法。在这个方法中，用户可以添加自己对最终采集的图像的处理
  - context_ptr
    - 用户自己的传入的上下文。这个对象将会作为回调函数里的参数。用户可以在回调函数中使用自己传入的上下文
- 返回
  - 成功
    - MOQI_STATUS_OK
  - 失败
    - MOQI_ERR_INVALID_IMAGE_TYPE 非法的图像类型
    - MOQI_ERR_WRONG_RAW_IMAGE_SIZE Raw 格式的图像类型不对
    - MOQI_ERR_INVALID_CAPTURE_IMAGE_STATE 获取图像状态不合法
    - MOQI_ERR_CAPTURE_IMAGE_FAIL 获取图像失败
    - MOQI_ERR_GRPC_REQUEST_FAIL 跟设备服务通信失败
    - MOQI_ERR_CAMERA_IS_NOT_CONNECTED 相机没有联通

MOQI_AutoCapture

- 原型
  - `int MOQI_WINAPI MOQI_AutoCapture(int handle, int finger_index, MOQI_IMAGE_TYPE preview_image_type, MOQI_IMAGE_TYPE capture_`
- 描述
  - 调用此方法进入自动采集模式，自动采集模式下，设备根据用户手指的位置角度自动判断是否进入采集状态，并进行采集
- 参数
  - handle
    - 设备服务对象的句柄
  - finger_index
    - 指位（指头的位置）， 1 到 5 分别表示右手拇指到小指，6 到 10 分别表示左手的拇指到小指
  - preview_image_type
    - 获取的预览图片格式，当前仅支持 RAW（推荐），BMP
  - capture_image_type
    - 获取的预览图片格式，当前仅支持 RAW（推荐），BMP
  - cb
    - 获取预览图像以及采集图像的回调函数。方法由用户定义。在回调函数中，用户可根据预览或者采集图像来分别增加对应的处理
  - context_ptr
    - 用户自己的传入的上下文。这个对象将会作为回调函数里的参数。用户可以在回调函数中使用自己传入的上下文
- 返回
  - 成功
    - MOQI_STATUS_OK
  - 失败
    - MOQI_ERR_INVALID_IMAGE_TYPE 非法的图像类型
    - return MOQI_ERR_INVALID_PREVIEW_IMAGE_STATE 非法的预览图像状态

        - MOQI_ERR_WRONG_RAW_IMAGE_SIZE Raw 格式的图像类型不对
        - MOQI_ERR_INVALID_CAPTURE_IMAGE_STATE 获取图像状态不合法
        - MOQI_ERR_CAPTURE_IMAGE_FAIL 获取图像失败
        - MOQI_ERR_GRPC_REQUEST_FAIL 跟设备服务通信失败
        - MOQI_ERR_CAMERA_IS_NOT_CONNECTED 相机没有联通

MOQI_RawConvertBmp

- 原型
    - `int MOQI_WINAPI MOQI_RawConvertBmp(int handle, MOQI_IMAGE raw_image, char *bmp_ptr)`
- 描述
    - 把图像从 raw 转成 bmp，用户需要在外部定义存储 bmp 的 buffer
- 参数
    - handle
        - 设备服务对象的句柄
    - raw_image
        - CaptureImage_Callback 中 获得的 raw 图像
    - bmp_ptr
        - 指向 bmp 数据内存的指针。用户调用这个方法前，需要定义好存储 bmp buffer 的空间。 BMP 头文件54字节。BMP 调色板为 1024 字节。如果一个图像宽 640 像素，高 640 像素。那么这个 BMP 文件的大小就是 54 + 1024 + 640*640 = 410678. 所以，用户可以定义 char bmp_ptr[410678], 然后传入这个 bmp_ptr。
- 返回
    - 成功
        - MOQI_STATUS_OK
    - 失败
        - MOQI_ERR_ERROR_ADDING_BMP_HEADER

MOQI_IsNFIQ2Supported

- 原型
    - `int MOQI_WINAPI MOQI_IsNFIQ2Supported(int handle)`
- 描述
    - 查看当前是否支持 nfiq2 的计算
- 参数
    - handle
        - 设备服务对象的句柄
- 返回
    - 成功
        - MOQI_STATUS_OK
    - 失败
        - MOQI_ERR_NO_SUPPORT_FOR_PRO_METHOD

MOQI_ComputeRawNfiq2Score

- 原型
    - `int MOQI_WINAPI MOQI_ComputeRawNfiq2Score(int handle, const char *image_ptr, int* score)`
- 描述
    - 计算图像 NFIQ2 的清晰度值
- 参数
    - handle
        - 设备服务对象的句柄
    - image_ptr
        - 指向图像的指针
    - score
        - 指向最终 NFIQ2 结果的指针
- 返回
    - Success
        - MOQI_STATUS_OK
    - Error
        - MOQI_ERR_CALCULATE_NFIQ2_SCORE 在计算 NFIQ2 分数的时候发生了错误
        - MOQI_ERR_NO_SUPPORT_FOR_PRO_METHOD 这是一个 pro 的方法，标准版无法调用

MOQI_IsNBISSupported

- 原型
  - `int MOQI_WINAPI MOQI_IsNBISSupported(int handle)`
- 描述
  - 查看当前是否支持重复指位检测 的计算
- 参数
  - handle
    - 设备服务对象的句柄
- 返回
  - 成功
    - MOQI_STATUS_OK
  - 失败
    - MOQI_ERR_NO_SUPPORT_FOR_PRO_METHOD

MOQI_GetRawMinutiae

- 原型
  - `int MOQI_WINAPI MOQI_GetRawMinutiae(int handle, const char *raw_image, int raw_image_size, MOQI_MINUTIAE *&minutiae_ptr)`
- 描述
  - 获得 raw 图像的特征
- 参数
  - handle
    - 设备服务对象的句柄
  - raw_image
    - Raw 图像的指针
  - raw_image_size
    - Raw 图像的大小
  - minutiae_ptr
    - 指向特征点的指针。通过这个方法，SDK 会内部开辟存储特征点的内存并把结果写入。内存的的地址赋给 minutiae_ptr 指针变量。当用户结束使用该指针（如结束重复指位检测 MOQI_CheckDuplicateMinutiae() ），应释放特征所占用的内存 MOQI_ReleaseMOQIMINUTIAE()
- 返回
  - 成功
    - MOQI_STATUS_OK
  - 失败
    - MOQI_ERR_GET_MINUTIAE_ERROR 获取特征点失败
    - MOQI_ERR_NO_SUPPORT_FOR_PRO_METHOD 这是一个 pro 的方法，标准版无法调用

MOQI_GetRawFileMinutiae

- 原型
  - `int MOQI_WINAPI MOQI_GetRawFileMinutiae(int handle, char *file_path, MOQI_MINUTIAE *&minutiae_ptr)`
- 描述
  - 获得 raw 图像的特征
- 参数
  - handle
    - 设备服务对象的句柄
  - file_path
    - 保存 raw image 的文件路径
  - minutiae_ptr
    - 指向特征点的指针。通过这个方法，SDK 会内部开辟存储特征点的内存并把结果写入。内存的的地址赋给 minutiae_ptr 指针变量。当用户结束使用该指针（如结束重复指位检测 MOQI_CheckDuplicateMinutiae() ），应释放特征所占用的内存 MOQI_ReleaseMOQIMINUTIAE()
- 返回
  - 成功
    - MOQI_STATUS_OK
  - 失败
    - MOQI_ERR_GET_MINUTIAE_ERROR 获取特征失败
    - MOQI_ERR_NO_SUPPORT_FOR_PRO_METHOD 这是一个 pro 的方法，标准版无法调用

MOQI_SaveImage

- 原型

    - `int MOQI_WINAPI MOQI_SaveImage(int handle, const char *file_path_ptr, int path_size, const char *image_ptr, int size)`
- 描述
    - 存储文件
- 参数
    - handle
        - 设备服务对象的句柄
    - file_path_ptr
        - 保存文件的路径字符数组指针
    - path_size
        - 保存文件的路径字符数组长度
    - image_ptr
        - 指向图像的指针
    - size
        - 图像的大小
- 返回
    - 成功
        - MOQI_STATUS_OK, if successful.
    - 失败
        - MOQI_ERR_IMAGE_DATA_EMPTY 图像数据为空
        - MOQI_ERR_FILE_PATH_EMPTY 图像文件路径为空

MOQI_CheckDuplicateMinutiae

- 原型
    - `int MOQI_WINAPI MOQI_CheckDuplicateMinutiae(int handle, MOQI_MINUTIAE *probe_minutiae_ptr, MOQI_MINUTIAE *target_minutiae_p`
- 描述
    - 重复指位检测
- 参数
    - handle
        - 设备服务对象的句柄
    - probe_minutiae_ptr
        - 发比对的图像特征的指针
    - target_minutiae_ptr
        - 被比对图像的特征的指针
    - score_ptr
        - 指向 NBIS 分数的指针
- 返回
    - 成功
        - MOQI_STATUS_OK
    - 失败
        - MOQI_ERR_MINUTIAE_TO_INNER_MINUTIAE 转化内部特征数据失败
        - MOQI_ERR_CHECK_DUPLICATE_MINUTIAE 重复指位检测失败
        - MOQI_ERR_RELEASE_MOQI_MINUTIAE 释放内部特征失败
        - MOQI_ERR_NO_SUPPORT_FOR_PRO_METHOD 这是一个 pro 的方法，标准版无法调用

MOQI_ReleaseMOQIMINUTIAE

- 原型
    - `int MOQI_WINAPI MOQI_ReleaseMOQIMINUTIAE(int handle, MOQI_MINUTIAE *target)`
- 描述
    - 释放由 MOQI_GetRawMinutiae 或者 MOQI_GetRawFileMinutiae 获得特征外部特征。
- 参数
    - handle
        - 设备服务对象的句柄
    - target
        - 指向要被释放内存的特征的指针
- 返回
    - 成功
        - MOQI_STATUS_OK, if successful.
    - 失败
        - MOQI_STATUS_FAIL
        - MOQI_ERR_NO_SUPPORT_FOR_PRO_METHOD 这是一个 pro 的方法，标准版无法调用

MOQI_GetLastSdkErrorCode

- 原型
  - `int MOQI_WINAPI MOQI_GetLastSdkErrorCode()`
- 描述
  - 获取上一次成功调用 MOQI_GetInstance 以及 MOQI_ReconnectUSBCamera 后保存的详细 SDK 错误代码
  - 若上述两个函数未被调用过或者调用时返回 MOQI_STATUS_FAIL，则本函数固定返回 MOQI_STATUS_OK
- 参数
  - 无
- 返回
  - MOQI_STATUS_OK (0) 没有错误发生
  - MOQI_ERR_SDK_NOT_INITIALIZED (300) SDK 尚未初始化
  - MOQI_ERR_SDK_ABNROMAL_DRIVER (301) 驱动有问题（没有安装&没安装成功）
  - MOQI_ERR_SDK_DEVICE_NOT_FOUND (302) 未检测到 A1 设备
  - MOQI_ERR_SDK_A1_SERVER_MALFUNCTION (303) A1 server 故障
  - MOQI_ERR_SDK_ADB_PORT_ERROR (304) adb 端口错误（比如被占用）

MOQI_GetLastCameraErrorCode

- 原型
  - `int MOQI_WINAPI MOQI_GetLastCameraErrorCode(int handle)`
- 描述
  - 获取上一次成功调用 MOQI_GetCameraState 后保存的详细相机状态
  - 若上述函数未被调用过或者调用时返回 MOQI_STATUS_FAIL，则本函数固定返回 -1
- 参数
  - handle
    - 设备服务对象的句柄
- 返回
  - 0 没有错误发生
  - 101 磁盘空间不足
  - 102 内存容量异常,少于 4G
  - 201 A1 玻璃脏污
  - 202 npu 故障, transfer 进程没有启动
  - 203 npu 内存故障
  - 204 npu inference 结果出错
  - 205 letctl set 失败,即 单片机通信故障

MOQI_GetLastCaptureState

- 原型
  - `int MOQI_WINAPI MOQI_GetLastCaptureState(int handle)`
- 描述
  - 获取上一次进行指纹采集后保存的详细采集状态代码
- 参数
  - handle
    - 设备服务对象的句柄
- 返回
  - MOQI_CAPTURE_STATE_CODE_CAPTURE_OK 0 采集成功
  - MOQI_CAPTURE_STATE_CODE_FINGER_INCORRECT_POSITION 3 手指位置不正确
  - MOQI_CAPTURE_STATE_CODE_UNKNOWN_ERROR 98 未知错误
  - MOQI_CAPTURE_STATE_CODE_CAPTURE_ERROR 99 相机故障
  - MOQI_CAPTURE_STATE_CODE_UNDISTORT_ERROR 103 A1 undistort 错误
  - MOQI_CAPTURE_STATE_CODE_SL_INIT_ERROR 104 A1 sl init error
  - MOQI_CAPTURE_STATE_CODE_SLDETECT_ERROR 105 A1 结构光模型检测失败
  - MOQI_CAPTURE_STATE_CODE_SLORDER_ERROR 106 A1 结构光顺序检查失败
  - MOQI_CAPTURE_STATE_CODE_BUILDFD_ERROR 107 A1 三维重建及方向计算错误
  - MOQI_CAPTURE_STATE_CODE_UNROLLING_ERROR 108 A1 展开步骤发生错误
  - MOQI_CAPTURE_STATE_CODE_BADCALIB_ERROR 109 A1 散点纠正失败
  - MOQI_CAPTURE_STATE_CODE_BLURMASK_ERROR 110 A1 模糊判断不达标造成失败
  - MOQI_CAPTURE_STATE_CODE_ENHANCE_ERROR 111 A1 模型增强失败
  - MOQI_CAPTURE_STATE_CODE_RECALIB_ERROR 112 A1 需要重新校准

MOQI_GenerateIso19794File

- 原型
  - `int MOQI_WINAPI MOQI_GenerateIso19794File(const char** image_array, const int* image_size_array, const int* finger_index_ar`
- 描述
  - 将一组指纹图片保存为 ISO/IEC19794-4 格式，文件数据保存在 file_binary 数组中，使用完毕后需要使用 MOQI_ReleaseCharArray 函数释放占用的资源
- 参数
  - image_array
    - 指纹图片数据数组，每一项为一个数组指针，指向一张图片的数据数组
  - image_size_array
    - image_array 中每张指纹图片的数据长度
  - finger_index_array
    - image_array 中每张图片对应的指位代码
  - num_fingers
    - 指纹图片数量
  - file_binary
    - 生成文件的数据数组
  - file_size
    - file_binary 的长度
- 返回
  - 成功
    - MOQI_STATUS_OK, if successful.
  - 失败
    - MOQI_STATUS_FAIL

MOQI_ReleaseCharArray

- 原型
  - `int MOQI_WINAPI MOQI_ReleaseCharArray(char* char_array)`
- 描述
  - 释放 sdk 申请的 `char*` 数组
- 参数
  - char_array
    - sdk 申请的字符数组指针
- 返回
  - 成功
    - MOQI_STATUS_OK, if successful.

回调方法

CaptureImage_Callback

- 原型
  - `typedef int (MOQI_CALLBACK CaptureImage_Callback) (const int handle, uint64_t request_id, MOQI_CAPTURE_IMAGE image, int fin`
- 描述
  - MOQI_CaptureImage 的回调函数。当获取图片完成或者失败的时候，会触发这个方法。当方法被触发的时候，用户应优先获取图像的 capture state，只有在图像的 state 是 ok 的条件下，再进行进一步的图像处理，比如 MOQI_RawConvertBmp() 可以把 raw 图像变成 BMP 图像等。
- 参数
  - handle
    - 设备服务对象的句柄
  - request_id
    - 请求 ID, 用户可在回调函数中，分辨是否属于同一个会话的请求
  - image
    - 当前获取的图像。这个图像只在当前回调函数内存在。如果用户希望在回调函数外使用。需要把该图像保存到预先准备好的内存区域。
  - finger_index
    - 用户的指位
  - context_ptr
    - 用户调用 MOQI_CaptureImage()传入的上下文
- 返回
  - 成功

        - MOQI_STATUS_OK
    - 失败
        - MOQI_STATUS_FAIL

PreviewImage_Callback

- 原型
    - `typedef int (MOQI_CALLBACK PreviewImage_Callback) (const int handle, uint64_t request_id, MOQI_PREVIEW_IMAGE image, int* ca`
- 描述
    - MOQI_PreviewImage() 的回调函数。当预览图像准备好的时候， 这个回调函数就会被触发。在设备处于预览模式下，这个方法会每秒会被触发数次，来处理预览图像的视频流。
- 参数
    - handle
        - 设备服务对象的句柄
    - request_id
        - 请求 ID, 用户可在回调函数中，分辨是否属于同一个会话的请求
    - image
        - 当前获取的图像。这个图像只在当前回调函数内存在。如果用户希望在回调函数外使用。需要把该图像保存到预先准备的内存区域。
    - cancel_flag_ptr
        - 用户可以设置这个指针执行的标志位，当这个值再这个回调函数中为设置成 1 时，则取消当前预览流程
    - context_ptr
        - 用户调用 MOQI_CaptureImage()传入的上下文
- 返回
    - 成功
        - MOQI_STATUS_OK
    - 失败
        - MOQI_STATUS_FAIL

AutoCapture_Callback

- 原型
    - `typedef int (MOQI_CALLBACK *AutoCapture_Callback) (const int handle, int finger_index, MOQI_AUTO_CAPTURE_STATE state, MOQI_`
- 描述
    - MOQI_AutoCapture() 的回调函数。当预览图像或者采集的图像生成好的时候， 这个回调函数就会被触发。在设备处于预览模式下，这个方法会每秒会被触发数次，来处理预览图像的视频流，当用户手指位置正确，则会进入采集状态，许等若干秒才得到最后处理过后的采集图像。
- 参数
    - handle
        - 设备服务对象的句柄
    - finger_index
        - 请求采集的指位
    - state
        - 当前相机的模式，开始的时候是预览模式，当用户深入手指，并且位置正确会进入采集模式
    - preview_image
        - 当前相机在预览模式下，获得的预览图像
    - cancel_flag_ptr
        - 用户可以设置这个指针执行的标志位，当这个值再这个回调函数中为设置成 1 时，则取消当前预览流程
    - current_preview_number
        - 用户当前预览的第几张图像
    - capture_image
        - 当前相机在采集模式下，获得的采集图像
    - context_ptr
        - 用户调用 MOQI_AutoCapture()传入的上下文
- 返回
    - 成功
        - MOQI_STATUS_OK
    - 失败
        - MOQI_STATUS_FAIL

Data Structure

MOQI_IMAGE_TYPE 枚举

当前 SDK 中所有的数据类型

```
/* RAW 图像 */
MOQI_IMAGE_FORMAT_RAW = 5,       // MOQI_CaptureImage

/* BMP 图像*/
MOQI_IMAGE_FORMAT_BMP = 0,      // 未实现

/* PNG 图像 */
MOQI_IMAGE_FORMAT_PNG = 1,      // 未实现

/* JPEG 图像 */
MOQI_IMAGE_FORMAT_JPEG = 2,     // MOQI_PreviewImage

/* JPEG 2000 图像 */
MOQI_IMAGE_FORMAT_JPEG2000 = 3, // 未实现

/* WSQ 图像 */
MOQI_IMAGE_FORMAT_WSQ = 4       // 未实现
```

AUTO_MOQI_CAPTURE_STATE 枚举

在自动采集的过程中，表示当前设备采集的状态。在自动采集过程中，会先进入预览状态，
这个时候，如果用户深入手指进行采集，如果手指位置标准且正确， 则进入采集状态

```
/* 当前处于预览状态 */
MOQI_AUTO_CAPTURE_STATE_PREVIEW = 0,

/* 当前处于采集状态 */
MOQI_AUTO_CAPTURE_STATE_CAPTURE = 1
```

MOQI_CAPTURE_STATE 枚举

获取图像的时候所有的状态。即 CaptureImage_Callback 中 MOQI_CAPTURE_IMAGE 的 state。

```
/* 获取图像正确 */
MOQI_CAPTURE_STATE_CAPTURE_OK = 0,

/* 手指位置不正确 */
MOQI_CAPTURE_STATE_FINGER_INCORRECT_POSITION = 3,

/* 未知错误 */
MOQI_CAPTURE_STATE_UNKNOWN_ERROR = 98,

/* 相机获取图像错误 */
MOQI_CAPTURE_STATE_CAPTURE_ERROR = 99,
```

MOQI_PREVIEW_STATE 枚举

预览图像的时候所有的状态。即 PreviewImage_Callback 中 *MOQI_PREVIEW_IMAGE *的 state。

```
    /* Finger position is ok*/
    MOQI_PREVIEW_STATE_PREVIEW_OK = 0,              // 手指位置合适，可以采集

    /* Finger position is right need move left*/
    MOQI_PREVIEW_STATE_MOVE_LEFT = 1,               // 手指位置偏右，需要左移

    /* Finger position is left need move right*/
    MOQI_PREVIEW_STATE_MOVE_RIGHT = 2,              // 手指位置偏左，需要右移

    /* Finger position is back need move forward*/
    MOQI_PREVIEW_STATE_MOVE_FORWARD = 3,            // 手指位置偏后，需要前移

    /* Finger position is in front need move back*/
    MOQI_PREVIEW_STATE_MOVE_BACKWARD = 4,           // 手指位置偏前，需要后移

    /* move finger to left and forward*/
    MOQI_PREVIEW_STATE_MOVE_LEFT_AND_FORWARD = 5,   // 手指位置偏右后，需要左前移

    /* move finger to left and back*/
    MOQI_PREVIEW_STATE_MOVE_LEFT_AND_BACKWARD = 6,  // 手指位置偏右前，需要左后移

    /* move finger to right and front*/
    MOQI_PREVIEW_STATE_MOVE_RIGHT_AND_FORWARD = 7,  // 手指位置偏左后，需要右前移

    /* move finger to right and back*/
    MOQI_PREVIEW_STATE_MOVE_RIGHT_AND_BACKWARD = 8, // 手指位置偏左前，需要右后移

    /* move finger up */
    MOQI_PREVIEW_STATE_MOVE_UP = 9,                 // 手指位置下，需要上移

    /* move finger down */
    MOQI_PREVIEW_STATE_MOVE_DOWN = 10,              // 手指位置上，需要下移

    /* rotate finger left */
    MOQI_PREVIEW_STATE_ROTATE_LEFT = 11,            // 手指位置偏右，需要向左转

    /* rotate finger right */
    MOQI_PREVIEW_STATE_ROTATE_RIGHT = 12,           // 手指位置偏左，需要向右转

    /* preview finished */
    MOQI_PREVIEW_STATE_PREVIEW_FINISHED = 98,       // 预览结束，开始自动采集

    /* Finger has preview error */
    MOQI_PREVIEW_STATE_PREVIEW_ERROR = 99,          // 相机故障
```

MOQI_IMAGE 结构体

一个由指向图像数据头部的指针和大小表示的图像

```
typedef struct tag_MOQI_IMAGE {
    /* 指向图像数据的指针。这个指针不可以被直接保存，其图像数据需要保存到其他系统的缓存，
    以做后续处理。因为当回调函数返回时，该指针指向的地址失效 */
    const char *frame;

    /* 图像大小 */
    int size;
}
```

MOQI_CAPTURE_IMAGE 结构体

获取图像中 MOQI_CaptureImage， MOQI_CAPTURE_IMAGE 数据在 CaptureImage_Callback 中使用，表示获取到的图像数据

```
typedef struct tag_MOQI_CAPTURE_IMAGE {
    /* 获取的平面指纹图像 */
    MOQI_IMAGE *plain_image;

    /* 获取的平面滚动捺印图像 */
    MOQI_IMAGE *roll_image;

    /* 获取图像的图像状态 */
    MOQI_CAPTURE_STATE state;
} MOQI_CAPTURE_IMAGE;
```

MOQI_PREVIEW_IMAGE 结构体

当相机开始 MOQI_PreviewImage 预览图像时，MOQI_PREVIEW_IMAGE 数据在 PreviewImage_Callback 中使用，表示预览的图像数据

```
typedef struct tag_MOQI_PREVIEW_IMAGE {
    /* 一次预览的图片（当前为两张） */
    MOQI_IMAGE **images;

    /* 一次预览的图片数量 */
    int num;

    /* 当前预览图像的状态 */
    MOQI_PREVIEW_STATE state;
}
```

MOQI_MINUTIA 结构体

MOQI_MINUTIA 包括一枚指纹的一个特征点及其相关信息

```c
typedef struct tag_MOQI_MINUTIA {
    /* 特征点的 x 坐标 */
    int x;

    /* 特征点的 y 坐标 */
    int y;

    /* 保留字 */
    int ex;

    /* 保留字 */
    int ey;

    /* 方向 */
    int direction;

    /* 可靠性 */
    double reliability;

    /* 0 - 分叉
       1 - 端点 */
    int type;

    /* 保留字 */
    int appearing;

    /* 保留字 */
    int feature_id;

    /* 保留字 */
    int *nbrs;

    /* 保留字 */
    int *ridge_counts;

    /* 保留字 */
    int num_nbrs;
}
```

MOQI_MINUTIAE 结构体

MOQI_MINUTIAE 保存了一个指纹图像所有特征点。

```c
typedef struct tag_MOQI_MINUTIAE {
    /* 分配的特征点 */
    int alloc;

    /* 实际特征个数 */
    int num;

    /* 指向一个成员是特征点地址的数组 */
    MOQI_MINUTIA **list;
}
```

MOQI_STRING 结构体

MOQI_STRING 保存了一个字符串

```c
typedef struct tag_MOQI_STRING {
  /* 指向字符串的指针 */
  const char* string_pointer;

  /* 字符串的长度不包括最后的字符串结束符 '\0' */
  int string_size;
} MOQI_STRING;
```

MOQI_POST_FLAG 枚举

设备自检时得到的设备状态。在 MOQI_INFO 结构体中可以得到该信息

```
typedef enum enum_MOQI_POST_FLAG {
  POST_OK = 0,
  DISK_FULL = 101, //磁盘空间不足
  MEM_LESS4 = 102, //内存容量异常,少于 4G
  NPUTRANS_FAIL = 202, // npu 故障, transfer 进程没有启动
  NPUMEM_FAIL = 203, // npu 内存故障
  NPUINFER_FAIL = 204, // npu inference 结果出错
  LEDSET_FAIL = 205 //letctl set 失败,即 单片机通信故障
} MOQI_POST_FLAG;
```

MOQI_INFO 结构体

MOQI_INFO 保存了 SDK 及其设备的基本信息

```
typedef struct tag_MOQI_INFO {
  /* 当前设备的版本 */
  MOQI_STRING machine_version;

  /* 当前 SDK 版本 */
  MOQI_STRING sdk_version;

  /* 设备 SN 码 */
  MOQI_STRING device_sn;

  /* 预览图片的数量 */
  int preview_image_num;

  /* 是否是 pro 版 */
  bool is_pro;

  MOQI_POST_FLAG post_flag;

  MOQI_STRING clf_lang;

  MOQI_STRING machine_lang;

} MOQI_INFO;
```