# ArcSoft Face Detection

## 开发指导文档

ArcSoft Corporation
46601 Fremont Blvd.
Fremont, CA 94538
http://www.arcsoft.com

**Trademark or Service Mark Information**

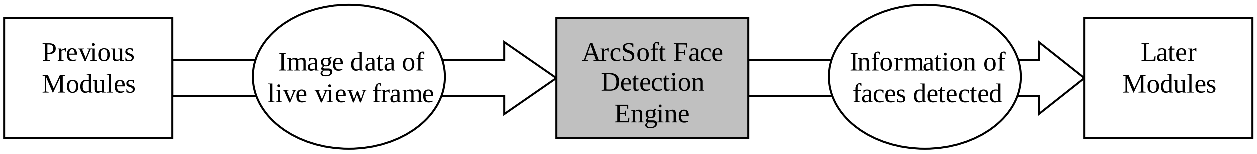
ArcSoft Inc. and ArcWare are registered trademarks of ArcSoft Inc.

Other product and company names mentioned herein may be trademarks and/or service marks of their respective owners. The absence of a trademark or service mark from this list does not constitute a waiver of ArcSoft Inc.'s trademark or other intellectual property rights concerning that trademark or service mark.

The information contained in this document is for discussion purposes only. None of the information herein shall be interpreted as an offer or promise to any of the substance herein nor as an agreement to contract or license, or as an implication of a transfer of rights. Any and all terms herein are subject to change at the discretion of ArcSoft. Copying, distributing, transferring or any other reproduction of these documents or the information contained herein is expressly prohibited, unless such activity is expressly permitted by an authorized representative of ArcSoft, Inc.

# 1. 概述

虹软人脸检测引擎工作流程图：


Previous Modules
Image data of live view frame
ArcSoft Face Detection Engine
Information of faces detected
Later Modules

## 1.1. 运行环境

- Linux x64

## 1.2. 系统要求

- 库依赖 GLIBC 2.19 及以上
- 编译器 GCC 4.8.2 及以上

## 1.3. 依赖库

- libsqlite3
- libcurl

# 2. 结构与常量

## 2.1. 基本类型

```
typedef MInt32 AFD_FSDK_OrientPriority;
typedef MInt32 AFD_FSDK_OrientCode;
```

所有基本类型在平台库中有定义。 定义规则是在 ANSIC 中的基本类型前加上字母“M”同时将类型的第一个字母改成大写。例如“long” 被定义成“MLong”

## 2.2. 数据结构与枚举

### 2.2.1. AFD_FSDK_FACERES

**描述**

检测到的脸部信息

**定义**

```
typedef struct{
    MInt32    nFace;
    MRECT     *rcFace;
    AFD_FSDK_OrientCode     *lfaceOrient;
} AFD_FSDK_FACERES, *LPAFD_FSDK_FACERES;
```

**成员变量**

| | |
|---|---|
| rcFace | 人脸矩形框信息 |
| nFace | 人脸个数 |
| lfaceOrient | 人脸角度信息 |

### 2.2.2. AFD_FSDK_VERSION

**描述**

SDK 版本信息

**定义**

```
 typedef struct
 {
   MInt32 lCodebase;
```

```
    MInt32 lMajor;
    MInt32 lMinor;
    MInt32 lBuild;
    MPChar Version;
    MPChar BuildDate;
    MPChar CopyRight;
 } AFD_FSDK_Version;
```

**成员描述**

| | |
|---|---|
| lCodebase | 代码库版本号 |
| lMajor | 主版本号 |
| lMinor | 次版本号 |
| lBuild | 编译版本号，递增 |
| Version | 字符串形式的版本号 |
| BuildDate | 编译时间 |
| CopyRight | 版权信息 |

## 2.2.3. AFD_FSDK_OrientPriority

**描述**

定义脸部检测角度的优先级

**定义**

```
enum _AFD_FSDK_OrientPriority{
    AFD_FSDK_OPF_0_ONLY           = 0x1,
    AFD_FSDK_OPF_90_ONLY          = 0x2,
    AFD_FSDK_OPF_270_ONLY         = 0x3,
    AFD_FSDK_OPF_180_ONLY         = 0x4,
    AFD_FSDK_OPF_0_HIGHER_EXT     = 0x5
};
```

**成员描述**

| | |
|---|---|
| AFD_FSDK_OPF_0_ONLY | 检测 0 度（±45 度）方向 |
| AFD_FSDK_OPF_90_ONLY | 检测 90 度（±45 度）方向 |
| AFD_FSDK_OPF_270_ONLY | 检测 270（±45 度）度方向 |
| AFD_FSDK_OPF_180_ONLY | 检测 180 度（±45 度）方向 |
| AFD_FSDK_OPF_0_HIGHER_EXT | 检测 0， 90， 180， 270 四个方向, 0 度更优先 |

## 2.2.4. AFD_FSDK_OrientCode

**描述**

定义检测结果中的人脸角度

**定义**

```
enum _AFD_FSDK_OrientCode{
    AFD_FSDK_FOC_0   = 0x1,
    AFD_FSDK_FOC_90  = 0x2,
    AFD_FSDK_FOC_270 = 0x3,
    AFD_FSDK_FOC_180 = 0x4,
    AFD_FSDK_FOC_30  = 0x5,
    AFD_FSDK_FOC_60  = 0x6,
    AFD_FSDK_FOC_120 = 0x7,
    AFD_FSDK_FOC_150 = 0x8,
    AFD_FSDK_FOC_210 = 0x9,
    AFD_FSDK_FOC_240 = 0xa,
    AFD_FSDK_FOC_300 = 0xb,
    AFD_FSDK_FOC_330 = 0xc
};
```

**成员描述**

| | |
|---|---|
| AFD_FSDK_FOC_0 | 0 度 |
| AFD_FSDK_FOC_90 | 90 度 |
| AFD_FSDK_FOC_270 | 270 度 |
| AFD_FSDK_FOC_180 | 180 度 |
| AFD_FSDK_FOC_30 | 30 度 |
| AFD_FSDK_FOC_60 | 60 度 |
| AFD_FSDK_FOC_120 | 120 度 |
| AFD_FSDK_FOC_150 | 150 度 |
| AFD_FSDK_FOC_210 | 210 度 |
| AFD_FSDK_FOC_240 | 240 度 |
| AFD_FSDK_FOC_300 | 300 度 |
| AFD_FSDK_FOC_330 | 330 度 |

### 2.2.5. 支持的颜色格式

| 定义 | 说明 |
|---|---|
| ASVL_PAF_I420 | 8-bit Y 通道，8-bit 2x2 采样 U 通道，8-bit 2x2 采样 V 通道 |
| ASVL_PAF_NV12 | 8-bit Y 通道，8-bit 2x2 采样 U 与 V 分量交织通道 |
| ASVL_PAF_NV21 | 8-bit Y 通道，8-bit 2x2 采样 V 与 U 分量交织通道 |
| ASVL_PAF_YUYV | YUV 分量交织，V 与 U 分量 2x1 采样，按 Y0, U0, Y1, V0 字节序排布 |
| ASVL_PAF_RGB24_B8G8R8 | RGB 分量交织，按 B, G, R, B 字节序排布 |

# 3. API 说明

## 3.1. AFD_FSDK_InitialFaceEngine

**原型**

```
MRESULT AFD_FSDK_InitialFaceEngine(
       MPChar                             AppId,
       MPChar                             SDKKey,
       MByte                              *pMem,
       MInt32                             lMemSize,
       MHandle                            *pEngine,
       AFD_FSDK_OrientPriority            iOrientPriority,
       MInt32                             nScale,
       MInt32                             nMaxFaceNum
);
```

**描述**

初始化脸部检测引擎

**参数**

| | | |
|---|---|---|
| AppId | [in] | 用户申请SDK时获取的App Id |
| SDKKey | [in] | 用户申请SDK时获取的SDK Key |
| pMem | [in] | 分配给引擎使用的内存地址 |
| lMemSize | [in] | 分配给引擎使用的内存大小 |
| pEngine | [out] | 引擎handle |
| iOrientPriority | [in] | 期望的脸部角度的检测范围 |
| nScale | [in] | 用于数值表示的最小人脸尺寸 有效值范围[2,50] 推荐值 16。<br>该尺寸是人脸相对于所在图片的长边的占比。例如，如果用户想检测到的最小人脸尺寸是图片长度的1/8，那么这个nScale就应该设置为8 |
| nMaxFaceNum | [in] | 用户期望引擎最多能检测出的人脸数 有效值范围[1,50] |

**返回值**

成功返回MOK，否则返回失败code。失败codes如下所列:

| | |
|---|---|
| MERR_INVALID_PARAM | 参数输入非法 |
| MERR_NO_MEMORY | 内存不足 |

## 3.2. AFD_FSDK_StillImageFaceDetection

**原型**

```
MRESULT AFD_FSDK_StillImageFaceDetection(
     MHandle                  hEngine,
     LPASVLOFFSCREEN          pImgData,
     LPAFD_FSDK_FACERES      *pFaceRes
);
```

**描述**

根据输入的图像检测出人脸位置，一般用于静态图像检测

**参数**

| | | |
|---|---|---|
| hEngine | [in] | 引擎 handle |
| pImgData | [in] | 待检测图像信息 |
| pFaceRes | [out] | 人脸检测结果 |

**返回值**

成功返回 MOK，否则返回失败 code。

## 3.3. AFD_FSDK_UninitialFaceEngine

**原型**

```
MRESULT AFD_FSDK_UninitialFaceEngine(
       MHandle       hEngine
);
```

**描述**

销毁引擎，释放相应资源

**参数**

| | | |
|---|---|---|
| hEngine | [in] | 引擎 handle |

**返回值**

成功返回 MOK，否则返回失败 code。失败 codes 如下所列:

| | |
|---|---|
| MERR_INVALID_PARAM | 参数输入非法 |

## 3.4. AFD_FSDK_GetVersion

**原型**

```
const AFD_FSDK_Version * AFD_FSDK_GetVersion(
        MHandle         hEngine
);
```

**描述**

获取 SDK 版本信息

**参数**

| | | |
|---|---|---|
| hEngine | [in] | 引擎 handle |

# 4. 示例代码

注意,使用时请替换申请的 APPID 和 SDKKEY，并设置好文件路径和图像尺寸

```
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <errno.h>
#include <assert.h>


#include "arcsoft_fsdk_face_detection.h"
#include "merror.h"

//#define APPID     "your appid"
//#define SDKKEY    "your sdkkey"

#define INPUT_IMAGE_FORMAT  ASVL_PAF_I420
#define INPUT_IMAGE_PATH    "your_input_image.yuv"
#define INPUT_IMAGE_WIDTH   (640)
#define INPUT_IMAGE_HEIGHT  (480)

#define WORKBUF_SIZE        (40*1024*1024)
#define MAX_FACE_NUM        (50)

int fu_ReadFile(const char* path, uint8_t **raw_data, size_t* pSize) {
    int res = 0;
    FILE *fp = 0;
    uint8_t *data_file = 0;
    size_t size = 0;

    fp = fopen(path, "rb");
    if (fp == nullptr) {
        res = -1;
        goto exit;
    }

    fseek(fp, 0, SEEK_END);
    size = ftell(fp);
    fseek(fp, 0, SEEK_SET);

    data_file = (uint8_t *)malloc(sizeof(uint8_t)* size);
    if (data_file == nullptr) {
        res = -2;
        goto exit;
    }

    if (size != fread(data_file, sizeof(uint8_t), size, fp)) {
        res = -3;
        goto exit;
    }

    *raw_data = data_file;
    data_file = nullptr;
exit:
    if (fp != nullptr) {
        fclose(fp);
    }

    if (data_file != nullptr) {
        free(data_file);
    }

    if (nullptr != pSize) {
        *pSize = size;
    }

    return res;
}


int main(int argc, char* argv[]) {

    MByte *pWorkMem = (MByte *)malloc(WORKBUF_SIZE);
    if(pWorkMem == nullptr){
        fprintf(stderr, "fail to malloc workbuf\r\n");
        exit(0);
    }

    MHandle hEngine = nullptr;

    int ret = AFD_FSDK_InitialFaceEngine(APPID, SDKKEY, pWorkMem, WORKBUF_SIZE,
                                         &hEngine, AFD_FSDK_OPF_0_HIGHER_EXT,
16, MAX_FACE_NUM);
    if (ret != 0) {
        fprintf(stderr, "fail to AFD_FSDK_InitialFaceEngine(): 0x%x\r\n", ret);
        free(pWorkMem);
        exit(0);
    }

    const AFD_FSDK_Version*pVersionInfo = AFD_FSDK_GetVersion(hEngine);
    printf("%d %d %d %d\r\n", pVersionInfo->lCodebase, pVersionInfo->lMajor,
                                 pVersionInfo->lMinor, pVersionInfo->lBuild);
    printf("%s\r\n", pVersionInfo->Version);
    printf("%s\r\n", pVersionInfo->BuildDate);
    printf("%s\r\n", pVersionInfo->CopyRight);

    ASVLOFFSCREEN inputImg = { 0 };
    inputImg.u32PixelArrayFormat = INPUT_IMAGE_FORMAT;
    inputImg.i32Width = INPUT_IMAGE_WIDTH;
    inputImg.i32Height = INPUT_IMAGE_HEIGHT;
    inputImg.ppu8Plane[0] = nullptr;
    fu_ReadFile(INPUT_IMAGE_PATH, (uint8_t**)&inputImg.ppu8Plane[0], nullptr);
     if (!inputImg.ppu8Plane[0]) {
        fprintf(stderr, "fail to fu_ReadFile(%s): %s\r\n", INPUT_IMAGE_PATH, 
strerror(errno));
        AFD_FSDK_UninitialFaceEngine(hEngine);
        free(pWorkMem);
        exit(0);
    }

    if (ASVL_PAF_I420 == inputImg.u32PixelArrayFormat) {
        inputImg.pi32Pitch[0] = inputImg.i32Width;
        inputImg.pi32Pitch[1] = inputImg.i32Width/2;
        inputImg.pi32Pitch[2] = inputImg.i32Width/2;
        inputImg.ppu8Plane[1] = inputImg.ppu8Plane[0] + inputImg.pi32Pitch[0] 
* inputImg.i32Height;
        inputImg.ppu8Plane[2] = inputImg.ppu8Plane[1] + inputImg.pi32Pitch[1] 
* inputImg.i32Height/2;
    } else if (ASVL_PAF_NV12 == inputImg.u32PixelArrayFormat) {
        inputImg.pi32Pitch[0] = inputImg.i32Width;
        inputImg.pi32Pitch[1] = inputImg.i32Width;
        inputImg.ppu8Plane[1] = inputImg.ppu8Plane[0] + (inputImg.pi32Pitch[0] 
* inputImg.i32Height);
    } else if (ASVL_PAF_NV21 == inputImg.u32PixelArrayFormat) {
        inputImg.pi32Pitch[0] = inputImg.i32Width;
        inputImg.pi32Pitch[1] = inputImg.i32Width;
        inputImg.ppu8Plane[1] = inputImg.ppu8Plane[0] + (inputImg.pi32Pitch[0] 
* inputImg.i32Height);
    } else if (ASVL_PAF_YUYV == inputImg.u32PixelArrayFormat) {
        inputImg.pi32Pitch[0] = inputImg.i32Width*2;
    } else if (ASVL_PAF_I422H == inputImg.u32PixelArrayFormat) {
        inputImg.pi32Pitch[0] = inputImg.i32Width;
        inputImg.pi32Pitch[1] = inputImg.i32Width / 2;
        inputImg.pi32Pitch[2] = inputImg.i32Width / 2;
        inputImg.ppu8Plane[1] = inputImg.ppu8Plane[0] + inputImg.pi32Pitch[0] 
* inputImg.i32Height;
        inputImg.ppu8Plane[2] = inputImg.ppu8Plane[1] + inputImg.pi32Pitch[1] 
* inputImg.i32Height;
    } else if (ASVL_PAF_LPI422H == inputImg.u32PixelArrayFormat) {
        inputImg.pi32Pitch[0] = inputImg.i32Width;
        inputImg.pi32Pitch[1] = inputImg.i32Width;
        inputImg.ppu8Plane[1] = inputImg.ppu8Plane[0] + (inputImg.pi32Pitch[0] 
* inputImg.i32Height);
    } else if (ASVL_PAF_RGB24_B8G8R8 == inputImg.u32PixelArrayFormat) {
        inputImg.pi32Pitch[0] = inputImg.i32Width*3;
    } else if (ASVL_PAF_RGB24_R8G8B8 == inputImg.u32PixelArrayFormat) {
        inputImg.pi32Pitch[0] = inputImg.i32Width * 3;
    } else if (ASVL_PAF_RGB32_B8G8R8A8 == inputImg.u32PixelArrayFormat) {
        inputImg.pi32Pitch[0] = inputImg.i32Width * 4;
    } else {
        fprintf(stderr, "unsupported Image format: 
0x%x\r\n",inputImg.u32PixelArrayFormat);
        free(inputImg.ppu8Plane[0]);
         AFD_FSDK_UninitialFaceEngine(hEngine);
        free(pWorkMem);
        exit(0);
    }
    LPAFD_FSDK_FACERES faceResult;
    ret = AFD_FSDK_StillImageFaceDetection(hEngine, &inputImg, &faceResult);
    if (ret != 0) {
        fprintf(stderr, "fail to AFD_FSDK_StillImageFaceDetection(): 0x%x\r\n", 
ret);
        free(inputImg.ppu8Plane[0]);
        AFD_FSDK_UninitialFaceEngine(hEngine);
        free(pWorkMem);
        exit(0);
    }


    for (int i = 0; i < faceResult->nFace; i++) {
        printf("face %d:(%d,%d,%d,%d)\r\n", i,
               faceResult->rcFace[i].left, faceResult->rcFace[i].top,
               faceResult->rcFace[i].right, faceResult->rcFace[i].bottom);

    }


    free(inputImg.ppu8Plane[0]);
    AFD_FSDK_UninitialFaceEngine(hEngine);
    free(pWorkMem);

    return 0;
}
```