

Csp 病理统一格式 SDK 使用说明书

文档版本

发布日期

2024-12-13



目 录

1 概述.....1

2 SDK 软件说明.....2

2.1 软件信息 2

2.2 SDK 压缩包说明 2

2.3 注意事项 3

3 SDK 接口使用说明.....4

3.1 扫描接口 4

3.1.1 扫描接口调用流程 4

3.1.2 扫描接口调用代码示例（C 语言） 5

3.2 阅片接口 6

3.2.1 阅片接口调用流程 6

3.2.2 阅片接口调用代码示例（C 语言） 6

3.3 约束 8

3.3.1 参数约束 8

3.3.2 使用约束 9

4 问题 FAQ.....10

4.1 常见问题及解决方案 10

1 概述

本文档为 Csp 病理统一格式 SDK 使用说明文档，包含 SDK 压缩包说明，库运行环境以及资源消耗，接口使用以及约束，问题 FAQ 等

2 SDK 软件说明

2.1 软件信息

名称	Csp 病理统一格式 SDK
版本	1.0.0
用途	集成该 SDK 软件，生成和调阅 Csp 病理统一格式

2.2 SDK 压缩包说明

系统版本	软件包	名称	描述
Windows 版本	CspSDK_sdk_windows_AMD64_1.0.0.release.zip	sdk/conf/version.txt	CspSDK 版本说明
		sdk/include/csp_api.h	SDK 提供的 API
		sdk/lib/libcsp_sdk.dll	libcsp_sdk.dll 为需要集成的 sdk 动态库，其余 dll 为运行所需要的动态链接库，版本为 gcc7.3。
		sdk/lib/libgcc_s_seh-1.dll	
		sdk/lib/libstdc++-6.dll	
		sdk/lib/libwinpthread-1.dll	
	CspSDK_sdk_windows_AMD64_1.0.0.release.zip.cms		CspSDK 签名文件
	CspSDK_sdk_windows_AMD64_1.0.0.release.zip.crl		
	Csp-SDK-		SDK 说明文档

系统版本	软件包	名称	描述
	description.pdf		
Linux 版本	CspSDK_sdk_x86_64_1.0.0.release.tar.gz	conf/version.txt	CspSDK 版本说明
		include/csp_api.h	SDK 提供的 API
		lib/ libcsp_sdk.so	libcsp_sdk.so 为需要集成的 sdk 动态库
	CspSDK_sdk_x86_64_1.0.0.release.tar.gz.cms		CspSDK 签名文件
	CspSDK_sdk_x86_64_1.0.0.release.tar.gz.crl		
	Csp-SDK-description.pdf		SDK 说明文档

2.3 注意事项

SDK 运行环境需求

SDK 库名	系统	建议运行环境
libcsp_sdk.so	建议 Linux Ubuntu20.04.2 及以上	gcc7.3.0 及以上，glibc 2.35 及以上
libcsp_sdk.dll	建议 Windows10 及以上	SDK 中包含的动态链接库或其他更高版本

资源消耗情况

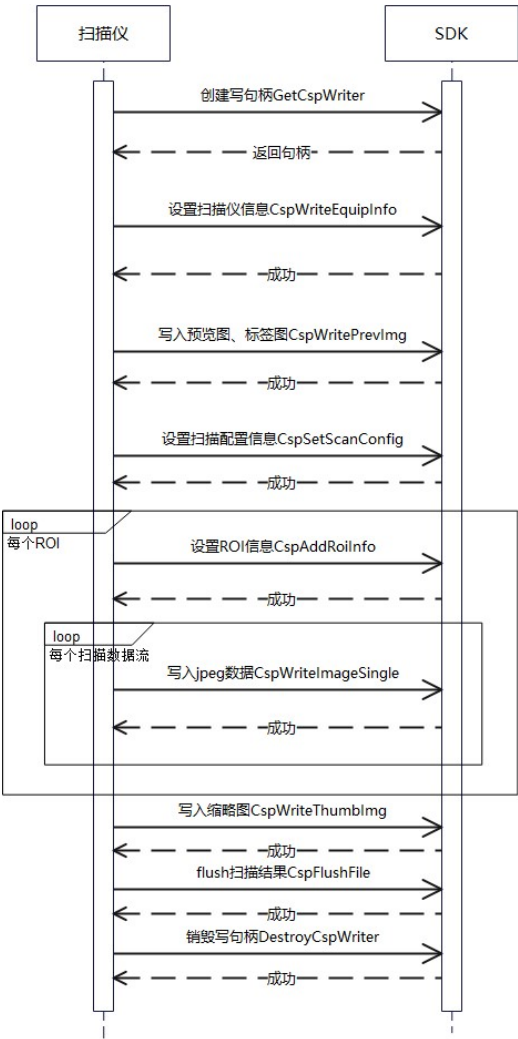
SDK 库名	运行资源消耗
libcsp_sdk.so	建议至少使用 4 核 CPU（支持多线程）
libcsp_sdk.dll	以 i7-12700，2.10 GHz 的 CPU 为例，建议至少需要预留 40%的 CPU 资源、200MB 运行内存来加载运行 sdk

3 SDK 接口使用说明

3.1 扫描接口

3.1.1 扫描接口调用流程

接口说明见 csp_api.h 头文件，下图为扫描接口调用流程图：



3.1.2 扫描接口调用代码示例（C 语言）

```

{
    const char* fname = "test.csp";
    void* fp = GetCspWriter(fname, 4); //创建写 csp 文件句柄, 4 代表并发数
    assert(fp != NULL);
    CspScannerInfo scannerInfo;
    // fill scannerInfo
    int32_t ret = CspWriteEquipInfo(fp, &scannerInfo); // 写入扫描仪信息
    assert(ret == 0);
    void* data = malloc(200); // 用户可根据需要修改申请内存大小
    // fill preview image data
    ret = CspWritePrevImg(fp, 100, 100, 200, data); // 写入预览图, 100、100、200 分别表示预览
    图的宽、高、图像字节长度
    assert(ret == 0);
    // fill label image data
    ret = CspWriteLabelImg(fp, 100, 100, 200, data); // 写入标签图, 100、100、200 分别表示标签
    图的宽、高、图像字节长度
    assert(ret == 0);
    // fill thumbnail image data
    ret = CspWriteThumbImg(fp, 100, 100, 200, data); // 写入缩略图, 100、100、200 分别表示缩略图
    的宽、高、图像字节长度
    assert(ret == 0);
    CspConfig cfg; // cfg 实际使用时, 应该作为 json 传入
    cfg.tileWidth = 256; // 切块基本宽度
    cfg.tileHeight = 256; // 切块基本高度
    cfg.imageWidth = 10000; // 图像宽度
    cfg.imageHeight = 20000; // 图像高度
    cfg.scanRatio = 40.0; // 当前切块对应的扫描倍率
    cfg.downsamplingRatio = 2.0; // 下采样倍率
    ret = CspSetScanConfig(fp, &cfg); // 设置扫描配置信息
    assert(ret == 0);

    // CspAddRoiInfo 添加 ROI 信息, 0、0 分别表示 ROI 的起始坐标 X、Y
    ret = CspAddRoiInfo(fp, 0, 0, cfg.imageWidth, cfg.imageHeight);
    assert(ret == 0);

    // 计算图像的每行、每列瓦片的个数
    uint32_t rowNum = (cfg.imageHeight + cfg.tileHeight - 1) / cfg.tileHeight;
    uint32_t colNum = (cfg.imageWidth + cfg.tileWidth - 1) / cfg.tileWidth;
    // use 'CspWriteImageSingle' to write tiles
    for (uint32_t r = 0; r < rowNum; r++) {
        for (uint32_t c = 0; c < colNum; c++) {
            CspImageInfo tile;
            tile.x = c * cfg.tileWidth;
            tile.y = r * cfg.tileHeight;
            tile.width = min(cfg.tileWidth, cfg.imageWidth - tile.x);
            tile.height = min(cfg.tileHeight, cfg.imageHeight - tile.y);
            // fill tile.data and tile.dataLen;
            ret = CspWriteImageSingle(fp, &tile); // 写入 tile 图像块, 也就是瓦片
            assert(ret == 0);
            // free tile.data
        }
    }
    ret = CspFlushFile(fp); // 结束写入图片
}

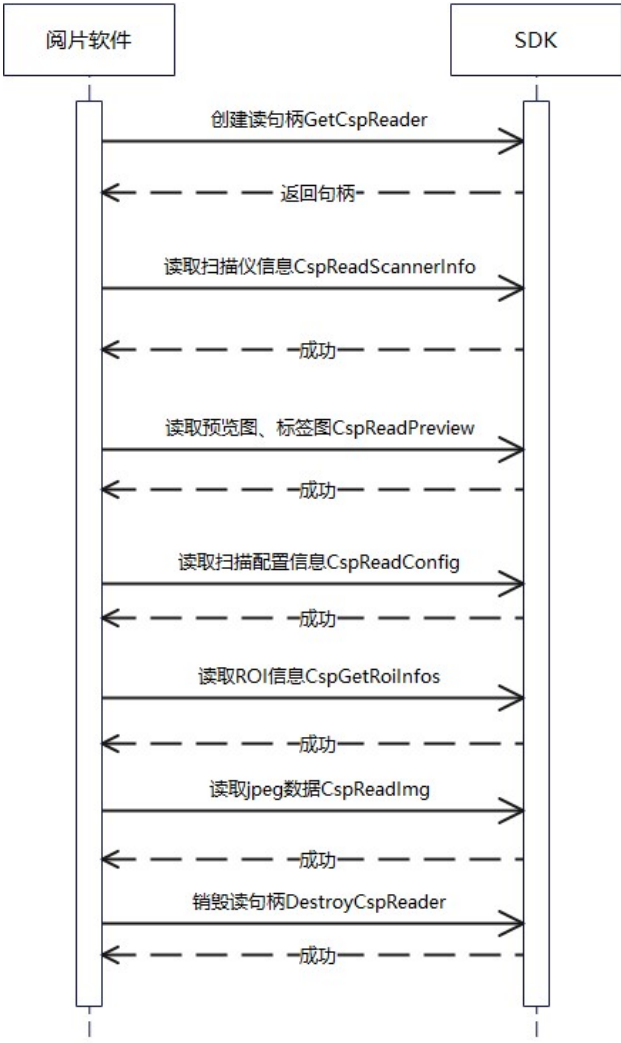
```

```
    assert(ret == 0);
    DestroyCspWriter(fp);    // 销毁写文件句柄
}
```

3.2 阅片接口

3.2.1 阅片接口调用流程

接口说明见 csp_api.h 头文件，下图为阅片接口调用流程图：



3.2.2 阅片接口调用代码示例（C 语言）

```
{
    const char* fname = "test.csp";
    void* fp = GetCspReader(fname);    // 创建 csp 读文件句柄
    assert(fp != NULL);

    // 读取预览图
    CspImageInfo imageInfo;
```

```

int32_t ret = CspReadPreview(fp, &imageInfo);
assert(ret == 0);
/* process with imageInfo
...
*/
CspDestroyImage(&imageInfo);          // 销毁预览图文件块

// 读取标签图
ret = CspReadLabel(fp, &imageInfo);
assert(ret == 0);
/* process with imageInfo
...
*/

// 读取缩略图
CspDestroyImage(&imageInfo);          // 销毁预览图标签块
ret = CspReadThumb(fp, &imageInfo);
assert(ret == 0);
/* process with imageInfo
...
*/
CspDestroyImage(&imageInfo);          // 销毁缩略图标签块

// 读取扫描仪信息
CspScannerInfo scanInfo;
ret = CspReadScannerInfo(fp, &scanInfo);
assert(ret == 0);

// 读取扫描配置信息
CspConfig cfg;
ret = CspReadConfig(fp, &cfg);
assert(ret == 0);

// 获取扫描层数
uint32_t layerNum;
ret = CspGetLayerNum(fp, &layerNum);
assert(ret == 0);
float scale = cfg.scanRatio;
uint32_t imageWidth = cfg.imageWidth;
uint32_t imageHeight = cfg.imageHeight;
for (uint32_t i = 0; i < layerNum; i++) {
    uint32_t rowNum = (imageHeight + cfg.tileHeight - 1) / cfg.tileHeight;    // 计算每一层行和列的瓦片个数
    uint32_t colNum = (imageWidth + cfg.tileWidth - 1) / cfg.tileWidth;
    for (uint32_t r = 0; r < rowNum; r++) {
        for (uint32_t c = 0; c < colNum; c++) {
            CspImageInfo tile;
            tile.x = c * cfg.tileWidth;
            tile.y = r * cfg.tileHeight;
            tile.width = min(cfg.tileWidth, imageWidth - tile.x);
            tile.height = min(cfg.tileHeight, imageHeight - tile.y);

            // 读取图像信息
            ret = CspReadImg(fp, scale, &tile);
            assert(ret == 0);
            /* process with imageInfo
            ...

```

```
        */
        CspDestroyImage(&tile);    // 销毁图像块
    }
}
scale /= cfg.downsamplingRatio;    // 循环, 读取下一层图像
imageWidth /= cfg.downsamplingRatio;
imageHeight /= cfg.downsamplingRatio;
}
DestroyCspReader(fp);    // 销毁读 csp 文件句柄
}
```

3.3 约束

3.3.1 参数约束

1、在 csp_api.h 定义的 CspConfig 结构体中

```
typedef struct {
    uint32_t tileWidth;    // 切块基本宽度, 典型值 256, 扫描时的配置项, 最大不超过 65536
    uint32_t tileHeight;    // 切块基本高度, 典型值 256, 最大不超过 65536
    uint32_t imageWidth;    // 扫描图像宽度, 最大不超过 (128 * 1024 * 1024)
    uint32_t imageHeight;    // 扫描图像高度, 最大不超过 (128 * 1024 * 1024)
    float scanRatio;    // 扫描倍率, 如 40.0, 20.0 等
    float downsamplingRatio;    // 下采样倍率, 需要是浮点数, 只支持 2: 1、4:1
    float mpp;    // 每个像素的微米数 (microns per pixel)
} CspConfig;
```

瓦片宽高不能大于生成的图片宽高, 即必须满足 (tileWidth <= imageWidth && tileHeight <= imageHeight)

建议扫描时的瓦片个数大于 1。如果只设置 1 个瓦片, 并且 tileWidth == imageWidth、tileHeight == imageHeight 时, 会返回错误。

2、ROI 结构体定义如下

```
typedef struct {
    uint32_t x;
    uint32_t y;
    uint32_t width;
    uint32_t height;
} CspRoiInfo;
```

其中, x、y、width、height 是相对于用户定义的最大倍率的图片的坐标和宽高, 可以定义多个 ROI, ROI 之间不能相交, 两个 ROI 不能公用一个顶点坐标或者一条边, 输入的切块 (tile) 图片必须都包含在 ROI 中, ROI 可以包含切块图片的一部分, 不能输入没有用到的切块, 否则接口会报错。

3.3.2 使用约束

- 1、切片的宽和高推荐使用 256
- 2、每张输入 tile 瓦片的大小应该小于 2MB。

4 问题 FAQ

4.1 常见问题及解决方案

问题描述	切片的标签图、缩略图无法显示 ¹
原因分析	扫描仪生成标签图、缩略图时，未使用 JPEG 格式，导致前段无法识别数据
解决方案	扫描仪修改，将标签图、缩略图输出格式改为 JPEG

问题描述	调阅 Csp 切片时加载慢 ¹
原因分析	SDK 内部只生成了 7 层图像，对于大分辨率切片，最小层的分辨率超过 256*256，导致加载时需要实时采样计算得到小于 256*256 的层，耗时高
解决方案	调整最大分辨率，SDK 已支持计算 10 层图像

问题描述	无法调阅 Csp 格式数据 ¹
原因分析	扫描仪未升级到最新版 SDK，导致生成的切片最小层过大，无法加载显示
解决方案	SDK 已支持计算 10 层图像，规避切片最小层过大的问题

问题描述	调阅Csp切片时，出现core dump导致进程异常退出
原因分析	调用 Csp 接口时，漏掉了 CspFlushFile，表示结束写入数据
解决方案	开发前参考 Csp 的 API 说明和调用实例，需要调用 CspFlushFile 函数，完成写入数据

问题描述	XX私有格式切片转换为Csp后，从1.2G变为1.5G
------	-----------------------------

原因分析	XX 私有格式切片采用 4:1 进行下采样，转换成 Csp 文件时设置成 2:1 进行下采样，导致数据膨胀
解决方案	扫描仪厂商自己转换与使用 SDK 转换时保持下采样倍率一致

问题描述	SDK的内存占用率过高
原因分析	SDK 内未设置并发数上限，导致大量数据堆积在内存中
解决方案	SDK 做了最大并发数的限制，同时请厂商设置合理的并发数