1 Overview

Supported by the Pathology Branch of the Chinese Medical Association and the Pathology Physicians Branch of the Chinese Medical Doctor Association, the adoption of the digital pathology Slice sharing format has been promoted. The sharing format uses the suffix .CSP, which stands for Chinese Society of Pathology. This document defines specifications for the CSP digital pathology image data format, integrating pathological workflow characteristics to establish a unified format specification from two perspectives: “more comprehensive” and “more efficient. ”

- By leveraging common access patterns of pathological images and correlating metadata with data, image loading speed is improved, thereby optimizing the user Slice viewing experience;
- By optimizing data layout according to the multi-frame data characteristics typical in pathology, file storage requirements are reduced; furthermore, secondary compression algorithms can be applied to further decrease storage space usage;
- Provides layout layer annotations for AI algorithms, with simultaneous loading of images and annotation data, tailored to AI application features, effectively improving AI diagnostic efficiency;
- Enhances data verification mechanisms to ensure data security and maintain data consistency during pathological file transmission and storage.

2 SDK User Instructions

2.1 Software Information

Name	Csp Pathology Unified Format SDK
Version	1.0.1
Purpose	Integrates this SDK software to generate and retrieve Csp Pathology Unified Format files

2.2 SDK Compressed Package Description

System Version	Software Package	Name	Description
Windows Version	CspSDK_sdk_windows_AMD64_1.0.1.release.20251224165646.zip	sdk/conf/version.txt	CspSDK Version Description
		sdk/include/csp_api.h	SDK ProvidedAPIs
		sdk/lib/libcsp_sdk.dll	libcsp_sdk.dll is the SDK dynamic library to be integrated; other dll files are runtime dynamic link libraries, version gcc7.3.
		sdk/lib/libgcc_s_seh-1.dll	
		sdk/lib/libstdc++-6.dll	
		sdk/lib/ libwinpthread-1.dll	
	CspSDK_sdk_windows_AMD64_1.0.1.release.20251224165646.zip.cms		CspSDK Signature File
	CspSDK_sdk_windows_AMD64_1.0.1.release.20251224165646.zip.crl		
	Csp-SDK-description.pdf		SDK Documentation
Linux Version	CspSDK_sdk_x86_64_1.0.1.release.20251224165646.tar.gz	conf/version.txt	CspSDK Version Description
		include/csp_api.h	SDK ProvidedAPIs
		lib/ libcsp_sdk.so	libcsp_sdk.so SDK Dynamic Library for Integration
	CspSDK_sdk_x86_64_1.0.1.release.20251224165646.tar.gz.cms		CspSDK Signature File
	CspSDK_sdk_x86_64_1.0.1.release.20251224165646.crl		
	Csp-SDK-description.pdf		SDK Documentation
ARM Version	CspSDK_sdk_arch64_1.0.1.release.20251224	conf/version.txt	CspSDK Version Description

System Version	Software Package	Name	Description
	165646.tar.gz		
		include/csp_api.h	SDK ProvidedAPIs
		lib/ libcsp_sdk.so	libcsp_sdk.so SDK Dynamic Library for Integration
	CspSDK_sdk_aarch64_1.0.1.release.20251224165646.tar.gz.cms		CspSDK Signature File
	CspSDK_sdk_aarch64_1.0.1.release.20251224165646.20251224165646.crl		
	Csp-SDK-description.pdf		SDK Documentation

2.3 Runtime Environment Description

SDK Runtime Environment Requirements

SDK Library Name	System	Recommended Runtime Environment
libcsp_sdk.so	Recommended Linux Ubuntu 20.04.2 or Later	gcc7.3.0 or Later, glibc 2.35or Later
libcsp_sdk.dll	Recommended Windows 10 or Later	Dynamic Link Libraries or Higher Versions Included in SDK

Resource Consumption Status

SDK Library Name	Runtime Resource Consumption
libcsp_sdk.so	Recommended Minimum Use of 4Core CPU(Supporting Multithreading)
libcsp_sdk.dll	Using the CPU i7-12700 at 2.10 GHz as an example, it is recommended to reserve at least 40% of CPU resources and 200 MB of RAM to load and run the SDK.

2.4 SDK Scanner Parameter Configuration Description

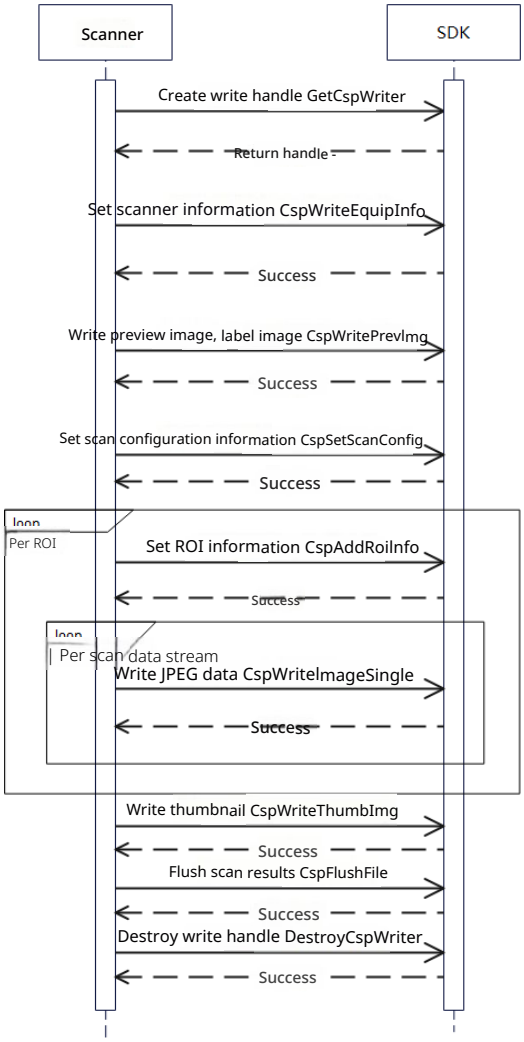
SDK Parameters	Specification
Tile Size	256*256
Compression Method	JPEG
Compression Quality	70
Scaling Ratio (MPP)	Recommended 0.5, adjustable
Scan Magnification	Recommended 20, adjustable

3 SDK Interface Usage Description

3.1 Scanner Interface

3.1.1 Scanner Interface Invocation Process

The interface description is provided in the csp_api.h header file. The following diagram illustrates the scanner interface invocation flow:



3.1.2 Scanner Interface Call Code Example (C language)

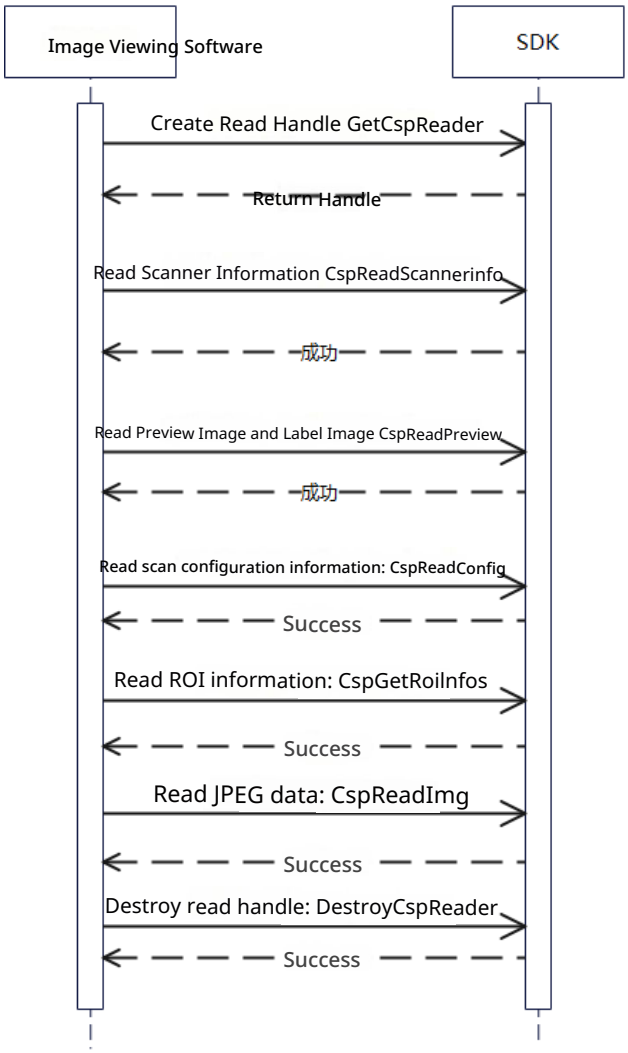
```
{
        const  char*  fname  =  "test.csp";
        void*  fp  =  GetCspWriter(fname,  4);        // Create a handle for writing CSP files; 4 denotes the concurrency level
        assert(fp  !=  NULL);
        CspScannerInfo  scannerInfo;
        //  fill  scannerInfo
        int32_t  ret  =  CspWriteEquipInfo(fp,  &scannerInfo);          // Write Scanner information
        assert(ret  ==  0);
        void*  data  =  malloc(200);          // Users may adjust the allocated memory size as needed
        //  fill  preview  image  data
        ret = CspWritePrevImg(fp,  100,  100,  200, data);  // Write Preview Image; 100, 100, and 200 denote the preview image's width, height, and byte length, respectively
        assert(ret  ==  0);
        //  fill  label  image  data
        ret = CspWriteLabelImg(fp,  100,  100,  200, data);  // Writing the Label Map; 100, 100, and 200 represent the Label Map's width, height, and image byte length, respectively.
        assert(ret  ==  0);
        //  fill  thumbnail  image  data
```

```
        ret = CspWriteThumbImg(fp, 100, 100, 200, data); // Write thumbnail; 100, 100, 200 represent the thumbnail's width, height, and image byte length, respectively
        assert(ret == 0);
        CspConfig cfg;                          // cfg should be passed as JSON in practical use
        cfg.tileWidth = 256;                    // Basic tile width
        cfg.tileHeight = 256;                   // Basic tile height
        cfg.imageWidth = 10000;                 // Image width
        cfg.imageHeight = 20000;                // Image width
        cfg.scanRatio = 40.0;                   // Current tile's corresponding scanning magnification
        cfg.downsamplingRatio = 2.0;            // Downsampling Factor
        ret = CspSetScanConfig(fp, &cfg);       // Set scanning configuration information
        assert(ret == 0);

         // CspAddRoiInfo adds ROI information; 0, 0 represent the ROI start coordinates X and Y, respectively
        ret = CspAddRoiInfo(fp, 0, 0, cfg.imageWidth, cfg.imageHeight);
        assert(ret == 0);

        // Calculate the number of tiles per image row and column
        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);     // Write tile image block
                assert(ret == 0);
                // free tile.data
            }
        }
        ret = CspFlushFile(fp); // Complete image writing
        assert(ret == 0);
        DestroyCspWriter(fp);       // Destroy write file handle
}
```

3.2 Image Viewing Interface

3.2.1 Image Viewing Interface Invocation Process

Interface specification is provided in the csp_api.hheader file; the diagram below illustrates the image viewing interface invocation workflow:



3.2.2 Image Viewing Interface Invocation Code Example (CLanguage)

```
{
        const  char*  fname  =  "test.csp";
        void*  fp  =  GetCspReader(fname);     // Create CSP read file handle
        assert(fp  !=  NULL);

        // Read preview image
        CspImageInfo  imageInfo;
        int32_t  ret  =  CspReadPreview(fp,  &imageInfo);
        assert(ret  ==  0);
        /*  process  with  imageInfo
          ...
        */
        CspDestroyImage(&imageInfo);          // Destroy preview image file block

        // Read label map
        ret  =  CspReadLabel(fp,  &imageInfo);
        assert(ret  ==  0);
        /*  process  with  imageInfo
          ...
        */
```

```
        // Read thumbnail
        CspDestroyImage(&imageInfo);          // Destroy preview image label block
        ret = CspReadThumb(fp, &imageInfo);
        assert(ret == 0);
        /* process with imageInfo
          ...
        */
        CspDestroyImage(&imageInfo);          // Destroy thumbnail label block

        // Read scanner information
        CspScannerInfo scanInfo;
        ret = CspReadScannerInfo(fp, &scanInfo);
        assert(ret == 0);

        // Read scanning configuration information
        CspConfig cfg;
        ret = CspReadConfig(fp, &cfg);
        assert(ret == 0);

        // Obtain scanning layers
        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; // Calculate the number of tiles per row and column for each layer
            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);

                    // Read image information
                    ret = CspReadImg(fp, scale, &tile);
                    assert(ret == 0);
                    /* process with imageInfo
                    ...
                    */
                    CspDestroyImage(&tile);     // Destroy image tile
                }
            }
            scale /= cfg.downsamplingRatio;    // Loop to read the next image pyramid level
            imageWidth /= cfg.downsamplingRatio;
            imageHeight /= cfg.downsamplingRatio;
        }
        DestroyCspReader(fp);     // Destroy the CSP file reader handle
}
```

3.3 Constraints

3.3.1 Parameter Constraints

1 Within the CspConfig structure defined in csp_api.h

```
typedef  struct  {
        uint32_t  tileWidth;          // Basic block width; typical value: 256; scanner configuration parameter; maximum value not to exceed 65536
        uint32_t  tileHeight;         // Basic block height, typical value 256, maximum not exceeding 65536
        uint32_t  imageWidth;         // Scanned image width, maximum does not exceed (128 * 1024 * 1024)
        uint32_t  imageHeight;        // Scanned image height, maximum not to exceed (128 * 1024 * 1024)
        float  scanRatio;             // Scanning magnification, for example, 40.0, 20.0, etc.
        float  downsa  mplingRatio;   // Downsampling factor, must be a floating-point number, supporting only 2:1 and 4:1
        float  mpp;                   // Microns per pixel
}  CspConfig;
```

Tile width and height must not exceed the generated image width and height; specifically, the condition (tileWidth <= imageWidth && tileHeight <= image-Height) must be met.

It is recommended that the number of tiles during scanning be greater than 1 . If only 1 tile is configured and tileWidth == imageWidth and tileHeight == imageHeight , an error will be returned.

2、 The ROI structure is defined as follows

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

Among them, x , y , width , and height represent the coordinates and dimensions relative to the image at the user-defined maximum magnification. Multiple ROIs may be defined; however, ROIs must not intersect, and two ROIs may not share a vertex coordinate or an edge. All input tiles must be fully contained within an ROI . An ROI can include only a portion of a tile image. Tiles that are not used must not be input; otherwise, the interface will report an error.

The tile position relative to the top-left coordinate of the ROI must be an integer multiple of the tile's width and height. For example, tile.x - roi.x must be an integer multiple of the tile width, and tile.y - roi.y must be an integer multiple of the tile height.

3.3.2 Usage Constraints

1. The size of each input tile shall be less than 2MB.

4 IssuesFAQ

4.1 Common Issues and Solutions

Issue Description	The label image and thumbnail of the slide cannot be displayed.
Cause Analysis	When the scanner generates the label image and thumbnail, it does not use the JPEG format, causing the frontend to be unable to recognize the data.
Solution	Modify the scanner to output the label image and thumbnail in JPEG format.

Issue Description	Slow loading when retrieving Csp slides.
Cause Analysis	The SDK internally generates only 7 image levels. For high-resolution slides, the lowest level resolution exceeds 256*256, requiring real-time sampling calculation during loading to produce a level smaller than 256*256, resulting in high latency.
Solution	Adjust the maximum resolution,SDK supports processing of 10 image layers

Issue Description	Unable to retrieve Csp format data
Cause Analysis	The scanner has not been upgraded to the latest version of the SDK,resulting in an excessively large minimum tile layer in the generated slide, which cannot be properly loaded Display
Solution	The SDK now supports processing of 10 image layers to prevent issues caused by excessively large minimum tile layers

Issue Description	A core dump occurs when retrieving Csp slides, causing abnormal process termination
Cause Analysis	When invoking the Csp interface, the CspFlushFile function was omitted, which signifies the completion of data writing
Solution	Before development, refer to the Csp API documentation and usage examples; it is necessary to call the CspFlushFile function to complete data writing

Issue Description	After converting XXprivate format slides to Csp, the file size increased from 1.2G to 1.5G

Cause Analysis	The private format slices adopt 4:1 downsampling; however, when converting to a Csp file, it is set to 2:1 downsampling, causing data inflation.
Solution	Scanner manufacturers should ensure consistent downsampling ratios when converting and utilizing the SDK.

Issue Description	The memory consumption of the SDK is excessively high.
Cause Analysis	The SDK does not enforce a concurrency limit, resulting in significant data accumulation in memory.
Solution	The SDK has a maximum concurrency limit; manufacturers are advised to configure an appropriate concurrency level.

4.2 Error Codes

Error codes returned by the SDK interface are negative; convert them to positive values and then to hexadecimal. The lower 8 bits specify the error cause. For instance, -1610441475 converted to positive is 1610441475, which corresponds to hexadecimal 0x5FFD6303. The lowest 8 bits, 0x03, represent the error cause. The detailed explanation is as follows:

Error Code	Error Description	Solution
0x02	Failed to read CSPfile	Common causes include insufficient user permissions, file deletion during the reading process, or file corruption. Please ensure the file can be accessed and read correctly.
0x03	Internal error	Contact engineer for resolution
0x04	Input parameter error	Verify input parameters
0x05	Memory error	Ensure sufficient software runtime memory and configure an appropriate concurrency level.