SSZN 深视智能

SR 系列 3D 相机 通信库

——参考手册

V3.3.2.17

2022.07

www.cnsszn.com

软件使用许可协议

使用本软件前，客户需同意下述软件使用许可协议 （以下简称“本协议”）的内容。
客户使用或复制 SR 系列 3D 相机通信库（以下简称“本软件”）的部分或全部功能时，将视为客户已同意本协议中规定的所有内容，且本协议成立。

第 1 条 （使用权的授予）

1. 在客户遵守本协议内容的前提下，深圳市深视智能科技有限公司（以下简称“本公司”）将对客户授予本软件的非独占性使用权。

第 2 条 （禁止事项）

客户不可对本软件实施以下操作或改动。

a. 更改本软件现有功能或向本软件添加新功能。
b. 对本软件实施的所有逆向工程行为，如：反编译、反汇编等。
c. 向第三方二次销售、转让、二次分配、授予、租赁本软件及本公司提供的本软件授权密钥等。但允许客户将本软件与使用本软件编写的应用程序共同进行二次分配。

第 3 条 （著作权等）

与本软件及本软件手册相关的所有知识产权 （如著作权），归本公司所有。

第 4 条 （免责）

客户或第三方因使用本软件而遭受的所有损害，本公司概不负责！

第 5 条 （支持）

本公司将基于本协议，根据客户针对本软件提出的问题，提供技术支持。但并不保证本公司提供的技术支持服务可使客户达成期望目的。

第 6 条 （协议终止）

1. 当客户进行废弃本软件及其复制品等以停止使用本软件时，本协议自动终止。
2. 当客户违反本协议中规定的任一条款时，本公司可单方面解除本协议。同时，客户应立即废弃本软件及其复制品，或将之返还至本公司。
3. 因客户违反本协议，而使本公司蒙受损失时，客户应向本公司赔偿相关损失。

第 7 条 （准据法）

本协议遵从中华人民共和国法律。

前言

使用前请务必阅读本用户手册。
阅读后，请妥善保管，以便日后查阅。

SR 系列 3D 相机通信库，为客户提供通过应用程序控制 SR 系列 3D 相机的通信接口。具体使用方法，请向本公司索取示例程序。

本手册的内容是本着准确无误的目标进行编制的。但是如果发现有不清楚、错误或含糊的内容，请联系本公司销售部们。如有缺页或装订错误，本公司予以更换。

联系我们

深圳市深视智能科技有限公司
电　　话：0755-29655424
传　　真：0755-27369408
电子邮件：support@cnsszn.com
网　　址：http://www.cnsszn.com
地　　址：深圳市南山区留仙大道 3370 号南山智园崇文园区 2 号楼第 5 层。
邮　　编：518071

感谢选用深视智能的机器视觉相关产品

为回报客户，我们将以一流的机器视觉产品、完善的售后服务、高效的技术支持，帮助您建立自己的机器视觉系统。

深视智能的更多信息

深视智能的网址是 http://www.cnsszn.com。在我们的网页上可以获得更多关于公司和产品的信息，包括：公司简介、产品介绍、技术支持、产品最新发布等。

您也可以通过电话（0755-29655424）咨询关于公司和产品的更多信息。

技术支持和售后服务

您可以通过以下途径获得我们的技术支持和售后服务：

电子邮件：support@cnsszn.com

电　　话：0755-29655424

地　　址：深圳市南山区留仙大道 3370 号南山智园崇文园区 2 号楼第 5 层

邮　　编：518000

用户手册的用途

用户通过阅读本手册，能够熟悉机器视觉传感产品 SR 系列 3D 相机通信库的基本使用，能够在实际工程项目中进行开发。

用户手册的使用对象

本用户手册适用于，对软件开发调试操作有一定了解的工程人员。

用户手册的主要内容

本手册由 8 节内容组成。详细介绍了 SR 系列 3D 相机通信库的运行环境、文件构成、嵌入方式、变量类型、结构体定义、函数详细说明、开发例程等。

相关文件

关于 SR 系列 3D 相机的硬件安装，请参阅随本产品配套的《SR 系列 3D 相机——硬件用户手册》。关于 SR 系列 3D 相机的软件调试，请参阅随本产品配套的《SR 系列 3D 相机——软件使用手册》。

文档版本

编号	版本号	修订日期
	1.04	2017 年 10 月 10 日
	2.0.1	2017 年 11 月 23 日
	2.2.2	2018 年 03 月 11 日
	2.2.3	2018 年 06 月 01 日
	3.0.1	2018 年 06 月 29 日
	3.3.1	2018 年 12 月 12 日
	3.3.1	2019 年 02 月 25 日
	3.3.2	2019 年 10 月 08 日
	3.3.2.1	2019 年 11 月 06 日
	3.3.2.2	2020 年 04 月 10 日
	3.3.2.3	2020 年 08 月 25 日
	3.3.2.4	2020 年 11 月 25 日
	3.3.2.5	2020 年 12 月 03 日
	3.3.2.6	2021 年 01 月 13 日
	3.3.2.7	2021 年 01 月 28 日
	3.3.2.8	2021 年 01 月 24 日
	3.3.2.9	2021 年 03 月 12 日
	3.3.2.10	2021 年 03 月 29 日
	3.3.2.11	2021 年 05 月 21 日
	3.3.2.12	2021 年 07 月 2 日
	3.3.2.13	2021 年 07 月 9 日
	3.3.2.14	2021 年 08 月 24 日
	3.3.2.15	2021 年 09 月 16 日
	3.3.2.16	2022 年 01 月 18 日
	3.3.2.17	2022 年 07 月 13 日

目录

1 运行环境

操作系统	Windows 10 32/64 位系统（Professional/Enterprise） Windows 8.1 32/64 位系统（Professional/Enterprise） Windows7 32/64 位系统（SP1 或更高版本）（Professional/Ultimate）
CPU	Core i3 2.3 GHz 或更快的处理器
内存容量	8 GB 以上
2 级缓存	2 MB 以上
硬盘空间	10 GB 以上
接口	Ethernet 1000BASE-T/100BASE-TX

* 不保证连接到 LAN 和使用路由器连接时的操作状况。

1.1 运行环境

以下内容是在使用 SR 系列 3D 相机通信库执行应用程序时的基本环境。

1. **Microsoft C Runtime Library**

   这是执行 DLL 所必需的运行环境。

2. **Microsoft .NET Framework**

   这是执行示例程序所必需的运行环境。

1.2 测试函数运行时间所用电脑配置

电脑型号	戴尔 OptiPlex 3050
操作系统	Microsoft Windows 7 旗舰版 (64 位/Service Pack 1)
CPU	(英特尔)Intel(R) Core(TM) i5-7500 CPU @ 3.40GHz(3401 MHz)
主板	戴尔 0Y4H34
内存	8.00 GB (2400 MHz)

2 文件构成

SR7Link.dll	DLL 本体。
SR7Link.lib	SR7Link.dll 的引入库。
SR7Link.h	定义通信接口的头文件。
Source	示例源文件夹。

3 嵌入方法

3.1 C++

1. 链接
   明链接、暗链接均可使用。
   进行暗链接时，请链接 “SR7Link.lib”
2. 包含文件
   请务必将下列头文件放置在源文件中。
   - SR7Link.h

4 变量类型

本文中的变量类型定义如下。

unsigned char	无符号的 8 bit 整数
unsigned short	无符号的 16 bit 整数
unsigned long	无符号的 32 bit 整数
int	有符号的 32 bit 整数
unsigned int	无符号的 32 bit 整数
long long	有符号的 64bit 整数
double	有符号的 64bit 浮点数

5 结构体定义

名称	Ethernet 设定结构
定义	Typedef struct { unsigned char abyIpAddress[4]; } SR7IF_ETHERNET_CONFIG;
描述	Ethernet 通信连接时的通信设定。 **abyIpAddress** 连接控制器的 IP 地址。 地址为 192.168.0.10 时， 设定为 abyIpAddress[0]=192 abyIpAddress[1]=168 abyIpAddress[2]=0 abyIpAddress[3]=10 依此类推
备注	–

名称	批处理回调函数结果信息结构体
定义	typedef struct { int xPoints; int BatchPoints; unsigned int BatchTimes; double xPixth; unsigned int startEncoder; int HeadNumber; int returnStatus; } SR7IF_STR_CALLBACK_INFO;

描述	回调函数传入批处理结果信息。变量说明：
	int xPoints; //x 方向数据数量
	int BatchPoints; //当前批处理扫描行数
	unsigned int BatchTimes; //已进行的批处理次数
	double xPixth; //x 方向点间距
	unsigned int startEncoder; //批处理开始时的编码器值
	int HeadNumber; //相机头数量
	int returnStatus; //0:正常批处理，<0：请参见错误码

备注	–

6 函数返回码定义

返回码	定义	说明
-999	功能/设备不存在	● 未连接设备
-998	该命令不支持	
-997	参数错误	● 传入不支持的参数或空指针
-996	功能未实现	
-995	句柄无效	
-994	内存（溢出/定义）错误	
-993	操作超时	● 一般为批处理超时，设置的批处理等待时间过短
-992	数据缓冲区过小	
-991	数据流错误	
-990	接口已关闭，不可用	
-989	当前版本不支持	
-988	操作被终止	
-987	操作与当前配置冲突	
-986	批处理帧丢失	
-985	无限循环溢出异常	
-984	无限循环读数据忙	
-983	批处理模式冲突	● 开启了循环模式，但调用非循环模式函数；或者未开启循环模式，但调用循环模式函数
-982	传感头不在线	
-100	批处理停止	● 批处理过程中，输入 IO（停止）有输入信号； ● 批处理过程中，调用停止批处理函数；
-1	一般性错误	
0	操作成功	

7 函数

7.1 函数一览

7.1.1 建立/断开与控制器之间的通信路径

函数名	概要
SR7IF_EthernetOpen	通过 Ethernet 建立连接
SR7IF_CommClose	断开连接
SR7IF_SearchOnline	查询在线设备

7.1.2 测量控制

函数名	概要
SR7IF_StartMeasure	开始测量
SR7IF_StopMeasure	停止测量

7.1.3 系统控制

函数名	概要
SR7IF_GetError	获取控制器的系统错误信息
SR7IF_ClearError	解除控制器的系统错误
SR7IF_GetVersion	获取当前使用的通信库的版本号
SR7IF_GetModels	获取当前连接的传感头型号
SR7IF_GetHeaderSerial	获取当前连接传感头序列号
SR7IF_SetOutputPortLevel	设置控制器输出端口电平
SR7IF_GetInputPortLevel	读取控制器输入端口电平
SR7IF_ExportParameters	导出当前系统配置参数
SR7IF_LoadParameters	导入系统配置参数，覆盖当前配置参数
SR7IF_GetLicenseKey	获取产品剩余使用天数
SR7IF_GetCurrentEncoder	读取当前编码器值
SR7IF_GetCameraTemperature	读取传感头温度，单位 0.01 摄氏度
SR7IF_GetCameraBoardTemperature	读取传感头主板温度，单位 1℃

7.1.4 测量结果的获取

- 公共数据获取设置函数

函数名	概要
SR7IF_SwitchProgram	切换相机配置的参数
SR7IF_GetOnlineCameraB	获取 B 相机是否在线
SR7IF_StartMeasure	开始批处理,立即执行批处理程序
SR7IF_StartIOTriggerMeasure	开始批处理,硬件 IO 触发开始批处理
SR7IF_StopMeasure	停止批处理
SR7IF_ProfilePointSetCount	当前批处理设定行数
SR7IF_ReceiveData	阻塞方式获取数据
SR7IF_ProfilePointCount	获取批处理实际获取行数
SR7IF_ProfileDataWidth	获取数据宽度
SR7IF_ProfileData_XPitch	数据 x 方向间距
SR7IF_GetSingleProfile	取当前一条轮廓（批处理下不更新）
SR7IF_SetSetting	参数设定
SR7IF_GetSetting	参数值获取
SR7IF_SetBatchRollProfilePoint	循环模式设定终止行数
SR7IF_SetMultiEncoderInterval	设置多组编码器触发间隔
SR7IF_GetStartIOTriggerCount	获取 IO 触发测量开始信号的统计计数

- 非连续循环获取数据
  - 阻塞方式批处理相关函数

函数名	概要
SR7IF_GetEncoder	获取编码器值
SR7IF_GetProfileData	阻塞方式获取轮廓数据
SR7IF_GetIntensityData	阻塞方式获取亮度数据

  - 非阻塞方式批处理相关函数

函数名	概要
SR7IF_GetEncoderContiune	非阻塞方式获取编码器值
SR7IF_GetProfileContiuneData	非阻塞方式获取轮廓数据
SR7IF_GetIntensityContiuneData	非阻塞获取亮度数据

- 连续循环获取数据

函数名	概要
SR7IF_GetBatchRollData	无终止循环获取数据
SR7IF_GetBatchRollError	无终止循环获取数据异常计算值

- 高速数据通信

函数名	概要
SR7IF_CALLBACK	高度数据通信接口定义
SR7IF_HighSpeedDataEthernetCommunication	高度数据通信接口初始化

- 回调获取数据

函数名	概要
SR7IF_BatchOneTimeCallBack	回调函数定义
SR7IF_SetBatchOneTimeDataHandler	回调函数注册接口
SR7IF_StartMeasureWithCallback	回调方式开始批处理（只需调用 1 次）
SR7IF_GetBatchProfilePoint	回调函数中获取高度数据
SR7IF_GetBatchIntensityPoint	回调函数中获取亮度数据
SR7IF_GetBatchEncoderPoint	回调函数中获取编码器数据
SR7IF_TriggerOneBatch	批处理软件触发开始

7.2 函数参考

7.2.1 建立/断开与控制器之间的通信连接

- Ethernet **通信连接**

格式	int SR7IF_EthernetOpen (unsigned int lDeviceId, SR7IF_ETHERNET_CONFIG* pEthernetConfig);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **pEthernetConfig(in)** Ethernet 通信设定。 有关各参数，请参照 “5 结构体定义 ”。
返回值	<0：失败 0：成功
说明	建立连接，使设备能够与通过 Ethernet 连接的控制器进行通信。
运行用时	约 125~128ms
支持版本	3.3.1

- **断开通信连接**

格式	int SR7IF_CommClose(unsigned int lDeviceId);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。
返回值	<0：失败 0：成功
说明	断开 Ethernet 的连接。
运行用时	约 16~24ms
支持版本	3.3.1

- **搜索在线相机**

格式	SR7IF_ETHERNET_CONFIG *SR7IF_SearchOnline(int *ReadNum, int timeOut);
参数	**ReadNum (out)** 搜索到的设备个数。 **timeOut（in）** 搜索超时时间
返回值	NULL：失败 SR7IF_ETHERNET_CONFIG*：返回已搜索到的设备的 IP 地址指针（结构体说明请参照“5 结构体定义”）
说明	搜索当前网络中的设备，使用时需关闭系统的网络防火墙或者将开发软件加入防火墙的白名单。
运行用时	约 500ms 及以上
支持版本	3.3.2.29

7.2.2 系统控制

- **获取系统错误信息**

<table>
<tr><td>格式</td><td>int SR7IF_GetError(unsigned int lDeviceId, int * pbyErrCnt, int * pwErrCode);</td></tr>
<tr><td>参数</td><td>pbyErrCnt(out)
用于接收系统错误信息数量的缓存。
pwErrCode(out)
用于接收系统错误信息的缓存。根据由新到旧的顺序，存储 *pbyErrCnt 个数量的系统错误信息。</td></tr>
<tr><td>返回值</td><td><0：失败 0：成功</td></tr>
<tr><td>说明</td><td>获取控制器的系统错误信息。
有关所返回的错误代码含义：
<table>
<tr><th>错误码</th><th>说明</th></tr>
<tr><td>0x010206</td><td>传感头只与传感头连接器 B 相连</td></tr>
<tr><td>0x010205</td><td>两个传感头型号不一致</td></tr>
<tr><td>0x010208</td><td>找不到传感头</td></tr>
<tr><td>0x010207</td><td>在上一次启动之际，连接了两个传感头，但是当前只识别到一个传感头</td></tr>
<tr><td>0x010001</td><td>设备打开失败</td></tr>
<tr><td>0x010002</td><td>Sensor 初始化失败</td></tr>
<tr><td>0x010003</td><td>广播失败</td></tr>
<tr><td>0x010004</td><td>系统启动超时</td></tr>
<tr><td>0x010005</td><td>Linux 系统内存错误</td></tr>
<tr><td>0x010110</td><td>A 相机线缆通信异常</td></tr>
<tr><td>0x010111</td><td>B 相机线缆通信异常</td></tr>
<tr><td>0x010201</td><td>相机发生更改,相机需要恢复原厂设置</td></tr>
<tr><td>0x010105</td><td>读相机型号失败</td></tr>
<tr><td>0x010106</td><td>Sensor 板不在线</td></tr>
<tr><td>0x010107</td><td>AUX 读取失败</td></tr>
<tr><td>0x010108</td><td>激光器板不在线</td></tr>
<tr><td>0x010109</td><td>读 SSR ID 失败</td></tr>
<tr><td>0x01010a</td><td>设备名称不存在</td></tr>
<tr><td>0x01010b</td><td>标定初始化失败</td></tr>
<tr><td>0x01010c</td><td>DDR1(FPGA) 测试失败</td></tr>
<tr><td>0x01010d</td><td>DDR2(FPGA) 测试失败</td></tr>
<tr><td>0x01010e</td><td>DDR3(FPGA) 测试失败</td></tr>
<tr><td>0x010202</td><td>相机 AUX 板 EEPROM 读错误</td></tr>
<tr><td>0x010203</td><td>相机 Sensor 板 EEPROM 读错误</td></tr>
<tr><td>0x010204</td><td>相机主板 EEPROM 读错误</td></tr>
<tr><td>0x010209</td><td>相机数据不存在</td></tr>
</table></td></tr>
</table>

	0x01020a	控制器硬件 DDR 错误
运行用时	约 10~33ms	
支持版本	3.3.2.18	

- **解除系统错误**

格式	int SR7IF_ClearError(unsigned int lDeviceId, unsigned short wErrCode);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **wErrCode(in)** 需要解除的错误的错误代码。
返回值	<0：失败 0：成功
说明	解除控制器中发生的系统错误。 在发生的所有系统错误被成功解除后，控制器才能正常开始测量。
运行用时	约 1ms
支持版本	3.3.1

- **读取库的版本信息**

格式	const char *SR7IF_GetVersion();
参数	无
返回值	当前库的版本号
说明	查询当前使用的二次开发动态库的版本号。
运行用时	约 1ms
支持版本	3.3.1

- **读取传感头的型号**

格式	const char *SR7IF_GetModels(unsigned int lDeviceId);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。
返回值	返回传感头的型号
说明	获取当前连接设备的传感头型号。
运行用时	约 1ms
支持版本	3.3.1

- **读取传感头的序列号**

格式	const char *SR7IF_GetHeaderSerial(unsigned int lDeviceId, int Head);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **Head(in)** 指定哪个传感头：0：传感头 A； 1：传感头 B。
返回值	返回传感头的型号
说明	获取当前连接设备指定传感头的序列号。
运行用时	约 1ms
支持版本	3.3.2.9

- **设置控制器输出端口电平**

格式	int SR7IF_SetOutputPortLevel(unsigned int lDeviceId, unsigned int Port, bool Level);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **Port(in)** 指定控制器的输出端口，范围 0~7。 **Level(in)** 指定端口输出电平，0/False：低电平；1/True：高电平。
返回值	<0：失败 0：成功
说明	设置控制器上指定输出端口的电平。 所有端口上电初始化为高电平。
运行用时	约 1ms
支持版本	3.3.1

- **读取控制器输入端口电平**

格式	int SR7IF_GetInputPortLevel (unsigned int lDeviceId, unsigned int Port, bool *Level);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **Port(in)** 指定控制器的输出端口，范围 0~7。 **Level(out)** 返回指定端口的输入电平，0/False：低电平；1/True：高电平。
返回值	<0：失败 0：成功

说明	读取控制器上指定输入端口的电平。
运行用时	约 1ms
支持版本	3.3.1

- **导出系统配置参数**

格式	const char *SR7IF_ExportParameters(unsigned int lDeviceId, unsigned int *size);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **size(out)** 返回参数表的大小。
返回值	返回参数表的字符指针
说明	导出系统当前使用的配置参数。
运行用时	约 10ms
支持版本	3.3.1

- **导入系统配置参数**

格式	int SR7IF_LoadParameters(unsigned int lDeviceId, const char *pSettingdata, unsigned int size);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **pSettingdata (in)** 导入参数表的字符指针。 **size(in)** 导入参数表的字符指针大小。
返回值	<0：失败　0：成功
说明	导入系统配置参数，覆盖当前使用的配置参数。
运行用时	约 10ms
支持版本	3.3.1

- **读取产品的可使用剩余天数**

格式	int SR7IF_GetLicenseKey(unsigned int lDeviceId, unsigned short *RemainDay);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **RemainDay (out)** 返回剩余天数的指针。
返回值	<0：失败 0：成功
说明	返回产品的可使用剩余天数。
运行用时	约 1ms
支持版本	3.3.1

- **读取当前的编码器值**

格式	int SR7IF_GetCurrentEncoder(unsigned int lDeviceId, unsigned int *value);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **value (out)** 返回当前编码器的指针。
返回值	<0：失败 0：成功
说明	读取当前编码器值。 系统的编码器计数初始值为 2147483648，在此基础上加减，总的计数范围为 0～4294967295。 *注：计数值 0－1 = 4294967295；计数值 4294967295 +1 = 0。*
运行用时	约 1ms
支持版本	3.3.2

- **读取传感头温度**

格式	int SR7IF_GetCameraTemperature(unsigned int lDeviceId, unsigned int *tempA, unsigned int *tempB);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **tempA (out)** 返回传感头 A 温度值。 **tempB(out)** 返回传感头 B 温度值，如果传感头 B 不在线则 tempB 不会赋值。
返回值	<0：失败 0：成功
说明	读取当前 A、B 传感头的温度值。 温度值单位为 0.01℃。 *注意：不支持 SR6000 系列。*
运行用时	约 1ms
支持版本	3.3.2.13

- **读取传感头主板温度**

格式	int SR7IF_GetCameraBoardTemperature(unsigned int lDeviceId, unsigned int *tempA, unsigned int *tempB);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **tempA (out)** 返回传感头 A 主板温度值，-1000000:表示读温度不支持。 **tempB(out)** 返回传感头 B 主板温度值，如果传感头 B 不在线则 tempB 不会赋值，-1000000:表示读温度不支持。
返回值	<0：失败 0：成功
说明	读取当前 A、B 传感头主板的温度值。 温度值单位为 1℃。 *注意：不支持 SR6000 系列。*
运行用时	约 1ms
支持版本	3.3.2.61

- **读取 IO 触发测量开始统计计数**

格式	int SR7IF_GetStartIOTriggerCount(unsigned int lDeviceId, const SR7IF_Data DataObj, unsigned int *pTriggerCount);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **DataObj (in)** 预留，设置为 NULL。 **pTriggerCount (out)** 返回触发次数，从上电时开始统计。
返回值	<0：失败 0：成功
说明	读取当前统计的 IO 触发测量开始信号的计数，即控制器 14 号脚接收到的有效输入信号计数。 *注：计数断电重启后清零；*
运行用时	约 1ms
支持版本	3.3.2.44

7.2.3 测量控制

- **切换程序**

格式	int SR7IF_SwitchProgram(unsigned int lDeviceId, int No);
参数	**lDeviceId(in)** 指定哪台设备进行切换，默认为 0。 **No (in)** 任务参数列表编号 0 - 63。
返回值	<0：失败 0：成功
说明	更换程序
运行用时	约 1ms
支持版本	3.3.1

- **获取传感头 B 是否在线**

格式	int SR7IF_GetOnlineCameraB(unsigned int lDeviceId)
参数	**lDeviceId(in)** 指定哪台设备进行切换，默认为 0。
返回值	<0：-982:传感头 B 不在线 其他:获取失败 0：传感头 B 在线
说明	当接单台相机时，必须确保 A 相机在线；查询 B 相机是否在线。
运行用时	小于 1ms
支持版本	3.3.1

- **开始批处理**

格式	int SR7IF_StartMeasure(unsigned int lDeviceId);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **Timeout (in)** 超时时间，默认为 50000。
返回值	<0：失败 0：成功
说明	开始批处理,立即执行批处理程序
运行用时	约 3~6ms
支持版本	3.3.1

- **开始批处理, 硬件 I0 触发开始批处理**

格式	int SR7IF_StartIOTriggerMeasure(unsigned int lDeviceId, int Timeout, int restart);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **Timeout (in)** 超时时间，默认为 50000。 **restart (in)** 预留接口，默认为 0。
返回值	<0：失败 0：成功
说明	开始批处理,并等待硬件 IO 触发开始批处理
运行用时	
支持版本	3.3.1

- **结束批处理**

格式	int SR7IF_StopMeasure(unsigned int lDeviceId);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。
返回值	<0：失败　0：成功
说明	结束批处理。若批处理尚未开始，将不做任何处理，也不会返回错误信息。若在批处理处理过程中结束，获取的轮廓数为已扫描的轮廓个数。
运行用时	约 1~4ms
支持版本	3.3.1

- **当前批处理设定行数**

格式	int SR7IF_ProfilePointSetCount (unsigned int lDeviceId, const SR7IF_Data DataObj);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **DataObj (in)** 预留,默认 NULL
返回值	返回实际批处理设定行数
说明	当前批处理设定行数
运行用时	小于 1ms
支持版本	3.3.1

- **阻塞方式获取数据**

格式	int SR7IF_ReceiveData(unsigned int lDeviceId, SR7IF_Data DataObj);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **DataObj (in)** 预留,默认 NULL
返回值	<0：失败 0：成功
说明	阻塞方式获取数据
运行用时	约 1400~20000ms（和批处理行数，网络带宽有关）
支持版本	3.3.1

- **获取批处理实际获取行数**

格式	int SR7IF_ProfilePointCount(unsigned int lDeviceId, const SR7IF_Data DataObj);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **DataObj (in)** 预留,默认 NULL
返回值	<0：失败 0：成功
说明	返回批处理实际获取行数
运行用时	小于 1ms
支持版本	3.3.1

- **获取数据宽度**

格式	int SR7IF_ProfileDataWidth(unsigned int lDeviceId, const SR7IF_Data DataObj);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **DataObj (in)** 预留,默认 NULL
返回值	返回数据宽度(单位像素)
说明	返回 x 方向上像素点数
运行用时	小于 1ms
支持版本	3.3.1

- **数据 x 方向间距**

格式	double SR7IF_ProfileData_XPitch (unsigned int lDeviceId, const SR7IF_Data DataObj);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **DataObj (in)** 预留,默认 NULL
返回值	返回数据 x 方向间距(mm)
说明	数据 x 方向点间距
运行用时	小于 1ms
支持版本	3.3.1

- **获取当前一条轮廓**

格式	int SR7IF_GetSingleProfile (unsigned int lDeviceId, int *pProfileData, unsigned int *pEncoder);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **pProfileData (out)** 返回轮廓的指针。 **pEncoder (out)** 返回编码器的指针。
返回值	<0：失败　0：成功
说明	获取非批处理模式下的最新的一条轮廓，批处理模式下不更新。
运行用时	小于 1ms
支持版本	3.3.1

- **参数设定**

格式	int SR7IF_SetSetting(unsigned int lDeviceId, int Depth, int Type, int Category, int Item, int Target[4], void *pData, int DataSize);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **Depth (in)** 设置的值的级别。 **Type (in)** 设置类型。 **Category (in)** 设置种类。 **Item (in)** 设置项目。 **Target[4] (in)** 根据发送 / 接收的设定，可能需要进行相应的指定。无需设定时，指定为 0 **pData (in)** 设置数据。 **DataSize** (in) 设置数据的长度。
返回值	<0：失败　0：成功
说明	参数设定。 **Depth**: 0x01 ：将参数写入非保存区域，参数生效，但关闭电源后设定将不被保存。 （重新启动时，保存区域的设定生效）

0x02：将参数写入保存区域，参数生效，关闭电源后设定仍将被保存到控制器。

Type：-1：返回当前配方的设定；

0x10h 至 0x4Fh：配方 0 至配方 63 中的哪一项设定。

Category：指定参数页，0：触发设定页面；1：拍摄设定页面；2：轮廓设定页面。

Item：指定发送 / 接收 Category 指定项目中的哪一项设定。

Target[4]：指定修改目标。

pData：写入的具体数据。

DataSize：写入数据的长度。

支持项目	Category	Item	Target[0]	pData	DataSize
触发模式	0x00	0x01	0	0：连续触发 1：外部触发 2：编码器触发	1
采样周期	0x00	0x02	0	10~67000	4
批处理开关	0x00	0x03	0	0：批处理 OFF 1：批处理 ON	1
编码器输入模式	0x00	0x07	0	0：1 相 1 递增； 1：2 相 1 递增； 2：2 相 2 递增； 3：2 相 4 递增；	1
细化点数（触发间隔）	0x00	0x09	0	1~10000	2
批处理点数	0x00	0x0a	0	50~15000	2
编码器类型	0x00	0x0b	0	0：单端 1：差分	1
循环模式	0x00	0x10	0	0：关闭 1：打开	1
批处理输出	0x00	0x21	0	0：轮廓+亮度 1：轮廓	1
Z 方向测量范围	0x01	0x03	0	*注：只支持 SR8020/SR8060。* 0：840 1：768 2：512 3：384 4：256 5：192 6：128 7：96 8：64 9：48	1

				10：32	
感光灵敏度	0x01	0x05	相机 A：0 相机 B：1	0：高精度 1：高动态范围 1 2：高动态范围 2 3：高动态范围 3 4：高动态范围 4 5：自定义高动态	1
曝光时间	0x01	0x06	相机 A：0 相机 B：1	0：10us 1：15us 2：30us 3：60us 4：120us 5：240us 6：480us 7：960us 8：1920us 9：2400us 10：4900us 11：9800us	1
光亮控制	0x01	0x0B	相机 A：0 相机 B：1	0：自动 1：手动	1
激光亮度上限	0x01	0x0C	相机 A：0 相机 B：1	1~99	1
激光亮度下限	0x01	0x0D	相机 A：0 相机 B：1	1~99	1
峰值灵敏度	0x01	0x0F	相机 A：0 相机 B：1	1~5	1
峰值选择	0x01	0x11	相机 A：0 相机 B：1	0：标准 1：near 2：far 3：使之转为无效数据 4：连续	1
X 轴压缩设定	0x02	0x02	0	1：OFF 2：2 4：4 8：8 16：16 *注：2.5D 模式下不能设置。*	1
X 轴中位数滤波	0x02	0x0A	相机 A：0 相机 B：1	1：OFF 3：3 点 5：5 点 7：7 点 9：9 点	1

	时间轴中位数滤波	0x02	0x0C	相机 A：0 相机 B：1	1：OFF 3：3 点 5：5 点 7：7 点 9：9 点	1
	平滑滤波	0x02	0x0B	相机 A：0 相机 B：1	1：1 次 2：2 次 4：4 次 8：8 次 16：16 次 32：32 次 64：64 次	1
	平均滤波	0x02	0x0D	相机 A：0 相机 B：1	1：1 次 2：2 次 4：4 次 8：8 次 16：16 次 32：32 次 64：64 次 128：128 次 256：256 次	2
	3D/2.5D 模式切换	0x30	0x0	0	0：3D 1：2.5D *注：2.5D 模式下，“X 轴压缩设定”自动变更为默认值。*	1

举例：

触发设定：

int data;

int depth = 0x01;//掉电不保存

int Type = 0x10;//设置配方 1

int target[4] = {0};

连续触发：data = 0x00; SR7IF_SetSetting(0, depth, Type, 0, 0x01, target, data, 1);

外部触发：data = 0x01; SR7IF_SetSetting(0, depth, Type, 0, 0x01, target, data, 1);

编码器触发：data = 0x02; SR7IF_SetSetting(0, depth, Type, 0, 0x01, target, data, 1);

批处理设定：

批处理关闭：data = 0x00; SR7IF_SetSetting(0, depth, Type, 0, 0x03, target, data, 1);

批处理使能：data = 0x01; SR7IF_SetSetting(0, depth, Type, 0, 0x03, target, data, 1);

支持版本	3.3.2.21

- **获取参数值**

格式	int SR7IF_GetSetting(unsigned int lDeviceId, int Type, int Category, int Item, int Target[4], void *pData, int DataSize);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **Type (in)** 指定获取类型。 **Category (in)** 指定获取种类。 **Item (in)** 指定获取项目。 **Target[4] (in)** 指定获取目标。 **pData (out)** 返回参数指针。 **DataSize (in)** 指定参数长度。
返回值	<0：失败 0：成功
说明	获取参数值： Type：程序 0 至程序 63(10h 至 4Fh）中的哪一项设定。 Category：指定参数页，0：触发设定页面；1：拍摄设定页面；2：轮廓设定页面。 Item：指定参数项，具体参数定义请参见“*参数设定*”中的说明。 Target[4]：通用参数时输入 0；传感头 A 参数输入 0；传感头 B 参数输入 1。 pData：返回的参数。 DataSize：获取参数的长度，具体参数长度请参见“*参数设定*”中的说明。
运行用时	小于 1ms
支持版本	3.3.2.21

- **设置多组编码器触发间隔**

格式	int SR7IF_SetMultiEncoderInterval(unsigned int lDeviceId, const SR7IF_Data DataObj, unsigned int enable, unsigned short Point1, unsigned short Interval1, unsigned short Point2, unsigned short Interval2, unsigned short Point3, unsigned short Interval3, unsigned short Point4, unsigned short Interval4, unsigned short Point5, unsigned short Interval5, unsigned short Point6, unsigned short Interval6, unsigned short Point7, unsigned short Interval7, unsigned short Point8, unsigned short Interval8);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **DataObj (in)** 预留,默认 NULL **enable (in)** 是否使能多组编码器触发间隔，1：开启；0：关闭。 **Point1~ Point8（in）** 指定“interval”开始生效的起始帧数。例如：Point1 为 100，Interval1 为 10，那么表示批处理的帧数从第 100 行开始，之后扫描的编码器触发间隔为 10； 参数有效范围：0~15000；其中 0：表示当前组不生效。 *注意：当设置多组的时候，编号大的帧数必须大于编号小的（帧数为0 除外），即 Point1<Point2<…<point8。* **Interval1~Interval8（in）** 指定批处理扫描过程中的编码器触发间隔。 参数有效范围：1~10000。
返回值	<0：失败 0：成功
说明	最多可以设置 8 组不同的编码器间隔，可使得在一次批处理过程中，分段按不同的编码器触发间隔（Y 向间距）扫描图像。 *注意：设置的参数断电不保存。*
运行用时	小于 1ms
支持版本	3.3.2.36

- **获取编码器值**

格式	int SR7IF_GetEncoder (unsigned int lDeviceId, const SR7IF_Data DataObj, unsigned int *Encoder);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **DataObj (in)** 预留,默认 NULL **Encoder (out)** 数据指针
返回值	<0：失败 0：成功
说明	获取编码器值
运行用时	小于 1ms
支持版本	3.3.1

- **非阻塞方式获取编码器值**

格式	int SR7IF_GetEncoderContiune (unsigned int lDeviceId, const SR7IF_Data DataObj, unsigned int *Encoder, unsigned int GetCnt);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **DataObj (in)** 预留,默认 NULL **Encoder (out)** 数据指针 **GetCnt (in)** 获取数据长度
返回值	<0：失败 >=0：实际返回的数据长度
说明	非阻塞方式获取编码器值
运行用时	小于 1ms
支持版本	3.3.1

- **阻塞方式获取轮廓数据**

格式	int SR7IF_GetProfileData(unsigned int lDeviceId, const SR7IF_Data DataObj, int *Profile);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **DataObj (in)** 预留,默认 NULL **Profile (out)** 数据指针。 返回数据为 Int 型，单位为 10nm。其中无效值以-10000*10^5 表示；
返回值	<0：失败　0：成功
说明	阻塞方式获取轮廓数据
运行用时	约 6~68ms
支持版本	3.3.1

- **非阻塞方式获取轮廓数据**

格式	int SR7IF_GetProfileContiuneData(unsigned int lDeviceId, const SR7IF_Data DataObj, int *Profile, unsigned int GetCnt);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **DataObj (in)** 预留,默认 NULL **Profile (out)** 数据指针，返回数据为 Int 型，单位为 10nm。其中无效值以-10000*10^5 表示； **GetCnt (in)** 获取数据长度
返回值	<0：失败　>=0：实际返回的数据长度
说明	非阻塞方式获取轮廓数据
运行用时	小于 1ms
支持版本	3.3.1

- **阻塞方式获取亮度数据**

格式	int SR7IF_GetIntensityData(unsigned int lDeviceId, const SR7IF_Data DataObj, unsigned char *Intensity);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **DataObj (in)** 预留,默认 NULL **Intensity (out)** 数据指针
返回值	<0：失败　0：成功
说明	阻塞方式获取亮度数据
运行用时	约 1~18ms
支持版本	3.3.1

- **非阻塞获取亮度数据**

格式	int SR7IF_GetIntensityContiuneData(unsigned int lDeviceId, const SR7IF_Data data, unsigned char *Intensity, unsigned int GetCnt);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **DataObj (in)** 预留,默认 NULL **Intensity (out)** 数据指针 **GetCnt (in)** 获取数据长度
返回值	<0：失败　>=0：实际返回的数据长度
说明	非阻塞获取亮度数据
运行用时	小于 1ms
支持版本	3.3.1

- **无终止循环设定终止行数**

格式	int SR7IF_SetBatchRollProfilePoint(unsigned int lDeviceId, const SR7IF_Data DataObj, unsigned int points);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **DataObj (in)** 预留，默认 NULL。 **Points (in)** 设定行数，范围（0：无终止循环，≥15000：设定终止行数，其他无效）。 *注：1.本接口设置的行数断电后不保存；* *2.行数默认为0。*
返回值	<0：失败 >=0：成功
说明	设置循环模式下，完成指定行数批处理时，退出循环。
运行用时	小于 1ms
支持版本	3.3.2.24

- **无终止循环获取数据**

格式	int SR7IF_GetBatchRollData(unsigned int lDeviceId, const SR7IF_Data DataObj, int *Profile, unsigned char *Intensity, unsigned int *Encoder, long long *FrameId, unsigned int *FrameLoss, unsigned int GetCnt);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3。 **DataObj (in)** 预留,默认 NULL **Profile (out)** 返回轮廓数据指针返回数据为 Int 型，单位为 10nm。其中无效值以-10000*10^5 表示； **Intensity (out)** 返回亮度数据指针 **Encoder (out)** 返回编码器数据指针 **FrameId (out)** 返回帧编号数据指针 **FrameLoss (out)** 返回批处理过快掉帧数量数据指针 **GetCnt (in)** 获取数据长度
返回值	-100：完成指定行数/IO 停止等；其他<0：失败； >0：实际返回的数据长度
说明	无终止循环获取数据
运行用时	小于 1ms
支持版本	3.3.1

- **无终止循环获取数据异常计算值**

格式	int SR7IF_GetBatchRollError (unsigned int lDeviceId, int *EthErrCnt, int *UserErrCnt);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3 **DataObj (in)** 预留,默认 NULL **EthErrCnt(out)** 返回网络传输导致错误的数量 **UserErrCnt(out)** 返回用户获取导致错误的数量
返回值	<0：失败 0：成功
说明	无终止循环获取数据异常计算值
运行用时	小于 1ms
支持版本	3.3.1

- **高度数据通信接口定义**

格式	typedef void (*SR7IF_CALLBACK)(char* pBuffer, unsigned int dwSize, unsigned int dwCount, unsigned int dwNotify, unsigned int dwDeviceId);
参数	**pBuffer (in)** 指向储存概要数据的缓冲区的指针 **dwSize (in)** 每个单元(行)的字节数量. **dwCount (in)** 存储在 pBuffer 中的内存的单元数量 **dwNotify (in)** 中断或批量结束等中断的通知 **dwDeviceId (in)** 执行回调的设备 ID 号
返回值	无
说明	高速数据通信的回调函数接口。 该接口会在主程序之外被调用，注意定义这个回调函数的线程安全。 每次获取新的数据前必须重新调用开始批处理函数。 **dwNotify 返回值定义:**

	位号 | 定义 0x00000004 | 批处理开始 0x00000008 | 超时错误 0x00000010 | 其他错误（停止 IO 信号输入等） 0x00010000 | 正常结束 0x80000000 | 批处理重新开始 其他 | 未定义
支持版本	3.3.1

- **高度数据通信接口初始化**

格式	int SR7IF_HighSpeedDataEthernetCommunicationInitalize(unsigned int lDeviceId, SR7IF_ETHERNET_CONFIG* pEthernetConfig, int wHighSpeedPortNo, SR7IF_CALLBACK pCallBack, unsigned int dwProfileCnt, unsigned int dwThreadId);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3 **pEthernetConfig (in)** Ethernet 通信设定 **wHighSpeedPortNo (in)** Ethernet 通信端口设定 **pCallBack (in)** 高速通信中数据接收的回调函数 **dwProfileCnt (in)** 回调函数被调用的频率 **dwThreadId (in)** 线程号
返回值	<0：失败 0：成功
说明	初始化以太网高速数据通信。
运行用时	约 1ms
支持版本	3.3.1

- **回调函数定义**

格式	typedef void (*SR7IF_BatchOneTimeCallBack) (const void *info, const SR7IF_Data *data);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3 **data (in)** 内部数据通信指针
返回值	无
说明	回调函数接口。 该接口会在主程序之外被调用，注意定义这个回调函数的线程安全。 每次获取新数据前无需调用开始批处理接口，在每次批处理结束后会自动调用 1 次。
运行用时	约 1ms
支持版本	3.3.2.8

- **回调函数设置**

格式	int SR7IF_SetBatchOneTimeDataHandler(unsigned int lDeviceId, SR7IF_BatchOneTimeCallBack CallFunc);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3 **CallFunc (in)** 回调函数
返回值	<0：失败　0：成功
说明	设置回调函数。
运行用时	约 1ms
支持版本	3.3.2.8

- **回调函数的开始批处理接口**

格式	int SR7IF_StartMeasureWithCallback(int iDeviceId, int ImmediateBatch);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3 **ImmediateBatch (in)** 0:立即开始批处理 1:等待外部硬件信号开始批处理.
返回值	<0：失败 0：成功
说明	回调方式的开始批处理函数。在第 1 次批处理前调用 1 次即可，后续的数据获取流程中无需再调用。
运行用时	约 1ms
支持版本	3.3.2.8

- **获取批处理的轮廓数据**

格式	const int *SR7IF_GetBatchProfilePoint(const SR7IF_Data *DataIndex, int Head);
参数	**DataIndex (in)** 内部数据通信指针，由回调函数传入。 **Head (in)** 指定接收哪个传感头的数据。 0: 传感头 A 1: 传感头 B
返回值	!=NULL：返回数据指针 ==NULL：失败，无数据或者相应头不存在 返回轮廓数据为 Int 型，单位为 10nm。其中无效值以-10000*10^5 表示；
说明	在回调函数中使用，获取批处理后的所有轮廓数据。
运行用时	约 1ms
支持版本	3.3.2.8

- **获取批处理的亮度数据**

格式	const unsigned char *SR7IF_GetBatchIntensityPoint(const SR7IF_Data *DataIndex, int Head);
参数	**DataIndex (in)** 内部数据通信指针，由回调函数传入。 **Head (in)** 指定接收哪个传感头的数据。 0: 传感头 A 1: 传感头 B
返回值	!=NULL：返回数据指针 ==NULL：失败，无数据或者相应头不存在

说明	在回调函数中使用，获取批处理后的所有灰度数据。
运行用时	约 1ms
支持版本	3.3.2.8

- **获取批处理的编码器值**

格式	const unsigned int *SR7IF_GetBatchEncoderPoint(const SR7IF_Data *DataIndex, int Head);
参数	**DataIndex (in)** 内部数据通信指针，由回调函数传入。 **Head (in)** 指定接收哪个传感头的数据。 0: 传感头 A 1: 传感头 B
返回值	!=NULL：返回数据指针； ==NULL：失败，无数据或者相应头不存在；
说明	在回调函数中使用，获取批处理后的每帧数据对应的编码器值。
运行用时	约 1ms
支持版本	3.3.2.8

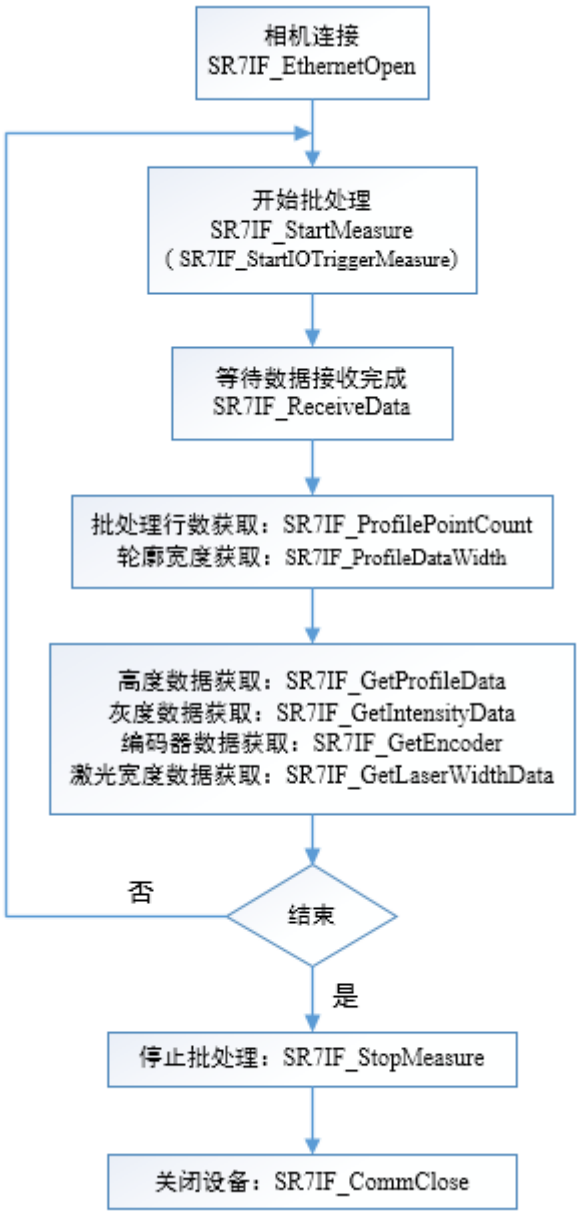
- **批处理软件触发开始**

格式	int SR7IF_TriggerOneBatch(int iDeviceId);
参数	**lDeviceId(in)** 指定通过哪台通信设备进行通信，范围 0-3
返回值	<0：失败 0：成功
说明	在回调函数的方式下使用，可以软件触发相机进入批处理，无需外部硬件触发信号。 *注意：如果当前相机处于批处理中，该函数会中断当前的批处理，然后再开启新一次的批处理。*
运行用时	约 1ms
支持版本	3.3.2.8

8 软件开发接口函数调用顺序例程

8.1 阻塞方式获取数据

基本流程图如下：



示例代码：

```
1.  int _tmain(int argc, _TCHAR* argv[])
2.  {
3.      //相机 IP 地址
4.      SR7IF_ETHERNET_CONFIG SREthernetConFig;
5.      SREthernetConFig.abyIpAddress[0] = 192;
6.      SREthernetConFig.abyIpAddress[1] = 168;
```

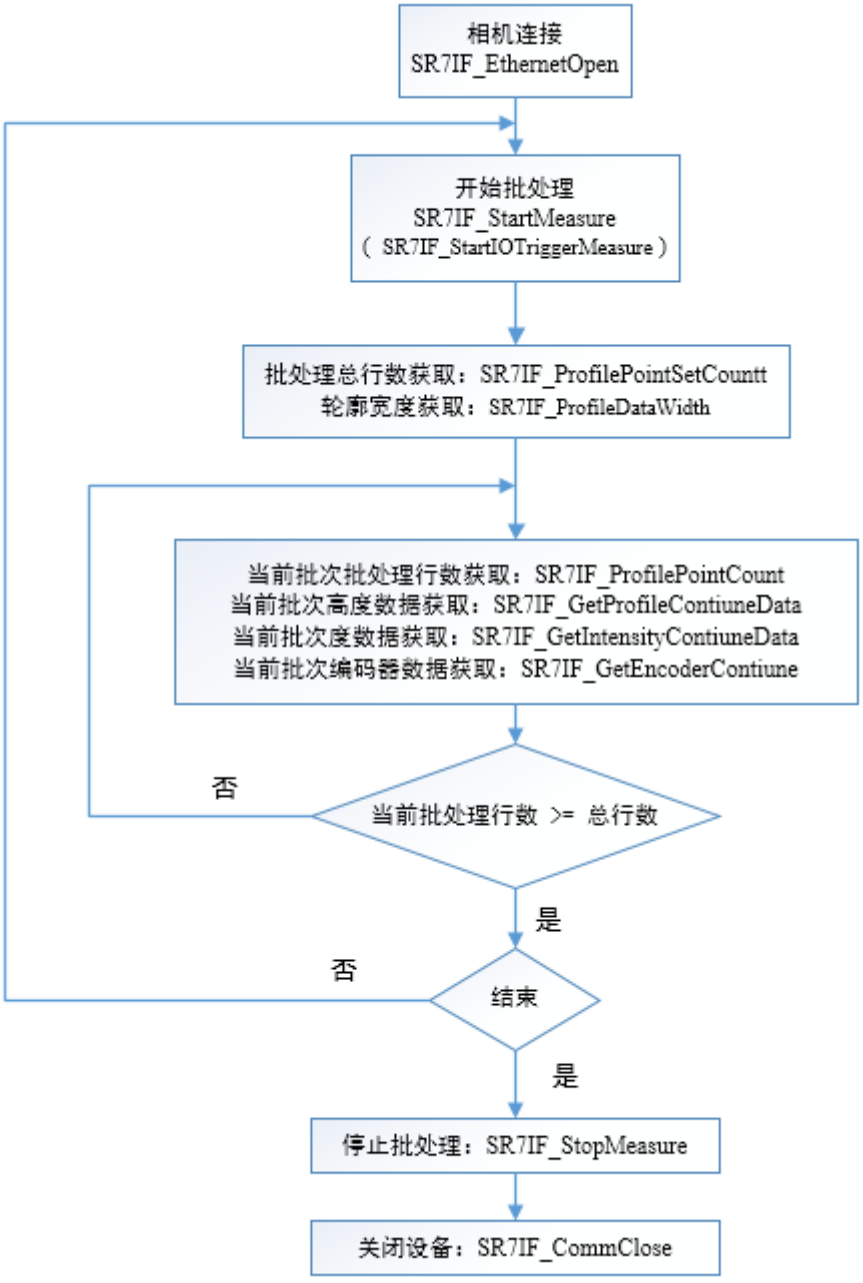
```
7.      SREthernetConFig.abyIpAddress[2] = 0;
8.      SREthernetConFig.abyIpAddress[3] = 122;
9.
10.     //设备 ID
11.     const int DEVICE_ID = 0;
12.
13.     //连接相机
14.     SR7IF_EthernetOpen(DEVICE_ID, &SREthernetConFig);
15.
16.     //开始批处理---------IO 触发时调用函数 SR7IF_StartIOTriggerMeasure(DEVICE_ID, 20000)
17.     SR7IF_StartMeasure(DEVICE_ID, 20000);
18.
19.     //等待数据接收
20.     SR7IF_Data DataObject = NULL;
21.     SR7IF_ReceiveData(0, DataObject);
22.
23.     //获取批处理行数
24.     int BatchPoint = SR7IF_ProfilePointCount(DEVICE_ID, DataObject);
25.
26.     //获取轮廓宽度
27.     int m_DataWidth = SR7IF_ProfileDataWidth(DEVICE_ID, DataObject);
28.
29.     //获取高度数据
30.     //定义高度数据缓存
31.     int * HeightData = new int[BatchPoint * m_DataWidth];
32.     SR7IF_GetProfileData(DEVICE_ID, DataObject, HeightData);
33.     //获取灰度数据
34.     //定义灰度数据缓存
35.     unsigned char* grayData = new unsigned char[BatchPoint * m_DataWidth];
36.     SR7IF_GetIntensityData(DEVICE_ID, DataObject, grayData);
37.
38.     //获取编码器数据
39.     //定义编码器数据缓存
40.     unsigned int* Encoder = new unsigned int[BatchPoint];
41.     SR7IF_GetEncoder(DEVICE_ID, DataObject, Encoder);
42.
43.     /******用户代码******/
44.
45.
46.     /********************/
47.
48.     //内存释放
49.     delete[] HeightData;
50.     delete[] grayData;
51.     delete[] Encoder;
```

```
52.
53.     //停止批处理
54.     SR7IF_StopMeasure(DEVICE_ID);
55.
56.     //关闭设备
57.     SR7IF_CommClose(DEVICE_ID);
58.     return 0;
59. }
```

详细代码见：DataBlockReceive.cpp 文件

8.2 非阻塞方式有限循环获取数据

基本流程图如下：



示例代码：

```
1.  int _tmain(int argc, _TCHAR* argv[])
2.  {
3.      //相机 IP
4.      SR7IF_ETHERNET_CONFIG SREthernetConFig;
5.      SREthernetConFig.abyIpAddress[0] = 192;
6.      SREthernetConFig.abyIpAddress[1] = 168;
7.      SREthernetConFig.abyIpAddress[2] = 0;
8.      SREthernetConFig.abyIpAddress[3] = 122;
```

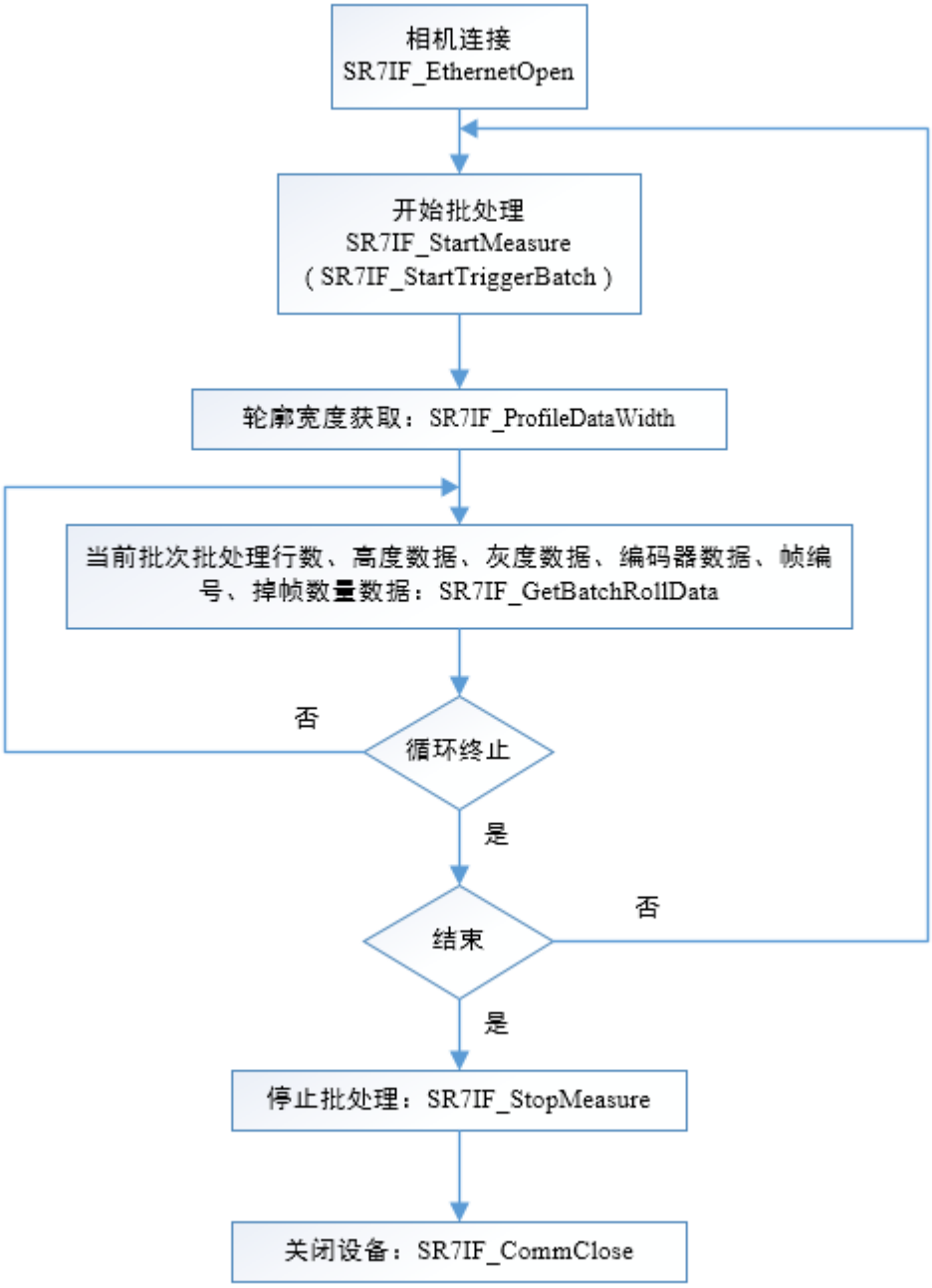
```
9.
10.     //设备 ID
11.     const int DEVICE_ID = 0;
12.     SR7IF_Data DataObject = NULL;
13.
14.     //连接相机
15.     SR7IF_EthernetOpen(DEVICE_ID, &SREthernetConFig);
16.
17.     //开始批处理---------IO 触发时调用函数 SR7IF_StartIOTriggerMeasure(DEVICE_ID, 20000)
18.     SR7IF_StartMeasure(DEVICE_ID, 20000);
19.
20.     //批处理总行数获取
21.     int BatchPoint = SR7IF_ProfilePointSetCount(DEVICE_ID, DataObject);       //获取总行数
22.
23.     //获取轮廓宽度
24.     int m_DataWidth = SR7IF_ProfileDataWidth(DEVICE_ID, DataObject);
25.
26.     //循环获取批处理数据
27.     int lastBatchPoint_CurNo = 0;                                             //上一次接收到的批处理数
    据行数编号
28.     int * HeightData = new int[BatchPoint * m_DataWidth];                     //高度数据缓存
29.     unsigned char* grayData = new unsigned char[BatchPoint * m_DataWidth];    //灰度数据缓存
30.     unsigned int* Encoder = new unsigned int[BatchPoint];                     //编码器数据缓存
31.     do
32.     {
33.         //获取当前批次的批处理行数编号
34.         int BatchPoint_CurNo = SR7IF_ProfilePointCount(DEVICE_ID, DataObject);
35.         //当前批次批处理行数
36.         int m_curBatchPoint = BatchPoint_CurNo - lastBatchPoint_CurNo;
37.         //获取当前批次高度数据
38.         SR7IF_GetProfileContiuneData(DEVICE_ID, DataObject, &HeightData[lastBatchPoint_CurNo * m_Data
    Width], m_curBatchPoint);
39.         //获取当前批次灰度数据
40.         SR7IF_GetIntensityContiuneData(DEVICE_ID, DataObject, &grayData[lastBatchPoint_CurNo * m_Data
    Width], m_curBatchPoint);
41.         //获取当前批次编码器数据
42.         SR7IF_GetEncoderContiune(DEVICE_ID, DataObject, &Encoder[lastBatchPoint_CurNo], m_curBatchPoi
    nt);
43.         lastBatchPoint_CurNo = BatchPoint_CurNo;
44.         Sleep(50);
45.     }
46.     while(lastBatchPoint_CurNo < BatchPoint);
47.
48.     /******用户代码******/
49.
```

```
50.
51.     /********************/
52.
53.     //内存释放
54.     delete[] HeightData;
55.     delete[] grayData;
56.     delete[] Encoder;
57.
58.     //停止批处理
59.     SR7IF_StopMeasure(DEVICE_ID);
60.
61.     //关闭设备
62.     SR7IF_CommClose(DEVICE_ID);
63.     return 0;
64. }
```

详细代码见：DataRollReceive.cpp 文件

8.3 非阻塞方式无限循环获取数据

基本流程图如下：



示例代码：

```
1.  int _tmain(int argc, _TCHAR* argv[])
2.  {
3.      SR7IF_ETHERNET_CONFIG SREthernetConFig;
4.      SREthernetConFig.abyIpAddress[0] = 192;
5.      SREthernetConFig.abyIpAddress[1] = 168;
6.      SREthernetConFig.abyIpAddress[2] = 0;
7.      SREthernetConFig.abyIpAddress[3] = 122;
```

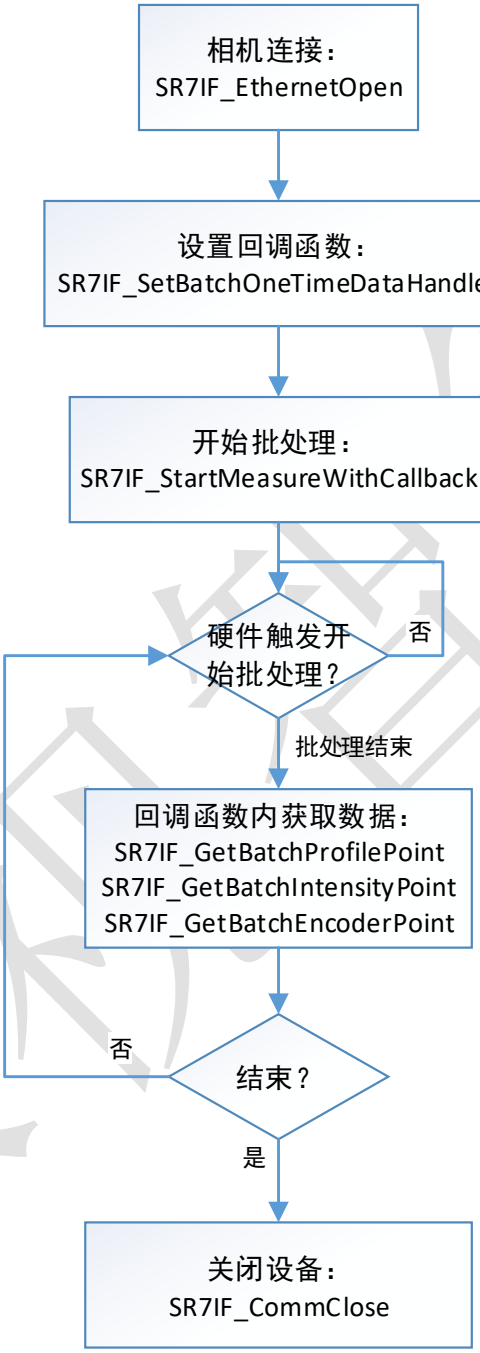
```
8.      const int DEVICE_ID = 0;
9.      SR7IF_Data DataObject = NULL;
10.
11.     //连接相机
12.     SR7IF_EthernetOpen(DEVICE_ID, &SREthernetConFig);
13.
14.     //开始批处理---------IO 触发时调用函数 SR7IF_StartIOTriggerMeasure(DEVICE_ID, 20000)
15.     SR7IF_StartMeasure(DEVICE_ID, 20000);
16.
17.     //获取轮廓宽度
18.     int m_DataWidth = SR7IF_ProfileDataWidth(DEVICE_ID, DataObject);
19.
20.     bool b_stop =  true;                                          //循环停止标志
21.     int* HeightData = new int[1000 * m_DataWidth];                //当前批次高度数据缓存
22.     unsigned char* GrayData = new unsigned char[1000 * m_DataWidth];  //当前批次灰度数据缓存
23.     long long BatchPoint_CurNo = 0;                               //当前批处理编号
24.     unsigned int* FrameLoss =  new unsigned int [1000];           //批处理过快掉帧数量数据缓存
25.     long long *FrameId = new long long[1000];                     //帧编号数据缓存
26.     unsigned int* Encoder = new unsigned int [1000];              //编码器数据缓存
27.     long long OverFlowStartId = 0;                                //溢出起始帧号
28.     int FrameLossID = 0;                                          //丢帧数
29.     int EncoderID = 0;                                            //编码器值
30.
31.     //循环接收数据
32.     do
33.     {
34.         /****循环跳出条件设置***/
35.
36.
37.         /**********************/
38.         //数据接收
39.         //当前批次高度数据、灰度数据、编码器数据、帧编号、掉帧数量数据
40.         int m_curBatchPoint = SR7IF_GetBatchRollData(DEVICE_ID, DataObject, HeightData, GrayData, Encode
    r, FrameId, FrameLoss, 500);
41.         if(m_curBatchPoint == 0)
42.             continue;
43.         int TmpID = m_curBatchPoint - 1;
44.         OverFlowStartId = FrameId[TmpID];
45.         FrameLossID = FrameLoss[TmpID];
46.         EncoderID = Encoder[TmpID];
47.         BatchPoint_CurNo += m_curBatchPoint;
48.         Sleep(50);
49.     }while(b_stop);
50.
51.     /****用户程序处理****/
```

```
52.
53.
54.     /********************/
55.
56.     //内存释放
57.     delete[] HeightData;
58.     delete[] GrayData;
59.     delete[] FrameLoss;
60.     delete[] Encoder;
61.     delete[] FrameId;
62.
63.     //停止批处理
64.     SR7IF_StopMeasure(DEVICE_ID);
65.
66.     //关闭设备
67.     SR7IF_CommClose(DEVICE_ID);
68.     return 0;
69. }
```

详细代码见：DataLoopReceive.cpp 文件

8.4 回调方式获取数据

基本流程图如下：



示例代码：

```
1.  static void BatchOneTimeCallBack(const void *info, const SR7IF_Data *data)
2.  {
3.      const SR7IF_STR_CALLBACK_INFO * conInfo = (const SR7IF_STR_CALLBACK_INFO*) info;
4.
5.      int mNumD = conInfo->BatchPoints * conInfo->xPoints;
6.
7.      //高度数据获取--相机 A
```

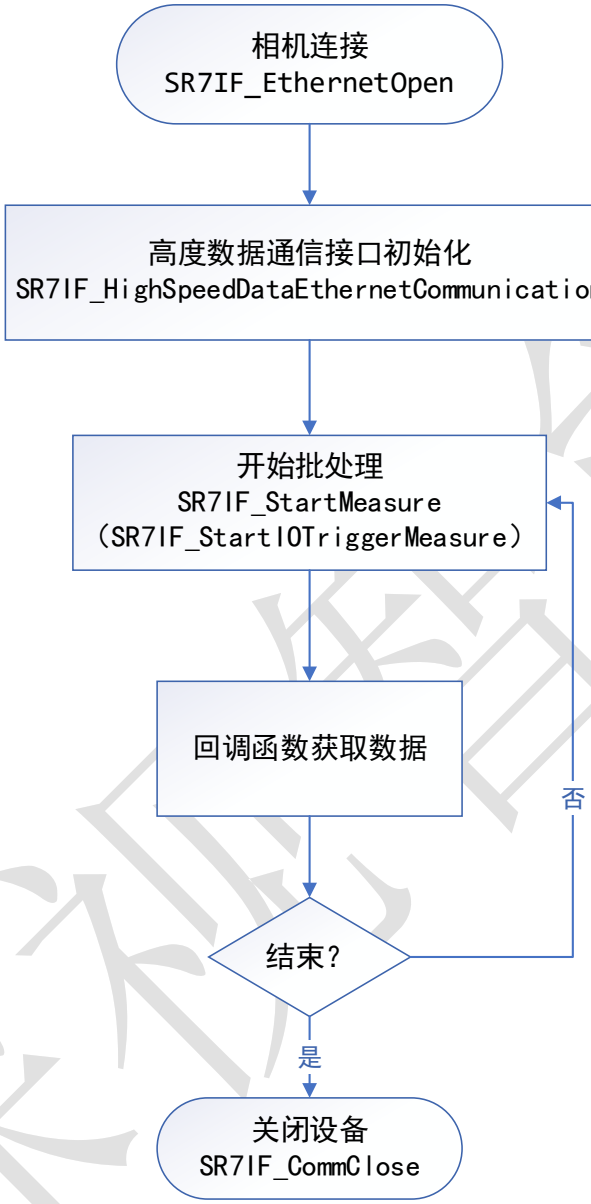
```
8.      const int* mTmpData = SR7IF_GetBatchProfilePoint(data, 0);
9.      int mNumP = sizeof(int) * conInfo->BatchPoints * conInfo->xPoints;
10.
11.     //高度数据缓存
12.     int *mProfileDataA = new int[mNumD];
13.
14.     if(mTmpData !=  NULL)
15.     {
16.         memset(mProfileDataA, -1000000000, mNumP);
17.         memcpy(mProfileDataA, &mTmpData[0], mNumP);
18.     }
19.
20.     //灰度数据获取
21.     int mNumG = sizeof(unsigned char) * conInfo->BatchPoints * conInfo->xPoints;
22.     const unsigned char* mTmpGraydata = SR7IF_GetBatchIntensityPoint(data, 0);
23.
24.     //灰度数据缓存
25.     char *mIntensityDataA = new char[mNumD];
26.
27.     if(mTmpGraydata != NULL)
28.     {
29.         memset(mIntensityDataA, 0, mNumG);
30.         memcpy(mIntensityDataA, &mTmpGraydata[0], mNumG);
31.     }
32.
33.     //编码器数据获取
34.     int mNumE = sizeof(unsigned int) * conInfo->BatchPoints;
35.     const unsigned int* mTmpEncoderdata  = SR7IF_GetBatchEncoderPoint(data, 0);
36.
37.     //编码器数据缓存
38.     int *mEncoderDataA = new int[conInfo->BatchPoints];
39.
40.     if(mTmpEncoderdata != NULL)
41.     {
42.         memset(mEncoderDataA, 0, mNumE);
43.         memcpy(mEncoderDataA, &mTmpEncoderdata[0], mNumE);
44.     }
45.
46.     if(conInfo->HeadNumber == 2)
47.     {
48.         //相机 B 数据
49.         const int* mTmpDataB = SR7IF_GetBatchProfilePoint(data, 1);
50.
51.         //高度数据缓存
52.         int *mProfileDataB = new int[mNumD];
```

```
53.
54.         if(mTmpData !=  NULL)
55.         {
56.             memset(mProfileDataB, -1000000000, mNumP);
57.             memcpy(mProfileDataB, &mTmpDataB[0], mNumP);
58.         }
59.
60.         const unsigned char* mTmpGraydataB = SR7IF_GetBatchIntensityPoint(data, 1);
61.
62.         //灰度数据缓存
63.         char *mIntensityDataB = new char[mNumD];
64.
65.         if(mTmpGraydataB != NULL)
66.         {
67.             memset(mIntensityDataB, 0, mNumG);
68.             memcpy(mIntensityDataB, &mTmpGraydataB[0], mNumG);
69.         }
70.
71.         const unsigned int* mTmpEncoderdataB  = SR7IF_GetBatchEncoderPoint(data, 1);
72.
73.         //编码器数据缓存
74.         int *mEncoderDataB = new int[conInfo->BatchPoints];
75.
76.         if(mTmpEncoderdata != NULL)
77.         {
78.             memset(mEncoderDataB, 0, mNumE);
79.             memcpy(mEncoderDataB, &mTmpEncoderdataB[0], mNumE);
80.         }
81.     }
82.
83. }
84.
85. int _tmain(int argc, _TCHAR* argv[])
86. {
87.
88.     //网络配置结构体
89.     SR7IF_ETHERNET_CONFIG SREthernetConFig;
90.     SREthernetConFig.abyIpAddress[0] = 192;
91.     SREthernetConFig.abyIpAddress[1] = 168;
92.     SREthernetConFig.abyIpAddress[2] = 0;
93.     SREthernetConFig.abyIpAddress[3] = 124;
94.
95.     //设备 ID 号 0-3
96.     const int DEVICE_ID = 0;
97.     SR7IF_Data DataObject = NULL;
```

```
98.
99.     //连接相机
100.     int reT = SR7IF_EthernetOpen(DEVICE_ID, &SREthernetConFig);
101.     if(reT < 0)
102.     {
103.         printf("Error: 连接失败:%d\n", reT);
104.         system("pause");  //窗口暂停
105.         return -1;
106.     }
107.
108.     //回调函数设置
109.     reT = SR7IF_SetBatchOneTimeDataHandler(DEVICE_ID, BatchOneTimeCallBack);
110.
111.     if(reT < 0)
112.     {
113.         printf("Error: 回调函数设置失败:%d\n", reT);
114.
115.         //关闭设备
116.         reT = SR7IF_CommClose(DEVICE_ID);
117.         if(reT < 0)
118.             printf("Error: 关闭设备失败:%d\n", reT);
119.         system("pause");  //窗口暂停
120.         return -1;
121.     }
122.     else
123.     {
124.         // 等待外部硬件触发批处理
125.         reT = SR7IF_StartMeasureWithCallback(0, 1);
126.         if(reT < 0)
127.         {
128.             printf("Error: 开始批处理失败:%d\n", reT);
129.             system("pause");  //窗口暂停
130.             return -1;
131.         }
132.     }
133.
134.     system("pause");  //窗口暂停
135.     return 0;
136. }
```

8.5 高速数据通信方式获取数据

基本流程图如下：



示例代码：

```
1.  /*
2.   * DataReceiveCallBack.cpp
3.   *
4.   * SR7000 Sample
5.   * Copyright (C) 2018 by SSZN
6.   *
7.   * Redistributions of files must retain the above copyright notice.
8.   *
9.   * Purpose: Connect to SR7000 system and receive data in Batch Mode. SR7000 must be in
    Batch mode.
10.  * Ethernet output for the profile data must be enabled.
```

```
 *
 */

#include "stdafx.h"
#include <stdlib.h>
#include "Lib\SR7Link.h"
#include  <windows.h>

static void  ReceiveHighSpeedData(char* pBuffer, unsigned int  uSize, unsigned int  uCount, unsigned int  uNotify, unsigned int uUser)
{
    if(uCount == 0 || uSize == 0)
        return;

    static int m_curBatchNo = 0;     //当前批次批处理行数编号
    int nProfDataCnt = uSize / sizeof(int);   //轮廓宽度 ---- 双相机 6400 点  前 3200 点为 A 相机的轮廓数据，后 3200 点为 B 相机的轮廓数据
    int TmpDNum = m_curBatchNo * nProfDataCnt;

    int* ProfileDataCall = new int[uCount * nProfDataCnt];     //当前批次轮廓数据缓存
    unsigned int* EncoderDataCall = new unsigned int[uCount * 2];  //当前批次编码器数据缓存  双相机乘以系数 2

    //轮廓数据拷贝
    memcpy(ProfileDataCall, pBuffer, uSize * uCount);
    //获取编码器数据
    SR7IF_GetEncoderContiune(0, NULL, EncoderDataCall, uCount);

    //打印当前批次前 m_count 个数据
    int m_count = 8;
    if(m_count >= uCount)
        m_count = uCount;

    printf("当前批次每个相机前%d 个高度数据:  \n", m_count);
    for(int i = 0; i < m_count; i++)
        printf("相机 A: %d, 相机 B: %d,\n", ProfileDataCall[i], ProfileDataCall[i + nProfDataCnt]);

    printf("\n");

    printf("当前批次每个相机前%d 个编码器数据:  \n", m_count);
    for(int i = 0; i < m_count; i++)
        printf("相机 A: %u, 相机 B: %u,\n", EncoderDataCall[2 * i], EncoderDataCall[2 * i + 1]);

```

```
51.     printf("\n");
52.
53.     m_curBatchNo += uCount;
54.
55.     delete[] ProfileDataCall;
56.     delete[] EncoderDataCall;
57.
58.     if(uNotify != 0)
59.     {
60.         if(uNotify == 0x10000)
61.         {
62.             printf("数据接收完成!\n");
63.             m_curBatchNo = 0;
64.         }
65.         if(uNotify & 0x80000000)
66.         {
67.             printf("批处理重新开始!\n");
68.             m_curBatchNo = 0;
69.         }
70.         if(uNotify & 0x08)
71.         {
72.             printf("批处理超时!\n");
73.             m_curBatchNo = 0;
74.         }
75.         if(uNotify & 0x04)
76.             printf("新批处理!\n");
77.     }
78. }
79.
80. int _tmain(int argc, _TCHAR* argv[])
81. {
82.
83.     //网络配置结构体
84.     SR7IF_ETHERNET_CONFIG SREthernetConFig;
85.     SREthernetConFig.abyIpAddress[0] = 192;
86.     SREthernetConFig.abyIpAddress[1] = 168;
87.     SREthernetConFig.abyIpAddress[2] = 0;
88.     SREthernetConFig.abyIpAddress[3] = 124;
89.
90.     //设备 ID 号 0-3
91.     const int DEVICE_ID = 0;
92.     SR7IF_Data DataObject = NULL;
93.
94.     //连接相机
95.     int reT = SR7IF_EthernetOpen(DEVICE_ID, &SREthernetConFig);
```

```
96.     if(reT < 0)
97.     {
98.         printf("Error: 连接失败:%d\n", reT);
99.         system("pause");  //窗口暂停
100.         return -1;
101.     }
102.
103.     //高速数据通信初始化
104.     unsigned int dwProfileCnt = 10;
105.     reT = SR7IF_HighSpeedDataEthernetCommunicationInitalize(DEVICE_ID,
106.                 &SREthernetConFig,
107.                 0,
108.                 ReceiveHighSpeedData,
109.                 dwProfileCnt,
110.                 DEVICE_ID);
111.
112.     if(reT < 0)
113.     {
114.         printf("Error: 高速数据以太网通信初始化失败:%d\n", reT);
115.
116.         //关闭设备
117.         reT = SR7IF_CommClose(DEVICE_ID);
118.         if(reT < 0)
119.             printf("Error: 关闭设备失败:%d\n", reT);
120.         system("pause");  //窗口暂停
121.         return -1;
122.     }
123.
124.     //开始批处理----IO 触发时调用函 SR7IF_StartIOTriggerMeasure(DEVICE_ID, 20000)
125.     //20000---20000 毫秒内完成自动进入下一步，不需要等待 20000 毫秒；时间超过 20000 毫秒，超
    时
126.     reT = SR7IF_StartMeasure(DEVICE_ID, 20000);
127.
128.     if(reT == SR7IF_ERROR_MODE)
129.     {
130.         printf("Error: 当前批处理为循环模式\n");
131.         //停止批处理
132.         reT = SR7IF_StopMeasure(DEVICE_ID);
133.         if(reT < 0)
134.         {
135.             printf("Error: 停止批处理失败:%d\n", reT);
136.             system("pause");  //窗口暂停
137.             return -1;
138.         }
139.         //关闭设备
```

```
140.         reT = SR7IF_CommClose(DEVICE_ID);
141.         if(reT < 0)
142.         {
143.             printf("Error: 关闭设备失败:%d\n", reT);
144.             system("pause");  //窗口暂停
145.         }
146.         system("pause");  //窗口暂停
147.         return -1;
148.     }
149.     else if(reT < 0)
150.     {
151.         printf("Error: 开始批处理失败:%d\n", reT);
152.         //关闭设备
153.         reT = SR7IF_CommClose(DEVICE_ID);
154.         if(reT < 0)
155.             printf("Error: 关闭设备失败:%d\n", reT);
156.         system("pause");  //窗口暂停
157.         return -1;
158.     }
159.
160.     system("pause");  //窗口暂停
161.     return 0;
162. }
```