软件开发文档

Software Development Kit

ftcoreIMC

v1.3.0.9

2024.11.19

版本历史

v1.3.0.9 （2024-11-19）

（1）**修改** sdk 发布流程过程优化。

v1.3.0.0 （2024-10-23）

- **首次发布**，C 语言动态链接库，提供 .NET Standard 2.0 封装库。
- 基于 libmodbus v3.1.10（LGPL v2.1+，https://libmodbus.org/，2022 年 12 月 07 日）
- 采用批处理方式进行版本发布，生成的库文件目录包括 inc、lib 和 netstandard2.0。inc 目录下为头文件，lib 目录下按 x86 和 x64 两个版本分别存储 ftcoreimc.dll 和 ftcoreimc.lib，netstandard2.0 目录下存储 ftcoreimccs.dll（注：.NET Standard 2.0，不区分 x86 和 x64）。
- 主要功能如下：

（1）串口方式连接（MODBUS RTU），或网络方式连接（MODBUS-TCP，串口服务器，MODBUS 转换模式）；

（2）参数设置：细分、螺距、加速系数、减速系数、搜零速度、速度等。

（3）单轴控制：相对运动、绝对运动、向左点动、向右点动、停止运行、搜零等。

（4）单轴状态查询：位置、限位、运行状态（运行中或停止）。

（5）级联时，逐个控制器连接，修改控制器地址（**错误！未定义书签。**），注意记录好新地址；

（6）支持 windows 系统 COM10 及以上端口（内部会转换为“\\\\.\\COM10”）。

- 版本号定义：major.minor.patch.build

  1.1.1.0 ==> major=1，minor=0，patch=1，build=0

  版本号命名规则为“主版本号.次版本号.修订号.编译号”（<major>.<minor>.<patch>.<build>），API 有重大更新时主版本号增加 1（可能不向下兼容）、API 有新增时次版本号增加 1（确保 API 向下兼容）、bug 修复时修订号增加 1、内部测试不同版本时编译号增加 1（用于区分不同测试版本）。

目　录

1. 安装

1.1. 技术支持

1.2. 定位

SDK 用于为客户提供运动控制器的二次开发包，便于将运动控制集成至客户软件中。

1.3. 关键特性

- 简洁的 API
- 控制器全功能控制
- C 语言开发
- 操作系统支持 windows 和 linux/ubuntu （亦可适配国产中标麒麟系统 aarch64 架构）
- 额外提供 C# 封装库（.NET Standard 2.0[1]）
- 支持原生 modbus 指令

1.4. 安装

sdk 包含 3 个文件夹，inc、lib、netstandard2.0：

（1）inc 文件夹下为 C 语言头文件，包括 ftcoreimc.h；

（2）lib 文件夹下为 C 语言动态链接库 ftcoreimc.dll 和静态链接库 ftcoreimc.lib，分为 x64 和 x86 两个版本；

（3）netstandard2.0 文件夹下为 C# 封装的动态链接库 ftcoreimccs.dll，不区分 x86 和 x64。

1.4.1 C 应用程序

导入头文件 ftcoreimc.h，添加静态库引用 ftcoreimc.lib 进行静态链接。

```
#include "<sdk_path>/inc/ftcoreimc.h"
```

[1] https://dotnet.microsoft.com/zh-cn/platform/dotnet-standard

```
#pragma comment(lib, "<sdk_path>/lib/x64/ftcoreimc.lib")
```

特别说明：运行时需要动态链接库 ftcoreimc.dll，请将 sdk 文件夹 lib 下 x86 或 x64 版本对应的 ftcoreimc.dll 文件至可执行程序所在目录下。

详见示例程序 6.1。

1.4.2 C# 应用程序

C# 应用程序需先添加引用。

具体如下（VS 2022 社区版 v17.4.2，其它版本参照相应说明）：在项目上点击右键，弹出窗口依次选择 Add->Project Reference（见图 1 所示），在弹出的界面上左侧列表中选择 Browse（见图 2 所示），然后点击 Browse 按钮选择动态链接库 ftcoreimccs.dll 所在位置，点击添加按钮（见图 3 所示）。sdk 使用方法详见示例程序 6.2。

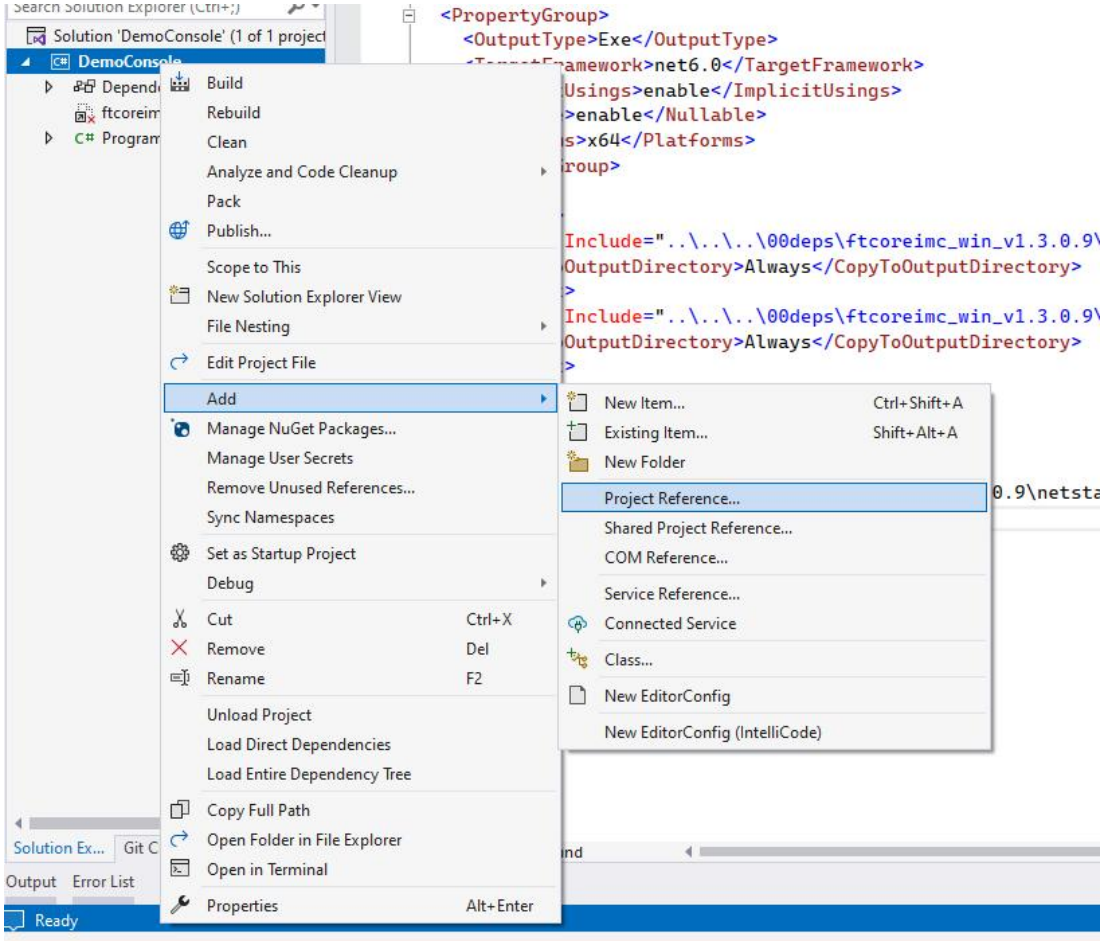


图 1 添加引用步骤 1（vs2022）

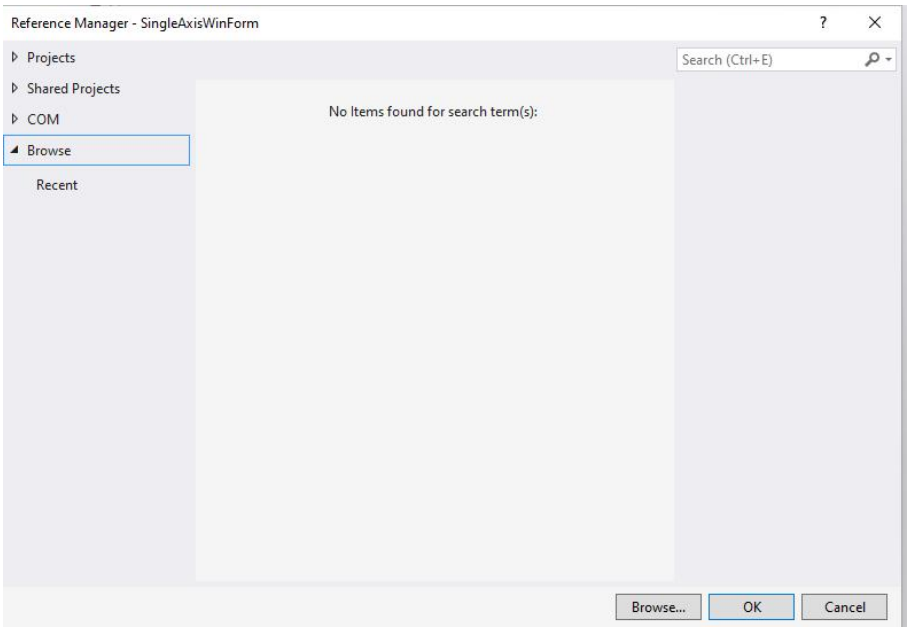


图 2 添加引用步骤 2（vs2022）

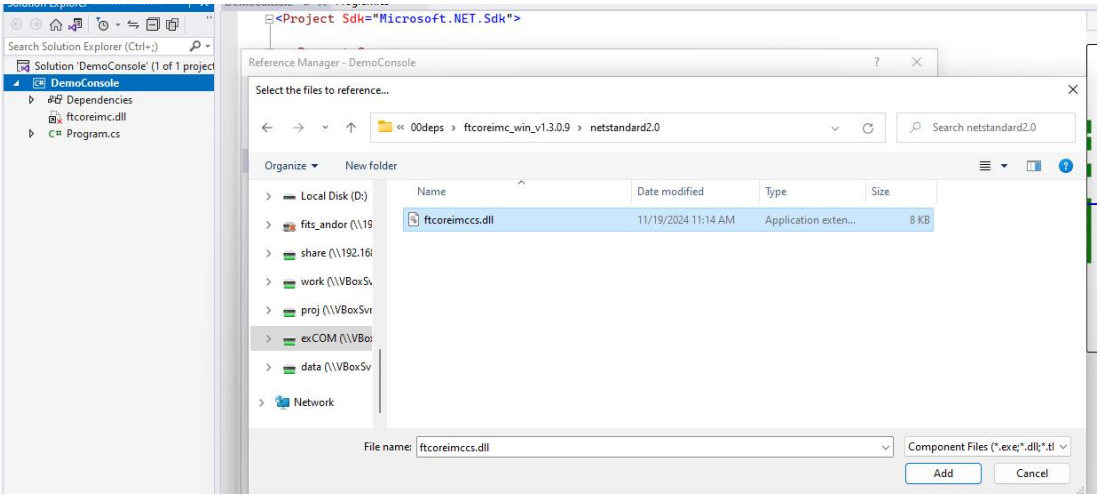


图 3 添加引用步骤 3（vs2022）

特别说明：运行时，需要 C 语言的底层库 ftcoreimc.dll，请将 sdk 文件夹 lib 完整拷贝至可执行程序所在路径下（见图 4 所示）。若可执行程序是确定的 x86 或 x64 版本，亦可仅拷贝对应的 ftcoreimc.dll 文件至可执行程序所在目录下。

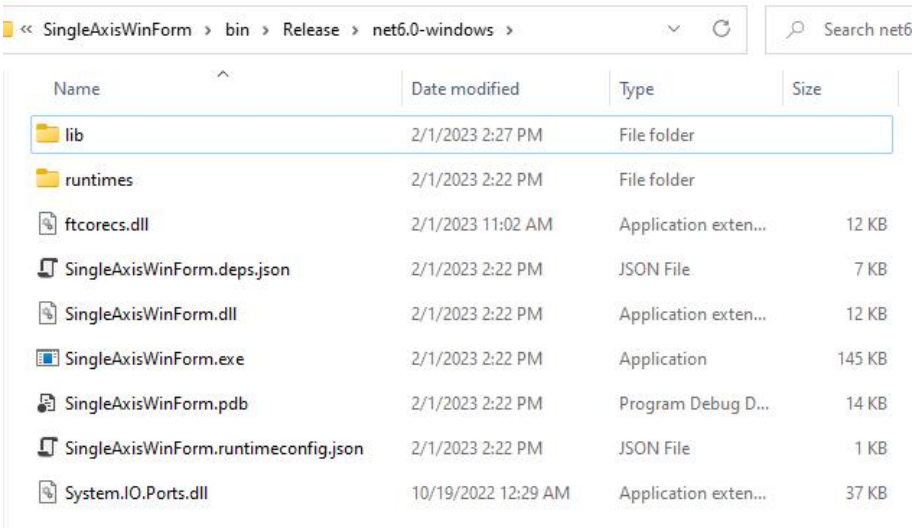


图 4 可执行程序目录结构示例（复制整个 lib 文件夹）

更推荐的方式：编辑工程文件 csproj，添加复制文件代码，如图 5 **所示，“CopyToOutputDirectory”，将根据编译选项复制相应版本的动态连接库 dll 文件到生成的可执行程序所在目录中。**

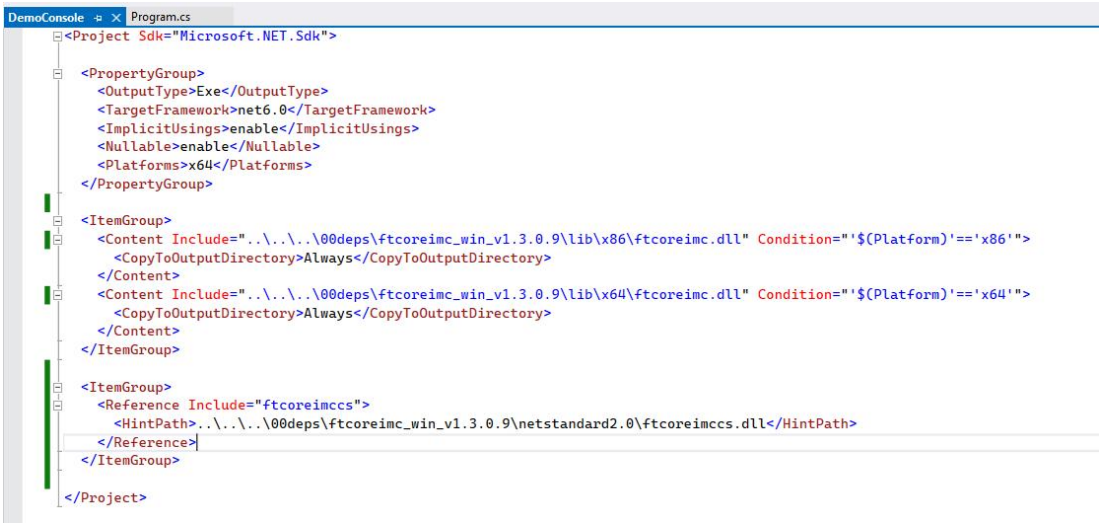


图 5 编辑 csproj 文件（vs2022）

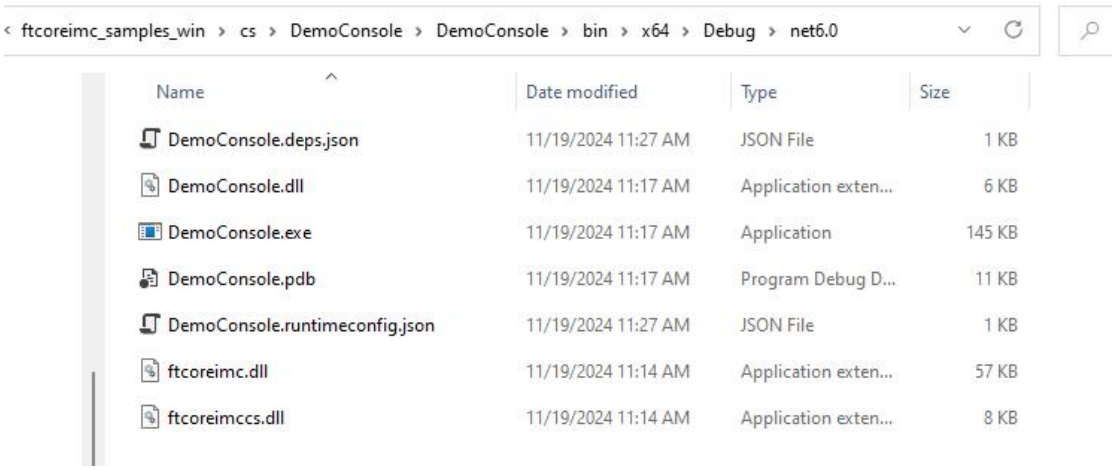


图 6 可执行程序目录结构示例 2（仅复制对应版本的 dll）

ftcoreimc.dll 缺失时将弹出如图 7 所示的错误提示框（Unable to load DLL，0x8007007E）。

ftcoreimc.dll 版本错误时将弹出如图 8 所示的错误提示框（incorrect format，0x8007000B）。（版本错误，如应使用 x64 版本，却错误地复制了 x86 版本）

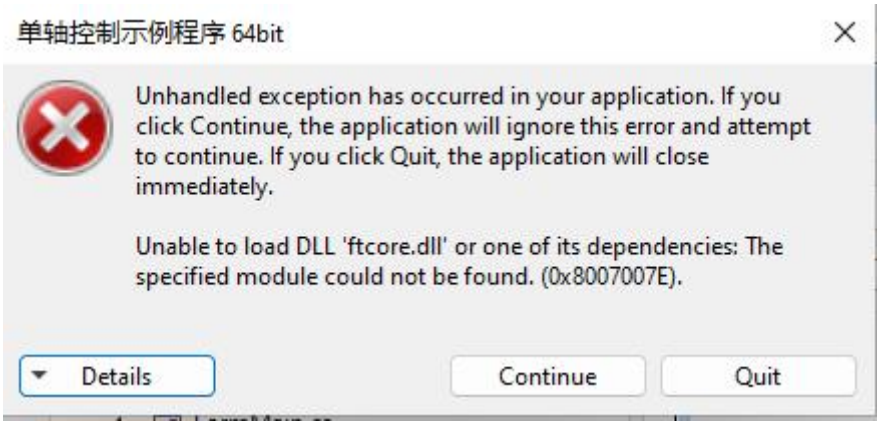


图 7 ftcoreimc.dll 缺失时的弹出信息

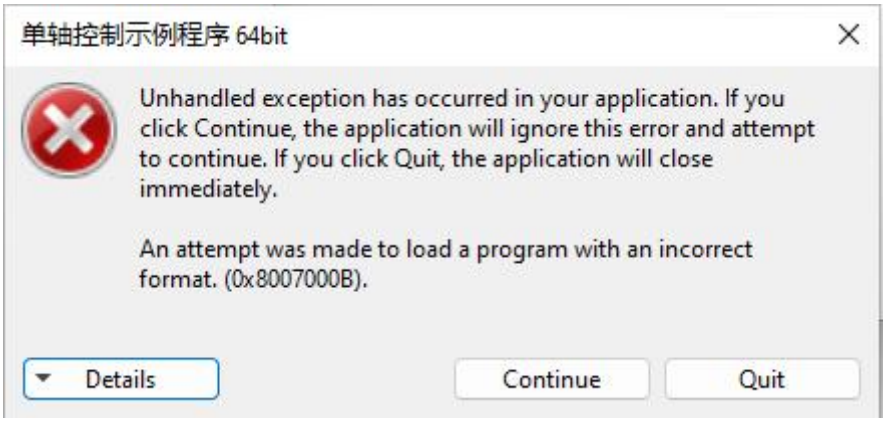


图 8 ftcoreimc.dll 版本错误时的弹出信息

2. 程序基本结构

程序基本结构如下：

（1）串口（Modbus-RTU）或网络（Modbus-TCP）建立连接；

（2）调用相应函数进行参数读取和设置；

（3）进行运动控制，绝对运动、相对运动、点动、停止等；

（4）查询状态，包括当前位置、运行中或停止、限位情况，通常用一个查询线程实现，需间隔一定时间，以避免通信阻塞；

（5）关闭设备连接。

参见 C 语言示例程序 6.1 和 C# 语言示例程序 6.2，以及 samples 中是 DemoConsole。

3. API（Application Program Interface）

3.1. 概述

SDK API 分为几大类，连接和关闭、参数设置、输出口电平设置、状态查询、基本运动控制。

3.2. 连接和关闭

支持 modbus RTU 和 modbus TCP，分别调用 fti_open_com 和 fti_open_tcp。

断开调用 fti_close 函数。

3.3. 参数设置

参数设置包括两类，一类是可以根据需要设置的，包括加减速系数、运行速度、搜零速度、软限位上下限位置以及软限位使能控制，另一类是位移台相关的，包括电机螺距、电机细分、行程等参数，需要根据位移台型号进行设定。

位移台基本参数：

（1）加速参数：fti_set_accel / fti_get_accel；

（2）减速参数：fti_set_decel / fti_get_decel；

（3）细分：fti_set_div / fti_get_div；

（4）螺距：fti_set_pitch / fti_get_pitch；

特别说明 1：上述位移台参数需按照位移台的具体型号，不得随意设置，否则可能运行不正常甚至导致硬件损坏。

特别说明 2：上述参数修改后，需调用参数保存函数（fti_save_params_permanently）方能断电保存。

软限位控制：

（1）软限位下限：fti_set_sw_p1 / fti_get_sw_p1；

（2）软限位上限：fti_set_sw_p2 / fti_get_sw_p2；

运行速度控制（参照位移台的最大速度）：

（1）运行速度：fti_set_vel / fti_get_vel。

3.4. 状态查询

状态查询包括当前位置、电机运行状态（停止或运行中）、限位触发状态（负限位是否触发、正限位是否触发），以及电机运行方向（向右或向左），单轴。

当前位置查询调用 fti_single_getpos 函数。

电机运行状态查询调用 fti_single_isrunning 函数。

限位触发状态查询调用 fti_single_getlimits 函数。

3.5. 基本运动控制

基本运动包括绝对移动、相对移动、连续左行、连续右行，以及停止运动、坐标清零、搜零和中止搜零。

绝对移动调用 fti_single_moveabs 函数。

相对移动调用 fti_single_move 函数。

连续左行调用 fti_single_jogleft 函数。

连续右行调用 fti_single_jogright 函数。

停止运动调用 fti_single_stop 函数。

坐标清零调用 fti_single_zero 函数。

搜零调用 fti_single_home 函数。

中止搜零调用 fti_single_aborthome 函数。

4. 函数说明

数据类型对应关系如下：

	.NET	LabVIEW	Matlab	
整型	int	I32	int32	32bit
无符号整型	uint	U32	uint32	32bit
无符号短整型	ushort	U16	uint6	16bit
浮点数	float			32bit
布尔	bool	Boolean	logical	1bit
字节	byte	U8	uint8	8bit，无符号
字符串	string	abc	string	

参考 MATLAB 官方文档[2][3]，MATLAB 会自动与 .NET 的数据类型进行转换，转换关系如上。Matlab 中默认情况以双精度浮点数形式（double）存储数据，需要使用相应函数转换成整型，包括数组的转换。如 .NET 中用 byte 表示控制器地址，MATLAB 中用 uint8(204)，将 0xCC(204) 转换为 byte 数据类型，然后调用相应函数。

LabVIEW 中，Numeric control 默认为 DBL（double）型，将 Representation 修改为相应的整型，如 I32、U32、U16、U8 等，或使用相应的转换函数 To Unsigned Long Integer(U32)。

轴号 Axis：轴号用"01"、"02"表示，轴号即为驱动器对应地址。

常数定义如下：

```
// 注： bool 类型用 byte 实现，用宏定义 FT_TRUE / FT_FALSE
#define FT_TRUE 1
#define FT_FALSE 0

// 用于 IO 控制和输入口电平判断，高电平 HIGH 或 低电平 LOW
#define FT_IO_HIGH 1
#define FT_IO_LOW 0

// 控制器类型，用于连接设备
#define CONTROLLER_TYPE_SMC 0
#define CONTROLLER_TYPE_NANO 2
```

[2] https://ww2.mathworks.cn/help/matlab/matlab_external/handling-net-data-in-matlab.html
[3] https://ww2.mathworks.cn/help/matlab/matlab_external/passing-net-data-in-matlab.html

```
#define CONTROLLER_TYPE_MINI04 3
#define CONTROLLER_TYPE_AMC 0   // 弃用


#define FT_SUCCESS 0
#define FT_ERR_INVALID_HANDLE 0x8001
#define FT_ERR_INVALID_FEATURE 0x8002
#define FT_ERR_NOTIMPLEMENTED 0x8003
#define FT_ERR_INVALID_AXIS 0x8004
```

4.1. 数据类型

句柄 FT_H、字节 byte 和短整型 ushort 为自定义数据类型，byte 主要用来表示布尔型 FT_TRUE/FT_FALSE，其定义如下：

```
// 64bit，用作句柄
typedef unsigned long long FT_H;

typedef unsigned char byte;
typedef unsigned short ushort;
```

4.2. 获取 SDK 版本号

4.2.1 fti_getsdkversion

```
const char* fti_getsdkversion()
```

描述

获取 sdk 版本号，如“1.1.0.0”。

参数

无

返回值

返回版本号字符串，如“1.1.0.0”

4.3. 连接和关闭

4.3.1 fti_open_com

```
int fti_open_com(const char* device, int baud, byte limit_isnegative,
                 FT_H* handle)
```

描述

连接串口（Modbus-RTU），初始化，返回句柄。

参数

device	字符串	串口设备标识,windows 系统下为“COM1”、“COM2”、“COM10”等，Linux 下为“/dev/ttyS0”、“/dev/ttyUSB0”等。
baud	整型	串口通信波特率，常用 9600、19200、38400、115200
limit_isnegative	字节/布尔	限位的电平逻辑： FT_TRUE：负逻辑（即低电平表示触发）。 FT_FALSE：正逻辑（即高电平表示触发）。
handle	FT_H*	**[返回]** 句柄

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.3.2 fti_open_tcp

```
int fti_open_tcp(const char* ip, int port, byte limit_isnegative,
                 FT_H* handle)
```

描述

连接网络（Modbus-TCP），初始化，返回句柄。

参数

ip	字符串	IP 地址，如“192.168.1.120”。
port	整型	端口号，如 10001
limit_isnegative	字节/布尔	限位的电平逻辑： FT_TRUE：负逻辑（即低电平表示触发）。 FT_FALSE：正逻辑（即高电平表示触发）。
handle	FT_H*	**[返回]** 句柄

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.3.3 fti_close

```
int fti_close(FT_H handle)
```

描述

关闭连接

参数

handle	FT_H	句柄

返回值

成功返回 **FT_SUCESS（0）**，失败返回错误码。

4.4. 设备适配（仅级联使用时需要）

4.4.1 fti_change_addr

```
int fti_change_addr(FT_H handle, const char* axis, ushort value)
```

描述

修改驱动器地址。出厂默认地址为 01。

特别说明：（1）请谨慎使用，用该函数修改驱动器地址后，后续的通信均需要使用新地址，务必记录好新地址。

（2）实际地址=基地址+拨码地址-1，该函数将根据电机型号自动转换（不同电机的拨码地址不同），确保实际地址为欲设置的驱动器地址。

（3）底层通过设置基地址实现，会查询电机型号。

参数

handle	FT_H	句柄
axis	字符串	轴号，用“01”、“02”、“03”等地址表示。
value	无符号短整型	驱动器基地址

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.5. 参数配置

4.5.1 fti_save_params_permanently

```
int fti_save_params_permanently(FT_H handle, const char* axis)
```

描述

永久保存当前参数，断电重启后保持。

特别说明：（1）带记忆寄存器的擦写寿命是 10 万次，请谨慎使用。建议不使用“自动保存参数”类似功能。

（2）部分参数需要调用该函数才能断电保存，否则断电后丢失，如螺距、闭环中的 PID 参数等。

（3）调用该函数时，所有带记忆寄存器都会保存。在未断电情况下，用户可以一次性把需要保存的参数设置好，再调用该函数。

参数

handle	FT_H	句柄
axis	字符串	轴号，用“01”、“02”、“03”等地址表示。

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.5.2 fti_set_accel

```
int fti_set_accel(FT_H handle, const char* axis, ushort value)
```

描述

设置加速系数。

参数

handle	FT_H	句柄
axis	字符串	轴号，用“01”、“02”、“03”等地址表示。
value	短整型	加速参数

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.5.3 fti_get_accel

```
int fti_get_accel(FT_H handle, const char* axis, ushort* value)
```

描述

获取加速系数。

参数

handle	FT_H	句柄
axis	字符串	轴号，用“01”、“02”、“03”等地址表示。
value	短整型	**[返回]** 加速参数

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.5.4 fti_set_decel

```
int fti_set_decel(FT_H handle, const char* axis, ushort value)
```

描述

设置减速系数。

参数

handle	FT_H	句柄

axis	字符串	轴号，用“01”、“02”、“03”等地址表示。
value	短整型	减速参数

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.5.5 fti_get_decel

```
int fti_get_decel(FT_H handle, const char* axis, ushort* value)
```

描述

获取减速系数。

参数

handle	FT_H	句柄
axis	字符串	轴号，用“01”、“02”、“03”等地址表示。
value	短整型	**[返回]** 减速参数

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.5.6 fti_set_div

```
int fti_set_div(FT_H handle, const char* axis, int value)
```

描述

设置细分。

特别说明：对于步进电机，细分是指电机旋转一周所需的脉冲数。

参数

handle	FT_H	句柄
axis	字符串	轴号，用“01”、“02”、“03”等地址表示。
value	整型	细分，如 1600

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.5.7 fti_get_div

```
int fti_get_div(FT_H handle, const char* axis, int* value)
```

描述

获取细分。

特别说明：对于步进电机，细分是指电机旋转一周所需的脉冲数。

参数

handle	FT_H	句柄
axis	字符串	轴号，用"01"、"02"、"03"等地址表示。
value	整型	**[返回]** 细分，如 1600

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.5.8 fti_set_pitch

```
int fti_set_pitch(FT_H handle, const char* axis, float value)
```

描述

设置螺距。

特别说明：对于平移台，螺距是指电机旋转一周实际平移的距离。

参数

handle	FT_H	句柄
axis	字符串	轴号，用"01"、"02"、"03"等地址表示。
value	浮点型	螺距，单位 mm

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.5.9 fti_get_pitch

```
int fti_get_pitch(FT_H handle, const char* axis, float* value)
```

描述

获取螺距。

特别说明：对于平移台，螺距是指电机旋转一周实际平移的距离。

参数

handle	FT_H	句柄
axis	字符串	轴号，用"01"、"02"、"03"等地址表示。
value	浮点型	**[返回]** 螺距，单位 mm

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.5.10 fti_set_vel

```
int fti_set_vel(FT_H handle, const char* axis, float value)
```

描述

设置运行速度。

参数

handle	FT_H	句柄
axis	字符串	轴号，用“01”、“02”、“03”等地址表示。
value	浮点型	运行速度，单位 mm/s

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.5.11 fti_get_vel

```
int fti_get_vel(FT_H handle, const char* axis, float* value)
```

描述

获取运行速度。

参数

handle	FT_H	句柄
axis	字符串	轴号，用“01”、“02”、“03”等地址表示。
value	浮点型	**[返回]** 运行速度，单位 mm/s

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.5.12 fti_set_sw_p1

```
int fti_set_sw_p1(FT_H handle, const char* axis, float value)
```

描述

设置软限位下限。

参数

handle	FT_H	句柄
axis	字符串	轴号，用“01”、“02”、“03”等地址表示。
value	浮点型	软限位下限，单位 mm

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.5.13 fti_get_sw_p1

```
int fti_get_sw_p1(FT_H handle, const char* axis, float* value)
```

描述

获取软限位下限。

参数

handle	FT_H	句柄
axis	字符串	轴号，用“01”、“02”、“03”等地址表示。
value	浮点型	**[返回]** 软限位下限，单位 mm

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.5.14 fti_set_sw_p2

```
int fti_set_sw_p2(FT_H handle, const char* axis, float value)
```

描述

设置软限位上限。

参数

handle	FT_H	句柄
axis	字符串	轴号，用“01”、“02”、“03”等地址表示。
value	浮点型	软限位上限，单位 mm

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.5.15 fti_get_sw_p2

```
int fti_get_sw_p2(FT_H handle, const char* axis, float* value)
```

描述

获取软限位上限。

参数

handle	FT_H	句柄
axis	字符串	轴号，用“01”、“02”、“03”等地址表示。
value	浮点型	**[返回]** 软限位上限，单位 mm

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.6. 状态查询

4.6.1 fti_single_getstatus

```
int fti_single_getstatus(FT_H handle, const char* axis, uint* value)
```

描述

单轴查询状态值。

注：根据状态值，再调用 fti_single_isrunning 解析是否在运动、调用 fti_single_getlimits 解析限位状态。

参数

handle	FT_H	句柄
axis	字符串	轴号，用"01"、"02"、"03"等地址表示。
value	无符号整形	**[返回]** 状态值

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.6.2 fti_single_isrunning

```
int fti_single_isrunning(uint status, byte* value)
```

描述

从状态值中解析是否正在运动。

注：状态值通过调用 fti_single_getstatus 函数查询获取。

参数

status	无符号整形	状态值
value	字节/布尔	**[返回]** 运行中 FT_TRUE，停止 FT_FALSE

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.6.3 fti_single_getlimits

```
int fti_single_getlimits(uint status, byte* values)
```

描述

从状态值中解析限位状态。

注：状态值通过调用 fti_single_getstatus 函数查询获取。

参数

status	无符号整形	状态值
value	字节/布尔数组	**[返回，数组]** 数组长度为 2，依次为负限位、正限位。 ● 触发：FT_TRUE ● 未触发：FT_FALSE

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.6.4 fti_single_getpos

```
int fti_single_getpos(FT_H handle, const char* axis, float* value)
```

描述

单轴获取当前位值（绝对坐标值）。

参数

handle	FT_H	句柄
axis	字符串	轴号，用"01"、"02"、"03"等地址表示。
value	浮点型	**[返回]** 当前位置，平移台的单位 mm，旋转台的单位为度。

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.7. 基本运动控制

特别说明 1：以下运动控制指令 fti_single_move、fti_single_moveabs、fti_single_home、fti_single_jogleft、fti_single_jogright 不会阻塞线程，通过状态判断运行情况；

特别说明 2：取消（中止）搜零必须调用 fti_single_aborthome 函数（该指令会停止轴的运动并终止搜零过程），而不是调用 fti_single_stop 函数。

4.7.1 fti_single_setenabled

```
int fti_single_setenabled(FT_H handle, const char* axis, byte value)
```

描述

单轴使能或失能马达（驱动器）。

参数

handle	FT_H	句柄
axis	字符串	轴号，AMC 的轴号为 XYZUVWAB 之一，如“X”、“Y”等，MINI04 则轴号，用“01”、“02”、“03”等地址表示。
value	字节/布尔	使能 FT_TRUE，失能 FT_FALSE

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.7.2 fti_single_stop

```
int fti_single_stop(FT_H handle, const char* axis)
```

描述

单轴停止运动。

参数

handle	FT_H	句柄
axis	字符串	轴号，用“01”、“02”、“03”等地址表示。

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.7.3 fti_single_zero

```
int fti_single_zero(FT_H handle, const char* axis)
```

描述

单轴将当前位置置零（注：若在运动则会先停止运动）。

参数

handle	FT_H	句柄
axis	字符串	轴号，用“01”、“02”、“03”等地址表示。

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.7.4 fti_single_home

```
int fti_single_home(FT_H handle, const char* axis)
```

描述

单轴搜零（注：触发内置搜零程序）。

参数

handle	FT_H	句柄
axis	字符串	轴号，AMC 的轴号为 XYZUVWAB 之一，如“X”、“Y”等，MINI04 则轴号，用“01”、“02”、“03”等地址表示。

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.7.5 fti_single_aborthome

```
int fti_single_aborthome(FT_H handle, const char* axis)
```

描述

单轴终止搜零。

参数

handle	FT_H	句柄
axis	字符串	轴号，AMC 的轴号为 XYZUVWAB 之一，如“X”、“Y”等，MINI04 则轴号，用“01”、“02”、“03”等地址表示。

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.7.6 fti_single_move

```
int fti_single_move(FT_H handle, const char* axis, float value)
```

描述

单轴相对运动。

参数

handle	FT_H	句柄
axis	字符串	轴号，用“01”、“02”、“03”等地址表示。
value	浮点型	相对运动距离，单位 mm

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.7.7 fti_single_moveabs

```
int fti_single_moveabs(FT_H handle, const char* axis, float value)
```

描述

单轴绝对运动。

参数

handle	FT_H	句柄
axis	字符串	轴号，用“01”、“02”、“03”等地址表示。
value	浮点型	绝对运动坐标，单位 mm 或度

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.7.8 fti_single_jogleft

```
int fti_single_jogleft(FT_H handle, const char* axis)
```

描述

单轴向左连续运动。

参数

handle	FT_H	句柄
axis	字符串	轴号，用“01”、“02”、“03”等地址表示。

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.7.9 fti_single_jogright

```
int fti_single_jogright(FT_H handle, const char* axis)
```

描述

单轴向右连续运动。

参数

handle	FT_H	句柄

axis	字符串	轴号，用“01”、“02”、“03”等地址表示。

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.8. 通用设置读取函数

特别说明：以下函数时通用函数，谨慎使用，调用时需确保地址、数据类型正确，特别时设置时，否则可能导致控制器异常。

4.8.1 fti_set_uint32

```
int fti_set_uint32(FT_H handle, const char* axis, const int reg_addr,
                   uint32_t value)
```

描述

设置寄存器的值，UINT32 类型。

特别说明：（1）请谨慎使用，确保地址正确、类型正确。

（2）若实际值为负数，则强制类型转换为 uint32 类型。

参数

handle	FT_H	句柄
axis	字符串	轴号，用“01”、“02”、“03”等地址表示。
reg_addr	整型	寄存器地址
value	无符号整型	参数

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.8.2 fti_get_uint32

```
int fti_get_uint32(FT_H handle, const char* axis, const int reg_addr,
                   uint32_t* value)
```

描述

设置寄存器的值，UINT32 类型。

特别说明：（1）请谨慎使用，确保地址正确、类型正确。

（2）若实际值为负数，则强制类型转换为 uint32 类型。

参数

handle	FT_H	句柄
axis	字符串	轴号，用“01”、“02”、“03”等地址表示。
reg_addr	整型	寄存器地址
value	无符号整型	【返回值】参数

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.8.3 fti_set_uint16

```
int fti_set_uint16(FT_H handle, const char* axis, const int reg_addr,
                   uint16_t value)
```

描述

设置寄存器的值，UINT16 类型。

特别说明：（1）请谨慎使用，确保地址正确、类型正确。

（2）若实际值为负数，则强制类型转换为 uint16 类型。

参数

handle	FT_H	句柄
axis	字符串	轴号，用“01”、“02”、“03”等地址表示。
reg_addr	整型	寄存器地址
value	无符号短整型	参数

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

4.8.4 fti_get_uint16

```
int fti_get_uint16(FT_H handle, const char* axis, const int reg_addr,
                   uint16_t* value)
```

描述

设置寄存器的值，UINT16 类型。

特别说明：（1）请谨慎使用，确保地址正确、类型正确。

（2）若实际值为负数，则强制类型转换为 uint16 类型。

参数

handle	FT_H	句柄

axis	字符串	轴号，用“01”、“02”、“03”等地址表示。
reg_addr	整型	寄存器地址
value	无符号短整型	【返回值】参数

返回值

成功返回 FT_SUCESS（0），失败返回错误码。

5. 错误码

138	ETIMEDOUT	Timeout
112345678+1	EMBXILFUN	Illegal function
112345678+2	EMBXILADD	Illegal data address
112345678+3	EMBXILVAL	Illegal data value
112345678+4	EMBXSFAIL	Slave device or server failure
112345678+5	EMBXACK	Acknowledge
112345678+6	EMBXSBUSY	Slave device or server is busy
112345678+7	EMBXNACK	Negative acknowledge
112345678+8	EMBXMEMPAR	Memory parity error
112345678+10	EMBXGPATH	Gateway path unavailable
112345678+11	EMBXGTAR	Target device failed to respond
112345678+12	EMBBADCRC	Invalid CRC
112345678+13	EMBBADDATA	Invalid data
112345678+14	EMBBADEXC	Invalid exception code
112345678+15	EMBMDATA	Too many data
112345678+16	EMBBADSLAVE	Response not from requested slave
0x8001	FT_ERR_INVALID_HANDLE	Invalid Handle 句柄无效，通常为未连接或已关闭
0x8002	FT_ERR_INVALID_FEATURE	Invalid Feature 针对 SMC，地址获取失败时返回
0x8003	FT_ERR_NOTIMPLEMENTED	Method NotImplemented 对应功能暂不支持，如 MINI 类型控制器不支持螺距读取和设置，调用 ft_get_pitch 时将返回该错误。
0x8004	FT_ERR_INVALID_AXIS	轴号错误

6. 示例程序

6.1. C 语言示例（DemoConsole）

```
#include <iostream>
////////////////////////////////////////////////////////////////////////
//  第一步：引入头文件，根据实际路径修改
#include "../../../00deps/ftcoreimc_win_v1.3.0.9/inc/ftcoreimc.h"
// 引入 lib 文件，根据实际路径修改，此处使用 x64 版本
// 请根据工程设置选择 x86 版本或 x64 版本
// 特别说明：需将相应的 dll 文件复制到可执行程序目录下，否则运行时将报错。
#pragma comment(lib,
"../../../00deps/ftcoreimc_win_v1.3.0.9/lib/x64/ftcoreimc.lib")

int main() {
  FT_H fthandle;
  int ret;

  const char* sdkver = fti_getsdkversion();
  printf("SDK Ver: %s\n", sdkver);

  ////////////////////////////////////////////////////////////////////////
  // 第二步：连接控制器。
  //   ft_open_com: 串口方式（MODBUS——RTU）
  //   ft_open_tcp: 网络方式（MODBUS-TCP），通常通过串口服务器实现，
  //                MODBUS 转码模式（非透传模式）。
  // 返回句柄 fthandle，此变量（句柄）是后续通信函数的第一个输入参数。
  ret = fti_open_com("COM1", 19200, true, &fthandle);
  // ret = fti_open_tcp("10.0.2.15", 10001, true, &fthandle);
  printf("ret:%d, fthandle:handle=0x%llX\n", ret, fthandle);

  // 检查返回值，返回值为 FT_SUCESS 表示函数执行成功，否则为相应错误码。
  // 特别说明：一定要检查返回值，以便正确处理错误，所有函数均需检查。
  // 最常见错误是 138（0x8A），表示通信超时，
  // 可能原因有：控制器未加电开机、控制器与电脑未正确连接等。
  //
  // 特别说明 2：ft_open_com/ft_open_tcp 函数仅仅建立连接，并未实际通信，
  //          此函数执行成功并不能判定与控制器的通信连接正常，
  //          可通过读取位置 ft_single_getpos 进行实际通信以判定
  //          与控制器的通信是否正常。
  if (ret != FT_SUCCESS) {
```

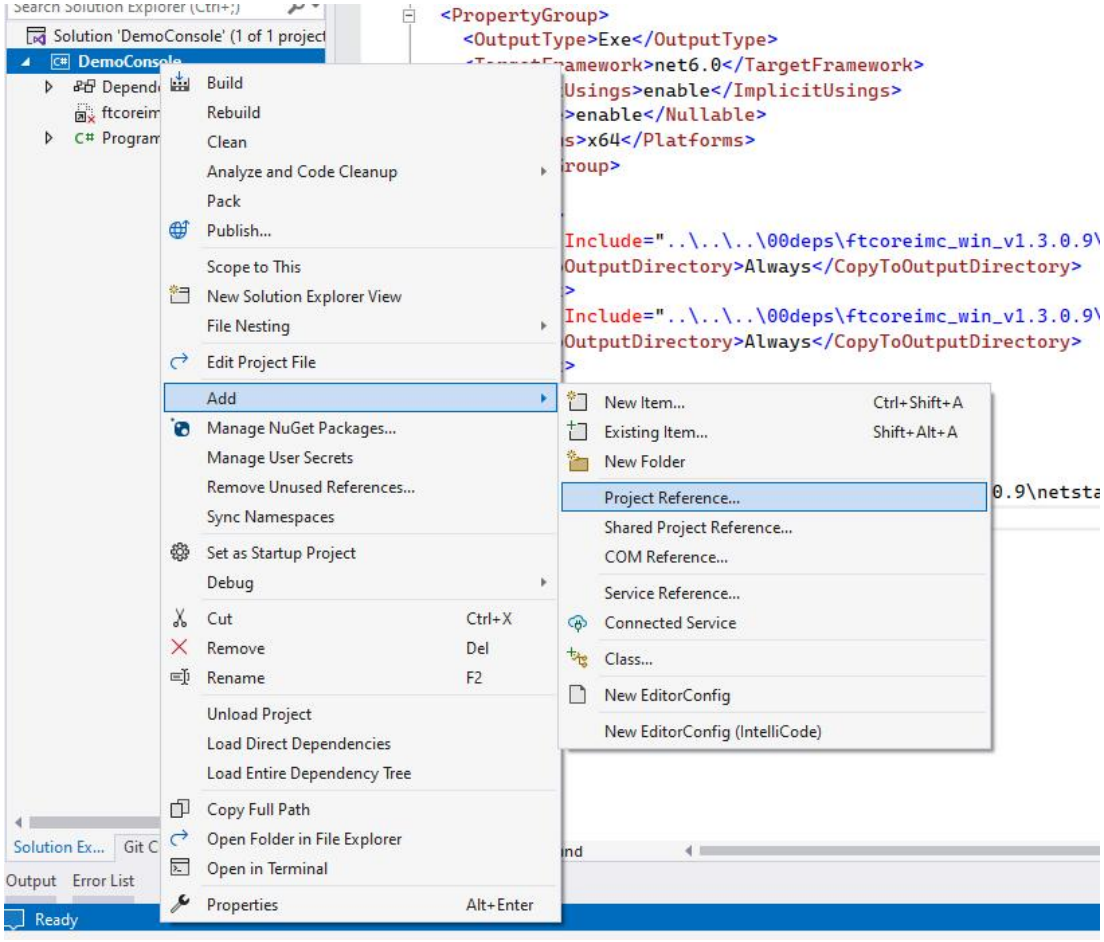
```
    printf("ft_open_com 失败！\n");
    return -1;
  } else {
    printf("ft_open_com 成功！\n");
  }

  ////////////////////////////////////////////////////////////////////////
  // 第三步：与控制器通信，读取/设置轴基本参数、控制、读取状态等。详见 SDK 文档。
  //   特别说明：基本参数通常出厂已配置好，若确需修改，请与厂家联系确认参数的具体值（务必与硬件匹配）。
  //   通常，控制采用主线程，状态查询新建线程。
  //   控制主要有：相对运动、绝对运动、点动、搜零等。
  //   状态包括： 位置、运行状态、限位状态等。
  //
  // 轴号 axis 为字符串："01"、"02"等。
  const char* axis = "X";

  // 读取/设置轴基本参数：螺距、细分、加速系数等。
  // 特别说明：更改螺距、细分、加减速系数等后，需调用"参数保存"函数，否则断电后会被复位。
  printf("\n");
  float pitch;
  int div;
  ushort accel;
  ret = fti_get_pitch(fthandle, axis, &pitch);
  if (ret != FT_SUCCESS) {
    printf("ft_get_pitch 失败。错误码：0x%X\n", ret);
  }
  ret = fti_get_div(fthandle, axis, &div);
  ret = fti_get_accel(fthandle, axis, &accel);
  printf("pitch:%.4f, div:%d, accel:%d\n", pitch, div, accel);

  // 读取状态：位置、运行状态、限位状态等。
  printf("\n");
  float pos;
  uint32_t status;
  byte isrunning;
  byte limits[2];
  ret = fti_single_getpos(fthandle, axis, &pos);
  if (ret != FT_SUCCESS) {
    printf("ft_single_getpos 失败。错误码：0x%X\n", ret);
  }
  ret = fti_single_getstatus(fthandle, axis, &status);
  ret = fti_single_isrunning(status, &isrunning);
  ret = fti_single_getlimits(status, limits);
```

```
// 搜零，搜零后坐标才有实际物理意义。
// 注意：上电后一定要先搜零，并等待搜零完成后再进行其它操作。
printf("\n");
printf("Try to ft_single_home...\n");
ret = fti_single_home(fthandle, axis);
if (ret != FT_SUCCESS) {
  printf("ft_single_home 失败。错误码：0x%lx\n", ret);
}

// 运动控制
// 相对运动 ft_single_move、绝对运动 ft_single_moveabs
printf("\n");
printf("Try to ft_single_move: 1.5f...\n");
ret = fti_single_move(fthandle, axis, 1.5f);
// ret = fti_single_moveabs(fthandle, axis, value);
if (ret != FT_SUCCESS) {
  printf("ft_single_move 失败。错误码：0x%lx\n", ret);
} else {
  //  通常需要等待运动到位，通过 ft_single_isrunning 等状态综合判断
  //  当然，更优的方式是新建一个线程定时查询位置等状态信息。
  printf("Wait until moving is stopped or some error is occured.\n");
  while (true) {
    ret = fti_single_getstatus(fthandle, axis, &status);
    ret = fti_single_isrunning(status, &isrunning);
    if (ret != FT_SUCCESS) {
      printf("ft_single_isrunning 失败。错误码：0x%lx\n", ret);
      break;
    }

    if (isrunning != FT_TRUE) {
      printf("STOPPED\n");
      break;
    }
  }
}

// 再次获取位置
ret = fti_single_getpos(fthandle, axis, &pos);
if (ret != FT_SUCCESS) {
  printf("ft_single_getpos 失败。错误码：0x%lx\n", ret);
} else {
  printf("pos: %.4f", pos);
}
```

```
    ////////////////////////////////////////////////////////////////////////
    // 最后一步：断开连接。
    fti_close(fthandle);

    printf("\n");
    printf("EXIT.\n");
    return 0;
}
```

6.2. C# 语言示例（DemoConsole）

在项目上点击右键，弹出窗口依次选择 Add->Project Reference（VS2022），然后选择动态链接库所在位置，添加。




```
// 第一步：引入依赖。
using ftcoreimccs;
using System.Runtime.InteropServices;
```

```
ulong fthandle;
int ret;

string? sdkver = Marshal.PtrToStringAnsi(ftcoreimc.fti_getsdkversion());
Console.WriteLine(string.Format("SDK Ver: {0}", sdkver));

////////////////////////////////////////////////////////////////////////
// 第二步：连接控制器。
//   fti_open_com: 串口方式（MODBUS——RTU）
//   fti_open_tcp: 网络方式（MODBUS-TCP），通常通过串口服务器实现，
//                 MODBUS 转码模式（非透传模式）。
// 返回句柄 fthandle，此变量（句柄）是后续通信函数的第一个输入参数。
ret = ftcoreimc.fti_open_com("COM1", 19200, 1, out fthandle);
// ret = ftcoreimc.fti_open_tcp("10.0.2.15", 10001, 1, out fthandle);
Console.WriteLine(string.Format("ret:{0}, fthandle:0x{1:X}", ret,
fthandle));

// 检查返回值，返回值为 FT_SUCESS 表示函数执行成功，否则为相应错误码。
// 特别说明：一定要检查返回值，以便正确处理错误，所有函数均需检查。
// 最常见错误是 138（0x8A），表示通信超时，
// 可能原因有：控制器未加电开机、控制器与电脑未正确连接等。
//
// 特别说明 2：ft_open_com/ft_open_tcp 函数仅仅建立连接，并未实际通信，
//         此函数执行成功并不能判定与控制器的通信连接正常，
//         可通过读取位置 ft_single_getpos 进行实际通信以判定
//         与控制器的通信是否正常。
if (ret != ftcoreimc.FT_SUCESS) {
    Console.WriteLine("fti_open_com 失败！错误码：0x{0:X}", ret);
    return;
} else {
    Console.WriteLine("fti_open_com 成功！");
}

////////////////////////////////////////////////////////////////////////
///////
// 第三步：与控制器通信，读取/设置轴基本参数、控制、读取状态等。详见 SDK 文档。
//   特别说明：基本参数通常出厂已配置好，若确需修改，请与厂家联系确认参数的具体值（务必
与硬件匹配）。
//   通常，控制采用主线程，状态查询新建线程。
//   控制主要有：相对运动、绝对运动、点动、搜零等。
//   状态包括： 位置、运行状态、限位状态等。
//
// 轴号 axis 为字符串："01"、"02"等，驱动器地址作为轴号使用。
// 特别说明：（1）需要级联使用时，用"修改驱动器地址 fti_change_addr"函数。
```

```
//          （2）修改地址后，需要使用新地址作为轴号进行通信。
// ret = ftcoreimc.fti_change_addr(fthandle, axis, value: 2);

string axis = "01";
uint status;
byte isrunning;

// 读取/设置轴基本参数：螺距、细分、加速系数等。
// 特别说明：更改螺距、细分、加减速系数等后，需调用"参数保存"函数，否则断电后会被复位。
Console.WriteLine();
ret = ftcoreimc.fti_get_pitch(fthandle, axis, out float pitch);
if (ret != ftcoreimc.FT_SUCESS) {
    Console.WriteLine(string.Format("fti_get_pitch 失败。错误码：0x{0:X}",
ret));
}
ret = ftcoreimc.fti_get_div(fthandle, axis, out int div);
ret = ftcoreimc.fti_get_accel(fthandle, axis, out ushort accel);
Console.WriteLine(string.Format("pitch:{0:f4}, div:{1}, accel:{2}",
    pitch, div, accel));

// 注意：参数保存函数。
//      所有参数设置好后，再调用该函数一次性保存。
// ret = ftcoreimc.fti_save_params_permanently(fthandle, axis);

// 读取状态：位置、运行状态、限位状态等。
Console.WriteLine();
ret = ftcoreimc.fti_single_getpos(fthandle, axis, out float value);
if (ret != ftcoreimc.FT_SUCESS) {
    Console.WriteLine(string.Format("fti_single_getpos 失败。错误码：0x{0:X}",
ret));
}
ret = ftcoreimc.fti_single_getstatus(fthandle, axis, out status);
ret = ftcoreimc.fti_single_isrunning(status, out isrunning);
byte[] limits = new byte[2];
ret = ftcoreimc.fti_single_getlimits(status, limits);

// 搜零，搜零后坐标才有实际物理意义。
// 注意：上电后一定要先搜零，并等待搜零完成后再进行其它操作。
Console.WriteLine();
Console.WriteLine("Try to fti_single_home...");
ret = ftcoreimc.fti_single_home(fthandle, axis);
if (ret != ftcoreimc.FT_SUCESS) {
    Console.WriteLine(string.Format("fti_single_home 失败。错误码：0x{0:X}",
ret));
```

```
}

// 运动控制
// 相对运动 fti_single_move、绝对运动 fti_single_moveabs
Console.WriteLine();
Console.WriteLine(string.Format("Try to fti_single_move: 1.5f..."));
ret = ftcoreimc.fti_single_move(fthandle, axis, value: 1.5f);
//ret = ftcore.fti_single_moveabs(fthandle, axis, value);
if (ret != ftcoreimc.FT_SUCESS) {
    Console.WriteLine(string.Format("fti_single_move 失败。错误码：0x{0:X}",
ret));
} else {
    // 通常需要等待运动到位，通过 fti_single_isrunning 等状态综合判断
    // 当然，更优的方式是新建一个线程定时查询位置等状态信息。
    Console.WriteLine("Wait until moving is stopped or some error is
occured.");
    while (true) {
        ret = ftcoreimc.fti_single_getstatus(fthandle, axis, out status);
        ret = ftcoreimc.fti_single_isrunning(status, out isrunning);
        if (ret != ftcoreimc.FT_SUCESS) {
            Console.WriteLine(string.Format("fti_single_isrunning 失败。错误
码：0x{0:X}", ret));
            break;
        }

        if (isrunning != ftcoreimc.FT_TRUE) {
            Console.WriteLine(string.Format("STOPPED"));
            break;
        }
    }
}

// 再次获取位置
ret = ftcoreimc.fti_single_getpos(fthandle, axis, out value);
if (ret != ftcoreimc.FT_SUCESS) {
    Console.WriteLine(string.Format("fti_single_getpos 失败。错误码：0x{0:X}",
ret));
} else {
    Console.WriteLine(string.Format("pos: {0:f4}", value));
}

////////////////////////////////////////////////////////////////////////
// 最后一步：断开连接。
ret = ftcoreimc.fti_close(fthandle);
```

```
Console.WriteLine();
Console.WriteLine("EXIT.");
```

7. MODBUS 协议简介

控制器底层为 MODBUS 协议，可直接用 MODBUS、通过串口进行通信。

串口参数为：8N1（数据位 8、停止位 1、无校验），波特率作为参数，常用 19200bps。

Modbus 是由 Modicon（现为施耐德电气公司的一个品牌）在 1979 年发明的，是全球第一个真正用于工业现场的总线协议。Modbus 协议是应用于电子控制器上的一种通用语言。通过此协议，控制器相互之间、控制器经由网络（例如以太网）和其它设备之间可以通信。它已经成为一通用工业标准。有了它，不同厂商生产的控制设备可以连成工业网络，进行集中监控。

Modbus 是一个请求/应答协议，并且提供功能码规定的服务。

Modbus 协议包括 ASCII、RTU、TCP 等，并没有规定物理层。协议定义了控制器能够认识和使用的消息结构，而不管它们是经过何种网络进行通信的。Modbus 的 ASCII、RTU 协议规定了消息、数据的结构、命令和对答的方式，数据通讯采用 Maser（主站）/Slave（从站）方式，主站发出数据请求消息， 从站接收到正确消息后就可以发送数据到主站以响应请求；主站也可以直接发消息修改从站的数据，实现双向读写。

MODBUS 规定，只有主站具有主动权，从站只能被动的响应，包括回答出错信息。

具体地，使用的是 MODBUS-RTU 协议，控制器处于 MODBUS 的从机地位，上位机软件为 MODBUS 主机。

MODBUS 通用数据帧如下：

地址码	功能码	数据区	错误校验码
8Bits	8Bits	N×8Bits	16Bits

常用功能码如下：

Modbus 功能码	名称	功能	对应的地址类型
01	读线圈状态	读位（读 N 个 Bits）	**0x**
02	读输入离散量	读位	**1x**
03	读多个寄存器	读整型、字符型、状态字、浮点型（读 N 个 Words）	**4x**
04	读输入寄存器	读整型、状态字、浮点型	**3x**
05	写单个线圈	写位（写一个 Bit）	**0x**
06	写单个寄存器	写整型、字符型、状态字、浮点型（写一个 Word）	**4x**
15	写多个线圈	写位（写 N 个 Bits）	**0x**
16	写多个寄存器	写整型、字符型、状态字、浮点型（写 N 个 Words）	**4x**

控制器地址码默认为 “01”，寄存器（地址）详见“”。寄存器为 INT16 型，两个寄存器合并为 INT32，浮点数需按一定精度（倍率）转换为整形存储，如螺距内部按 1000.0 的倍率存储，对应精度为 1 um。

8. 控制器相关信息

8.1. 控制器（IMC）固定地址

控制器（IMC）地址列表（部分）

地址	读写	用途	值说明
0x04-5	读	电机实时位置	INT32，pulses
0x06-7	读	运行及输入口状态寄存器	UINT32， bit0：X0 输入状态，1 为有输入（高电平）。用作负限位输入。 bit1：X1 输入状态，1 为有输入（高电平）。用作正限位输入。 bit8-9：运行状态，00-空闲、01-即将启动、10-即将停止、11-正在运行。
0x21	读写	螺距	UINT16 用作存储螺距，单位为 mm。x1000 转换为整数。如存储 1000 表示螺距为 1.0mm。
0x24-25	读写	细分	UINT32 每转所需脉冲数。
0x66	读写	驱动器基地值	UINT16
0x6E-6F	读写	软件负限位设置	INT32
0x70-71	读写	软件正限位设置	INT32
0x98	读写	加速时间	UINT16 从启动速度到目标速度需要的时间。单位为 ms。
0x99	读写	减速时间	UINT16 从目标速度到停止速度需要的时间。单位为 ms。
0xC8	写	运行指令	UINT16 主要用作停止、点动。
0xC9	写	回原点执行寄存器	UINT16 用作搜零。
0xCE-CF	写	相对坐标移动	INT32
0xD0-D1	写	绝对坐标移动	INT32
0xD2	写	设置当前点击位置	INT32 通常用作置零。
0xD4	写	使能控制	UINT16
0xDC	写	断电保存命令	UINT16

地址	读写	用途	值说明