

SCATECH 软控

PROGRAMMING MANUAL • 编程手册

SCMotion 软卡 SDK 编程手册

用 C / C++ / C# 通过软卡 SDK 动态库驱动 EtherCAT 真机：连接、控制权、全部 SCM_* 指令说明与案例工程

适用软件：SCMotion 4.4.16 及以上

文档版本：V1.0

发布日期：2026-10

PREFACE 前言

关于本手册

本手册面向用软卡 SDK 编写上位机程序的开发人员：.NET（C#）或 C/C++ 程序通过 SDK 连接 SCMotion 主软件，读状态、拿控制权、使能轴、发运动、读写 I/O。工程、总线、从站、轴参数都在主软件里配置，见《用户使用手册》。

指令语义以 SCMotion 内置帮助的「软卡 SDK 指令说明」为准，本手册第 5 章即由它整理而来；两者不一致时以主软件里的帮助为准。每条指令固定为：函数原型 / 参数表 / 返回值 / 权限 / 说明 / 注意。「注意」段是现场踩坑区，写上位机前必须看。

修订记录

版本	日期	说明
V1.0	2026-10	首次发布。依据 SCMotion 4.4.16 内置帮助与案例包 README 整理，新增 SDK 概述、开发环境配置、案例工程说明。

阅读约定

- C 与 .NET 指令完全同名同参：函数名、参数名、参数顺序、常量名、结构体字段名一致，差别只有引哪个库。
- 轴号、从站位置、位号、I/O 全局序号都从 0 开始。界面上显示的「1 轴」调用时是 `axis = 0`。
- 所有指令返回 `int` 错误码，`SCM_OK` = 0。正式程序每一步都要判返回码。

危险 第一次跑真实伺服，速度和距离用保守值，急停在手边。`SCM_AxisMoveVel` 不给终点，持有控制权的进程崩溃后轴仍会继续运动；安全相关场合必须配硬件急停。

CONTENTS 目录

第1章 SDK 概述	8
1.1 软卡 SDK 是什么	8
1.2 标准调用顺序	8
1.3 指令分组	8
第2章 软卡 SDK 上手	10
2.1 谁用它、前提是什么	10
2.2 SDK 包里有什么	10
2.3 连接配置文件	10
2.4 最短路径	11
2.5 最小程序	11
2.6 第一根轴现场步骤	14
2.7 C/C++ 工程设置	14
第3章 开发环境配置	16
3.1 C# / .NET 工程	16
3.2 C# 的第二种接法：P/Invoke 直接调用 C 动态库	16
3.3 WinForms 高 DPI（界面模糊）	17
3.4 C / C++ 工程	17
3.5 运行前检查	18
第4章 软卡设置	19
4.1 从哪里进	19
4.2 页面上有什么	19
4.3 软卡 SDK 服务是什么	19
4.4 控制权	20
4.5 生成 SDK 包	20
4.6 常见问题	20
第5章 软卡 SDK 指令说明	21
5.1 怎么读这一章	21
5.2 入口与命名	21
5.3 通用参数与返回码	21
5.4 正确调用顺序	22
5.5 异步受理与使能前置条件	22
5.6 I/O 点号规则	22
5.7 指令索引	23

5.8 连接与生命周期	24
5.8.1 SCM_Open	24
5.8.2 SCM_OpenProjectConfig	25
5.8.3 SCM_Close	25
5.9 EtherCAT 总线	26
5.9.1 SCM_EtherCATStart	27
5.9.2 SCM_EtherCATStop	28
5.9.3 SCM_EtherCATGetWkc	28
5.9.4 SCM_EtherCATGetSlaveCount	29
5.10 控制权（属主模型）	30
5.10.1 SCM_RequestOwner	30
5.10.2 SCM_ReleaseOwner	30
5.10.3 SCM_RequestMaintenance	31
5.11 轴使能与回零	32
5.11.1 SCM_AxisPower	32
5.11.2 SCM_AxisHome	32
5.11.3 SCM_AxisSetHomeParam	33
5.11.4 SCM_AxisGetHomeParam	34
5.11.5 SCM_AxisSetHomeParams	34
5.11.6 SCM_AxisGetHomeParams	35
5.11.7 SCM_AxisGetHomingState	35
5.11.8 SCM_AxisAbortHoming	36
5.11.9 SCM_AxisGetHomingResult	36
5.12 点位运动	37
5.12.1 SCM_AxisMoveAbs	37
5.12.2 SCM_AxisMoveRel	38
5.12.3 SCM_AxisMoveVel	39
5.12.4 SCM_AxisStop	40
5.12.5 SCM_AxisEStop	40
5.12.6 SCM_AxisHalt	41
5.12.7 SCM_AxisJog	41
5.12.8 SCM_AxisReset	42
5.12.9 SCM_AxisSetPosition	43
5.13 轴状态读取	43
5.13.1 SCM_AxisGetStatus	43
5.13.2 SCM_AxisReadStatus	44
5.13.3 SCM_AxisGetPosition	45

5.13.4	SCM_AxisGetParam	45
5.13.5	SCM_AxisGetVelocity	46
5.14	I/O	47
5.14.1	SCM_WriteOutputBit	47
5.14.2	SCM_ReadInputBit	48
5.14.3	SCM_WriteOutput	48
5.14.4	SCM_ReadInput	49
5.14.5	SCM_WriteOutputWord	49
5.14.6	SCM_ReadInputWord	50
5.14.7	SCM_WriteAnalogOutput	50
5.14.8	SCM_ReadAnalogInput	51
5.14.9	SCM_WriteChannelRaw	51
5.14.10	SCM_ReadChannelRaw	52
5.15	坐标组插补 (3~8 轴)	52
5.15.1	SCM_GroupEnable	53
5.15.2	SCM_GroupSetLimits	54
5.15.3	SCM_GroupConfigAxis	55
5.15.4	SCM_GroupConfigPath	56
5.15.5	SCM_GroupMoveLinearAbs	56
5.15.6	SCM_GroupMoveLinearRel	57
5.15.7	SCM_GroupMoveCircularAbs	58
5.15.8	SCM_GroupMoveCircularRad	59
5.15.9	SCM_GroupMoveCircularRel	60
5.15.10	SCM_GroupSetOverride	61
5.15.11	SCM_GroupFlush	62
5.15.12	SCM_GroupResetPath	62
5.15.13	SCM_GroupStop	63
5.15.14	SCM_GroupGetSpace	63
5.15.15	SCM_GroupGetStatus	64
5.15.16	SCM_GroupDisable	65
5.15.17	SCM_GroupBufIO	65
5.15.18	SCM_GroupBufDelay	66
5.15.19	SCM_GroupBufAnalog	67
5.16	电子齿轮	67
5.16.1	SCM_GearIn	68
5.16.2	SCM_GearOut	69
5.16.3	SCM_GearGetState	70

5.16.4 齿轮状态 SCM_GEAR_STATE (SCM_GearGetState 的返回值)	70
5.17 电子凸轮	71
5.17.1 SCM_CamTableBegin	72
5.17.2 SCM_CamTableAddPoint	72
5.17.3 SCM_CamTableEnd	73
5.17.4 SCM_CamIn	74
5.17.5 SCM_CamOut	75
5.17.6 SCM_CamSwitchTable	75
5.17.7 SCM_CamGetState	76
5.17.8 段曲线 SCM_CAM_CURVE (SCM_CamTableAddPoint 的 curve)	76
5.17.9 凸轮状态 SCM_CAM_STATE (SCM_CamGetState 的返回值)	77
5.17.10 啮合模式 SCM_CAM_ENGAGE_MODE (SCM_CamIn 的 engageMode)	77
5.18 诊断、错误与常量表	77
5.18.1 SCM_GetDiagnostics	77
5.18.2 SCM_GetVersion	78
5.18.3 SCM_GetOpenErrorText	78
5.18.4 SCM_GetLastError	79
5.18.5 SCM_GetLastReason	79
5.18.6 SCM_GetLastMessage	80
5.18.7 SCM_GetErrorText	80
5.18.8 错误码 SCM_RESULT	81
5.18.9 错误处理约定 (适用于全部指令)	81
5.18.10 结构化拒绝原因码 (SCM_GetLastReason 的出参)	82
5.18.11 回零参数号 SCM_HOME_PARAM	82
5.18.12 回零阶段 SCM_HOMING_PHASE (SCM_AxisGetHomingState 的返回值)	83
5.18.13 回零失败原因 SCM_HOMING_RESULT (SCM_AxisGetHomingResult 的返回值)	83
5.18.14 SCM_AXIS_STATUS 字段	84
5.18.15 SCM_DIAGNOSTICS 字段	85
5.18.16 SCM_AXIS_PLC_STATUS 字段 (PLCopen 定义)	85
5.18.17 轴参数号 SCM_AXIS_PARAM (SCM_AxisGetParam 的 paramId)	86
5.19 CNC / G 代码通道	87
5.19.1 SCM_CncOpen	87
5.19.2 SCM_CncClose	88
5.19.3 SCM_CncSetParam	89
5.19.4 SCM_CncGetParam	89
5.19.5 SCM_CncLoadFile	90
5.19.6 SCM_CncLoadText	90

5.19.7	SCM_CncCompile	91
5.19.8	SCM_CncGetErrorCount	91
5.19.9	SCM_CncGetErrorAt	92
5.19.10	SCM_CncCycleStart	92
5.19.11	SCM_CncFeedHold	93
5.19.12	SCM_CncStop	93
5.19.13	SCM_CncReset	94
5.19.14	SCM_CncSetOverride	94
5.19.15	SCM_CncSetSingleBlock	95
5.19.16	SCM_CncGetStatus	95
5.19.17	CNC 通道状态 SCM_CNC_STATE (SCM_CNC_STATUS.state)	96
5.19.18	CNC 通道参数号 SCM_CNC_PARAM (SCM_CncSetParam 的 paramId)	96
5.19.19	CNC 通道状态 SCM_CNC_STATUS 字段	99
5.20	机器人	100
5.20.1	SCM_RobotOpen	101
5.20.2	SCM_RobotClose	101
5.20.3	SCM_RobotSetParam	102
5.20.4	SCM_RobotGetParam	102
5.20.5	SCM_RobotMoveJ	103
5.20.6	SCM_RobotMoveL	104
5.20.7	SCM_RobotJogJoint	105
5.20.8	SCM_RobotJogCart	105
5.20.9	SCM_RobotStop	106
5.20.10	SCM_RobotGetState	106
5.20.11	SCM_RobotTableBegin	107
5.20.12	SCM_RobotTableAddPoint	107
5.20.13	SCM_RobotTableEnd	108
5.20.14	SCM_RobotRunTable	109
5.20.15	机器人参数号 SCM_ROBOT_PARAM_* (SCM_RobotSetParam / SCM_RobotGetParam 的 paramId)	109
5.20.16	机器人状态结构 SCM_ROBOT_STATE (SCM_RobotGetState 的出参)	111
5.20.17	机器人故障码 SCM_ROBOT_FAULT_* (SCM_ROBOT_STATE.faultCode)	111
第6章	软卡排障与版本匹配	112
6.1	排查顺序	112
6.2	连不上	112
6.3	被拒	112
6.4	诊断值不对	113

6.5 轴不动	113
6.6 症状速查	113
6.7 版本与配置匹配	114
第7章 案例工程说明	115
7.1 案例清单	115
7.2 每个案例目录里有什么	116
7.3 运行前提	117
7.4 代码约定	117
7.5 复制一个函数到别的项目要带走什么	117
7.6 常见拒绝原因	118
附录A 案例教程索引	119

第1章 SDK 概述

软卡 SDK 的定位、架构、调用顺序和指令全貌。

1.1 软卡 SDK 是什么

软卡 SDK 是 SCMotion 提供给客户程序的动态库接口。客户程序不直接碰 EtherCAT 过程数据：总线、分布时钟和进入 OP 都由主软件负责，客户程序通过共享内存连到主软件内部的"软卡 SDK 服务"，像调用一块运动控制卡一样调用 `SCM_*` 指令。



图 1-1 软卡 SDK 架构

- 真机运行时主软件必须常驻，并处于「实时运行」、「激活配置参数」进入 OP，客户程序才能拿控制权。
- 离线开发把主软件切到「仿真」：切进去就有完整的离线运行时，客户程序照常连，代码和真机一字不差。
- 「配置调试」下 SDK 的写命令（拿控制权、轴命令、写输出）会被拒，读状态和诊断不受限。
- 同一时刻只有一个程序能下发运动和写输出（属主）；其余连接只能读。

1.2 标准调用顺序



只监控：连上后直接读 `SCM_GetDiagnostics` / `SCM_AxisGetStatus` / `SCM_ReadInputBit`，不调 `SCM_RequestOwner`

图 1-2 SDK 标准调用顺序

1.3 指令分组

分组	条数	典型指令
连接与生命周期	3	<code>SCM_OpenProjectConfig</code> 、 <code>SCM_Close</code>
EtherCAT 总线	4	<code>SCM_EtherCATGetWkc</code> 、 <code>SCM_EtherCATGetSlaveCount</code>

控制权（属主模型）	3	SCM_RequestOwner、SCM_ReleaseOwner
轴使能与回零	9	SCM_AxisPower、SCM_AxisHome
点位运动	9	SCM_AxisMoveAbs、SCM_AxisJog、SCM_AxisStop
轴状态读取	5	SCM_AxisGetStatus、SCM_AxisReadStatus
I/O	10	SCM_WriteOutputBit、SCM_ReadInputBit、模拟量
坐标组插补（3~8 轴）	19	SCM_GroupEnable、SCM_GroupMoveLinearAbs
电子齿轮 / 电子凸轮	3 / 7	SCM_GearIn、SCM_CamIn
诊断、错误与常量表	7	SCM_GetDiagnostics、SCM_GetLastReason
CNC / G 代码通道	16	SCM_CncLoadText、SCM_CncCycleStart
机器人	14	SCM_RobotMoveJ、SCM_RobotMoveL

共 109 条，逐条说明见第 5 章。

第2章 软卡 SDK 上手

给第一次用软卡 SDK 写客户程序的人：拿到包先看什么、怎么连上主软件、怎么把第一根轴安全地跑起来。

2.1 谁用它、前提是什么

软卡 SDK 给客户自己的程序用：.NET 或 C/C++ 程序通过它连主软件，读状态、拿控制权、使能轴、发运动、读写 I/O。工程、总线、从站、轴参数都在主软件里配，客户程序不碰 EtherCAT 过程数据。

- 真机运行时主软件必须常驻。总线、分布时钟和进入 OP 都由主软件负责，客户程序连上来时总线在跑。
- 主软件处于「实时运行」并「激活配置参数」进入 OP，客户程序才能拿控制权。
- 离线开发切到「仿真」：切进去就有完整的离线运行时，客户程序照常连，代码和真机一字不差。组插补、CNC、齿轮/凸轮、机器人在仿真里都能跑。
- 「配置调试」下 SDK 的写命令（拿控制权、轴命令、写输出）会被拒，读状态和诊断不受限。

2.2 SDK 包里有什么

SDK 包由主软件「软卡设置」页的「生成SDK包」生成，见第4章「软卡设置」。包里只有写程序必须的东西：

目录或文件	用途
sd\dotnet\SCmotionSoftCard.Core.dll	.NET 库。C# 工程只引这一个。
sd\dotnet\interp_core.dll	插补规划器。只有用到插补时才需要，放到 exe 同目录。
sd\c\include\scmotion_card.h	C/C++ 头文件。
sd\c\win-x64\SCmotionSoftCard.dll 与 SCmotionSoftCard.lib	C 动态库（x64）和导入库。DLL 随客户程序一起部署。
docs\软卡SDK指令说明.html	指令说明，从生成包的那份主软件帮助里抽出，用浏览器打开。内容与「软卡 SDK 指令说明」一章相同；两者不一致时以主软件里的帮助为准。
README.txt	引哪个库、怎么连、调用顺序。

包里没有示例工程，本章的最小程序就是示例。

2.3 连接配置文件

客户程序不手填轴数、I/O 点数、周期和共享内存名，这些都在连接配置文件里：

- 路径固定 `C:\ProgramData\SoftSC\softcard.runtime.json`。
- 主软件「激活配置参数」进入 OP 或切到「仿真」时自动写出，内容永远是当前工程。
- 客户程序用 `SCM_OpenProjectConfig(路径, clientId, 句柄)` 读它连接。不要复制旧文件，不要手工编辑。
- 文件不存在，多半是主软件还没进 OP，不是路径写错。

- `clientId` 是你程序自己的身份号，多个程序同时连要各不相同。连接配置文件里也有一个 `clientId` 字段，那是主软件自己的，不要照抄。

2.4 最短路径

`SCM_OpenProjectConfig` → `SCM_RequestOwner` → `SCM_AxisPower(轴, 1)` → 轮询 `SCM_AxisGetStatus` 的 `powered` → `SCM_AxisMoveRel` → 轮询到位 → `SCM_AxisPower(轴, 0)` → `SCM_ReleaseOwner` → `SCM_Close`。

- `SCM_RequestOwner` 只带句柄一个参数。拿到控制权就自动武装，可以直接发轴命令，不需要再调 `SCM_EtherCATStart`。
- `SCM_AxisPower` 是异步受理：返回 0 只表示收下了。完成看 `powered`；等待超时不要短于 15 秒。轴没使能就发 `SCM_AxisMove*` 会被直接拒绝。
- `SCM_ReleaseOwner` 归还控制权并解除武装，但不停正在跑的运动。要连运动一起收掉，先 `SCM_AxisStop`，或用 `SCM_EtherCATStop`。
- 轴号从 0 开始，第一根轴是 0。
- 所有指令返回 `int` 错误码，0 是 `SCM_OK`。正式程序每一步都要判返回码。

2.5 最小程序

.NET 与 C 两份做的是同一件事：连接、拿控制权、使能 0 号轴、走 10 个用户单位、下电、归还、关闭。距离和速度只是示例，现场按机械比例和限位改。

注意 危险。第一次跑真实伺服，速度和距离用保守值，急停在手边。先确认整条链路正常，再加大行程。

.NET (引 `SCmotionSoftCard.Core.dll`) :

```
using System;
using System.Text;
using System.Threading;
using SCmotion.SoftCard.Sdk;
using static SCmotion.SoftCard.Sdk.ScMotionApi;

int rc = SCM_OpenProjectConfig(@"C:\ProgramData\SoftSC\softcard.runtime.json", 300, out var card);
if (rc != SCM_OK)
{
    var why = new StringBuilder(1024);
    SCM_GetOpenErrorText(why, 1024);           // 逐项列出双方的轴数、点数、周期
    Console.WriteLine($"连接失败 {rc}: {why}");
    return;
}

try
{
    rc = SCM_RequestOwner(card);                // 只带句柄；成功即自动武装
    if (rc != SCM_OK) { Console.WriteLine($"拿不到控制权 {rc}"); return; }
```

```

SCM_AxisPower(card, 0, 1); // 异步受理, 返回 0 只表示收下了
if (!Wait(() => { SCM_AxisGetStatus(card, 0, out var st); return st.powered && !st.faulted; }, 15000))
{ Console.WriteLine("使能超时"); return; }

SCM_AxisMoveRel(card, 0, 10.0, 5.0, 0, 0, 0); // 走 10 个用户单位, 5 单位/s; 后三个 0 = 用轴页默认
Thread.Sleep(50);
if (!Wait(() => { SCM_AxisReadStatus(card, 0, out var rs); return rs.valid && !rs.discreteMotion; }, 30000))
{ Console.WriteLine("到位超时"); SCM_AxisStop(card, 0); return; }

SCM_AxisGetStatus(card, 0, out var done);
Console.WriteLine($"到位 actualPosition={done.actualPosition}");

SCM_AxisPower(card, 0, 0); // 下电同样是异步受理
Wait(() => { SCM_AxisGetStatus(card, 0, out var st); return !st.powered; }, 15000);
}
finally
{
    SCM_ReleaseOwner(card); // 归还控制权, 不停正在跑的运动
    SCM_Close(card);
}

static bool Wait(Func<bool> done, int timeoutMs)
{
    long t0 = Environment.TickCount64;
    while (!done())
    {
        if (Environment.TickCount64 - t0 > timeoutMs) return false;
        Thread.Sleep(50);
    }
    return true;
}

```

C (引 `scmotion_card.h`, 链接 `SCmotionSoftCard.lib`):

```

#include <stdio.h>
#include <windows.h>
#include "scmotion_card.h"

/* 轮询 powered 到指定值, 超时返回 0 */
static int wait_powered(SCM_CARD_HANDLE card, int axis, int want, int timeoutMs)
{
    SCM_AXIS_STATUS st;
    DWORD t0 = GetTickCount();
    for (;;) {
        SCM_AxisGetStatus(card, axis, &st);
        if (want ? (st.powered && !st.faulted) : !st.powered) return 1;
        if (GetTickCount() - t0 > (DWORD)timeoutMs) return 0;
        Sleep(50);
    }
}

```

```

/* 轮询定位运动结束 (discreteMotion 落下), 超时返回 0 */
static int wait_in_position(SCM_CARD_HANDLE card, int axis, int timeoutMs)
{
    SCM_AXIS_PLC_STATUS rs;
    DWORD t0 = GetTickCount();
    Sleep(50);
    for (;;) {
        SCM_AxisReadStatus(card, axis, &rs);
        if (rs.valid && !rs.discreteMotion) return 1;
        if (GetTickCount() - t0 > (DWORD)timeoutMs) return 0;
        Sleep(50);
    }
}

int main(void)
{
    SCM_CARD_HANDLE card = 0;
    SCM_AXIS_STATUS st;
    char why[1024];
    int rc;

    rc = SCM_OpenProjectConfig("C:\\ProgramData\\SoftSC\\softcard.runtime.json", 300, &card);
    if (rc != SCM_OK) {
        SCM_GetOpenErrorText(why, sizeof why); /* 逐项列出双方的轴数、点数、周期 */
        printf("open failed %d: %s\\n", rc, why);
        return rc;
    }

    rc = SCM_RequestOwner(card); /* 只带句柄; 成功即自动武装 */
    if (rc != SCM_OK) {
        printf("request owner failed %d\\n", rc);
        SCM_Close(card);
        return rc;
    }

    SCM_AxisPower(card, 0, 1); /* 异步受理, 返回 0 只表示收下了 */
    if (!wait_powered(card, 0, 1, 15000)) { printf("power on timeout\\n"); goto done; }

    SCM_AxisMoveRel(card, 0, 10.0, 5.0, 0, 0, 0); /* 走 10 个用户单位, 5 单位/s; 后三个 0 = 用轴页默认 */
    if (!wait_in_position(card, 0, 30000)) { printf("move timeout\\n"); SCM_AxisStop(card, 0); goto done; }

    SCM_AxisGetStatus(card, 0, &st);
    printf("in position, actualPosition=%.3f\\n", st.actualPosition);

    SCM_AxisPower(card, 0, 0); /* 下电同样是异步受理 */
    wait_powered(card, 0, 0, 15000);

done:
    SCM_ReleaseOwner(card); /* 归还控制权, 不停正在跑的运动 */
    SCM_Close(card);
}

```

```
return 0;
}
```

说明 提示。连接失败时只看错误码不够。`SCM_GetOpenErrorText` 会逐项列出双方的轴数、I/O 点数、周期，照着对就知道哪里不一致。它是全表唯一不带句柄的指令，因为连接失败时还没有句柄。

2.6 第一根轴现场步骤

按下面的顺序走，每一步看到预期结果再进下一步。

1. 运动前检查：急停回路有效；正负限位、软限位和运动方向在主软件轴页低速点动确认过；驱动器无故障。
2. 只读诊断：连上后先 `SCM_GetDiagnostics`，不拿控制权。`cycleCounter` 要持续增长，`workCounter` 稳定，`missedCycleCount` 不持续增长。
3. 读轴状态：`SCM_AxisGetStatus(句柄, 0, 状态)`。`faulted` 为真先记下 `statusWord` 和 `errorCode`，去主软件排故障；`actualPosition` 一直是 0 或不变化，先查反馈 PDO 映射。
4. 拿控制权：`SCM_RequestOwner`。返回 `SCM_REJECTED` 说明控制权在别的程序手上；返回 `SCM_INVALID_STATE` 说明主软件没进 OP 或当前模式不允许下发。失败就停在这里，不要往下发命令。
5. 使能：`SCM_AxisPower(句柄, 0, 1)`，轮询 `powered` 为真且 `faulted` 为假。
6. 小距离相对运动：`SCM_AxisMoveRel`，距离和速度取保守值，后三个参数传 0 用轴页「动态」默认值。看 `actualPosition` 按预期方向变化；方向反了，停下来回主软件轴页和驱动器参数查方向，不要在程序里用取反来弥补。
7. 收尾：还在动就 `SCM_AxisStop` → `SCM_AxisPower(句柄, 0, 0)` → `SCM_ReleaseOwner` → `SCM_Close`。程序异常退出而未归还控制权时，主软件约 5 秒后判定其失联，下一次 `SCM_RequestOwner` 自动收回。

跑不动、连不上、被拒，按第6章「软卡排障与版本匹配」的顺序查。

2.7 C/C++ 工程设置

Visual Studio 工程按下表配。SDK 包里没有 `sdk\c\win-x64\SCmotionSoftCard.dll` 时，让生成包的人补齐后重新生成一份，不要在工程里硬配不存在的库。

设置项	值
平台	x64。不要用 Win32。
C/C++ → 附加包含目录	SDK 包的 <code>sdk\c\include</code>
链接器 → 附加库目录	SDK 包的 <code>sdk\c\win-x64</code>
链接器 → 附加依赖项	<code>SCmotionSoftCard.lib</code>
运行目录	<code>SCmotionSoftCard.dll</code> 放 exe 同目录；用到插补再放 <code>interp_core.dll</code> 。

运行前检查：

- 编译找不到头文件：包含目录没配。

- 链接找不到库：库目录没配，或包里没有 `.lib`。
- 启动报找不到 DLL：exe 同目录没有 `SCmotionSoftCard.dll`。
- 结构体字段读数离谱：头文件和 DLL 不是同一个包里的。升级 SDK 后清掉旧头文件和旧库，重新编译。
- 每次调用都判返回码，非 0 就把返回码、`SCM_GetLastReason` 的原因码和诊断值一起写进日志。

第3章 开发环境配置

C#、C/C++、Qt 工程怎么引用 SDK，以及常见的编译、运行问题。

3.1 C# / .NET 工程

案例工程使用 .NET 8 WinForms，`.csproj` 关键配置如下，新建工程照抄即可：

```
<PropertyGroup>
  <OutputType>WinExe</OutputType>
  <TargetFramework>net8.0-windows</TargetFramework>
  <UseWindowsForms>true</UseWindowsForms>
  <PlatformTarget>x64</PlatformTarget>          <!-- 必须 x64 -->
  <Prefer32Bit>false</Prefer32Bit>
  <ApplicationHighDpiMode>PerMonitorV2</ApplicationHighDpiMode>
</PropertyGroup>
<ItemGroup>
  <Reference Include="SCmotionSoftCard.Core">
    <HintPath>sdk\SCmotionSoftCard.Core.dll</HintPath>
    <Private>true</Private>
  </Reference>
  <None Include="sdk\interp_core.dll">          <!-- 用到插补时需要，放 exe 同目录 -->
    <Link>interp_core.dll</Link>
    <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
  </None>
</ItemGroup>
```

代码里写 `using SCmotion.SoftCard.Sdk;`，之后用 `ScMotionApi.SCM_AxisPower(...)`；或写 `using static SCmotion.SoftCard.Sdk.ScMotionApi;` 直接 `SCM_AxisPower(handle, 0, 1)`，和 C 一样。出参在 .NET 侧是 `out` 参数，字符串缓冲是 `StringBuilder`。

3.2 C# 的第二种接法：P/Invoke 直接调用 C 动态库

C# WPF 版案例不引用 `SCmotionSoftCard.Core.dll`，而是用 `DllImport` 直接调用 C 动态库 `SCmotionSoftCard.dll`。每个案例目录带一个 `ScmSdk.cs`，只声明本案例用到的指令，拷走即可用。声明示例：

```
[DllImport("SCmotionSoftCard.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int SCM_OpenProjectConfig([MarshalAs(UnmanagedType.LPArray)] byte[] configPath, int
clientId, out IntPtr handle);

[DllImport("SCmotionSoftCard.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int SCM_AxisPower(IntPtr handle, int axis, int enable);

// 结构体字段顺序与 C 头文件一致；C 侧 int 型的 powered / faulted 在这里也是 int
[StructLayout(LayoutKind.Sequential)]
public struct SCM_AXIS_STATUS { public int axis, state, powered, faulted, homed;
```

```
public double actualPosition, targetPosition, actualVelocity, followingError;
public ushort statusWord; public int errorCode; public double commandVelocity; }
```

P/Invoke 三条铁律	原因
CallingConvention = CallingConvention.Cdecl 必须写	不写默认 StdCall，调用约定对不上。
句柄用 IntPtr 接，不能用 int	x64 下指针 8 字节，int 只有 4 字节，高位被截断。
传字符串走 byte[] + UTF-8	C 侧是 UTF-8，.NET 默认封送在中文 Windows 上按 GBK 走，路径含中文会乱码。

平台固定 x64，AnyCPU 会报 BadImageFormatException。SCmotionSoftCard.dll、interp_core.dll 要在 exe 同目录，原生 DLL 不会往上层目录找。

3.3 WinForms 高 DPI（界面模糊）

现象：VS 设计器里清晰，跑起来就糊——进程被当成 DPI-unaware，Windows 对整窗做位图拉伸。在 Program.cs 里显式处理：

```
Application.SetHighDpiMode(HighDpiMode.PerMonitorV2); // 必须在创建任何窗体之前
Application.EnableVisualStyles();
Application.SetCompatibleTextRenderingDefault(false); // false = TextRenderer(GDI) 画字，清晰
Application.Run(new MainForm());
```

仍然模糊时，检查 Windows 有没有给这个 exe 打兼容性标记：exe 右键 → 属性 → 兼容性 → 「更改高 DPI 设置」→ 勾掉「替代高 DPI 缩放行为」。这个标记的优先级高于代码和 manifest。

3.4 C / C++ 工程

设置项	值
平台	x64。不要用 Win32。
C/C++ → 附加包含目录	SDK 包的 sdk\c\include
链接器 → 附加库目录	SDK 包的 sdk\c\win-x64
链接器 → 附加依赖项	SCmotionSoftCard.lib
运行目录	SCmotionSoftCard.dll 放 exe 同目录；用到插补再放 interp_core.dll。

Qt 工程（qmake）以 C++ Qt 版案例的 .pro 为准（Kit: Desktop Qt 6.11.2 MSVC 2022 64-bit）：

```
QT += widgets
CONFIG += c++17 release windows
QMAKE_CXXFLAGS += /utf-8 # 源码里的中文字符串
```

```
INCLUDEPATH += $$PWD/sdk          # scmotion_card.h
LIBS        += -L$$PWD/sdk -lSCmotionSoftCard  # SCmotionSoftCard.Lib

DESTDIR = $$PWD                  # exe 落在本目录, 和 SCmotionSoftCard.dll 放一起
```

路径等字符串传给 SDK 前用 `QString::toUtf8()` 转成 UTF-8；句柄类型 `SCM_CARD_HANDLE` 在头文件里是 `void*`。

3.5 运行前检查

现象	原因
编译找不到头文件	包含目录没配。
链接找不到库	库目录没配，或包里没有 <code>.lib</code> 。
启动报找不到 DLL	exe 同目录没有 <code>SCmotionSoftCard.dll</code> 。
结构体字段读数离谱	头文件和 DLL 不是同一个包里的。升级 SDK 后清掉旧头文件和旧库，重新编译。
连接返回失败、找不到 <code>softcard.runtime.json</code>	主软件还没「激活配置参数」进入 OP，或没切到「仿真」。不是路径写错。
案例 exe 挪到别处就起不来	每个 exe 只认自己那个 dll，apphost 里嵌的是原 dll 名；要挪就重新编译。

第4章 软卡设置

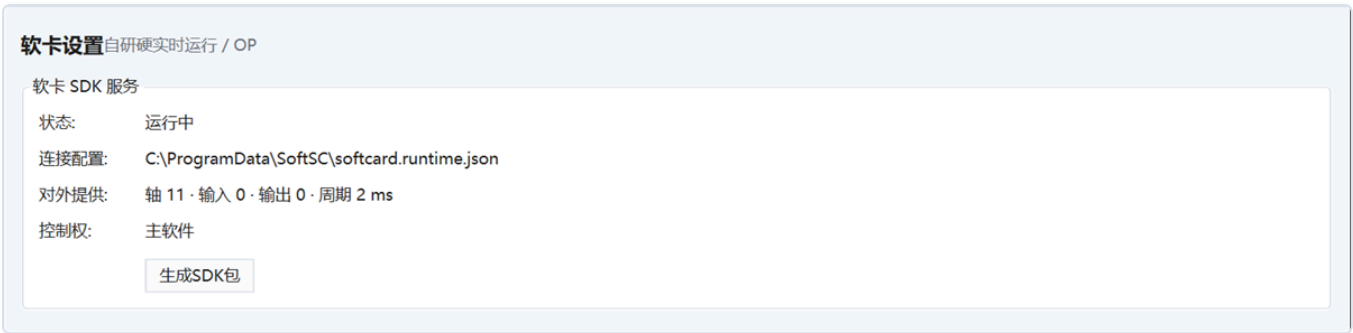
主软件里给软卡 SDK 用的一页：看软卡 SDK 服务的状态和控制权在谁手上、生成给客户的 SDK 包。SDK 指令本身见第5章「软卡 SDK 指令说明」。

4.1 从哪里进

左侧工程树「SYSTEM - 配置」下点「软卡设置」。页面右上角显示当前模式。

4.2 页面上有什么

一个区块「软卡 SDK 服务」，四行状态，外加一个「生成SDK包」按钮。状态在打开这一页时每秒刷新。



软卡设置页：服务状态、连接配置、对外提供、控制权四行，加一个「生成SDK包」按钮

字段	看什么
状态	「运行中」表示客户程序现在就能连上来。「未运行」时后面括号里写了怎么让它跑起来：真机「激活配置参数」进入 OP，或者切到「仿真」模式。
连接配置	连接配置文件的完整路径（ <code>C:\ProgramData\SoftSC\softcard.runtime.json</code> ）。客户程序用 <code>SCM_OpenProjectConfig</code> 读这份文件连上来，不用手填共享内存名与轴数。
对外提供	客户程序连上来会看到的轴数、输入输出点数与周期。这些值以服务启动时的工程为准：之后改了工程（重扫拓扑、改周期、增删轴），要重新「激活配置参数」才会跟着变。
控制权	「主软件」表示没有客户程序持有控制权；客户程序拿走后显示是哪个客户端、以及它最近一次活动是多以前。

4.3 软卡 SDK 服务是什么

它是主软件内部的一块服务：主软件进入 OP（或切到仿真模式）时自动启动，把当前工程的轴与 I/O 通过共享内存交给客户程序使用；退出运行时自动停止。客户程序不需要另外启动任何进程，也不需要导出任何配置文件。

离线开发（手边没有真机）就把主软件切到「仿真」模式，客户程序照样连得上、跑得动，只是不驱动真实伺服。

4.4 控制权

同一时刻只有一个程序能下发运动和写输出，这个程序就是属主。其余连接只能读。

- 客户程序用 `SCM_RequestOwner` 取得控制权，主软件这边的运动与输出下发自动让出；客户程序 `SCM_ReleaseOwner` 交还后恢复。
- 客户程序异常退出没有交还时，服务约 5 秒判定它已经不存在了，下一次 `SCM_RequestOwner` 会自动收回。
- 客户程序判断控制权在不在自己手上，要拿自己的 `clientId` 和 `SCM_GetDiagnostics` 的 `ownerClientId` 比对。

4.5 生成 SDK 包

1. 点「生成SDK包」，选一个输出目录。
2. 目录下生成 `SCmotionSoftCardSdk_工程名_时间` 文件夹，消息栏报「软卡 SDK 包已生成: …」。
3. 有文件缺失时消息栏报一条黄色提示，按提示补齐后重新生成一次。

包里有什么见第2章「软卡 SDK 上手」。包里的指令说明是从生成它的那份主软件帮助里抽出来的，升级主软件后重新生成一份。

不需要另外给客户程序导出任何文件：工程在主软件里配完，「激活配置参数」进入 OP 时连接配置自动写到 `C:\ProgramData\SoftSC\softcard.runtime.json`。

4.6 常见问题

现象	原因	处理
状态显示「未运行」	真机没有「激活配置参数」进入 OP，或者没有切到「仿真」模式。	按括号里的提示做；客户程序这时连接会收到「连不上主软件」。
客户程序报连接参数对不上	服务提供的是激活时的工程配置，之后工程又做了修改。	重新「激活配置参数」，客户程序重新读连接配置文件。
客户程序写输出、发运动被拒	没有取得控制权，或者控制权在别的客户程序手上。	看「控制权」这一行是谁；客户程序侧用 <code>SCM_GetLastReason</code> 读原因码。
生成的包缺 DLL	生成包的这台机器上没有对应文件。	看消息栏的提示，补齐后重新生成。

第5章 软卡 SDK 指令说明

全部 SDK 指令的功能、调用条件、参数、返回值和注意事项。写程序时查这一章；怎么连上、怎么跑第一根轴见第2章「软卡 SDK 上手」。

5.1 怎么读这一章

- 先看下面的「正确调用顺序」，再查具体指令。
- 查指令时重点看「返回值」「权限」「注意」三栏，不要只看函数名。
- 指令表按功能分组，开头有索引；末尾是错误码、拒绝原因码和结构体字段表。

5.2 入口与命名

- C 与 .NET 指令完全同名同参：函数名、参数名、参数顺序、常量名、结构体字段名一致，差别只有引哪个库。C/C++ 引 `scmotion_card.h` 和 `SCmotionSoftCard.dll`，.NET 引 `SCmotionSoftCard.Core.dll`。
- .NET 写 `using static SCmotion.SoftCard.Sdk.ScMotionApi;`，之后直接 `SCM_AxisPower(handle, 0, 1)`，和 C 一样。出参在 .NET 侧是 `out` 参数，字符串缓冲是 `StringBuilder`。
- 所有指令以 `SCM_` 开头，句柄类型 `SCM_CARD_HANDLE`，由 `SCM_OpenProjectConfig` 给出。
- 一个函数只有一种签名，没有默认参数。用不到的加速度、减速度、加加速度显式传 0，表示用轴页默认值。
- 轴号、从站位置、位号、I/O 全局序号都从 0 开始。

5.3 通用参数与返回码

名称	说明
<code>handle</code>	卡句柄。连接成功后所有指令的第一个参数。
<code>axis</code>	轴号，0 基。界面上显示的「1 轴」调用时是 <code>axis = 0</code> 。
<code>clientId</code>	你程序的身份号，连接时给。判断控制权在不在自己手上，拿它和 <code>SCM_DIAGNOSTICS</code> 的 <code>ownerClientId</code> 比。
<code>configPath</code>	连接配置文件路径， <code>C:\ProgramData\SoftSC\softcard.runtime.json</code> 。
返回值	一律 <code>int</code> 错误码， <code>SCM_OK</code> = 0。常见的： <code>SCM_REJECTED</code> 被拒、 <code>SCM_BACKEND_UNAVAILABLE</code> 软卡 SDK 服务不可用、 <code>SCM_INVALID_AXIS</code> 轴号非法、 <code>SCM_INVALID_STATE</code> 当前状态不允许。全表见末尾「错误码」。

- 收到 `SCM_REJECTED` 紧接着读 `SCM_GetLastReason`，按原因码分支：没控制权、控制权在别人手上、轴被别的通道占用、主软件未就绪等。
- 文案用 `SCM_GetErrorText`（查表）或 `SCM_GetLastMessage`（最近一次错误的现场描述）。文案只给人看，程序分支用原因码。
- 读函数失败时出参给 0 或零值并返回原因码，不抛异常、不崩溃。

5.4 正确调用顺序

SCM_OpenProjectConfig → SCM_RequestOwner → SCM_AxisPower → 轮询 powered → SCM_AxisMove* /
SCM_AxisHome → SCM_AxisStop → SCM_ReleaseOwner → SCM_Close。

- 只监控：连上后直接读 SCM_GetDiagnostics、SCM_AxisGetStatus、SCM_ReadInputBit，不调 SCM_RequestOwner。
- 要控制：先 SCM_RequestOwner，成功即自动武装。失败就停，不要继续发写命令。
- SCM_EtherCATStart / SCM_EtherCATStop 是本程序的武装 / 解除武装，不启停真实总线。正常流程用不到 Start；Stop 会拆掉本程序的插补组并停掉在动的轴，但不放弃控制权。
- SCM_ReleaseOwner 归还控制权、解除武装，不停正在跑的运动。退出前先 SCM_AxisStop，再归还，再 SCM_Close。归还后再发命令返回 SCM_REJECTED，重新拿到控制权即恢复。
- 建组（SCM_GroupEnable）和 CNC 前，成员轴都要 powered 为真，再等约 1 秒。建组是同步命令，返回 0 即建成；首次建组本身约 1 秒。
- 使能走完后主软件会收回控制权，所以紧挨着 SCM_GroupEnable 再调一次 SCM_RequestOwner，否则建组会因为不是属主被拒。

5.5 异步受理与使能前置条件

SCM_AxisPower、SCM_AxisReset、SCM_AxisHome 是异步受理：返回 0 只表示主软件已受理，动作还没做完。

- 使能完成看 SCM_AxisGetStatus 的 powered；等待超时不短于 15 秒。
- 复位完成看 faulted 转假，至少要 450 ms。
- 回零完成看 homed 为真，过程看 SCM_AxisGetHomingState。
- 轴没使能、或使能还没走完时发 SCM_AxisMove*、SCM_AxisHome、SCM_GroupEnable 会被直接拒绝，原因码 HostNotReady。如果每次发送都被拒，原因是没有等使能完成，并非设备故障。应先轮询等待完成，不要用被拒后重试来代替等待。
- 其余指令不是异步受理，返回码就是执行结果。

5.6 I/O 点号规则

- 新程序用 SCM_WriteOutputBit(handle, 从站位置, 位, 值) / SCM_ReadInputBit(handle, 从站位置, 位, 值)。从站位置是扫描顺序（0 基），位号是该从站内的输出 / 输入位（0 基），增删别的从站不影响。
- SCM_WriteOutput / SCM_ReadInput 按全局序号（所有点按扫描顺序 → PDO → 位序排成一列）。增删从站会让序号整体错位；没有说明文件的 I/O 模块也按设备报告的通道计入序号，原来跳过它们的序号会整体后移。用了全局序号的老程序，升级前逐个核对。
- 按从站读写一个没映射的点不报错：写了什么都不发生，读回 0 且返回 SCM_OK。接线阶段要到现场确认，不能只看返回码。
- 字量和模拟量通道用 SCM_WriteOutputWord / SCM_ReadInputWord / SCM_WriteAnalogOutput / SCM_ReadAnalogInput，见指令表 I/O 一组。

说明 C 与 .NET 的指令完全同名同参，差别只有引哪个库：C/C++ 引 scmotion_card.h + SCmotionSoftCard.dll，.NET 引 SCmotionSoftCard.Core.dll。

5.7 指令索引

分类	指令
连接与生命周期	SCM_Open 、 SCM_OpenProjectConfig 、 SCM_Close
EtherCAT 总线	SCM_EtherCATStart 、 SCM_EtherCATStop 、 SCM_EtherCATGetWkc 、 SCM_EtherCATGetSlaveCount
控制权（属主模型）	SCM_RequestOwner 、 SCM_ReleaseOwner 、 SCM_RequestMaintenance
轴使能与回零	SCM_AxisPower 、 SCM_AxisHome 、 SCM_AxisSetHomeParam 、 SCM_AxisGetHomeParam 、 SCM_AxisSetHomeParams 、 SCM_AxisGetHomeParams 、 SCM_AxisGetHomingState 、 SCM_AxisAbortHoming 、 SCM_AxisGetHomingResult
点位运动	SCM_AxisMoveAbs 、 SCM_AxisMoveRel 、 SCM_AxisMoveVel 、 SCM_AxisStop 、 SCM_AxisEStop 、 SCM_AxisHalt 、 SCM_AxisJog 、 SCM_AxisReset 、 SCM_AxisSetPosition
轴状态读取	SCM_AxisGetStatus 、 SCM_AxisReadStatus 、 SCM_AxisGetPosition 、 SCM_AxisGetParam 、 SCM_AxisGetVelocity
I/O	SCM_WriteOutputBit 、 SCM_ReadInputBit 、 SCM_WriteOutput 、 SCM_ReadInput 、 SCM_WriteOutputWord 、 SCM_ReadInputWord 、 SCM_WriteAnalogOutput 、 SCM_ReadAnalogInput 、 SCM_WriteChannelRaw 、 SCM_ReadChannelRaw
坐标组插补（3~8 轴）	SCM_GroupEnable 、 SCM_GroupSetLimits 、 SCM_GroupConfigAxis 、 SCM_GroupConfigPath 、 SCM_GroupMoveLinearAbs 、 SCM_GroupMoveLinearRel 、 SCM_GroupMoveCircularAbs 、 SCM_GroupMoveCircularRad 、 SCM_GroupMoveCircularRel 、 SCM_GroupSetOverride 、 SCM_GroupFlush 、 SCM_GroupResetPath 、 SCM_GroupStop 、 SCM_GroupGetSpace 、 SCM_GroupGetStatus 、 SCM_GroupDisable 、 SCM_GroupBufIO 、 SCM_GroupBufDelay 、 SCM_GroupBufAnalog
电子齿轮	SCM_GearIn 、 SCM_GearOut 、 SCM_GearGetState
电子凸轮	SCM_CamTableBegin 、 SCM_CamTableAddPoint 、 SCM_CamTableEnd 、 SCM_CamIn 、 SCM_CamOut 、 SCM_CamSwitchTable 、 SCM_CamGetState
诊断、错误与常量表	SCM_GetDiagnostics 、 SCM_GetVersion 、 SCM_GetOpenErrorText 、 SCM_GetLastError 、 SCM_GetLastReason 、 SCM_GetLastMessage 、 SCM_GetErrorText
CNC / G 代码通道	SCM_CncOpen 、 SCM_CncClose 、 SCM_CncSetParam 、 SCM_CncGetParam 、 SCM_CncLoadFile 、 SCM_CncLoadText 、 SCM_CncCompile 、 SCM_CncGetErrorCount 、 SCM_CncGetErrorAt 、 SCM_CncCycleStart 、 SCM_CncFeedHold 、 SCM_CncStop 、 SCM_CncReset 、 SCM_CncSetOverride 、 SCM_CncSetSingleBlock 、 SCM_CncGetStatus

机器人

SCM_RobotOpen、SCM_RobotClose、SCM_RobotSetParam、SCM_RobotGetParam、
 SCM_RobotMoveJ、SCM_RobotMoveL、SCM_RobotJogJoint、SCM_RobotJogCart、
 SCM_RobotStop、SCM_RobotGetState、SCM_RobotTableBegin、
 SCM_RobotTableAddPoint、SCM_RobotTableEnd、SCM_RobotRunTable

5.8 连接与生命周期

说明 两种打开方式都通过出参返回句柄。推荐 `SCM_OpenProjectConfig` —— 参数从主软件写出的连接契约里读，

无需手工核对双方参数。连接失败时只看返回码是不够的：详细原因要问 `SCM_GetOpenErrorText`。

5.8.1 SCM_Open

```
int SCM_Open(const char* mapName, int axisCount, int inputCount, int outputCount, int cyclePeriodUs, int
clientId, SCM_CARD_HANDLE* handle)
```

功能：按手工指定的共享内存名与规格，连接**已经在跑的**运行时宿主。

参数	类型	方向	说明
<code>mapName</code>	<code>const char*</code>	入	共享内存名。传 <code>NULL</code> 用默认映射名
<code>axisCount</code>	<code>int</code>	入	轴数，须与宿主一致
<code>inputCount</code>	<code>int</code>	入	数字量输入点数，须与宿主一致
<code>outputCount</code>	<code>int</code>	入	数字量输出点数，须与宿主一致
<code>cyclePeriodUs</code>	<code>int</code>	入	周期 (μs)，须与宿主一致
<code>clientId</code>	<code>int</code>	入	本程序的身份号。 多程序场景必须给不同值
<code>handle</code>	<code>SCM_CARD_HANDLE*</code>	出	成功时给回卡句柄；失败时为空

返回值：`SCM_OK` = 已连接；`SCM_BACKEND_UNAVAILABLE` = 宿主没起或共享内存不存在；`SCM_INVALID_ARGUMENT` = 规格与宿主对不上 / 出参指针为空。

权限：无需控制权（连上来默认是 Monitor 角色）。

说明：本函数**不会**代你把宿主拉起来 —— 宿主没在跑就直接返回 `SCM_BACKEND_UNAVAILABLE`。

四个规格值（轴数 / 输入点数 / 输出点数 / 周期）在握手时**逐一校验**，对不上就拒绝连接。手填极易错，正式用法请走 `SCM_OpenProjectConfig`。

`clientId` 是判断“控制权在不在自己手上”的依据：拿它与 `SCM_DIAGNOSTICS.ownerClientId` 比。只看 `controlRole` 会认错人 —— 别的程序持有 Owner 时，你这边读到的角色字段同样可能是 `Owner`。

说明 注意：⚠ 连接失败时**错误码只说"哪一类错"**。共享内存不存在 / 布局指纹不匹配 / 规格与宿主对不上，这三种都可能落到同一个码上 —— 紧接着调 `SCM_GetOpenErrorText`，那条消息里逐项列出双方的轴数、I/O 点数、周期，照着对就能定位。

⚠ 共享内存布局带**版本指纹**，宿主写入、连接时逐字段校验。**SDK 与主软件必须同版本一起升级**：旧版 SDK 连接新版宿主时会在握手阶段被直接拒绝，反之亦然。这样可以避免旧程序错位读取新宿主的状态区，读出看似正常、实则错误的位置与状态。

相关指令： `SCM_OpenProjectConfig`、`SCM_GetOpenErrorText`、`SCM_Close`

5.8.2 SCM_OpenProjectConfig

```
int SCM_OpenProjectConfig(const char* configPath, int clientId, SCM_CARD_HANDLE* handle)
```

功能：按工程连接契约文件连接宿主 —— 轴数 / 点数 / 周期 / 映射名全从文件读。**推荐的正式用法。**

参数	类型	方向	说明
<code>configPath</code>	<code>const char*</code>	入	连接契约文件路径，通常是 <code>C:\ProgramData\SoftSC\softcard.runtime.json</code>
<code>clientId</code>	<code>int</code>	入	本程序的身份号。 多程序场景必须给不同值
<code>handle</code>	<code>SCM_CARD_HANDLE*</code>	出	成功时给回卡句柄；失败时为空

返回值： `SCM_OK` = 已连接；`SCM_INVALID_ARGUMENT` = 契约文件不存在 / 读不通 / 内容非法；
`SCM_BACKEND_UNAVAILABLE` = 宿主没起或共享内存不存在。

权限：无需控制权（连上来默认是 Monitor 角色）。

说明：连接契约由**主软件进 OP 时自动写出**，里面是宿主当前实际使用的映射名、轴数、I/O 点数、周期。用它连接可保证双方参数一致。

契约文件里也有一个 `clientId` 字段，那是主软件自己用的编号；**本函数的 `clientId` 参数是你自己的身份号，请另给一个值，不要照抄契约里的。**

说明 注意：⚠ 找不到契约文件，通常意味着主软件还没"激活运行"进入 OP，而不是路径写错了 —— 先去主软件那边确认状态。

相关指令： `SCM_Open`、`SCM_GetOpenErrorText`、`SCM_Close`

5.8.3 SCM_Close

```
int SCM_Close(SCM_CARD_HANDLE handle)
```

功能：关闭句柄并释放相关资源。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄

返回值：`SCM_OK`。已关闭的句柄再关一次同样返回 `SCM_OK`（幂等）。

权限：无需控制权。

说明：关闭前如果还持有控制权，应先 `SCM_ReleaseOwner` 归还，让主软件及时恢复控制。

未归还就退出，控制权也不会被永久占用：宿主会自动回收失效的控制权（见 `SCM_RequestOwner`），但需要约 5 秒。

.NET 侧 `ScMotionCard` 同时实现 `IDisposable`，`using` 也可以。

说明 注意：⚠ 关闭之后继续调读写指令不会崩溃，但会返回 `SCM_NOT_OPEN` 并给默认值。

相关指令：`SCM_ReleaseOwner`、`SCM_Open`

5.9 EtherCAT 总线

说明 总线归主软件管，本节两条不启停总线——网卡、EtherCAT 总线、分布式时钟(DC)、进入 OP 全由主软件负责，上位程序连上来时总线通常已经在跑。

说明 本节的 `SCM_EtherCATStart` / `SCM_EtherCATStop` 是本程序的武装开关：

说明 || 含义 |

|---|---|

| **武装** | 本程序可以下发轴 / IO / 组命令 |

| **解除武装** | 本程序后续的命令被拒，驱动交还主软件 |

说明 `SCM_RequestOwner` 成功时会自动武装一次，所以最短路径就是

`SCM_Open` → `SCM_RequestOwner` → `SCM_AxisPower`，这两条一般不需要调用。

它们留给一种进阶处境：**不放弃控制权**，但想把驱动交还主软件（`Stop`），过一会儿再拿回来（`Start`）。

说明 ⚠ 别绕开宿主自行开总线（例如在主软件已占用的同一块网卡上另起一个不配 DC 的运行）——缺 DC 会导致伺服进 OP 即 Fault。

说明 ⚠⚠ 上面那条最短路径只到「轴使能」为止——要接着建组 / 跑 CNC，还有两步（只按上面的最短路径编写，会收到 `SCM_REJECTED`）：

说明 | # | 要多做的事 | 不做的后果 |

|---|---|---|

| ① | 只看 **powered** 不够，还要等主软件的使能确认。**SCM_AxisGetStatus** 的 **powered** 置 1 (= 6041 到 Operation Enabled) 之后，主软件 MC 层还要**连续 3 次检测确认**（主软件轴任务约每 50 ms 检测一次）才置位使能标志，而 **SCM_GroupEnable** / **SCM_CncCycleStart** 检查的是**该标志**，两者实际相差约 **73 ms**。因此在 **powered == 1** 后再等待约 **1 秒**（或轮询到建组不再被拒）| 建组返回 **SCM_REJECTED**，看起来像「使能了却建不了组」|

| ② | 建组前重新 **SCM_RequestOwner** 一次。**SCM_AxisPower** 的使能流程完成后，主软件会**收回控制权**（主软件日志：「控制权已收回，主软件恢复控制」）| 建组时已不是属主，同样返回 **SCM_REJECTED** |

说明 完整最短路径：

SCM_Open → **SCM_RequestOwner** → **SCM_AxisPower** (×N) → 轮询 **powered**
→ 宽限约 1 秒 → **SCM_RequestOwner** (再一次) → **SCM_GroupEnable** / **SCM_CncOpen** ...

5.9.1 SCM_EtherCATStart

```
int SCM_EtherCATStart(SCM_CARD_HANDLE handle)
```

功能：武装本程序，让它可以下发轴 / IO / 组命令。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄

返回值：**SCM_OK** = 已武装；**SCM_REJECTED** = 没有控制权(用 **SCM_GetLastReason** 分辨 **NoControl** / **NotOwner**)；**SCM_INVALID_STATE** = 主软件未进 OP，或当前模式不允许下发命令；**SCM_BACKEND_UNAVAILABLE** = 运行时后端不可用；**SCM_TIMEOUT** = 宿主未在命令超时内应答；**SCM_INVALID_ARGUMENT** = **handle** 为空。其余错误码见错误码表。

权限：需 Owner 或 Maintenance。

说明：拿到控制权时已经自动武装过一次，所以正常流程不必调用本条；重复调用是幂等的（已武装再武装直接返回 **SCM_OK**）。

本条**不启动真实总线**：总线、DC、进入 OP 都由主软件负责。它做的是校验主软件确实就绪，然后把驱动交给你这个程序——此后主软件那一侧的运动界面变成只读。

武装不成功的两个原因，都由 **SCM_GetLastMessage** 说清：主软件还没走完「连接 → 扫描 → 进入激活运行 (OP)」，或者当前模式不允许下发命令。

说明 注意：⚠ 武装状态是否真的生效，以 `SCM_GetDiagnostics` 出参的 `busState` 为准（`Started` = 已武装），不要只看本函数的返回码就往下走。

⚠ 本条**不属于异步受理**(异步受理的只有 `SCM_AxisPower` / `SCM_AxisReset` / `SCM_AxisHome`)，返回码给的是宿主执行这条命令的结果，不是"收下了吗"。

⚠ 不要绕开宿主自行开总线 —— 例如在主软件已占用的同一块网卡上另起一个不配 DC 的运行。缺 DC 会导致伺服进 OP 即 Fault。

相关指令： `SCM_EtherCATStop`、`SCM_RequestOwner`、`SCM_AxisPower`、`SCM_GetDiagnostics`

5.9.2 SCM_EtherCATStop

```
int SCM_EtherCATStop(SCM_CARD_HANDLE handle)
```

功能：解除本程序的武装，把驱动交还主软件（**不放弃控制权**）。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄

返回值： `SCM_OK` = 已解除武装；`SCM_REJECTED` = 没有控制权(用 `SCM_GetLastReason` 分辨 `NoControl` / `NotOwner`)；`SCM_INVALID_STATE` = 当前状态不允许；`SCM_BACKEND_UNAVAILABLE` = 运行时后端不可用；`SCM_TIMEOUT` = 宿主未在命令超时内应答；`SCM_INVALID_ARGUMENT` = `handle` 为空。其余错误码见错误码表。

权限：需 Owner 或 Maintenance。

说明：解除武装后状态退回 `Scanned`，控制权仍在你手上，可以再 `SCM_EtherCATStart` 拿回驱动。

本条会先停止本程序的全部运动：拆掉本程序建立的插补组，并停掉本程序驱动着的每一根轴。这样可以防止交还驱动后规划器仍按旧目标继续运动而造成飞车。

这是本条与 `SCM_ReleaseOwner` 的唯一区别：**放弃控制权不停运动**（与常见运动控制卡 Close 不停轴的做法一致），**交还驱动停运动**。想让轴接着跑就用 `SCM_ReleaseOwner`，想让轴停就用本条。

说明 注意：⚠ 正在回零的轴会以 `SCM_HOMING_BUS_STOPPED` 收场。

⚠ 本条**不停真实总线**，主软件那一侧照常运行 —— 它只解除你这个程序的武装。

相关指令： `SCM_EtherCATStart`、`SCM_ReleaseOwner`、`SCM_AxisStop`、`SCM_AxisPower`

5.9.3 SCM_EtherCATGetWkc

```
int SCM_EtherCATGetWkc(SCM_CARD_HANDLE handle, int* wkc)
```

功能：读总线工作计数器(WKC)。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄

wkc	int*	出	工作计数器当前值。失败时给 0
-----	------	---	-----------------

返回值： `SCM_OK` = 读到; `SCM_NOT_OPEN` = 句柄已关闭; `SCM_BACKEND_UNAVAILABLE` = 运行时后端不可用; `SCM_INVALID_ARGUMENT` = `handle` 或 `wkc` 为空。其余错误码见错误码表。

权限：只读，不需要控制权，Monitor 角色即可读。

说明：WKC 是 EtherCAT 报文里被从站累加的工作计数器，用来判断本周期的过程数据交换是否完整 —— 数值稳定即通信正常，逐周期跳变说明有站没应答。

同一个值也在 `SCM_GetDiagnostics` 出参的 `workCounter` 字段里，一次读诊断就能连别的运行时指标一起拿到。

说明 注意：⚠ **WKC 不是从站数量。** 每个从站按其读/写数据区在报文里累加，一个站可以贡献不止 1 —— 例如总线上有 35 个从站时，WKC 读到的是 **105**。因此不能用 WKC 判断拓扑（比如“读到几就是几个站”“值变了就是掉了几个站”）。要站数请用 `SCM_EtherCATGetSlaveCount`。
⚠ 读失败时出参给 0 并返回原因码。0 是失败后的默认值，不要把它单独当作“总线没通”的依据 —— 先看返回码。

相关指令： `SCM_EtherCATGetSlaveCount`、`SCM_GetDiagnostics`

5.9.4 SCM_EtherCATGetSlaveCount

```
int SCM_EtherCATGetSlaveCount(SCM_CARD_HANDLE handle, int* count)
```

功能：读在线从站数(总线上实际扫描到的站数)。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 <code>SCM_Open*</code> 取得的卡句柄
count	int*	出	在线从站数; -1 = 不可得 ，与“真的 0 站”是两回事。失败时给 -1

返回值： `SCM_OK` = 读到(出参可能仍是 -1，见下); `SCM_NOT_OPEN` = 句柄已关闭; `SCM_BACKEND_UNAVAILABLE` = 运行时后端不可用; `SCM_INVALID_ARGUMENT` = `handle` 或 `count` 为空。其余错误码见错误码表。

权限：只读，不需要控制权，Monitor 角色即可读。

说明：这是判断拓扑规模的**唯一正确读法**，与 `SCM_EtherCATGetWkc` 不是一回事。

同一个值也在 `SCM_GetDiagnostics` 出参的 `slaveCount` 字段里，含义完全相同(包括 -1)。

说明 注意：⚠ **-1 = 不可得，不是 0 站。** 宿主版本较旧没有写这个值，或后端本身拿不到站数时，出参就是 -1，此时返回码仍可能是 `SCM_OK`。把 -1 和 0 混在一起判，会把“读不到拓扑”误当成“总线上一个站都没有”。判断应分三种情况：**> 0** 有站 / **== 0** 确实空 / **< 0** 不可得(去查宿主与总线状态)。
⚠ 站数由主软件扫描拓扑后才有意义: 主软件还没走完「连接 → 扫描」时读到的是上一次扫描的结果或 -1。

相关指令： `SCM_EtherCATGetWkc`、`SCM_GetDiagnostics`

5.10 控制权（属主模型）

说明 同一时刻只有一个 Owner。取得控制权后主软件不再下发运动与输出（界面变为只读），归还后主软件恢复控制。

读操作一律不需要控制权。

5.10.1 SCM_RequestOwner

```
int SCM_RequestOwner(SCM_CARD_HANDLE handle)
```

功能：申请主控制权（Owner），取得后才能下发运动与写输出。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄

返回值：`SCM_OK` = 已取得（并已自动武装）；`SCM_REJECTED` = 控制权在别人手上（`SCM_GetLastReason` 给 `NotOwner`）；`SCM_INVALID_STATE` = 自动武装没成功（主软件未进 OP / 当前模式不允许下发），此时控制权也没给你；`SCM_BACKEND_UNAVAILABLE` = 宿主不可用。

权限：本条就是取得权限的入口，任何角色都可调。

说明：取得控制权后，主软件自动停止下发运动与输出（界面变为只读），你的程序独占下发权。

取得控制权 = 同时完成武装：本条成功时会替你走一遍 `SCM_EtherCATStart`，所以拿到控制权就能直接 `SCM_AxisPower`，最短路径是 `SCM_Open` → `SCM_RequestOwner` → `SCM_AxisPower`。

武装不成功就一点控制权都不给（`SCM_INVALID_STATE`）。这样可以避免出现"已取得控制权却无法下发命令"这种难以排查的状态。失败原因读 `SCM_GetLastMessage`，两种：主软件还没走完「连接 → 扫描 → 进入激活运行（OP）」，或当前模式不允许下发命令。

失效控制权自动回收：取得控制权后 SDK 会自动周期发送心跳（默认每秒一次，不需要调用任何 API）。持有者进程崩溃 / 被终止 / 未归还就退出时，宿主在约 5 秒内判定控制权失效，下一个 `SCM_RequestOwner` 会自动收回并转交控制权（主软件日志中有记录）。空闲不发命令不会丢失控制权——SDK 的自动心跳就表明程序仍在运行。

说明 注意：⚠ 判断"控制权是不是在我手上"，要拿自己的 `clientId` 与 `SCM_DIAGNOSTICS.ownerClientId` 比。只看 `controlRole` 会认错人——别的程序持有 Owner 时，你这边读到的角色字段同样可能是 `Owner`，据此启用界面上的写操作按钮，每次操作都会被拒。

相关指令： `SCM_ReleaseOwner`、`SCM_RequestMaintenance`、`SCM_GetDiagnostics`、`SCM_GetLastReason`

5.10.2 SCM_ReleaseOwner

```
int SCM_ReleaseOwner(SCM_CARD_HANDLE handle)
```

功能：归还控制权，退回 Monitor（只读）角色。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄

返回值： `SCM_OK` = 已归还； `SCM_REJECTED` = 你不是当前属主。

权限：只有**当前属主**能归还。

说明：归还后主软件恢复控制。程序退出前应显式调用它，不要依赖控制权自动回收——那需要约 5 秒，期间主软件一直处于只读状态。

归还控制权 = 同时解除武装（武装状态跟着控制权走）。所以后连接的程序**不会**继承前一个程序的武装状态，它得自己 `SCM_RequestOwner`；你自己要再下发也得重新申请。

说明 注意：⚠ 归还控制权**不会**停止正在进行的运动——解除武装只让**后续**命令被拒，已经在运动的轴照常运行（与常见运动控制卡 Close 不停轴的做法一致）。要连运动一起收掉，归还前先 `SCM_AxisStop` / `SCM_GroupStop`，或者改用 `SCM_EtherCATStop`（它会拆组停轴，但**不放弃**控制权）。

相关指令： `SCM_RequestOwner`、`SCM_EtherCATStop`、`SCM_AxisStop`、`SCM_Close`

5.10.3 SCM_RequestMaintenance

```
int SCM_RequestMaintenance(SCM_CARD_HANDLE handle)
```

功能：申请维护模式（Maintenance），用于人工干预与抢占接管。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄

返回值： `SCM_OK` = 已取得； `SCM_REJECTED` = 被拒（读 `SCM_GetLastReason`）； `SCM_BACKEND_UNAVAILABLE` = 宿主不可用。

权限：本条是取得权限的入口，任何角色都可调。

说明：维护模式与 Owner 同级下发权限，同样独占。用途是“我知道现在有人在控，但我要接管”的场景。

与 `SCM_RequestOwner` 一样：**成功即自动武装一次**，武装不成则一点权限都不给。

归还同样用 `SCM_ReleaseOwner`。

说明 注意：⚠ 抢占立即生效：原属主随后下发的写命令会开始收到 `SCM_REJECTED`，而它自己不会收到任何“你被抢了”的通知。要让原属主发现，只能靠它自己比对 `ownerClientId`。

相关指令： `SCM_RequestOwner`、`SCM_ReleaseOwner`、`SCM_GetDiagnostics`

5.11 轴使能与回零

说明 △ 本节的 `SCM_AxisPower` 与 `SCM_AxisHome` 都是**异步受理**：返回码只答"命令收下了吗"，不答"做完了吗"。完成条件、最长耗时与等待方法见各条的「说明」。

说明 回零参数一律**轴用户单位**（速度 / 偏移 / 行程上限），换算在宿主侧做，与「点位运动」那一族**完全同一套**：同一根 mm 轴上 `SCM_AxisMoveAbs(..., 5.0, ...)` 走 5 mm，这里的偏移 `5.0` 也是 5 mm。

5.11.1 SCM_AxisPower

```
int SCM_AxisPower(SCM_CARD_HANDLE handle, int axis, int enable)
```

功能：轴上电 / 下电（伺服使能）。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>axis</code>	<code>int</code>	入	轴号，0 基
<code>enable</code>	<code>int</code>	入	非 0 = 上电；0 = 下电

返回值：`SCM_OK` = 已受理（不是"已使能"）；`SCM_INVALID_STATE` = 总线未进 OP 或未取得控制权；`SCM_INVALID_AXIS` = 轴号非法；`SCM_REJECTED` = 被拒（读 `SCM_GetLastReason`）。

权限：需 Owner 或 Maintenance。

说明：上电前应确认总线已进 OP、驱动器无 Fault、急停与限位条件正常。

内部执行一段多步 CiA402 使能流程，需要若干个总线周期才能完成。

是否完成以轮询 `SCM_AxisGetStatus` 出参的 `powered` 为准，而不是本函数的返回码。耗时：成功时 ≥120ms、最长 8.1s；失败时最长 **12.62s**（流程中有两次状态等待，各最长 4s）。因此**等待超时不得短于 15 秒**，否则会在使能流程仍正常进行时误报超时。

说明 **注意**：△ 对已经使能的轴再次上电也会重新执行一遍使能流程，期间 6041 状态字会短暂离开 Operation Enabled（`powered` 可能短暂为 0），因此同样需要等待。
△ 不等 `powered` 就继续操作会被**拒绝**：`SCM_AxisHome`、`SCM_GroupEnable` 与 `SCM_AxisMove*` 在轴未使能时都会拒绝执行，返回 `SCM_REJECTED`，原因码 `HostNotReady`。表现为每次发送都被拒，看起来像"无法建组/轴无法运动"，实际原因是没有等待使能完成。请轮询等待，不要用"被拒后重试"代替等待。

相关指令：`SCM_AxisGetStatus`、`SCM_AxisReset`、`SCM_AxisHome`

5.11.2 SCM_AxisHome

```
int SCM_AxisHome(SCM_CARD_HANDLE handle, int axis)
```

功能：按该轴配置启动回零流程（异步受理）。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
axis	int	入	轴号，0 基

返回值： SCM_OK = 已受理（不是"已回零"）； SCM_REJECTED = 被拒，典型原因码 HomingNotEnabled（该轴没启用任何回零引擎） / HomingSwitchNotConfigured（启用了控制器回零但原点开关点位没配） / 轴未使能； SCM_INVALID_AXIS = 轴号非法。

权限：需 Owner 或 Maintenance。

说明：按轴配置自动分流三种方式（由回零参数 SCM_HOME_ENGINE 决定，取值见本章末「回零参数号」表）：

- 驱动器回零（引擎①）：写 CiA402 的 6098 让驱动器自主回零；
- 控制器回零（引擎②）：由控制器读 I/O 原点开关，走粗找 → 慢爬 → 偏移三阶段；
- 当前位置置零：不动电机，把当前位置定义为原点（绝对值编码器机型的常用方式）。

受理后轮询 SCM_AxisGetHomingState，收尾条件必须写成 HOMED 或 ERROR 或 ABORTED。

说明 注意：⚠ 只判 ERROR 不判 ABORTED，遇到"被中止"会一直等到你自己超时——而中止是很常见的正常操作。

⚠ 失败时按 SCM_AxisGetHomingResult 的原因码分支，别解析文案。其中

SCM_HOMING_MODE_SWITCH_FAILED（驱动器进不去回零模式）的处置与其它失败完全不同：改回零方式号、改接线都没用。

相关指令： SCM_AxisGetHomingState、SCM_AxisGetHomingResult、SCM_AxisAbortHoming、SCM_AxisSetHomeParams

5.11.3 SCM_AxisSetHomeParam

```
int SCM_AxisSetHomeParam(SCM_CARD_HANDLE handle, int axis, int id, double value)
```

功能：设置该轴的一项回零参数。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
axis	int	入	轴号，0 基
id	int	入	参数号，取 SCM_HOME_* 常量（见本章末「回零参数号」表）
value	double	入	参数值。速度 / 偏移 / 行程上限一律轴用户单位

返回值： `SCM_OK` = 已设置； `SCM_REJECTED` = 该轴回零正在进行中； `SCM_INVALID_ARGUMENT` = 参数号非法；
`SCM_INVALID_AXIS` = 轴号非法。

权限：需 Owner 或 Maintenance。

说明：适合只改一两项的场合。要整组下发用 `SCM_AxisSetHomeParams`。

单位换算在宿主侧做，上位程序别自己乘换算比。

说明 注意：⚠ 回零进行中改参数会被拒，以免回零过程使用新旧混杂的参数。

相关指令： `SCM_AxisGetHomeParam`、`SCM_AxisSetHomeParams`、`SCM_AxisHome`

5.11.4 SCM_AxisGetHomeParam

```
int SCM_AxisGetHomeParam(SCM_CARD_HANDLE handle, int axis, int id, double* value)
```

功能：读该轴的一项回零参数。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>axis</code>	<code>int</code>	入	轴号，0 基
<code>id</code>	<code>int</code>	入	参数号，取 <code>SCM_HOME_*</code> 常量（见本章末「回零参数号」表）
<code>value</code>	<code>double*</code>	出	参数值。失败时给 0

返回值： `SCM_OK` = 读到； `SCM_INVALID_ARGUMENT` = 参数号非法； `SCM_INVALID_AXIS` = 轴号非法。

权限：只读，不需要控制权。

说明：参数号非法或宿主版本较旧不认识该参数时，出参给 0。

说明 注意：⚠ 出参 0 既可能是"这一项真的是 0"，也可能是"没读到"。要区分就看返回码，别只看出参。

相关指令： `SCM_AxisSetHomeParam`、`SCM_AxisGetHomeParams`

5.11.5 SCM_AxisSetHomeParams

```
int SCM_AxisSetHomeParams(SCM_CARD_HANDLE handle, int axis, const SCM_HOME_PARAMS* homeParams)
```

功能：整组下发 12 项回零参数。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄

<code>axis</code>	<code>int</code>	入	轴号，0 基
<code>homeParams</code>	<code>const SCM_HOME_PARAMS*</code>	入	12 项回零参数（字段与 <code>SCM_HOME_*</code> 参数号一一对应）

返回值：`SCM_OK` = 12 项全部下发成功；否则返回**第一个失败项**的错误码；`SCM_REJECTED` = 该轴回零正在进行中。

权限：需 Owner 或 Maintenance。

说明：请在回零未进行时调用。

说明

注意：⚠ **本条不是原子的** —— 内部逐项下发，任一项失败即停下并返回该项的错误码，已下发的
那几项不回滚。失败后应把整组重读一遍（`SCM_AxisGetHomeParams`）确认实际状态，别假定"要么全成功
要么全没动"。

相关指令：`SCM_AxisGetHomeParams`、`SCM_AxisSetHomeParam`、`SCM_AxisHome`

5.11.6 `SCM_AxisGetHomeParams`

```
int SCM_AxisGetHomeParams(SCM_CARD_HANDLE handle, int axis, SCM_HOME_PARAMS* homeParams)
```

功能：整组读回 12 项回零参数。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>axis</code>	<code>int</code>	入	轴号，0 基
<code>homeParams</code>	<code>SCM_HOME_PARAMS*</code>	出	12 项回零参数。失败时给零值

返回值：`SCM_OK` = 读到；`SCM_INVALID_AXIS` = 轴号非法；`SCM_INVALID_ARGUMENT` = 出参指针为空。

权限：只读，不需要控制权。

说明：配完参数后读一遍核对，是回零调试的常规动作 —— 尤其在 `SCM_AxisSetHomeParams` 中途失败之后。

相关指令：`SCM_AxisSetHomeParams`、`SCM_AxisGetHomeParam`

5.11.7 `SCM_AxisGetHomingState`

```
int SCM_AxisGetHomingState(SCM_CARD_HANDLE handle, int axis)
```

功能：查该轴当前的回零阶段。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄

axis	int	入	轴号, 0 基
------	-----	---	---------

返回值: ⚠ 返回值本身就是回零阶段码 `SCM_HOMING_PHASE` (取值见本章末「回零阶段」表), 不是错误码。句柄无效 / 轴号非法时返回 `SCM_INVALID_ARGUMENT` (值 10, 越出阶段枚举范围, 可据此判别)。

权限: 只读, 不需要控制权。

说明: 阶段是粘滞的 —— 走到 `HOMED` 之后一直保持, 直到下一次回零开始。所以它能回答"这根轴回过零没有"这个问题; 更直接的判断方法是看 `SCM_AXIS_STATUS.homed`。

说明 注意: ⚠ 本条名字看着是"读", 实现上是发命令 (占用全局命令槽), 不要高频轮询。回零期间 100ms 一次足够。

⚠ 主软件仿真模式下没有真实回零开关, 观察到的阶段序列与实际设备可能不同 (部分阶段一闪而过) —— 回零流程的分支处理请以实际设备上的表现为准。

相关指令: `SCM_AxisHome`、`SCM_AxisGetHomingResult`、`SCM_AxisGetStatus`

5.11.8 SCM_AxisAbortHoming

```
int SCM_AxisAbortHoming(SCM_CARD_HANDLE handle, int axis)
```

功能: 中止该轴正在进行的回零。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 <code>SCM_Open*</code> 取得的卡句柄
axis	int	入	轴号, 0 基

返回值: `SCM_OK` = 中止命令已下发; `SCM_INVALID_AXIS` = 轴号非法。

权限: 需 Owner 或 Maintenance。

说明: 驱动器回零 (引擎①) 以 halt 方式减速停止, 停稳后以当前位置为起点切回 CSP 位置控制; 控制器回零 (引擎②) 取消状态机并停轴。

中止后阶段变为 `ABORTED`, `SCM_AxisGetHomingResult` 给 `SCM_HOMING_ABORTED`。

说明 注意: ⚠ **中止不等于错误:** 中止是"再来一次就行", 错误是"要去排查"。轮询回零阶段时把这两者分开处理, 不要都往报警里塞。

相关指令: `SCM_AxisHome`、`SCM_AxisGetHomingState`

5.11.9 SCM_AxisGetHomingResult

```
int SCM_AxisGetHomingResult(SCM_CARD_HANDLE handle, int axis)
```

功能: 查该轴回零失败 / 中止的**结构化原因**。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
axis	int	入	轴号，0 基

返回值：⚠ 返回值本身就是原因码 `SCM_HOMING_RESULT`（取值见本章末「回零失败原因」表），不是错误码。句柄无效 / 轴号非法时返回 10。

权限：只读，不需要控制权。

说明：只在阶段为 `ERROR` 或 `ABORTED` 时才有意义；其余时候给 `SCM_HOMING_NONE`。

说明 注意：⚠ 按本码分支，别去解析文案——文案随版本 / 语言变，码不变。

⚠ 以下三个码是驱动器回零（引擎①）特有的，在实际设备上会出现：`SCM_HOMING_DRIVE_HOMING_ERROR`（驱动器报回零错误，多为方式号与限位 / 原点开关接线方向不匹配）、`SCM_HOMING_BUS_STOPPED`（回零途中过程数据停了）、`SCM_HOMING_MODE_SWITCH_FAILED`（驱动器进不去回零模式）。

相关指令：`SCM_AxisGetHomingState`、`SCM_AxisHome`

5.12 点位运动

说明 单位：位置 = 轴用户单位，速度 = 轴用户单位/s，加减速 = 轴用户单位/s²，Jerk = 轴用户单位/s³——与主软件轴页完全同一套单位（轴页配 mm 就写 mm，写 10 就走 10 mm）。换算比与方向取反由宿主按轴页设置施加，上位程序别自己乘换算比。

说明 与「轴使能与回零」一节的回零参数使用同一套单位。

说明 `acceleration` / `deceleration` / `jerk` 传 0 = 用主软件轴页「动态」里给该轴配的默认值；要逐段自己控就显式传。两边都没有默认实参，必须写满。

说明 ⚠ 轴未使能（或使能还在进行中）时，本节的运动指令会被拒绝：返回 `SCM_REJECTED`，原因码 `HostNotReady`。
先轮询 `SCM_AxisGetStatus` 的 `powered` 为真再发，做法见 `SCM_AxisPower` 一条；**不要用"被拒后重试"代替等待。**

5.12.1 SCM_AxisMoveAbs

```
int SCM_AxisMoveAbs(SCM_CARD_HANDLE handle, int axis, double position, double velocity, double acceleration, double deceleration, double jerk)
```

功能：绝对定位——把轴移动到指定的绝对坐标。

参数	类型	方向	说明
----	----	----	----

handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
axis	int	入	轴号，0 基
position	double	入	目标位置（轴用户单位，绝对坐标）
velocity	double	入	运行速度（轴用户单位/s）
acceleration	double	入	加速度（轴用户单位/s ² ）。0 = 用轴页默认
deceleration	double	入	减速度（轴用户单位/s ² ）。0 = 用轴页默认
jerk	double	入	加加速度（轴用户单位/s ³ ）。0 = 用轴页默认

返回值： SCM_OK = 已下发； SCM_REJECTED = 无控制权 / 轴被别的运动通道占用； SCM_INVALID_AXIS = 轴号非法； SCM_FAULTED = 轴处于故障态。

权限： 需 Owner 或 Maintenance。

说明： 坐标系是主站侧的轴坐标系，受 SCM_AxisSetPosition 影响。

判断到位： 自行轮询 SCM_AxisGetPosition，或等待 SCM_AxisReadStatus 的 discreteMotion 位清零。

说明 注意： ⚠ 轴未使能（或使能还在进行中）时本条会被**拒绝**，返回 SCM_REJECTED、原因码 HostNotReady。发送运动指令前请先轮询到 SCM_AxisGetStatus 的 powered 为真，不要用"被拒后重试"代替等待。

相关指令： SCM_AxisMoveRel、SCM_AxisMoveVel、SCM_AxisStop、SCM_AxisSetPosition

5.12.2 SCM_AxisMoveRel

```
int SCM_AxisMoveRel(SCM_CARD_HANDLE handle, int axis, double distance, double velocity, double acceleration, double deceleration, double jerk)
```

功能： 相对定位 —— 在当前基准上移动指定的增量。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
axis	int	入	轴号，0 基
distance	double	入	移动增量（轴用户单位，带符号）
velocity	double	入	运行速度（轴用户单位/s）
acceleration	double	入	加速度（轴用户单位/s ² ）。0 = 用轴页默认

<code>deceleration</code>	<code>double</code>	入	减速度（轴用户单位/s ² ）。0 = 用轴页默认
<code>jerk</code>	<code>double</code>	入	加加速度（轴用户单位/s ³ ）。0 = 用轴页默认

返回值：同 `SCM_AxisMoveAbs`。

权限：需 Owner 或 Maintenance。

说明：⚠ 基准是上一条运动指令的目标位（setpoint），不是当前实际位。连续推多段相对运动时，后一段接在前一段的终点上，不会因为跟随误差而累积偏差。

说明 注意：⚠ 正因为基准是 setpoint 而不是实际位，"先读当前位置再算增量"这种写法在连续推段时会算错——直接给增量即可。

相关指令：`SCM_AxisMoveAbs`、`SCM_AxisGetPosition`

5.12.3 `SCM_AxisMoveVel`

```
int SCM_AxisMoveVel(SCM_CARD_HANDLE handle, int axis, double velocity, double acceleration, double deceleration, double jerk)
```

功能：速度模式——按给定速度连续运动，不给终点。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>axis</code>	<code>int</code>	入	轴号，0 基
<code>velocity</code>	<code>double</code>	入	目标速度（轴用户单位/s，带符号定向）
<code>acceleration</code>	<code>double</code>	入	加速度（轴用户单位/s ² ）。0 = 用轴页默认
<code>deceleration</code>	<code>double</code>	入	减速度（轴用户单位/s ² ）。0 = 用轴页默认
<code>jerk</code>	<code>double</code>	入	加加速度（轴用户单位/s ³ ）。0 = 用轴页默认

返回值：同 `SCM_AxisMoveAbs`。

权限：需 Owner 或 Maintenance。

说明：轴会一直走下去，直到 `SCM_AxisStop` / `SCM_AxisHalt` / `SCM_AxisEStop` 收手，或撞到限位保护。

说明 注意：⚠ 这条不给终点，**必须自己负责收手**。程序异常退出时轴不会自动停——持有控制权的进程崩溃后，轴仍会继续运动，要等宿主回收失效的控制权、主软件接管后才会停止。安全相关场合请配硬件急停。

相关指令：SCM_AxisStop、SCM_AxisHalt、SCM_AxisJog

5.12.4 SCM_AxisStop

```
int SCM_AxisStop(SCM_CARD_HANDLE handle, int axis)
```

功能：普通停止——让轴停到 StandStill 状态，不置故障。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
axis	int	入	轴号，0 基

返回值：SCM_OK = 已下发；SCM_REJECTED = 无控制权；SCM_INVALID_AXIS = 轴号非法。

权限：需 Owner 或 Maintenance。

说明：这是**最终停**——与可被抢占的 SCM_AxisHalt 语义不同。停下来之后要重新下发运动指令才会再动。

说明 注意：⚠ 别拿它当“暂停”用。要“减速中还能被新指令直接接管”的语义，用 SCM_AxisHalt。

相关指令：SCM_AxisHalt、SCM_AxisEStop、SCM_AxisMoveVel

5.12.5 SCM_AxisEStop

```
int SCM_AxisEStop(SCM_CARD_HANDLE handle, int axis)
```

功能：急停。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
axis	int	入	轴号，0 基

返回值：SCM_OK = 已下发；SCM_REJECTED = 无控制权；SCM_INVALID_AXIS = 轴号非法。

权限：需 Owner 或 Maintenance。

说明：以最快方式停轴。

说明 注意：⚠ 急停会把轴置为故障态（`faulted`）。恢复流程是 `SCM_AxisReset` → 等 `faulted` 变假 → `SCM_AxisPower` 重新使能 → 等 `powered`，两步都是异步受理。
⚠ 软件急停不能替代硬件急停回路。

相关指令： `SCM_AxisReset`、`SCM_AxisPower`、`SCM_AxisStop`

5.12.6 SCM_AxisHalt

```
int SCM_AxisHalt(SCM_CARD_HANDLE handle, int axis, double deceleration)
```

功能：可中断的受控停 —— 按给定减速度减速到 0，减速过程中可被新的运动指令直接接管。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>axis</code>	<code>int</code>	入	轴号，0 基
<code>deceleration</code>	<code>double</code>	入	减速度（轴用户单位/s ² ）。 <code><= 0</code> = 用该轴默认停止减速度

返回值： `SCM_OK` = 已下发；`SCM_REJECTED` = 无控制权；`SCM_INVALID_AXIS` = 轴号非法。

权限：需 Owner 或 Maintenance。

说明：与 `SCM_AxisStop` 的差别是**不锁轴，也不结束该轴的 CSP 运动控制** —— 减速途中直接发一条新的 `SCM_AxisMove*` 就能接管，不必等它停稳。适合做"松开按钮先减速，但随时可能再按下去"这类交互。

说明 注意：⚠ 需要"停了就别再动"的场合请用 `SCM_AxisStop`。把 `SCM_AxisHalt` 当最终停用，会出现"以为停了、结果一条迟到的运动指令把它带走"的情况。

相关指令： `SCM_AxisStop`、`SCM_AxisJog`

5.12.7 SCM_AxisJog

```
int SCM_AxisJog(SCM_CARD_HANDLE handle, int axis, int direction, double velocity, double acceleration, double deceleration)
```

功能：点动（手动寸动）。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>axis</code>	<code>int</code>	入	轴号，0 基
<code>direction</code>	<code>int</code>	入	<code>> 0</code> 正向 / <code>< 0</code> 反向 / <code>0</code> 收手（按 <code>deceleration</code> 减速停）

<code>velocity</code>	<code>double</code>	入	点动速度（轴用户单位/s）。取绝对值后按 <code>direction</code> 定向； <code>direction != 0</code> 时必须 <code>> 0</code>
<code>acceleration</code>	<code>double</code>	入	加速度（轴用户单位/s ² ）。 <code>0</code> = 用轴页默认
<code>deceleration</code>	<code>double</code>	入	减速度（轴用户单位/s ² ）。 <code>0</code> = 用轴页默认

返回值：`SCM_OK` = 已下发；`SCM_INVALID_ARGUMENT` = `direction != 0` 而 `velocity <= 0`；`SCM_REJECTED` = 无控制权；`SCM_INVALID_AXIS` = 轴号非法。

权限：需 Owner 或 Maintenance。

说明：典型用法是按钮按下发 `direction = ±1`，松开发 `direction = 0`。

方向由 `direction` 决定，**不是**由 `velocity` 的符号决定——传负速度不会让它反向，只会被取绝对值。

说明 注意：⚠ 松手一定要发 `direction = 0`。界面上如果只在按钮 Down 时发指令、Up 时忘了发收手，轴会一直走。

相关指令：`SCM_AxisMoveVel`、`SCM_AxisHalt`、`SCM_AxisStop`

5.12.8 SCM_AxisReset

```
int SCM_AxisReset(SCM_CARD_HANDLE handle, int axis)
```

功能：复位轴故障（异步受理）。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>axis</code>	<code>int</code>	入	轴号，0 基

返回值：`SCM_OK` = 已受理（不是“已复位”）；`SCM_REJECTED` = 无控制权；`SCM_INVALID_AXIS` = 轴号非法。

权限：需 Owner 或 Maintenance。

说明：内部执行一段 CiA402 故障复位流程，含约 160ms + 290ms 两段等待，**耗时 ≥450ms**。

是否完成以轮询到 `SCM_AxisGetStatus` 出参的 `faulted` 变为假（0）为准，而不是本函数的返回码。

复位成功后轴通常处于未使能态，需要再 `SCM_AxisPower` 使能一次。

说明 注意：⚠ 复位不能消除故障的原因。驱动器侧的硬故障（如功率板损坏）复位后会立刻再报——若复位后立即再次报故障，请不要反复重试，应检查硬件。

相关指令：`SCM_AxisPower`、`SCM_AxisGetStatus`、`SCM_AxisEStop`

5.12.9 SCM_AxisSetPosition

```
int SCM_AxisSetPosition(SCM_CARD_HANDLE handle, int axis, double position)
```

功能：位置定义 —— 把轴当前所在的物理点定义为指定坐标，**不产生运动**。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
axis	int	入	轴号，0 基
position	double	入	要把当前物理点定义成的坐标（轴用户单位）。清零传 0

返回值： SCM_OK = 已设置； SCM_REJECTED = 轴在运动中（原因码 AxisMoving）或无控制权；
SCM_INVALID_AXIS = 轴号非法。

权限：需 Owner 或 Maintenance。

说明：主站侧的坐标偏移实现，**不写驱动器的 6064**。定义之后 SCM_AxisGetPosition 的读数与绝对定位的目标位都在新坐标系里。

说明 注意：⚠ 轴运动中会被拒，必须先停稳。

⚠ 这条只改坐标，不改机械位置 —— 它不是“回零”。真要建立机械原点请用 SCM_AxisHome（其中“当前位置置零”方式与本条效果相近，但会同时更新回零状态）。

相关指令： SCM_AxisGetPosition、SCM_AxisHome、SCM_AxisStop

5.13 轴状态读取

说明 本节全是**纯读**：只读共享内存，不占命令槽，可以放心按界面刷新率轮询，也不需要控制权。

说明 跟随误差与驱动器故障码都在 SCM_AxisGetStatus 的结构体里，不另设单条读法。

说明 位置与速度读回来是**轴用户单位**（与「点位运动」下发时的单位一致，方向取反也已还原），所以 SCM_AxisMoveAbs(..., 10.0, ...) 走完再读就是 10.0，可以直接拿来判到位。

5.13.1 SCM_AxisGetStatus

```
int SCM_AxisGetStatus(SCM_CARD_HANDLE handle, int axis, SCM_AXIS_STATUS* status)
```

功能：读该轴的完整状态（位置 / 速度 / 跟随误差 / 状态字 / 故障码等）。

参数	类型	方向	说明
----	----	----	----

handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
axis	int	入	轴号，0 基
status	SCM_AXIS_STATUS*	出	轴状态结构体，字段表见本章末「SCM_AXIS_STATUS 字段」。失败时给零值

返回值： SCM_OK = 读到； SCM_INVALID_AXIS = 轴号非法（出参给零值）； SCM_NOT_OPEN = 句柄已关闭； SCM_INVALID_ARGUMENT = 出参指针为空。

权限： 只读，不需要控制权。

说明： 这是最常用的一条。 powered / faulted 两个字段分别用于判断 SCM_AxisPower 与 SCM_AxisReset 是否完成（等待方法见这两条的说明）。 actualVelocity 是驱动器实测速度（606C）， commandVelocity 是规划速度 —— 前者是"它实际多快"，后者是"要它走多快"，排查跟随问题时要一起看。

说明 注意： ⚠ 出参给零值时别把 powered = 0 当成"轴确实没使能"，先看返回码 —— 轴号非法同样给零值。

相关指令： SCM_AxisReadStatus、SCM_AxisGetPosition、SCM_AxisGetVelocity

5.13.2 SCM_AxisReadStatus

```
int SCM_AxisReadStatus(SCM_CARD_HANDLE handle, int axis, SCM_AXIS_PLC_STATUS* status)
```

功能： 读解码后的 PLCopen 轴状态位（与主软件 MC_ReadStatus 同源）。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
axis	int	入	轴号，0 基
status	SCM_AXIS_PLC_STATUS*	出	PLCopen 状态位，字段表见本章末「SCM_AXIS_PLC_STATUS 字段」。轴号非法时 valid = 0

返回值： SCM_OK = 读到； SCM_INVALID_AXIS = 轴号非法（出参 valid = 0）； SCM_NOT_OPEN = 句柄已关闭。

权限： 只读，不需要控制权。

说明： 解码规则与主软件 MC_ReadStatus 一字不差，界面上要显示"停稳 / 定位中 / 连续运动中 / 故障停"这类状态时用它，不必自己去解 CiA402 状态字。各状态位互斥，取第一个成立的显示即可。

说明 注意： ⚠ 先看 valid。 valid = 0 时其余位全是 0，直接显示会变成"什么状态都不是"。

相关指令：SCM_AxisGetStatus

5.13.3 SCM_AxisGetPosition

```
int SCM_AxisGetPosition(SCM_CARD_HANDLE handle, int axis, double* position)
```

功能：读该轴的实际位置。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
axis	int	入	轴号，0 基
position	double*	出	实际位置（轴用户单位）。失败时给 0

返回值：SCM_OK = 读到；SCM_INVALID_AXIS = 轴号非法（出参给 0）；SCM_NOT_OPEN = 句柄已关闭。

权限：只读，不需要控制权。

说明：读数在主站侧坐标系里，受 SCM_AxisSetPosition 的位置定义影响。

同一个值也在 SCM_AxisGetStatus 的 actualPosition 字段里；要同时看位置与状态，读一次 SCM_AxisGetStatus 更省事，也不会出现“位置与状态来自不同时刻”的错位。

说明 注意：⚠ 判到位时别只比位置。位置到了不等于运动结束（还可能在整定），也不等于没报警——一并看 SCM_AxisReadStatus 的 standStill 与 SCM_AxisGetStatus 的 faulted。

相关指令：SCM_AxisGetStatus、SCM_AxisSetPosition

5.13.4 SCM_AxisGetParam

```
int SCM_AxisGetParam(SCM_CARD_HANDLE handle, int axis, int paramId, double* value)
```

功能：读主软件轴页上配的轴参数（当量、软限位、动态默认值）。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
axis	int	入	轴号，0 基
paramId	int	入	参数号，取 SCM_AXIS_PARAM（见本章末「轴参数号」表）
value	double*	出	参数值。返回非 SCM_OK 时无意义

返回值：SCM_OK = 读到；SCM_INVALID_ARGUMENT = 轴号越界 / 参数号未知 / 宿主版本旧不认这条命令；SCM_NOT_OPEN = 句柄已关闭。

权限：只读，不需要控制权。**本族只读**——这些值由主软件轴页管理，客户端不能修改。

说明：你的程序依托主软件运行，轴页上配置的当量、软限位等就是这台设备的实际参数。连接契约 `.json` 里不含这些值（只有轴数/点数/周期/映射名），请用本条读取，不必对照轴页手工抄写。

最要紧的用法是建组前逐轴读 `SCM_AXIS_SCALE`，原样填给 `SCM_GroupConfigAxis` 的 `scale`。填错的后果是整段路径按错误的比例运动，而且**不会有任何报错**。

软限位则可以用来在自己界面上画行程范围，或在下发前先自查目标点是否越界。

启动自检：如果你的程序里把某根轴的当量、行程或方向写成了常量（点位族按脉冲发值时尤其常见），请在 `SCM_RequestOwner` 之后、发第一条运动之前，逐轴读一次 `SCM_AXIS_SCALE`（必要时连同 `SCM_AXIS_INVERT_DIRECTION`），与你程序里的假设比对，**不一致就拒绝启动，并把两个值都打出来**。主软件轴页改过换算比后只在改动当时弹一次提醒，它不知道你的程序里写了什么，也没有任何事件能告诉你"改过了"——因此只能由客户端在启动时自行检查。

说明 注意：⚠ **不可得看返回码，不看值。**这一族里 `0` 是好几项的合法值（软限位可以配成 `0`，"未配"也是 `0`），拿 `0` 当"没读到"会把"配成了 `0`"读成"读不到"。

⚠ 方向取反**不在当量里**（当量恒为正）。取反由主软件在下发与读回两侧对称施加，客户端不用也不该自己乘符号；`SCM_AXIS_INVERT_DIRECTION` 读出来只为显示。

⚠ 使用 `SCM_Cnc*` 时无需调用本条：`SCM_CncOpen` 会自动逐轴读取当量。

相关指令： `SCM_GroupConfigAxis`、`SCM_CncOpen`

5.13.5 SCM_AxisGetVelocity

```
int SCM_AxisGetVelocity(SCM_CARD_HANDLE handle, int axis, double* velocity)
```

功能：读该轴的实际速度。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>axis</code>	<code>int</code>	入	轴号，0 基
<code>velocity</code>	<code>double*</code>	出	实际速度（轴用户单位/s，来自驱动器 606C 实测）。失败时给 <code>0</code>

返回值： `SCM_OK` = 读到；`SCM_INVALID_AXIS` = 轴号非法（出参给 `0`）；`SCM_NOT_OPEN` = 句柄已关闭。

权限：只读，不需要控制权。

说明：这是实测速度。规划速度是另一回事，在 `SCM_AxisGetStatus` 的 `commandVelocity` 字段里。

说明 注意：⚠ 拿实测速度判"停了没有"要留裕量——伺服停稳时它会在 0 附近抖。判停请用 `SCM_AxisReadStatus` 的 `standStill`。

相关指令： `SCM_AxisGetStatus`、`SCM_AxisReadStatus`

5.14 I/O

说明 推荐按从站寻址：`slavePosition` = 从站拓扑位置（0 基，等于扫描顺序），`bit` = 站内 0 基位号。

增删其它从站不影响某个从站的 (`slavePosition`, `bit`) 地址，比全局序号稳。

解析在宿主侧完成，上位程序不用加载任何映射文件。

说明 写输出需 Owner / Maintenance；读输入 Monitor 亦可。

说明 △ 两种"读不到"要分清：未映射的从站位 / 通道 → 出参 0 且返回 `SCM_OK`（宿主判定"这个点没配"，

不是错误）；而全局序号越界（如纯输出拓扑 DI=0 时读输入）→ 出参 0 且返回 `SCM_INVALID_ARGUMENT`。

说明 △ 数字量的出参是 `int`（0/1）不是 `bool`，判断时用 `!= 0`。

说明 △ 本节的读指令实现上是发命令（占用全局命令槽），不要高频轮询。

5.14.1 SCM_WriteOutputBit

```
int SCM_WriteOutputBit(SCM_CARD_HANDLE handle, int slavePosition, int bit, int value)
```

功能：按从站写一个数字量输出位（推荐用法）。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>slavePosition</code>	<code>int</code>	入	从站拓扑位置，0 基（等于扫描顺序）
<code>bit</code>	<code>int</code>	入	该从站内的输出位号，0 基
<code>value</code>	<code>int</code>	入	非 0 置位，0 清位

返回值：`SCM_OK` = 已下发（该点未映射时同样是 `SCM_OK`，只是什么都没写）；`SCM_REJECTED` = 无控制权；`SCM_BACKEND_UNAVAILABLE` = 宿主不可用。

权限：需 Owner 或 Maintenance。

说明：地址不受其它从站增删影响，是接线固定后最稳的写法。

说明 注意：△ 写一个未映射的点不会报错。要确认写进去了，回读一次 `SCM_ReadInputBit` 对应的输入点，或看现场。

相关指令：`SCM_ReadInputBit`、`SCM_WriteOutput`

5.14.2 SCM_ReadInputBit

```
int SCM_ReadInputBit(SCM_CARD_HANDLE handle, int slavePosition, int bit, int* value)
```

功能：按从站读一个数字量输入位（推荐用法）。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
slavePosition	int	入	从站拓扑位置，0 基
bit	int	入	该从站内的输入位号，0 基
value	int*	出	0 / 1。未映射或失败时给 0

返回值：SCM_OK = 正常（含"该点未映射"）；SCM_NOT_OPEN = 句柄已关闭；SCM_BACKEND_UNAVAILABLE = 宿主不可用。

权限：只读，Monitor 亦可。

说明：出参是 int，判断时用 != 0。

说明 注意：⚠ 未映射的点返回 SCM_OK 且出参 0 —— 这是宿主的判定（"这个点没配"不算错误）。所以"读到 0 且返回 OK"有两种可能：真的是低电平，或者这个点根本没配。接线阶段要靠现场确认，不能只看读数。

相关指令：SCM_WriteOutputBit、SCM_ReadInput

5.14.3 SCM_WriteOutput

```
int SCM_WriteOutput(SCM_CARD_HANDLE handle, int index, int value)
```

功能：按全局序号写一个数字量输出点（兼容保留）。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
index	int	入	全局输出序号：所有输出位按 扫描顺序 → PDO → 位序 排成的扁平序号
value	int	入	非 0 置位，0 清位

返回值：SCM_OK = 已下发；SCM_INVALID_ARGUMENT = 序号越界；SCM_REJECTED = 无控制权。

权限：需 Owner 或 Maintenance。

说明：兼容保留的写法。

说明 注意：⚠ 增删从站会让全局序号整体错位：同一个序号在拓扑变化后可能指向另一个点。新代码请用 `SCM_WriteOutputBit` 按从站寻址。⚠ 没有说明文件的 I/O 模块也会按设备上报的通道计入全局序号（早期版本不计入），因此同一套拓扑升级后序号可能整体后移——升级前请核对程序中用到的全局序号，或改用按从站寻址。

相关指令： `SCM_WriteOutputBit`、`SCM_ReadInput`

5.14.4 SCM_ReadInput

```
int SCM_ReadInput(SCM_CARD_HANDLE handle, int index, int* value)
```

功能：按全局序号读一个数字量输入点（兼容保留）。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>index</code>	<code>int</code>	入	全局输入序号，编号规则同 <code>SCM_WriteOutput</code>
<code>value</code>	<code>int*</code>	出	0 / 1。越界或失败时给 0

返回值： `SCM_OK` = 读到； `SCM_INVALID_ARGUMENT` = 序号越界（例如纯输出拓扑 DI=0 时读任何输入）；
`SCM_NOT_OPEN` = 句柄已关闭。

权限：只读，Monitor 亦可。

说明：越界时出参给 0 并返回 `SCM_INVALID_ARGUMENT`，不抛异常、不崩溃。

说明 注意：⚠ 注意与 `SCM_ReadInputBit` 的差别：本条越界是 `SCM_INVALID_ARGUMENT`，而按从站读未映射点是 `SCM_OK`。两条的“读不到”走的是不同返回码，别用同一套判断。⚠ 全局序号的错位问题同 `SCM_WriteOutput`：没有说明文件的 I/O 模块也计入序号，与早期版本相比，升级后序号可能整体后移。

相关指令： `SCM_ReadInputBit`、`SCM_WriteOutput`

5.14.5 SCM_WriteOutputWord

```
int SCM_WriteOutputWord(SCM_CARD_HANDLE handle, int channelIndex, long long value)
```

功能：写一个整型 / 字通道（按通道整字写入）。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>channelIndex</code>	<code>int</code>	入	全局 I/O 通道序号
<code>value</code>	<code>long long</code>	入	要写入的整型值，按通道类型截断

返回值： `SCM_OK` = 已下发； `SCM_REJECTED` = 无控制权； `SCM_INVALID_ARGUMENT` = 通道序号非法。

权限：需 Owner 或 Maintenance。

说明：值与总线字节的编解码在**持有通道档案的宿主侧**完成，共享内存只透传原始整数。

说明 注意：⚠ `channelIndex` 是**通道序号**，与数字量点位的全局序号不是同一套编号，别混用。

相关指令： `SCM_ReadInputWord`、 `SCM_WriteChannelRaw`

5.14.6 SCM_ReadInputWord

```
int SCM_ReadInputWord(SCM_CARD_HANDLE handle, int channelIndex, long long* value)
```

功能：读一个整型 / 字通道。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>channelIndex</code>	<code>int</code>	入	全局 I/O 通道序号
<code>value</code>	<code>long long*</code>	出	通道当前值。未映射或失败时给 <code>0</code>

返回值： `SCM_OK` = 正常（含“该通道未映射”）； `SCM_NOT_OPEN` = 句柄已关闭； `SCM_BACKEND_UNAVAILABLE` = 宿主不可用。

权限：只读，Monitor 亦可。

说明：未映射通道给 `0` 且返回 `SCM_OK`，处理方式与 `SCM_ReadInputBit` 一致。

相关指令： `SCM_WriteOutputWord`、 `SCM_ReadChannelRaw`

5.14.7 SCM_WriteAnalogOutput

```
int SCM_WriteAnalogOutput(SCM_CARD_HANDLE handle, int channelIndex, double value)
```

功能：按**工程量**写模拟量输出。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>channelIndex</code>	<code>int</code>	入	全局 I/O 通道序号
<code>value</code>	<code>double</code>	入	工程量值。F32/F64 通道原生写入；整型通道按舍入写入

返回值： `SCM_OK` = 已下发； `SCM_REJECTED` = 无控制权； `SCM_INVALID_ARGUMENT` = 通道序号非法。

权限：需 Owner 或 Maintenance。

说明：工程量 ↔ 原始值的换算由宿主按通道档案完成，上位程序直接给物理量即可。
要绕过换算直接写原始码值，用 `SCM_WriteChannelRaw`。

相关指令： `SCM_ReadAnalogInput`、`SCM_WriteChannelRaw`

5.14.8 SCM_ReadAnalogInput

```
int SCM_ReadAnalogInput(SCM_CARD_HANDLE handle, int channelIndex, double* value)
```

功能：按工程量读模拟量输入。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>channelIndex</code>	<code>int</code>	入	全局 I/O 通道序号
<code>value</code>	<code>double*</code>	出	工程量值。未映射或失败时给 <code>0</code>

返回值： `SCM_OK` = 正常（含"该通道未映射"）；`SCM_NOT_OPEN` = 句柄已关闭。

权限：只读，Monitor 亦可。

说明：换算在宿主侧完成。

说明 注意：⚠ 出参 `0` 可能是"真的 0"，也可能是"通道没配"，两者都返回 `SCM_OK`。首次接线时请用现场信号确认。

相关指令： `SCM_WriteAnalogOutput`、`SCM_ReadChannelRaw`

5.14.9 SCM_WriteChannelRaw

```
int SCM_WriteChannelRaw(SCM_CARD_HANDLE handle, int channelIndex, long long raw)
```

功能：按通道类型写原始整数值（不走工程量换算）。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>channelIndex</code>	<code>int</code>	入	全局 I/O 通道序号
<code>raw</code>	<code>long long</code>	入	原始码值，按通道位宽截断

返回值： `SCM_OK` = 已下发；`SCM_REJECTED` = 无控制权；`SCM_INVALID_ARGUMENT` = 通道序号非法。

权限：需 Owner 或 Maintenance。

说明：调试模拟量标定、或需要精确控制码值时用它。

说明 注意：⚠ 原始值不经量程校验，写超出通道位宽的值会被截断。

相关指令：SCM_ReadChannelRaw、SCM_WriteAnalogOutput

5.14.10 SCM_ReadChannelRaw

```
int SCM_ReadChannelRaw(SCM_CARD_HANDLE handle, int channelIndex, long long* value)
```

功能：按通道类型读原始整数值（不走工程量换算）。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
channelIndex	int	入	全局 I/O 通道序号
value	long long*	出	原始码值。未映射或失败时给 0

返回值：SCM_OK = 正常（含“该通道未映射”）；SCM_NOT_OPEN = 句柄已关闭。

权限：只读，Monitor 亦可。

说明：与 SCM_ReadAnalogInput 读的是同一个点，差别只在要不要换算 —— 标定时两条对照着看最直观。

相关指令：SCM_WriteChannelRaw、SCM_ReadAnalogInput

5.15 坐标组插补（3~8 轴）

说明 由前瞻规划器执行。连接真实 EtherCAT 总线时，由宿主侧规划器以 1ms 周期向各轴输出 CSP 位置指令；

主软件仿真模式下使用的是同一个规划器 —— 两种模式下 API 完全一样。

⚠ 单位是「轴用户单位」，但换算比**由本节自己的`scale`定**，不取自轴页

说明 本节全部的 x/y/z、dx/dy/dz、w0..w4、velocity、vmax/amax/jmax、cornerDeviation、compressChord 一律是轴用户单位；SCM_GroupConfigAxis 的 scale 是脉冲当量（脉冲 / 用户单位），规划器出帧时才乘 scale 变脉冲。

scale 默认 1.0 ⇒ 没配过 scale 时组坐标就等于脉冲。

说明 ⚠ 与点位运动最容易混淆的差别就在这里：点位那一族也是轴用户单位，但换算比取自主软件轴页；组插补取的是这里传进来的 scale。两个来源不会自动一致 —— 同一根 mm 轴上，点位写 5.0 走 5 mm，而 scale 没配过时组里写 5.0 只走 5 个脉冲。两族混用时要自己对齐。

``crd`` = 插补坐标系号

说明 0 基，上限 8 个。各 `crd` 独立建组、独立推段、独立查状态，两条路径可以同时走。同一根轴不许被两个坐标系同时接管，后建的被拒（原因码 `AxisClaimedByOther`）。
`crd` 越界返回 `SCM_INVALID_ARGUMENT`，**不会**自动改用 0 号 —— 单坐标系应用恒传 0。

``segId`` = 段号（仅 ``SCM\_GroupMove\*``）

说明 传 0 = 内部自动编号；传非 0 = 你自己的编号（典型是 G 代码行号），原样透传，由 `SCM_GroupGetStatus` 的 `lastSegId` 回报**正在执行到哪一段**。
 自动编号与自定义编号可以混用，不会重号（自定义编号会使内部计数从该编号之后继续）。

随动轴 ``w0..w4``（组宽 3~8 轴）

说明 追加在参数表**最末**。两处含义**不同**，这是本节最容易写错的地方：
 - `SCM_GroupEnable` 的 `w0..w4` 填**轴号**，-1 = 该位不用，**组宽 = 3 + 非 -1 的个数**；槽位必须**从左往右连续填**，中间留空会被拒绝并返回 `SCM_INVALID_ARGUMENT`（不会自动跳过）。
 - `SCM_GroupMove*` 的 `w0..w4` 填该轴**本段终点**（Rel 系列填增量），NAN = 该轴本段不动。
 "不动"锚在**已投递路径的末端**，不是"保持轴当前位置" —— 连续推段时两者不同。
 填到组宽以外的槽位会被拒绝，不会被忽略。

说明 随动轴**不参与路径几何**，沿 XYZ 弧长线性推进（直线段上是 N 维线性插补，圆弧段上是"转多少角度、随动轴走多少"）；但**同样算成员轴**，也参与跨坐标系的轴占用检查。
 C 里 NAN 宏在 `<math.h>`。两边都没有默认实参，**5 个哨兵值必须显式传满**。

5.15.1 SCM_GroupEnable

```
int SCM_GroupEnable(SCM_CARD_HANDLE handle, int axisX, int axisY, int axisZ, int crd, int w0, int w1, int w2, int w3, int w4)
```

功能：建组 —— 把若干根轴编成一个插补坐标系，并启动各轴的 CSP 位置控制。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>axisX</code>	<code>int</code>	入	X 插补轴的轴号（必填）
<code>axisY</code>	<code>int</code>	入	Y 插补轴的轴号（必填）
<code>axisZ</code>	<code>int</code>	入	Z 插补轴的轴号（必填）
<code>crd</code>	<code>int</code>	入	插补坐标系号，0 基，上限 8 个
<code>w0</code>	<code>int</code>	入	随动轴 0 的 轴号 ；-1 = 该位不用
<code>w1</code>	<code>int</code>	入	随动轴 1 的 轴号 ；-1 = 该位不用

w2	int	入	随动轴 2 的轴号；-1 = 该位不用
w3	int	入	随动轴 3 的轴号；-1 = 该位不用
w4	int	入	随动轴 4 的轴号；-1 = 该位不用

返回值： `SCM_OK` = 已建组；`SCM_INVALID_ARGUMENT` = `crd` 越界 / 轴号重复 / 随动轴槽位中间空一格；`SCM_REJECTED` = 轴未使能、轴被别的运动通道占用（`AxisClaimedByOther`）、或组通道不可用（`SoftRealtimeNotAllowed`）。

权限：需 Owner 或 Maintenance。

说明：组内各轴须互不相同、都是主软件里已映射的**硬件伺服轴**，且**已经使能**。

组宽 = 3 + 非 -1 的随动轴个数，最多 8 轴。

说明 注意：⚠ **轴必须先真正使能到位再建组。** `SCM_AxisPower` 是异步受理，返回 `SCM_OK` 只表示收下了——不等 `powered` 就建组会因轴未使能被拒绝，表现为每次建组都被拒。

⚠ 轴被其它运动通道（插补页 / CNC / 凸轮 / 齿轮 / 机器人）占用时会被拒。**随动轴同样算成员轴**——`crd0` 占了 0/1/2/3 之后，`crd1` 只能从 4 开始。

⚠ 组插补只在硬实时的内核组通道上提供。组通道不可用时（例如主软件未运行在硬实时内核上），本条**直接被拒并给出原因**，不会降级为缺少保护的运行方式。处置：确认主软件已激活"实时运行"再重试。

⚠ **同步命令：返回 `SCM_OK` 即组已建成**（不是异步受理，不用轮询）。宿主**首次**建组需要一次性初始化，4 轴约需 1s，轴数越多耗时越长，因此 SDK 对本条的等待上限**单独放宽到 5s**（其余命令为 1s）。若仍拿到 `SCM_TIMEOUT`：**组可能已在宿主侧建成**（超时答的是"没等到"，不是"没做"）——同一轴集重发本命令**幂等**（返回"插补组已存在(成员轴相同, 不重复建立)"），不要把超时当作致命错误直接退出，否则会留下一个无人管理的已建组（处理方法：`SCM_GroupDisable`，或在主软件侧停止）。

相关指令： `SCM_GroupDisable`、`SCM_GroupSetLimits`、`SCM_GroupConfigAxis`、`SCM_AxisPower`

5.15.2 SCM_GroupSetLimits

```
int SCM_GroupSetLimits(SCM_CARD_HANDLE handle, double vmax, double amax, int crd)
```

功能：设置该坐标系的**组级**速度 / 加速度上限。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 <code>SCM_Open*</code> 取得的卡句柄
vmax	double	入	路径速度上限（用户单位/s）
amax	double	入	路径加速度上限（用户单位/s ² ）
crd	int	入	插补坐标系号

返回值： `SCM_OK` = 已设置；`SCM_INVALID_ARGUMENT` = `crd` 越界；`SCM_REJECTED` = 无控制权。

权限：需 Owner 或 Maintenance。

说明：影响前瞻限速。每段的实际路径速度按 $\min(\text{段 velocity}, v_{\max})$ 封顶。

没有单独配过 `SCM_GroupConfigAxis` 的轴，回落到这一对上限，**依然受限**。

相关指令： `SCM_GroupConfigAxis`、`SCM_GroupMoveLinearAbs`

5.15.3 SCM_GroupConfigAxis

```
int SCM_GroupConfigAxis(SCM_CARD_HANDLE handle, int axisIndex, double scale, double vmax, double amax, double jmax, int crd)
```

功能：设置组内**某一根轴**的脉冲当量与动力学上限。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>axisIndex</code>	<code>int</code>	入	组内位置 <code>0..7</code> ：0=X / 1=Y / 2=Z， 3..7 = 随动轴 W0..W4 （与建组槽位一一对应）。 不是轴号
<code>scale</code>	<code>double</code>	入	脉冲当量（脉冲 / 用户单位）。 <code><= 0</code> = 保持原值。默认 <code>1.0</code>
<code>vmax</code>	<code>double</code>	入	该轴速度上限（用户单位/s）。 <code><= 0</code> = 不设置，回落组级
<code>amax</code>	<code>double</code>	入	该轴加速度上限（用户单位/s ² ）。 <code><= 0</code> = 不设置，回落组级
<code>jmax</code>	<code>double</code>	入	该轴加加速度上限（用户单位/s ³ ）。 <code><= 0</code> = 不设置，回落组级
<code>crd</code>	<code>int</code>	入	插补坐标系号

返回值：`SCM_OK` = 已设置；`SCM_INVALID_ARGUMENT` = `axisIndex` 或 `crd` 越界。

权限：需 Owner 或 Maintenance。

说明：下一次运动会生效（不是立即改变正在走的路径）。

`axisIndex` 是**组内位置**不是轴号——这是本条最容易搞错的地方。建组时填在 `w0` 的那根轴，这里就是 `3`。未配过的随动轴回落组级、依然受限；建议逐根配是因为随动轴的机械条件通常与 XYZ 不同。

说明 注意：⚠ 加了随动轴后整条路径可能明显变慢，这是正常现象不是故障。路径速度要同时满足每根随动轴自己的上限：某根随动轴本段行程是 XYZ 弧长的十倍时，路径速度就被压到十分之一。想快请调高**该轴**的 `vmax`，而不是去掉限值。

⚠ 组插补的尺度由这里的 `scale` 决定，不取自主软件轴页的换算比——这是它与点位运动最容易混淆的差别：两族都叫“轴用户单位”，但换算比是两个来源。`scale` 保持默认 `1.0` 时组坐标就是脉冲，而同一根轴的点位坐标却按轴页换算比走——两族混用时要自己对齐。

相关指令： `SCM_GroupSetLimits`、`SCM_GroupEnable`

5.15.4 SCM_GroupConfigPath

```
int SCM_GroupConfigPath(SCM_CARD_HANDLE handle, double compressAngleDeg, double compressChord, int lookaheadWindow, int crd)
```

功能：设置路径规划参数（小线段压缩阈值与前瞻窗口）。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
compressAngleDeg	double	入	小线段压缩的角度阈值（度）。0 = 不压缩
compressChord	double	入	小线段压缩的弦高阈值（用户单位）。0 = 不压缩
lookaheadWindow	int	入	前瞻段数。≤ 0 = 保持默认（128）
crd	int	入	插补坐标系号

返回值：SCM_OK = 已设置；SCM_INVALID_ARGUMENT = crd 越界。

权限：需 Owner 或 Maintenance。

说明：下一次运动会生效。

针对小线段密集的轨迹（如 CAM 出的刀路）：把在阈值内的连续小线段合并，减少规划压力、提高路径速度。

说明 注意：⚠ 拐角容差不在这里 —— 那个是随每一段传的 cornerDeviation。别把两者混为一谈。

相关指令：SCM_GroupMoveLinearAbs、SCM_GroupSetLimits

5.15.5 SCM_GroupMoveLinearAbs

```
int SCM_GroupMoveLinearAbs(SCM_CARD_HANDLE handle, double x, double y, double z, double velocity, double jerk, double cornerDeviation, unsigned int segId, int crd, double w0, double w1, double w2, double w3, double w4)
```

功能：推一段直线（绝对终点）。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
x	double	入	终点 X（用户单位，绝对）
y	double	入	终点 Y（用户单位，绝对）
z	double	入	终点 Z（用户单位，绝对）
velocity	double	入	本段路径速度（用户单位/s），按组级与逐轴上限封顶

jerk	double	入	加加速度（用户单位/s ³ ）。> 0 启用 7 段 S 曲线
cornerDeviation	double	入	拐角容差（用户单位）。0 = 拐角处精确停
segId	unsigned int	入	段号。0 = 内部自动编号
crd	int	入	插补坐标系号
w0	double	入	随动轴 0 本段终点；NAN = 本段不动
w1	double	入	随动轴 1 本段终点；NAN = 本段不动
w2	double	入	随动轴 2 本段终点；NAN = 本段不动
w3	double	入	随动轴 3 本段终点；NAN = 本段不动
w4	double	入	随动轴 4 本段终点；NAN = 本段不动

返回值：SCM_OK = 已投递；SCM_INVALID_ARGUMENT = crd 越界 / 给到组宽以外的随动轴槽位；SCM_REJECTED = 组未建 / 无控制权；SCM_INVALID_STATE = 段缓冲已满（先看 SCM_GroupGetSpace）。

权限：需 Owner 或 Maintenance。

说明：连续推段时由前瞻在接点处平滑过渡，cornerDeviation 控制允许多大的拐角偏差。

说明 注意：⚠ 连续投递场景必须用 SCM_GroupFlush 收尾，否则最后几段会一直等后续段而不结束。

相关指令：SCM_GroupMoveLinearRel、SCM_GroupFlush、SCM_GroupGetSpace

5.15.6 SCM_GroupMoveLinearRel

```
int SCM_GroupMoveLinearRel(SCM_CARD_HANDLE handle, double dx, double dy, double dz, double velocity, double jerk, double cornerDeviation, unsigned int segId, int crd, double w0, double w1, double w2, double w3, double w4)
```

功能：推一段直线（相对增量，G91 语义）。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
dx	double	入	X 增量（用户单位）
dy	double	入	Y 增量（用户单位）
dz	double	入	Z 增量（用户单位）
velocity	double	入	本段路径速度（用户单位/s）
jerk	double	入	加加速度（用户单位/s ³ ）。> 0 启用 7 段 S 曲线

cornerDeviation	double	入	拐角容差（用户单位）。0 = 拐角处精确停
segId	unsigned int	入	段号。0 = 内部自动编号
crd	int	入	插补坐标系号
w0	double	入	随动轴 0 本段增量；NAN = 本段不动
w1	double	入	随动轴 1 本段增量；NAN = 本段不动
w2	double	入	随动轴 2 本段增量；NAN = 本段不动
w3	double	入	随动轴 3 本段增量；NAN = 本段不动
w4	double	入	随动轴 4 本段增量；NAN = 本段不动

返回值：同 SCM_GroupMoveLinearAbs。

权限：需 Owner 或 Maintenance。

说明：⚠ 增量锚在已投递路径的末端，不是轴此刻的实际位置。连续推段时这一点很关键——否则每段都会把跟随误差算进去。

相关指令：SCM_GroupMoveLinearAbs、SCM_GroupResetPath

5.15.7 SCM_GroupMoveCircularAbs

```
int SCM_GroupMoveCircularAbs(SCM_CARD_HANDLE handle, int plane, int clockwise, double x, double y, double z,
double centerOffA, double centerOffB, double velocity, double jerk, double cornerDeviation, unsigned int segId,
int crd, double w0, double w1, double w2, double w3, double w4)
```

功能：推一段圆弧（圆心偏移式，绝对终点）。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
plane	int	入	圆弧平面：0 = XY / 1 = XZ / 2 = YZ
clockwise	int	入	0 = 逆时针 / 1 = 顺时针
x	double	入	终点 X（用户单位，绝对）
y	double	入	终点 Y（用户单位，绝对）
z	double	入	终点 Z（用户单位，绝对）
centerOffA	double	入	圆心减起点在平面第一轴上的分量
centerOffB	double	入	圆心减起点在平面第二轴上的分量
velocity	double	入	本段路径速度（用户单位/s）

jerk	double	入	加加速度（用户单位/s ³ ）
cornerDeviation	double	入	拐角容差（用户单位）
segId	unsigned int	入	段号。0 = 内部自动编号
crd	int	入	插补坐标系号
w0	double	入	随动轴 0 本段终点；NAN = 本段不动
w1	double	入	随动轴 1 本段终点；NAN = 本段不动
w2	double	入	随动轴 2 本段终点；NAN = 本段不动
w3	double	入	随动轴 3 本段终点；NAN = 本段不动
w4	double	入	随动轴 4 本段终点；NAN = 本段不动

返回值： SCM_OK = 已投递； SCM_INVALID_ARGUMENT = 平面号非法 / crd 越界 / 圆心偏移定不出合法圆；其余同 SCM_GroupMoveLinearAbs。

权限： 需 Owner 或 Maintenance。

说明： 终点等于起点即整圆 —— 要走整圆只能用这种给法（半径式定不出圆心）。

圆心偏移是相对起点的向量，与终点是绝对还是相对无关。

相关指令： SCM_GroupMoveCircularRad、SCM_GroupMoveCircularRel

5.15.8 SCM_GroupMoveCircularRad

```
int SCM_GroupMoveCircularRad(SCM_CARD_HANDLE handle, int plane, int clockwise, double x, double y, double z,
double radius, double velocity, double jerk, double cornerDeviation, unsigned int segId, int crd, double w0,
double w1, double w2, double w3, double w4)
```

功能： 推一段圆弧（半径式，G 代码 R 式）。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
plane	int	入	圆弧平面：0 = XY / 1 = XZ / 2 = YZ
clockwise	int	入	0 = 逆时针 / 1 = 顺时针
x	double	入	终点 X（用户单位，绝对）
y	double	入	终点 Y（用户单位，绝对）
z	double	入	终点 Z（用户单位，绝对）
radius	double	入	半径。> 0 = 劣弧（≤180°），< 0 = 优弧（>180°）
velocity	double	入	本段路径速度（用户单位/s）

jerk	double	入	加加速度（用户单位/s ³ ）
cornerDeviation	double	入	拐角容差（用户单位）
segId	unsigned int	入	段号。0 = 内部自动编号
crd	int	入	插补坐标系号
w0	double	入	随动轴 0 本段终点；NAN = 本段不动
w1	double	入	随动轴 1 本段终点；NAN = 本段不动
w2	double	入	随动轴 2 本段终点；NAN = 本段不动
w3	double	入	随动轴 3 本段终点；NAN = 本段不动
w4	double	入	随动轴 4 本段终点；NAN = 本段不动

返回值：SCM_OK = 已投递；SCM_INVALID_ARGUMENT = 起点与终点重合（R 式定不出圆心）/ 半径小于起终点距离的一半（够不着）/ 平面号非法 / crd 越界。

权限：需 Owner 或 Maintenance。

说明：半径的符号决定走劣弧还是优弧，与 clockwise 是两个独立的选择。

说明 注意：⚠ 起终点重合会被拒 —— R 式定不出圆心。整圆请用圆心偏移式

SCM_GroupMoveCircularAbs。

⚠ 半径小于起终点距离一半也会被拒（几何上够不着）。浮点误差会让"恰好等于一半"这种边界情形不稳定，工程上留一点裕量。

相关指令：SCM_GroupMoveCircularAbs、SCM_GroupMoveCircularRel

5.15.9 SCM_GroupMoveCircularRel

```
int SCM_GroupMoveCircularRel(SCM_CARD_HANDLE handle, int plane, int clockwise, double dx, double dy, double dz,
double centerOffA, double centerOffB, double velocity, double jerk, double cornerDeviation, unsigned int segId,
int crd, double w0, double w1, double w2, double w3, double w4)
```

功能：推一段圆弧（圆心偏移式，相对终点）。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
plane	int	入	圆弧平面：0 = XY / 1 = XZ / 2 = YZ
clockwise	int	入	0 = 逆时针 / 1 = 顺时针
dx	double	入	终点 X 增量（用户单位）
dy	double	入	终点 Y 增量（用户单位）

dz	double	入	终点 Z 增量（用户单位）
centerOffA	double	入	圆心减起点在平面第一轴上的分量
centerOffB	double	入	圆心减起点在平面第二轴上的分量
velocity	double	入	本段路径速度（用户单位/s）
jerk	double	入	加加速度（用户单位/s³）
cornerDeviation	double	入	拐角容差（用户单位）
segId	unsigned int	入	段号。0 = 内部自动编号
crd	int	入	插补坐标系号
w0	double	入	随动轴 0 本段增量；NAN = 本段不动
w1	double	入	随动轴 1 本段增量；NAN = 本段不动
w2	double	入	随动轴 2 本段增量；NAN = 本段不动
w3	double	入	随动轴 3 本段增量；NAN = 本段不动
w4	double	入	随动轴 4 本段增量；NAN = 本段不动

返回值：同 SCM_GroupMoveCircularAbs。

权限：需 Owner 或 Maintenance。

说明：只有终点变成增量，圆心偏移本来就是相对起点的，语义不变。增量同样锚在已投递路径的末端。

相关指令：SCM_GroupMoveCircularAbs、SCM_GroupMoveLinearRel

5.15.10 SCM_GroupSetOverride

```
int SCM_GroupSetOverride(SCM_CARD_HANDLE handle, double ratio, int crd)
```

功能：设置进给倍率。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
ratio	double	入	倍率 0 ~ 1.5（0% ~ 150%）。0 = 进给保持
crd	int	入	插补坐标系号

返回值：SCM_OK = 已设置；SCM_INVALID_ARGUMENT = 倍率越界 / crd 越界。

权限：需 Owner 或 Maintenance。

说明： `ratio = 0` 就是进给保持 / 暂停：轴在路径上停住，队列一段不少，调回非 0 就从停处继续跑。这是暂停加工的正确做法 —— 不要用 `SCM_GroupStop`（那个会清掉规划状态）。

倍率变化走斜坡（满量程约 0.5s），不瞬跳。

说明 注意： ⚠ 上限钳在 150%，因为倍率 >100% 会把规划加速度放大到 `ratio2` 倍 —— 再高就可能超出机械与驱动器能力。

⚠ `SCM_GroupDisable` 会把倍率复位回 100%，`SCM_GroupResetPath` 不会。解完组再建组时倍率是 100%，但只清路径不解组时倍率还是你上次设的值。

相关指令： `SCM_GroupStop`、`SCM_GroupFlush`、`SCM_GroupDisable`

5.15.11 SCM_GroupFlush

```
int SCM_GroupFlush(SCM_CARD_HANDLE handle, int crd)
```

功能： 收尾当前路径 —— 把已投递的段冲刷出来，轨迹走完自然停。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>crd</code>	<code>int</code>	入	插补坐标系号

返回值： `SCM_OK` = 已下发；`SCM_INVALID_ARGUMENT` = `crd` 越界；`SCM_REJECTED` = 组未建。

权限： 需 Owner 或 Maintenance。

说明： 不解组，各轴的 CSP 控制保持不变。走完之后组还在，可以继续推新的段。

说明 注意： ⚠ 连续投递场景必须用它收尾。前瞻规划器要看到后续段才能决定当前段的末速度，最后几段没有“后续”，不 Flush 就会一直等下去，表现为“轨迹走到快结束时停住不动了”。

相关指令： `SCM_GroupMoveLinearAbs`、`SCM_GroupGetStatus`

5.15.12 SCM_GroupResetPath

```
int SCM_GroupResetPath(SCM_CARD_HANDLE handle, int crd)
```

功能： 清空当前路径，回到“未开始”状态。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>crd</code>	<code>int</code>	入	插补坐标系号

返回值： `SCM_OK` = 已下发；`SCM_INVALID_ARGUMENT` = `crd` 越界；`SCM_REJECTED` = 组未建。

权限：需 Owner 或 Maintenance。

说明：不解组，各轴的 CSP 控制保持不变。下一次 `SCM_GroupMove*` 会以当时的实际位置为起点重新起一个运动会话。投错了段想重来时用它。

说明 注意：⚠ 运动中调用会立即断掉后续帧 —— 应先 `SCM_GroupStop` 停稳再调。运动中直接 Reset 相当于把正在走的轨迹拦腰截断。
⚠ 它不会复位进给倍率（那是 `SCM_GroupDisable` 的行为）。

相关指令：`SCM_GroupStop`、`SCM_GroupDisable`

5.15.13 SCM_GroupStop

```
int SCM_GroupStop(SCM_CARD_HANDLE handle, double deceleration, int crd)
```

功能：沿原路径受控停止。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>deceleration</code>	<code>double</code>	入	减速度（用户单位/s ² ）
<code>crd</code>	<code>int</code>	入	插补坐标系号

返回值：`SCM_OK` = 已下发；`SCM_INVALID_ARGUMENT` = `crd` 越界；`SCM_REJECTED` = 组未建。

权限：需 Owner 或 Maintenance。

说明：沿原路径减速，不是各轴各自急停 —— 轨迹不会出圈，刀具不会切飞。

停下来之后组保持使能，可以继续追加 `SCM_GroupMove*` 从当前位置续走。

说明 注意：⚠ 只想暂停一会儿再继续的，用 `SCM_GroupSetOverride(handle, 0, crd)` 更合适 —— 那个队列一段不少。本条会消耗掉正在走的路径。

相关指令：`SCM_GroupSetOverride`、`SCM_GroupResetPath`

5.15.14 SCM_GroupGetSpace

```
int SCM_GroupGetSpace(SCM_CARD_HANDLE handle, int* space, int* capacity, int crd)
```

功能：读段缓冲的剩余可投递数与总容量。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>space</code>	<code>int*</code>	出	还能再投递多少段。组未建时给 -1

<code>capacity</code>	<code>int*</code>	出	段缓冲总容量。组未建时给 <code>-1</code>
<code>crd</code>	<code>int</code>	入	插补坐标系号

返回值： `SCM_OK`（组未建时也是 `SCM_OK`，靠出参 `-1` 判别）； `SCM_INVALID_ARGUMENT` = `crd` 越界。

权限：只读，不需要控制权。

说明：连续投递大量段时，用它做流控： `space` 见底就先别推，等消费掉一些再继续。

说明 注意：⚠ `space` 不等于 `capacity - segmentsQueued`。系统需要为受控停止保留一部分已执行段的缓冲位置，这些位置暂时不能复用。"还能不能再推 N 段"**必须看** `space` —— 用"容量减排队数"估会偏乐观，推到一半突然被拒。

相关指令： `SCM_GroupGetStatus`、 `SCM_GroupMoveLinearAbs`

5.15.15 SCM_GroupGetStatus

```
int SCM_GroupGetStatus(SCM_CARD_HANDLE handle, int* phase, unsigned long long* consumedFrames, unsigned int* lastSegId, int* segmentsQueued, int crd)
```

功能：查该坐标系的运行状态。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>phase</code>	<code>int*</code>	出	组阶段： <code>0</code> =Disabled / <code>1</code> =Enabled / <code>2</code> =Moving / <code>3</code> =Stopping / <code>4</code> =Fault
<code>consumedFrames</code>	<code>unsigned long long*</code>	出	已消费的插补帧数（每个总线周期一帧）
<code>lastSegId</code>	<code>unsigned int*</code>	出	正在执行到哪一段的段号
<code>segmentsQueued</code>	<code>int*</code>	出	队列里还排着多少段
<code>crd</code>	<code>int</code>	入	插补坐标系号

返回值： `SCM_OK` = 读到； `SCM_INVALID_ARGUMENT` = `crd` 越界。

权限：只读，不需要控制权。

说明： `lastSegId` 配合 `SCM_GroupMove*` 传入的自带段号，可以直接把"当前执行到 G 代码第几行"显示出来。判"这一批段是不是都走完了"： `phase` 回到 `1`（Enabled）且 `segmentsQueued` 为 0。

说明 注意：⚠ 本条名字看着是"读"，实现上是**发命令**（占用全局命令槽），不要按界面刷新率高频轮询——200ms 一次足够。

相关指令： `SCM_GroupGetSpace`、 `SCM_GroupFlush`

5.15.16 SCM_GroupDisable

```
int SCM_GroupDisable(SCM_CARD_HANDLE handle, int crd)
```

功能：解组 —— 释放组内各轴，解除对它们的占用。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
crd	int	入	插补坐标系号

返回值：SCM_OK = 已解组；SCM_INVALID_ARGUMENT = crd 越界。

权限：需 Owner 或 Maintenance。

说明：解组会把该坐标系的进给倍率复位回 100%。

解组之后这些轴才可以被别的坐标系或别的运动通道接管。

说明 注意：△ 解组前先把运动停干净（SCM_GroupStop 或等 SCM_GroupFlush 走完）。运动中解组会直接切断各轴的运动指令。

相关指令：SCM_GroupEnable、SCM_GroupStop、SCM_GroupSetOverride

5.15.17 SCM_GroupBufIO

```
int SCM_GroupBufIO(SCM_CARD_HANDLE handle, int slavePosition, int bit, int value, int crd)
```

功能：缓冲数字量输出 —— 把"置某个 DO 位"这个动作压进运动缓冲，等轨迹走到当前路径末端时才执行。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
slavePosition	int	入	从站拓扑位置（0 基），规则同 SCM_WriteOutputBit
bit	int	入	站内 0 基位号，规则同 SCM_WriteOutputBit
value	int	入	非 0 = ON，0 = OFF
crd	int	入	插补坐标系号

返回值：SCM_OK = 已挂上；SCM_INVALID_ARGUMENT = 从站/位号不存在 或 crd 越界；SCM_INVALID_STATE = 路径会话还没起（没推过段）；SCM_REJECTED = 组未建 / 组已故障。

权限：需 Owner 或 Maintenance。

说明：动作挂在**调用这一刻已投递路径的末端上**，等轨迹真走到那个点，才在该周期输出。点胶、激光开关、吹气都靠它做到"在工件的这个位置动作"。典型顺序是"先推段、再挂动作"，一段一挂：

① `SCM_GroupMoveLinearAbs(...)` 推到起胶点 → ② `SCM_GroupBufIO(card, 3, 0, 1, crd)` 挂"走到起胶点才开胶" → ③ `SCM_GroupMoveLinearAbs(...)` 推到收胶点 → ④ `SCM_GroupBufIO(card, 3, 0, 0, crd)` 挂"走到收胶点才断胶" → ⑤ `SCM_GroupFlush(card, crd)` 收尾。

说明 注意：⚠ 与 `SCM_WriteOutputBit` 的区别就是这个"等"。后者**立刻**写过程映像，与路径无关——在运动中调它，阀会在你调用的那一瞬间动，而不是在工件的那个位置上动。要位置精度就用本条。

⚠ **必须先推过至少一段运动再挂。**动作锚在规划器的段边界上，路径会话还没起时没有锚点可用，这时调用返回 `SCM_INVALID_STATE`，不会被忽略。

⚠ 执行时刻的精度是一个**总线周期**（1ms 周期下最多滞后一个周期，对应的位置偏差不超过一个周期内走过的行程），这是总线周期带来的固有限制。

⚠ 一帧只承载一个动作。同一个点上既要开阀又要写模拟量，拆成相邻两条即可，两者相隔一个总线周期。

⚠ `SCM_GroupStop`（受控停）会把剩余路径连同挂在上面的动作一起丢掉——停完重推路径时动作也要重挂。

⚠ 主软件**仿真模式**下这三条同样可用，且触发时序的规则与实际设备相同（同一个路径规划器、同一份工程 I/O 通道表，点位地址解析完全一致）。可先在仿真模式下验证**流程与顺序**，再到实际设备上验证电气动作。

相关指令： `SCM_GroupBufDelay`、`SCM_GroupBufAnalog`、`SCM_WriteOutputBit`

5.15.18 SCM_GroupBufDelay

```
int SCM_GroupBufDelay(SCM_CARD_HANDLE handle, int delayMs, int crd)
```

功能：缓冲延时——轨迹走到当前路径末端时，在**路径上停** `delayMs` 毫秒，然后自己继续走。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>delayMs</code>	<code>int</code>	入	驻留毫秒数（正数）
<code>crd</code>	<code>int</code>	入	插补坐标系号

返回值： `SCM_OK` = 已挂上；`SCM_INVALID_ARGUMENT` = 延时非正 或 超上限 或 `crd` 越界；`SCM_INVALID_STATE` = 路径会话还没起；`SCM_REJECTED` = 组未建 / 组已故障。

权限：需 Owner 或 Maintenance。

说明：规划器会**先把路径减速到该点停稳**，再驻留。这一步不可省略：若不减速直接驻留，指令速度会在一个周期内从巡航速度突变为 0（相当于无限大的减速度），造成机械冲击，并很可能立即触发跟随误差保护。驻留期间轴保持在该点不动，路径队列一段不少，驻留结束自动继续。

说明 注意：⚠ 上限 = **60000 个总线周期**（1ms 周期上就是 60 秒）。超过上限会**直接报错**，不会自动截断为上限值，以免请求 5 分钟的驻留实际只停 60 秒却返回成功。

⚠ 因为要停稳，这一段的实际节拍会比不加驻留时长：减速 + 驻留 + 重新加速。对节拍敏感的工艺请把驻留点放在本来就要停的地方。

相关指令： `SCM_GroupBufIO`、`SCM_GroupBufAnalog`

5.15.19 SCM_GroupBufAnalog

```
int SCM_GroupBufAnalog(SCM_CARD_HANDLE handle, int channelIndex, double value, int crd)
```

功能：缓冲模拟量输出 —— 把"写某个模拟量通道"这个动作压进运动缓冲，走到当前路径末端才执行。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>channelIndex</code>	<code>int</code>	入	全局 I/O 通道序号，规则同 <code>SCM_WriteAnalogOutput</code>
<code>value</code>	<code>double</code>	入	工程值，编解码同 <code>SCM_WriteAnalogOutput</code>
<code>crd</code>	<code>int</code>	入	插补坐标系号

返回值： `SCM_OK` = 已挂上；`SCM_INVALID_ARGUMENT` = 通道越界 / 不是输出通道 / 是数字量位 / 宽度不是 1·2·4 字节 / `crd` 越界；`SCM_INVALID_STATE` = 路径会话还没起；`SCM_REJECTED` = 组未建 / 组已故障。

权限：需 Owner 或 Maintenance。

说明：典型用法是随位置改胶量、改激光功率 —— 在几个路径点上各挂一次，速度与胶量就对得上位置。

说明 注意：⚠ 数字量位通道会被拒，请改用 `SCM_GroupBufIO`。原因是位通道要按位改写，而本条是按字节掩码写 —— 若允许写入，同一字节中相邻的位会被一并覆盖，而返回码仍是成功。

⚠ 其余注意事项（先推段、一帧一个动作、受控停会丢、仿真模式可用但电气动作以实际设备为准）同 `SCM_GroupBufIO`。

相关指令： `SCM_GroupBufIO`、`SCM_WriteAnalogOutput`

5.16 电子齿轮

说明 主从耦合：从轴按齿比实时跟随主轴（语义对齐 PLCopen `MC_GearIn` / `MC_GearOut`）。
以从轴为单位：一根从轴同一时刻只能有一个耦合；一主带多从 = 对每根从轴各发一条 `SCM_GearIn`。

单位是轴用户单位 —— 与点位/回零同一族

说明 本节全部量一律**轴用户单位**，与「点位运动」「回零参数」同一套：
追赶加减速 / 追赶加加速 / 同步速度窗 / 同步位置窗都按**从轴**的用户单位；
齿比也按用户单位：**从轴用户位移 = 主轴用户位移 × `ratioNumerator` / `ratioDenominator`**。
主从当量不同时（例如直线主轴 + 转台从轴）软件自动折算，上位程序不必自己乘。
分子可负 = 反向；分子 0 拒；分母必须 > 0。

主轴说明

说明 主轴 = 任意已映射的轴，位置源 = **实际位置**（6064 反馈），**主轴不要求使能** ——
idle 轴被外力 / 手轮拖动照样能当主；把外部编码器映射成一根轴，它就能当主轴。
没有单独的"虚主轴"指令：拿一根（虚拟 / 仿真）轴当 `masterAxis`，
用 `SCM_AxisMoveVel` / `SCM_AxisJog` / `SCM_AxisStop` 驱动它，就是虚主轴。

受理 ≠ 完成

说明 `SCM_GearIn` / `SCM_GearOut` 都是**异步受理**：返回码只答"收下了吗"。
是否完成通过轮询 `SCM_GearGetState` 判断：`SCM_GearIn` 等 `SYNCHRONIZED`、`SCM_GearOut` 等
`DECOUPLED`，**两条轮询的收尾条件都必须含 `FAULTED`** —— 只等目标态遇到故障会一直等下去。

中止语义（接管式）

说明 `SCM_AxisStop`（从轴）= **立即断耦合并停轴**；从轴掉使能 / 故障 → 状态变 `FAULTED`
（粘滞，下次 `SCM_GearIn` 清）。`SCM_GearOut` 才是平稳减速脱开。
主软件仿真模式下啮合瞬时完成，状态只会看到 0→2。

5.16.1 SCM_GearIn

```
int SCM_GearIn(SCM_CARD_HANDLE handle, int axis, int masterAxis, double ratioNumerator, double ratioDenominator,
double catchAcceleration, double catchJerk, double syncVelocityWindow, double syncPositionWindow)
```

功能：齿轮啮合 —— 让 `axis`（从轴）按齿比实时跟随 `masterAxis`（主轴）。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>axis</code>	<code>int</code>	入	从轴轴号（必须 已使能 ）
<code>masterAxis</code>	<code>int</code>	入	主轴轴号（任意已映射轴， 不要求使能 ；位置源 = 实际位置）
<code>ratioNumerator</code>	<code>double</code>	入	齿比分子。 可负 = 反向 ；0 被拒
<code>ratioDenominator</code>	<code>double</code>	入	齿比分母。必须 > 0

<code>catchAcceleration</code>	<code>double</code>	入	追赶段加速度上限（从轴用户单位/ s^2 ）。必须 > 0 ， 无默认值
<code>catchJerk</code>	<code>double</code>	入	追赶段 Jerk（从轴用户单位/ s^3 ）。 $\leq 0 =$ T 型（不限 Jerk）
<code>syncVelocityWindow</code>	<code>double</code>	入	判"已同步"的速度窗（从轴用户单位/s）。必须 > 0
<code>syncPositionWindow</code>	<code>double</code>	入	判"已同步"的位置窗（从轴用户单位）。必须 > 0

返回值：`SCM_OK` = 已受理（**不是已同步** —— 轮询 `SCM_GearGetState`）；`SCM_INVALID_ARGUMENT` = 分母 ≤ 0 / 分子为 0 / 追赶加速度 ≤ 0 / 同步窗 ≤ 0 / 主从同轴；`SCM_INVALID_AXIS` = 轴号越界；`SCM_INVALID_STATE` = 从轴未使能；`SCM_REJECTED` = 换主轴没先脱开、或无控制权。

权限：需 Owner 或 Maintenance。

说明：耦合从受理起生效：先按 `catchAcceleration` / `catchJerk` 限幅追赶主轴（状态 `COUPLING`），速度差与位置差都进了两个同步窗之后转 `SYNCHRONIZED`，此后从轴用户位移恒等于主轴用户位移 $\times$ 分子 / 分母。

已耦合的从轴再发本条：同主轴 = **在线更新齿比 / 参数**（齿比在线生效，常用于运行中改速比）；换主轴被拒 —— 先 `SCM_GearOut`。

说明 注意：⚠ **异步受理：**返回 `SCM_OK` 只表示收下了。轮询 `SCM_GearGetState` 至 `SYNCHRONIZED`，**收尾条件必须含 `FAULTED`** —— 只等 `SYNCHRONIZED` 遇到故障会一直等。

⚠ 从轴必须**真正使能到位**再啮合：`SCM_AxisPower` 是异步受理，不等 `powered` 就发本条会因从轴未使能被拒绝。

⚠ 从轴掉使能 / 故障时耦合会被自动解除，状态变 `FAULTED`（粘滞，下次 `SCM_GearIn` 清）。

相关指令：`SCM_GearOut`、`SCM_GearGetState`、`SCM_AxisMoveVel`、`SCM_AxisStop`

5.16.2 SCM_GearOut

```
int SCM_GearOut(SCM_CARD_HANDLE handle, int axis)
```

功能：平稳脱开 —— 从轴减速停在当前位置，解除齿轮耦合。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>axis</code>	<code>int</code>	入	从轴轴号

返回值：`SCM_OK` = 已受理（未耦合的轴调它也返回 `SCM_OK`，幂等）；`SCM_INVALID_AXIS` = 轴号越界；`SCM_REJECTED` = 无控制权。

权限：需 Owner 或 Maintenance。

说明：脱开后从轴保持位置、不锁轴、不置故障，主轴不受影响。要**立即**断开（不减速）用 `SCM_AxisStop`（从轴）—— 那是接管式中止。

说明 注意：⚠ **异步受理：**轮询 `SCM_GearGetState` 至 `DECOUPLED`，**收尾条件必须含 `FAULTED`**。

相关指令： `SCM_GearIn`、`SCM_GearGetState`、`SCM_AxisStop`

5.16.3 `SCM_GearGetState`

```
int SCM_GearGetState(SCM_CARD_HANDLE handle, int axis)
```

功能：查从轴的齿轮耦合状态。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>axis</code>	<code>int</code>	入	从轴轴号。越界不报错，返回 <code>SCM_GEAR_STATE_DECOUPLED</code> (0)

返回值：⚠ **返回值就是状态码 `SCM_GEAR_STATE`（不是错误码）**，取值见本章末「齿轮状态」表；句柄无效时返回 `SCM_INVALID_ARGUMENT` (10，越出状态枚举可判)。

权限：只读，不需要控制权。

说明：`SCM_GearIn` / `SCM_GearOut` 是否完成都通过本条判断：前者等 `SYNCHRONIZED` (2)，后者等 `DECOUPLED` (0)，收尾条件都必须含 `FAULTED` (4)。`FAULTED` 是粘滞的：保持到下次 `SCM_GearIn`。

说明 注意：⚠ 主软件仿真模式下啮合 / 脱开瞬时完成，只会看到 0 和 2，不会经过 1 / 3。

相关指令： `SCM_GearIn`、`SCM_GearOut`

5.16.4 齿轮状态 `SCM_GEAR_STATE`（`SCM_GearGetState` 的返回值）

常量	值	含义
<code>SCM_GEAR_STATE_DECOUPLED</code>	0	未耦合（初始态，也是 <code>SCM_GearOut</code> 的完成态）
<code>SCM_GEAR_STATE_COUPLING</code>	1	追赶啮合中（按 <code>catchAcceleration</code> / <code>catchJerk</code> 限幅追主轴）
<code>SCM_GEAR_STATE_SYNCHRONIZED</code>	2	已同步（速度差与位置差都进了同步窗； <code>SCM_GearIn</code> 的完成态）
<code>SCM_GEAR_STATE_DECOUPLING</code>	3	平稳脱开减速中（ <code>SCM_GearOut</code> 受理后）
<code>SCM_GEAR_STATE_FAULTED</code>	4	从轴掉使能 / 故障，耦合已自动解除。 粘滞： 下次 <code>SCM_GearIn</code> 清

-

5.17 电子凸轮

说明 从轴按一张**凸轮表**（主轴相位 → 从轴位移）实时跟随主轴
（语义对齐 PLCopen `MC_CamTableSelect` / `MC_CamIn` / `MC_CamOut`）。

以从轴为单位：一根从轴同一时刻只能有一个耦合，**且与电子齿轮共用这一个名额**——
同一根从轴不能既跑齿轮又跑凸轮。

单位是轴用户单位 —— 与点位/回零同一族

说明 本节全部量一律**轴用户单位**，与其余各族同一套。分清是**哪一根轴**的用户单位：
主轴模值 `masterModulo`、关键点的 `phase`、`phaseOffset`、`blendSpan` 按**主轴**；
关键点的 `position`、`catchVelocity` 按**从轴**。 `amplitudeScale` 无量纲。

说明 表是**先定义、后绑轴**的（`SCM_CamTableBegin/AddPoint/End` 只给表号，`SCM_CamIn` 才知道
主从是谁），所以表按用户单位存着，**啮合时软件按那一对轴的当量重编一份**——
同一张表挂到不同轴对上都对，上位程序不必为某一对轴预先折算。

下发方式：三段式逐点下发

说明 `SCM_CamTableBegin` → $N \times$ `SCM_CamTableAddPoint` → `SCM_CamTableEnd`。
命令通道每次只能传递一条定长命令、不支持数组参数，因此凸轮表需逐点下发，
与坐标组插补逐段下发的方式相同。
收表（End）才做整表校验；校验失败**保留旧表**（半张表绝不生效）。
表是卡上的**全局资源**：表号 0~15，单表 ≤ 256 点，所有客户端共用。

三种啮合模式

说明 | 模式 | 从轴位置 | 啮合瞬间 |
|---|---|---|
| `SCM_CAM_ENGAGE_ABSOLUTE` | 幅值 × 表值(主轴位置 + 相位偏移) | 从轴若不在表上，按 `catchVelocity` 限速追过去 |
| `SCM_CAM_ENGAGE_RELATIVE` | 啮合位 + 幅值 × (表值 - 啮合处表值) | **无跳变**（啮合点必然落在曲线上）。最常用 |
| `SCM_CAM_ENGAGE_ADDITIVE` | 啮合位 + 幅值 × 表值 | 表零点的偏差要追 |

主轴说明

说明 主轴 = 任意已映射的轴，位置源 = **实际位置**（6064 反馈），**主轴不要求使能**——
idle 轴被外力 / 手轮拖动照样能当主。**没有单独的"虚主轴"指令**：拿一根（虚拟 / 仿真）轴
当 `masterAxis`，用 `SCM_AxisMoveVel` / `SCM_AxisJog` / `SCM_AxisStop` 驱动它。

受理 ≠ 完成

说明 `SCM_CamIn` / `SCM_CamOut` / `SCM_CamSwitchTable` 都是**异步受理**。

是否完成通过轮询 `SCM_CamGetState` 判断：`SCM_CamIn` 等 `COUPLED`、`SCM_CamOut` 等 `DECOUPLED`，
两条轮询的收尾条件都必须含 `FAULTED` —— 只等目标态遇到故障会一直等下去。

中止语义（接管式）

说明 `SCM_AxisStop`（从轴）= **立即断耦合并停轴**；从轴掉使能 / 故障 → 状态变 `FAULTED`

（粘滞，下次 `SCM_CamIn` 清）。`SCM_CamOut` 才是平稳减速脱开。

主软件仿真模式下啮合瞬时完成、换表瞬时生效，状态只会看到 0→2。

5.17.1 SCM_CamTableBegin

```
int SCM_CamTableBegin(SCM_CARD_HANDLE handle, int table, double masterModulo)
```

功能：开始定义凸轮表 —— 清该表的暂存区，并记下主轴模值。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>table</code>	<code>int</code>	入	表槽位号，0~15
<code>masterModulo</code>	<code>double</code>	入	主轴模值（主轴用户单位）。必须 > 0 —— 表在这个周期上循环

返回值：`SCM_OK` = 暂存区已就绪；`SCM_INVALID_ARGUMENT` = 表号越界 / 模值 ≤ 0；`SCM_REJECTED` = 该表正被活动耦合使用、或无控制权。

权限：需 Owner 或 Maintenance。

说明：本条只动暂存区，**不动已生效的表** —— `Begin` / `AddPoint` 推到一半放弃，旧表照常跑。

说明 **注意：**⚠ **正被活动耦合使用的表拒绝重定义**（运行中替换正在使用的表会导致从轴指令跳变）。要在线改曲线，请定义到**另一个槽位**，再用 `SCM_CamSwitchTable` 平滑切过去。

相关指令：`SCM_CamTableAddPoint`、`SCM_CamTableEnd`、`SCM_CamSwitchTable`

5.17.2 SCM_CamTableAddPoint

```
int SCM_CamTableAddPoint(SCM_CARD_HANDLE handle, int table, double phase, double position, int curve)
```

功能：向暂存区追加一个凸轮表关键点。

参数	类型	方向	说明
----	----	----	----

handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
table	int	入	表槽位号（必须已 SCM_CamTableBegin）
phase	double	入	主轴相位（主轴用户单位），落在 [0, masterModulo)
position	double	入	该相位处的从轴位移（从轴用户单位）
curve	int	入	本点到下一点之间的运动规律，取 SCM_CAM_CURVE（本章末「段曲线」表）

返回值： SCM_OK = 已收下； SCM_INVALID_STATE = 该表没先 SCM_CamTableBegin； SCM_INVALID_ARGUMENT = 曲线号越界 / 相位或位移不是有限实数； SCM_REJECTED = 已达 256 点上限、或无控制权。

权限： 需 Owner 或 Maintenance。

说明： 可乱序推（收表时按相位排序）。末点到首点（+模值）会自动补一段闭合段，曲线取末点的 curve。

说明 注意： ⚠ 单表 256 点上限，超出上限会直接报错，不会自动截断——被截掉部分点的凸轮曲线看似正常，实际是错误的。

相关指令： SCM_CamTableBegin、SCM_CamTableEnd

5.17.3 SCM_CamTableEnd

```
int SCM_CamTableEnd(SCM_CARD_HANDLE handle, int table)
```

功能： 收表 —— 整表校验通过后，该表才生效。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
table	int	入	表槽位号

返回值： SCM_OK = 表已生效； SCM_INVALID_STATE = 该表没在定义中； SCM_INVALID_ARGUMENT = 整表校验不过（点数 < 2 / 相位重复 / 相位越出 [0, 模值)）； SCM_REJECTED = 无控制权。

权限： 需 Owner 或 Maintenance。

说明： 校验三条 —— 关键点 ≥ 2 个、相位互不重复、全部落在 [0, masterModulo)。

说明 注意： ⚠ 校验失败保留旧表（半张表绝不生效），暂存区一律丢弃 —— 要重来请重新 SCM_CamTableBegin。

相关指令： SCM_CamTableBegin、SCM_CamTableAddPoint、SCM_CamIn

5.17.4 SCM_CamIn

```
int SCM_CamIn(SCM_CARD_HANDLE handle, int axis, int masterAxis, int table, int engageMode, double phaseOffset,
double amplitudeScale, double catchVelocity)
```

功能：凸轮啮合 —— 让 `axis`（从轴）按表 `table` 实时跟随 `masterAxis` 的相位。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>axis</code>	<code>int</code>	入	从轴轴号（必须 已使能 ）
<code>masterAxis</code>	<code>int</code>	入	主轴轴号（任意已映射轴， 不要求使能 ；位置源 = 实际位置）
<code>table</code>	<code>int</code>	入	已生效的表槽位号，0~15
<code>engageMode</code>	<code>int</code>	入	啮合模式，取 <code>SCM_CAM_ENGAGE_MODE</code> （本章未「啮合模式」表）
<code>phaseOffset</code>	<code>double</code>	入	加在主轴相位上的偏移（主轴用户单位）
<code>amplitudeScale</code>	<code>double</code>	入	乘在表值上的幅值缩放。 不能为 0
<code>catchVelocity</code>	<code>double</code>	入	啮合追赶限速（从轴用户单位/s）。必须 > 0， 无默认值

返回值： `SCM_OK` = 已受理（**不是已耦合** —— 轮询 `SCM_CamGetState`）；`SCM_INVALID_ARGUMENT` = 表号 / 模式越界、幅值为 0、追赶速度 ≤ 0、主从同轴；`SCM_INVALID_AXIS` = 轴号越界；`SCM_INVALID_STATE` = 从轴未使能、表未定义或还在定义中；`SCM_REJECTED` = 换表 / 换模式 / 换主轴没先脱开、或无控制权。

权限：需 Owner 或 Maintenance。

说明：耦合从受理起生效：先按 `catchVelocity` 限速追赶到凸轮位（状态 `ENGAGING`），追上后转 `COUPLED`，此后从轴每个周期的位置都由主轴相位查表直接算出。

已耦合的从轴再发本条：同主轴 + 同表 + 同模式 = 在线更新参数（`phaseOffset` / `amplitudeScale` / `catchVelocity` 即刻生效）；**换表被拒** —— 用 `SCM_CamSwitchTable`（那条才有平滑过渡）；**换模式 / 换主轴被拒** —— 先 `SCM_CamOut`。

说明 注意：⚠ **异步受理：**返回 `SCM_OK` 只表示收下了。轮询 `SCM_CamGetState` 至 `COUPLED`，**收尾条件必须含 `FAULTED`**。

⚠ `amplitudeScale` 传 0 会被拒：在绝对模式下那等于**命令从轴一路开回坐标零点**，会造成从轴大幅意外运动，而不是“从轴不动”。

⚠ `catchVelocity` 传 0 会被拒：否则它会被内部下限处理为极小值，表现为“啮合永远追不上”。

⚠ **换模式必须先脱开：**相对 / 附加模式的基准（主轴原点、从轴基位、表基值）是在啮合那一刻记录的，中途换模式 = 基准换了含义，从轴指令会跳。

相关指令： `SCM_CamOut`、`SCM_CamSwitchTable`、`SCM_CamGetState`、`SCM_CamTableEnd`、`SCM_AxisStop`

5.17.5 SCM_CamOut

```
int SCM_CamOut(SCM_CARD_HANDLE handle, int axis)
```

功能：平稳脱开 —— 从轴沿当前速度减速停在当前位置，解除凸轮耦合。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
axis	int	入	从轴轴号

返回值：SCM_OK = 已受理（未耦合的轴调它也返回 SCM_OK，幂等）；SCM_INVALID_AXIS = 轴号越界；SCM_REJECTED = 无控制权。

权限：需 Owner 或 Maintenance。

说明：脱开后从轴保持位置、不锁轴、不置故障，主轴不受影响。要**立即**断开（不减速）用 SCM_AxisStop（从轴）—— 那是接管式中止。

说明 注意：⚠ **异步受理：**轮询 SCM_CamGetState 至 DECOUPLED，**收尾条件必须含 FAULTED**。带速骤停会造成机械冲击，因此这里采用减速停止；不要用它当急停。

相关指令：SCM_CamIn、SCM_CamGetState、SCM_AxisStop

5.17.6 SCM_CamSwitchTable

```
int SCM_CamSwitchTable(SCM_CARD_HANDLE handle, int axis, int table, double blendSpan)
```

功能：运行中平滑换表 —— 在给定的主轴相位跨度内，把旧表输出平滑过渡到新表。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
axis	int	入	从轴轴号（必须正在凸轮耦合中）
table	int	入	要切过去的表槽位号（模值必须与当前耦合一致）
blendSpan	double	入	过渡宽度 = 主轴相位跨度（主轴用户单位）。必须 > 0

返回值：SCM_OK = 已受理；SCM_INVALID_ARGUMENT = 表号越界 / 过渡宽度 ≤ 0；SCM_INVALID_STATE = 该轴未在凸轮耦合中、目标表未定义或还在定义中；SCM_REJECTED = 新表模值与当前耦合不一致、仍在 ENGAGING 追赶中、或无控制权。

权限：需 Owner 或 Maintenance。

说明：过渡用五次 smoothstep（端点速度与加速度都为零），**位置与速度都连续**，不产生阶跃。这是更换工艺配方 / 产品规格时的推荐做法：先把新曲线定义到另一个表槽位，再用本条切过去。

说明

注意：⚠ 过渡随**主轴相位**推进 —— 主轴不动，过渡就不推进。
⚠ 模值不同的表会被拒：同一个相位在两张不同模值的表上指的不是同一个位置，平滑过渡也无法处理。要换模值先 `SCM_CamOut`。
⚠ 仍在 `ENGAGING`（限速追赶）时会被拒：此时从轴尚未到达曲线，再叠加换表过渡无法保证收敛。等到 `COUPLED` 再换。

相关指令： `SCM_CamIn`、`SCM_CamTableEnd`、`SCM_CamGetState`

5.17.7 `SCM_CamGetState`

```
int SCM_CamGetState(SCM_CARD_HANDLE handle, int axis)
```

功能：查该从轴的凸轮耦合状态。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>axis</code>	<code>int</code>	入	从轴轴号

返回值：⚠ **返回值本身就是 `SCM_CAM_STATE`（不是错误码）**，取值见本章末「凸轮状态」表。句柄无效时返回 `SCM_INVALID_ARGUMENT`（10，越出状态枚举，可以判）。轴号越界回 `SCM_CAM_STATE_DECOUPLED`（0）。

权限：只读，**不需要控制权**。

说明：这是 `SCM_CamIn` / `SCM_CamOut` / `SCM_CamSwitchTable` 三条异步受理指令**判断是否完成的依据**。

说明

注意：⚠ 两条轮询（`CamIn` 等 `COUPLED`、`CamOut` 等 `DECOUPLED`）的**收尾条件都必须含 `FAULTED`** —— 只等目标态，遇到从轴掉使能会一直等下去。`FAULTED` 是**粘滞的**，下次 `SCM_CamIn` 才清。

相关指令： `SCM_CamIn`、`SCM_CamOut`、`SCM_CamSwitchTable`

5.17.8 段曲线 `SCM_CAM_CURVE`（`SCM_CamTableAddPoint` 的 `curve`）

关键点到**下一个**关键点之间的运动规律。同一升程下，峰值加速度 修正梯形 < 五次多项式 < 摆线。

常量	值	含义
<code>SCM_CAM_CURVE_LINE</code>	0	直线（匀速）。端点速度 不为零 —— 跨段会有速度台阶，适合驻留段
<code>SCM_CAM_CURVE_POLY5</code>	1	五次多项式（3-4-5）。端点速度与加速度都为零 ⇒ 跨段 C2。 默认选它
<code>SCM_CAM_CURVE_CYCLOID</code>	2	摆线。端点速度 / 加速度为零；峰值加速度比 <code>POLY5</code> 大，但加速度全程连续
<code>SCM_CAM_CURVE_MODTRAP</code>	3	修正梯形。端点条件同上， 峰值加速度最小 —— 要压驱动器峰值转矩就选它

5.17.9 凸轮状态 `SCM_CAM_STATE` (`SCM_CamGetState` 的返回值)

常量	值	含义
<code>SCM_CAM_STATE_DECOUPLED</code>	0	未耦合（初始态，也是 <code>SCM_CamOut</code> 的完成态）
<code>SCM_CAM_STATE_ENGAGING</code>	1	追赶啮合中（从轴按 <code>catchVelocity</code> 限速逼近凸轮位）
<code>SCM_CAM_STATE_COUPLED</code>	2	已耦合（从轴每个周期按表跟随； <code>SCM_CamIn</code> 的完成态）
<code>SCM_CAM_STATE_DISENGAGING</code>	3	平稳脱开减速中（ <code>SCM_CamOut</code> 受理后）
<code>SCM_CAM_STATE_FAULTED</code>	4	从轴掉使能 / 故障，耦合已自动解除。粘滞：下次 <code>SCM_CamIn</code> 清

5.17.10 啮合模式 `SCM_CAM_ENGAGE_MODE` (`SCM_CamIn` 的 `engageMode`)

常量	值	含义
<code>SCM_CAM_ENGAGE_ABSOLUTE</code>	0	绝对：从轴 = 幅值 × 表值(主轴位置 + 相位偏移)。表值直接就是从轴坐标
<code>SCM_CAM_ENGAGE_RELATIVE</code>	1	相对：相位自 啮合时刻 起算，从轴 = 啮合位 + 幅值 × (表值 - 啮合处表值)。 啮合无跳变 ，最常用
<code>SCM_CAM_ENGAGE_ADDITIVE</code>	2	附加：相位同相对，从轴 = 啮合位 + 幅值 × 表值（表零点的偏差要追）

5.18 诊断、错误与常量表

说明 周期计数 / 周期耗时 / 漏周期 / WKC / 站数 / 控制角色 / 属主，全在 `SCM_GetDiagnostics` 的结构体里，不另设单条读法。

说明 排障原则：按错误码分支，不要解析消息文字。消息文字随版本 / 语言变化，错误码不变。

5.18.1 `SCM_GetDiagnostics`

```
int SCM_GetDiagnostics(SCM_CARD_HANDLE handle, SCM_DIAGNOSTICS* diagnostics)
```

功能：读运行时诊断快照。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>diagnostics</code>	<code>SCM_DIAGNOSTICS*</code>	出	诊断结构体，字段表见本章末「 <code>SCM_DIAGNOSTICS</code> 字段」。后端不可用时给零值

返回值： `SCM_OK` = 读到； `SCM_NOT_OPEN` = 句柄已关闭； `SCM_BACKEND_UNAVAILABLE` = 宿主不可用（出参给零值）。

权限：只读，不需要控制权。

说明：一次读齐总线状态、规模、周期指标、WKC、站数、控制角色与属主。界面上的状态栏用它刷新最合适——一次读完不会出现“半新半旧”的画面。

判断“控制权在不在我手上”：拿自己的 `clientId` 与出参的 `ownerClientId` 比，**别只看 `controlRole`**。

说明 注意：⚠ 后端不可用时给的是**零值诊断**。`axisCount = 0` 要当“没读到”理解，不是“这台机器没有轴”。

相关指令： `SCM_EtherCATGetWkc`、`SCM_EtherCATGetSlaveCount`、`SCM_GetLastError`

5.18.2 SCM_GetVersion

```
int SCM_GetVersion(SCM_CARD_HANDLE handle, char* buffer, int bufferSize)
```

功能：读 SDK 版本串。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>buffer</code>	<code>char*</code>	出	版本串输出缓冲（.NET 侧是 <code>StringBuilder</code> ）
<code>bufferSize</code>	<code>int</code>	入	缓冲区大小（字节 / 字符数）

返回值： `SCM_OK` = 读到； `SCM_INVALID_ARGUMENT` = 缓冲区不够（内容被截断）或指针为空。

权限：只读，不需要控制权。

说明：报障时请把这个串一并附上——SDK 与主软件必须同版本，版本串是判断配套关系的第一手依据。

相关指令： `SCM_GetDiagnostics`

5.18.3 SCM_GetOpenErrorText

```
int SCM_GetOpenErrorText(char* buffer, int bufferSize)
```

功能：读**最近一次 `SCM_Open*` 失败**的详细原因（本线程）。

参数	类型	方向	说明
<code>buffer</code>	<code>char*</code>	出	文本输出缓冲（.NET 侧是 <code>StringBuilder</code> ）
<code>bufferSize</code>	<code>int</code>	入	缓冲区大小

返回值： `SCM_OK`（从没失败过时给空串）； `SCM_INVALID_ARGUMENT` = 缓冲区不够或指针为空。

权限： 无。

说明： ⚠ 全表唯一不带 `handle` 的指令 —— 打开失败时尚未取得句柄。

连接失败时只报错误码是不够的： `SCM_BACKEND_UNAVAILABLE` 只告诉你"连不上"，不告诉你"哪里对不上"。这条消息里逐项列出双方的轴数、I/O 点数、周期，照着对就能定位是规格不符、共享内存不存在，还是布局指纹不匹配。

成功打开会清空它。

说明 注意： ⚠ 它是按线程记录的。在别的线程里连接失败，回到主线程读不到内容。

相关指令： `SCM_Open`、`SCM_OpenProjectConfig`

5.18.4 SCM_GetLastError

```
int SCM_GetLastError(SCM_CARD_HANDLE handle, int* code)
```

功能： 读最近一次指令调用的错误码。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>code</code>	<code>int*</code>	出	最近一次调用的 <code>SCM_RESULT</code> 。成功即 <code>SCM_OK</code>

返回值： `SCM_OK` = 读到； `SCM_INVALID_ARGUMENT` = 出参指针为空。

权限： 只读，不需要控制权。

说明： 每次调用都会刷新，所以要紧跟在被查的那条调用之后读。

主要用途是读函数：读函数把值写进出参、错误码不在返回值里的场合（例如要区分"真的读到 0"与"读失败给的默认 0"），就靠这一条。

说明 注意： ⚠ 紧跟着读，中间别插别的 SDK 调用 —— 插一条它就被刷新了。

相关指令： `SCM_GetLastReason`、`SCM_GetErrorText`

5.18.5 SCM_GetLastReason

```
int SCM_GetLastReason(SCM_CARD_HANDLE handle, int* reason)
```

功能： 读最近一次被拒的结构化原因码。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄

reason	int*	出	原因码，取值见本章末「结构化拒绝原因码」表。拿不到时给 0（None）
--------	------	---	-------------------------------------

返回值：SCM_OK = 读到；SCM_INVALID_ARGUMENT = 出参指针为空。

权限：只读，不需要控制权。

说明：排障请按它分支 —— 这是全表最该用的一条诊断指令。收到 SCM_REJECTED 之后紧跟着读它，就能知道到底是"没控制权"还是"总线没启动"还是"轴被别人占了"。
拿不到码（非拒绝类失败）时为 None，此时才去看文案。

说明 注意：⚠ 别拿文案做程序分支。SCM_GetLastMessage 的内容随版本与语言变，码不变。

相关指令：SCM_GetLastMessage、SCM_GetLastError

5.18.6 SCM_GetLastMessage

```
int SCM_GetLastMessage(SCM_CARD_HANDLE handle, char* buffer, int bufferSize)
```

功能：读最近一次错误的文案。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
buffer	char*	出	文案输出缓冲（.NET 侧是 StringBuilder）
bufferSize	int	入	缓冲区大小

返回值：SCM_OK = 读到（没有错误时给空串）；SCM_INVALID_ARGUMENT = 缓冲区不够或指针为空。

权限：只读，不需要控制权。

说明：文案是给人看的 —— 打进日志、显示在界面上。

说明 注意：⚠ 不要拿它做程序分支。文案随版本 / 语言变，码不变；要分支请用 SCM_GetLastReason。

相关指令：SCM_GetLastReason、SCM_GetErrorText

5.18.7 SCM_GetErrorText

```
int SCM_GetErrorText(SCM_CARD_HANDLE handle, int code, char* buffer, int bufferSize)
```

功能：把错误码翻译成文案（纯查表，不看"最近一次"）。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄

code	int	入	要翻译的错误码
buffer	char*	出	文案输出缓冲（.NET 侧是 <code>StringBuilder</code> ）
bufferSize	int	入	缓冲区大小

返回值：`SCM_OK` = 翻到；`SCM_INVALID_ARGUMENT` = 缓冲区不够或指针为空。

权限：只读，不需要控制权。

说明：与 `SCM_GetLastMessage` 的差别：本条是纯查表，你给什么码它翻什么码；`SCM_GetLastMessage` 给的是"最近一次错误的现场描述"，通常信息更多。日志里两条一起打最有用。

相关指令：`SCM_GetLastMessage`、`SCM_GetLastError`

5.18.8 错误码 `SCM_RESULT`

常量	值	含义
<code>SCM_OK</code>	0	成功
<code>SCM_NOT_OPEN</code>	1	卡未打开
<code>SCM_ALREADY_OPEN</code>	2	卡已打开
<code>SCM_INVALID_STATE</code>	3	当前状态下该操作无效
<code>SCM_INVALID_AXIS</code>	4	轴号非法
<code>SCM_BACKEND_UNAVAILABLE</code>	5	运行时后端不可用（宿主没起 / 断连）
<code>SCM_TIMEOUT</code>	6	超时
<code>SCM_REJECTED</code>	7	被运行时拒绝（细分原因看 <code>SCM_GetLastReason</code> ）
<code>SCM_FAULTED</code>	8	轴或运行时处于故障态
<code>SCM_NOT_SUPPORTED</code>	9	该运行时不支持此操作
<code>SCM_INVALID_ARGUMENT</code>	10	参数非法

5.18.9 错误处理约定（适用于全部指令）

- 打开成功后的读写一律返回错误码，绝不崩溃调用方进程。
- 读函数失败时出参给默认值（0 / 零值）并返回原因码：
- 全局序号越界（如纯输出拓扑 DI=0）的 `SCM_ReadInput` → 出参 0，返回 `SCM_INVALID_ARGUMENT`；
- 轴号非法的 `SCM_AxisGet*` → 出参 0 / 零值，返回 `SCM_INVALID_AXIS`；
- 未映射的从站 / 通道读（`SCM_ReadInputBit` / `SCM_Read*Word` / `Analog` / `ChannelRaw`）→ 出参 0，返回 `SCM_OK`（宿主侧判定"这个点没配"，不是错误）。
- 被拒（`SCM_REJECTED`）时再读 `SCM_GetLastReason` 拿结构化原因。

5.18.10 结构化拒绝原因码（SCM_GetLastReason 的出参）

排障按这张表分支。拿不到码时为 `None`，那时才去看文案。

值	名称	含义与处置
0	<code>None</code>	未分类（旧宿主，或该失败没有结构化原因）—— 看文案
1	<code>NoControl</code>	未持有控制权：先 <code>SCM_RequestOwner</code>
2	<code>NotOwner</code>	控制权在别的程序手上
3	<code>BusNotStarted</code>	本程序未武装：先 <code>SCM_RequestOwner</code> （拿到控制权即自动武装），或 <code>SCM_EtherCATStart</code>
4	<code>InvalidAxis</code>	轴号越界，或该轴不是已映射的硬件轴
5	<code>SoftRealtimeNot Allowed</code>	后端不开放连续伺服运动（插补 / 凸轮 / 齿轮 / 机器人）。请确认主软件已在硬实时下激活“实时运行”。该名称沿用早期版本，名称与码值均保持不变
6	<code>AxisClaimedByOther</code>	该轴已被别的运动通道驱动（插补 / CNC / 软卡组 / 界面）
7	<code>HostNotReady</code>	主软件未就绪（过程数据未运行等前置不满足）
8	<code>HomingNotEnabled</code>	该轴没启用任何回零引擎：轴页既没勾“启用控制器回零”，回零模式也不是“驱动器回原点”
9	<code>HomingSwitchNot Configured</code>	启用了控制器回零，但原点开关点位没配，或在过程映像里解析不到
10	<code>AxisMoving</code>	轴在动（或被坐标组接管），该操作要求先停稳 —— 位置定义 / 当前位置置零式回零会用到
11	<code>HomingMethodUnsupported</code>	驱动器声明的 0x60E3（支持的回零方式）里没有当前配置的方式号。⚠ 只有驱动器 确实声明了 0x60E3 才会给这个码；未声明支持表的驱动器不做此项检查
99	<code>Other</code>	其它，细节见文案与主软件日志

5.18.11 回零参数号 SCM_HOME_PARAM

说明 速度 / 偏移 / 行程上限一律**轴用户单位**（与点位族同一套），换算在宿主侧做。
回零进行中禁止改参数（宿主会拒），否则状态机会读到半新半旧的一组。

常量	值	含义
<code>SCM_HOME_ENGINE</code>	1	回零方式：0=驱动器回零（引擎①，CiA402 6098） / 1=控制器回零（引擎②，读 I/O 原点开关） / 2=当前位置置零（无运动，对应 CiA402 method 35/37；绝对值编码器机型的常用方式）
<code>SCM_HOME_SWITCH_SLAVE</code>	2	引擎②：原点开关从站拓扑位置（0 基）； <code>-1</code> = 未配，回零被拒

SCM_HOME_SWITCH_BIT	3	引擎②：开关在该从站的站内输入位号（0 基）
SCM_HOME_SWITCH_ACTIVE_HIGH	4	引擎②：有效电平，1=高有效 / 0=低有效
SCM_HOME_DIRECTION	5	引擎②：粗找方向，+1 / -1
SCM_HOME_FAST_VELOCITY	6	引擎②：粗找速度（用户单位/s）
SCM_HOME_SLOW_VELOCITY	7	引擎②：慢爬速度（用户单位/s）；回零精度基本由它决定
SCM_HOME_OFFSET	8	回零偏移（用户单位）：找到开关沿后的停靠位置；两引擎共用
SCM_HOME_TIMEOUT_MS	9	无进展超时（ms）：轴静止超过它才判失败，运动中不计时
SCM_HOME_TRAVEL_LIMIT	1 0	搜索行程上限（用户单位）；0=不限行程（仅受 10 分钟总时长限制）
SCM_HOME_DRIVE_METHOD	1 1	引擎①：CiA402 回零方式号（写 6098），取值查驱动器手册
SCM_HOME_DRIVE_VELOCITY	1 2	引擎①：回零速度（用户单位/s，写 6099）

5.18.12 回零阶段 SCM_HOMING_PHASE （SCM_AxisGetHomingState 的返回值）

常量	值	含义
SCM_HOMING_PHASE_IDLE	0	空闲 / 未在回零（是否已回零看 SCM_AXIS_STATUS.homed）
SCM_HOMING_PHASE_SEARCHING	1	粗找：按方向快速移动找原点开关
SCM_HOMING_PHASE_APPROACHING	2	慢爬：反向慢速精定位开关沿
SCM_HOMING_PHASE_OFFSETTING	3	移到偏移停靠点
SCM_HOMING_PHASE_HOMED	4	回零完成（已置软件原点）
SCM_HOMING_PHASE_ERROR	5	回零错误（超时 / 撞行程上限 / 驱动故障），不含"被中止"
SCM_HOMING_PHASE_ABORTED	6	被中止（中止=再来一次就行，错误=要去排查）

5.18.13 回零失败原因 SCM_HOMING_RESULT （SCM_AxisGetHomingResult 的返回值）

常量	值	含义
----	---	----

SCM_HOMING_NONE	0	未失败，或宿主版本过旧不支持本功能
SCM_HOMING_FEEDBACK_INVALID	1	轴反馈无效（读不到位置 / 状态）
SCM_HOMING_NOT_POWERED	2	轴未使能：先 SCM_AxisPower 并等 powered
SCM_HOMING_DRIVE_FAULT	3	回零途中驱动器报故障
SCM_HOMING_TRAVEL_LIMIT_EXCEEDED	4	超行程上限：通常压根没碰到开关（方向反 / 点位配错 / 接线）
SCM_HOMING_STALL_TIMEOUT	5	无进展超时（轴静止超 SCM_HOME_TIMEOUT_MS）
SCM_HOMING_HARD_DEADLINE	6	总时长超过 10 分钟上限（多为往返振荡，请检查开关信号）
SCM_HOMING_ABORTED	7	被上层中止
SCM_HOMING_EXCEPTION	8	状态机异常
SCM_HOMING_DRIVE_HOMING_ERROR	9	引擎①特有： 驱动器报回零错误（6041 bit13）。与 DRIVE_FAULT（bit3 驱动器故障）是两回事——bit13 是“这次回零走不通”（方式号与限位 / 原点开关接线方向不匹配最常见），驱动器本身没坏
SCM_HOMING_BUS_STOPPED	10	引擎①特有： 回零途中过程数据停了（总线停 / 激活运行被中断），控制字不再下发
SCM_HOMING_MODE_SWITCH_FAILED	11	引擎①特有： 驱动器没能进入回零模式（6060=6 写不进，或写了但 6061 不回显 6）。 处置与其它失败完全不同 ——多半是这台驱动器不支持 EtherCAT 回零模式，或要求先停使能才允许切模式；改回零方式号、改接线都没用

5.18.14 SCM_AXIS_STATUS 字段

字段	C 类型	.NET 类型	含义
axis	int	int	轴号
state	int	AxisState	轴状态机
powered	int	bool	已使能（用于判断 SCM_AxisPower 是否完成）
faulted	int	bool	故障中（用于判断 SCM_AxisReset 是否完成）

homed	int	bool	已回零
actualPosition	double	double	实际位置（轴用户单位）
targetPosition	double	double	目标位置（轴用户单位）
actualVelocity	double	double	实际速度（轴用户单位/s，606C 实测；带符号，方向即符号）
followingError	double	double	跟随误差（轴用户单位）
statusWord	unsigned short	ushort	CiA402 状态字（6041）
errorCode	int	int	驱动器故障码
commandVelocity	double	double	规划速度 （带符号）：这是"要它走多快"，actualVelocity 是"它实际多快"

5.18.15 SCM_DIAGNOSTICS 字段

字段	C 类型	含义
busState	int	总线状态
axisCount / inputCount / outputCount	int	轴数 / 数字量输入点数 / 输出点数
cycleCounter	long long	周期计数
cyclePeriodUs	int	周期（μs）
lastCycleTimeUs / maxCycleTimeUs	double	最近 / 最大周期耗时（μs）
missedCycleCount	long long	漏周期数
workCounter	int	WKC（不是站数）
controlRole	int	本程序的控制角色（Monitor / Owner / Maintenance）
ownerClientId	int	当前属主的 clientId
ownerHeartbeatCounter	long long	属主心跳计数
slaveCount	int	在线从站数，-1 = 不可得（与"真的 0 站"分开）

5.18.16 SCM_AXIS_PLC_STATUS 字段（PLCopen 定义）

解码规则与主软件 MC_ReadStatus 一字不差：

字段	含义
----	----

valid	本次读取有效（轴号非法时为 0）
errorStop	故障
disabled	未使能且非故障
standStill	已使能、非故障、无活动运动
discreteMotion	定位运动中（SCM_AxisMoveAbs / SCM_AxisMoveRel）
continuousMotion	连续运动中（SCM_AxisMoveVel / SCM_AxisJog，含点动收手减速段）
homing	控制器回零进行中（引擎②；主软件 MC_Home 未绑定时恒 0，规则与主软件一致）
synchronizedMotion	保留，恒 0（同主软件）

5.18.17 轴参数号 SCM_AXIS_PARAM（SCM_AxisGetParam 的 paramId）

只读——这些值归主软件轴页所有。单位一律轴用户单位（与点位族、回零族同一套）。

号	名	含义
0	SCM_AXIS_SCALE	轴当量：脉冲/用户单位（= 轴页换算比的绝对值）。恒 > 0。建组时原样填给 SCM_GroupConfigAxis 的 scale
1	SCM_AXIS_INVERT_DIRECTION	方向取反 0 / 1。⚠ 不在当量里（当量恒为正）；取反由主软件两侧对称施加，客户端别自己乘符号
2	SCM_AXIS_SOFT_LIMIT_ENABLED	软限位是否启用 0 / 1
3	SCM_AXIS_SOFT_LIMIT_POSITIVE	软限位正限（用户单位）
4	SCM_AXIS_SOFT_LIMIT_NEGATIVE	软限位负限（用户单位）
5	SCM_AXIS_MAX_VELOCITY	最大速度限制（用户单位/s）= 轴页「动态」的默认速度上限；0 = 未配/不限。主软件 CNC/插补页建组给规划器的逐轴限速就是它，SCM_CncOpen 也会逐轴取它配组——想跑出与页面一致的限速，建组时把它填给 SCM_GroupConfigAxis 的 vmax
6	SCM_AXIS_DEFAULT_ACCELERATION	轴页「动态」默认加速度（用户单位/s²）；0 = 未配

7	SCM_AXIS_DE FAULT_DECELE RATION	轴页「动态」默认减速度（用户单位/s ² ）；0 = 未配
8	SCM_AXIS_DE FAULT_JERK	轴页「动态」默认 Jerk（用户单位/s ³ ）；0 = 梯形

⚠ 0 在这一族里是合法值（软限位可以配成 0，"未配"也是 0）。所以"读到没读到"一律看 SCM_AxisGetParam 的返回码，别看值。

5.19 CNC / G 代码通道

说明 G 代码程序在你自己的进程里解释并逐段下发，最终发往坐标组的就是 SCM_Group* 指令（你也可以直接调用它们）。G 代码解释与转换使用的是与主软件 CNC 页同一套实现。

说明 通道号就是 crd，不另立一套编号：跑起来之后 SCM_GroupGetStatus(crd) 查的就是同一个组，SCM_GroupGetSpace(crd) 看的就是一条队列。

说明 坐标与进给一律轴用户单位（G 代码本来就这么写）。当量无需手动设置——SCM_CncOpen 会逐轴向主软件查询；查询不到且未显式设置时，SCM_CncCycleStart 会拒绝启动，不会按 1.0 运行。

说明 典型顺序：SCM_CncOpen → SCM_CncSetParam（可选）→ SCM_CncLoadFile / SCM_CncLoadText → SCM_CncCompile → SCM_CncCycleStart → 轮询 SCM_CncGetStatus →（SCM_CncFeedHold / SCM_CncSetOverride / SCM_CncStop）→ SCM_CncReset / SCM_CncClose。

说明 方言与支持的 G/M 码见主软件帮助「CNC / G 代码通道」一章。⚠ F 的单位是"用户单位/秒"，不是 mm/min——按 FANUC 习惯写 F100 会差 60 倍。

5.19.1 SCM_CncOpen

```
int SCM_CncOpen(SCM_CARD_HANDLE handle, int crd, int axisX, int axisY, int axisZ, int axisA, int axisC)
```

功能：在坐标系 crd 上建一个 CNC 通道。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
crd	int	入	坐标系号，同时就是本 CNC 通道的编号
axisX	int	入	X 轴号，0 基

<code>axisY</code>	<code>int</code>	入	Y 轴号
<code>axisZ</code>	<code>int</code>	入	Z 轴号
<code>axisA</code>	<code>int</code>	入	第 4 轴 (A) 轴号; <code>-1</code> = 不用
<code>axisC</code>	<code>int</code>	入	第 5 轴 (C) 轴号; <code>-1</code> = 不用

返回值: `SCM_OK` = 建好; `SCM_INVALID_ARGUMENT` = `crd` 为负 / X、Y、Z 没给全; `SCM_INVALID_STATE` = 该 `crd` 上已经有 CNC 通道 (先 `SCM_CncClose`)。

权限: 建通道本身不动硬件, 不需要控制权; 真正下发要在 `SCM_CncCycleStart`, 那时需要控制权。

说明: 建通道时会逐轴向主软件问当量 (等价于 `SCM_AxisGetParam` 的 `SCM_AXIS_SCALE`), 填进本通道的 `SCM_CNC_PARAM_SCALE_*` ——你不用照着轴页抄。要覆盖再用 `SCM_CncSetParam`。
同时会向主软件问 **M 功能输出映射** (`SCM_CNC_PARAM` 的 30~38: 主轴正/反转位、冷却位、S 模拟量通道与增益/上限) ——在主软件 CNC 参数页配置后随工程保存, 程序中无需再配置。**通道对号约定:** `crd + 1` = 页面通道号 (页面 `CNC-001` ↔ SDK `crd 0`)。查询不到 (通道号不对应, 或主软件版本较旧) 时保持"未配" (slave/通道 = `-1`), 启动时再根据程序是否用到相应 M 码判断能否运行。
A/C 轴只在程序里真的出现过 `A` 或 `C` 字 (或用了 `G43.4`) 时才会入组。

说明 注意: ⚠ 当量既问不到、你也没显式设时, `SCM_CncCycleStart` 会**拒绝启动**。不要绕过该检查直接按 `1.0` 运行——当量填错会使整段路径按错误比例运动, 而返回码全部是成功。

相关指令: `SCM_CncClose`、`SCM_AxisGetParam`、`SCM_GroupConfigAxis`

5.19.2 `SCM_CncClose`

```
int SCM_CncClose(SCM_CARD_HANDLE handle, int crd)
```

功能: 关掉 CNC 通道, 内含解组交还轴。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	卡句柄
<code>crd</code>	<code>int</code>	入	通道号

返回值: `SCM_OK` = 关掉了; `SCM_INVALID_ARGUMENT` = 该 `crd` 上没有 CNC 通道。

权限: 需要控制权 (要下发解组)。

说明: `SCM_Close` 关卡时会自动收掉所有 CNC 通道, 正常退出路径不用逐个关。

相关指令: `SCM_CncOpen`、`SCM_CncReset`

5.19.3 SCM_CncSetParam

```
int SCM_CncSetParam(SCM_CARD_HANDLE handle, int crd, int paramId, double value)
```

功能：设通道参数。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	卡句柄
crd	int	入	通道号
paramId	int	入	参数号，取 SCM_CNC_PARAM（见本章末「CNC 通道参数号」表）
value	double	入	参数值，单位见参数号表

返回值：SCM_OK = 设好；SCM_INVALID_ARGUMENT = 参数号未知 / 该 crd 上没有通道。

权限：只改本地通道配置，不需要控制权。

说明：这些都是编译期参数——运行中改只对下一次 SCM_CncCompile 生效，已经推进规划器的段改不了。要在线调速度请用 SCM_CncSetOverride。

说明 注意：⚠ SCM_CNC_PARAM_SCALE_* 默认已由 SCM_CncOpen 从主软件取到，除非你确实要覆盖，否则别设。

相关指令：SCM_CncGetParam、SCM_CncSetOverride

5.19.4 SCM_CncGetParam

```
int SCM_CncGetParam(SCM_CARD_HANDLE handle, int crd, int paramId, double* value)
```

功能：读通道参数。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	卡句柄
crd	int	入	通道号
paramId	int	入	参数号，取 SCM_CNC_PARAM
value	double*	出	参数值。失败时给 0

返回值：SCM_OK = 读到；SCM_INVALID_ARGUMENT = 参数号未知 / 该 crd 上没有通道。

权限：只读，不需要控制权。

说明：没设过的参数读回默认值。想确认 SCM_CncOpen 到底从主软件取到了什么当量，读一下 SCM_CNC_PARAM_SCALE_X 就知道。

相关指令：SCM_CncSetParam

5.19.5 SCM_CncLoadFile

```
int SCM_CncLoadFile(SCM_CARD_HANDLE handle, int crd, const char* path)
```

功能：从磁盘载入 .nc 程序。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	卡句柄
crd	int	入	通道号
path	const char*	入	程序文件路径，UTF-8

返回值：SCM_OK = 载入了；SCM_INVALID_ARGUMENT = 路径为空 / 读不到文件 / 该 crd 上没有通道；SCM_INVALID_STATE = 程序正在运行（先 SCM_CncStop / SCM_CncReset）。

权限：只动本地缓存，不需要控制权。

说明：在你自己的进程里读盘，主软件不参与、也不需要看得见这个文件。载入只是把文本收下，语法错误要 SCM_CncCompile 才报。

相关指令：SCM_CncLoadText、SCM_CncCompile

5.19.6 SCM_CncLoadText

```
int SCM_CncLoadText(SCM_CARD_HANDLE handle, int crd, const char* text)
```

功能：直接传入程序文本，无需先写成文件。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	卡句柄
crd	int	入	通道号
text	const char*	入	程序正文，UTF-8；换行用 \n 或 \r\n 都行

返回值：同 SCM_CncLoadFile（少一个“读不到文件”）。

权限：只动本地缓存，不需要控制权。

说明：程序由你的软件直接生成（例如上位排版软件输出）时使用本条，无需先写入文件。

相关指令：SCM_CncLoadFile、SCM_CncCompile

5.19.7 SCM_CncCompile

```
int SCM_CncCompile(SCM_CARD_HANDLE handle, int crd, int* moveCount, double* totalLength)
```

功能：编译已载入的程序（词法/语法 + 刀补几何变换 + 全程几何试算）。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	卡句柄
crd	int	入	通道号
moveCount	int*	出	编译出的运动段数。失败时给 0
totalLength	double*	出	路径总长（轴用户单位）。失败时给 0

返回值：SCM_OK = 编译通过；SCM_INVALID_ARGUMENT = 有编译错误（用 SCM_CncGetErrorAt 逐条取，带源行号）；SCM_INVALID_STATE = 还没载入程序 / 程序正在运行。

权限：纯本地计算，不需要控制权。

说明：编译起点取轴当前位置——所以同一份程序在不同起点编出来的绝对坐标不同（G91 相对编程除外）。SCM_CncCycleStart 在没编译过时自己先编一次，所以你也可以跳过这条；单独调它的价值是在启动之前就把语法错误摆给操作员看。

说明 注意：⚠ 编译通过不等于能启动：程序坐标系与机床当前位置对不上时，SCM_CncCycleStart 仍会拒绝启动。

相关指令：SCM_CncGetErrorCount、SCM_CncGetErrorAt、SCM_CncCycleStart

5.19.8 SCM_CncGetErrorCount

```
int SCM_CncGetErrorCount(SCM_CARD_HANDLE handle, int crd, int* count)
```

功能：上次编译的错误条数。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	卡句柄
crd	int	入	通道号
count	int*	出	错误条数；0 = 编译通过

返回值：SCM_OK；SCM_NOT_OPEN = 句柄已关闭。

权限：只读，不需要控制权。

说明：拿它决定循环取几条错误。

相关指令：SCM_CncGetErrorAt

5.19.9 SCM_CncGetErrorAt

```
int SCM_CncGetErrorAt(SCM_CARD_HANDLE handle, int crd, int index, int* line, char* buffer, int bufferSize)
```

功能：取第 `index` 条编译错误的源行号与文案。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	卡句柄
<code>crd</code>	<code>int</code>	入	通道号
<code>index</code>	<code>int</code>	入	错误下标, 0 基
<code>line</code>	<code>int*</code>	出	源行号 (1 基)。失败时给 <code>0</code>
<code>buffer</code>	<code>char*</code>	出	错误文案, UTF-8, 以 <code>\0</code> 结尾
<code>bufferSize</code>	<code>int</code>	入	缓冲字节数, 含结尾 <code>\0</code>

返回值：`SCM_OK` = 取到；`SCM_INVALID_ARGUMENT` = 下标越界 / 该 `crd` 上没有通道 / 缓冲装不下（已截断）。

权限：只读，不需要控制权。

说明：行号是程序里的实际行号，不是段号——可以直接拿去在你的编辑器里定位。建议缓冲给 512 字节以上。

说明 注意：⚠ 缓冲区不足时内容会被截断，并返回 `SCM_INVALID_ARGUMENT` 提示发生了截断。

相关指令：`SCM_CncGetErrorCount`、`SCM_CncCompile`

5.19.10 SCM_CncCycleStart

```
int SCM_CncCycleStart(SCM_CARD_HANDLE handle, int crd)
```

功能：循环启动。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	卡句柄
<code>crd</code>	<code>int</code>	入	通道号

返回值：`SCM_OK` = 起来了；`SCM_INVALID_ARGUMENT` = 编译不过 / 投递被拒（文案见 `SCM_GetLastMessage`）；`SCM_INVALID_STATE` = 已在运行 / 当量不明 / 程序坐标系与机床位置对不上；`SCM_REJECTED` = 没有控制权、总线没起、轴被别的通道占着。

权限：需要控制权（`SCM_RequestOwner`）。

说明：一条指令四种语义，按当前状态定：没跑时=从头启动（必要时自己先编译一次）；单段模式下=推下一块；进给保持中=解除保持并从停处续跑；`M0 / M1` 程序停中（`state` = `SCM_CNC_STATE_PROGRAM_STOP`）=从停

点续跑。

说明 注意：⚠ 它会先做一道**起步跨度自检**：程序的首个目标点离机床当前位置远得离谱、而程序本身尺寸又很小时，说明这份程序是按另一个原点写的——**当场拒绝**，不会以一条巨大的快移开场。收到这个拒绝时请先回零、或用 `SCM_AxisSetPosition` 把机床对到程序零点，或改用相对编程（`G91`）。
⚠ 轴必须先**真正使能到位**。`SCM_AxisPower` 是异步受理，返回 `SCM_OK` 只表示收下了——不等 `powered` 就启动会在建组时被拒绝。

相关指令： `SCM_CncFeedHold`、`SCM_CncStop`、`SCM_CncGetStatus`

5.19.11 SCM_CncFeedHold

```
int SCM_CncFeedHold(SCM_CARD_HANDLE handle, int crd, int hold)
```

功能：进给保持 / 解除。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	卡句柄
<code>crd</code>	<code>int</code>	入	通道号
<code>hold</code>	<code>int</code>	入	非 <code>0</code> = 保持； <code>0</code> = 解除并续跑

返回值： `SCM_OK`；`SCM_INVALID_STATE` = 程序没在运行；`SCM_REJECTED` = 没有控制权。

权限：需要控制权。

说明：保持 = 把进给倍率压到 `0`，**队列一段不少**，轴停在路径上；解除（或再 `SCM_CncCycleStart`）从停处续跑。

说明 注意：⚠ 保持是**减速到零**不是急停——刚下这条时轴还在减速坡上，别拿那一刻的位置当“已停稳”。判断是否停稳请看 `SCM_CncGetStatus` 的 `pathVelocity`。
⚠ 要**丢弃**剩余路径请用 `SCM_CncStop`，那条停下来就回不去了。

相关指令： `SCM_CncCycleStart`、`SCM_CncStop`、`SCM_CncSetOverride`

5.19.12 SCM_CncStop

```
int SCM_CncStop(SCM_CARD_HANDLE handle, int crd)
```

功能：停止（受控停，剩余路径丢弃）。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	卡句柄
<code>crd</code>	<code>int</code>	入	通道号

返回值： `SCM_OK` ； `SCM_INVALID_STATE` = 程序没在运行； `SCM_REJECTED` = 没有控制权。

权限： 需要控制权。

说明： 按 `SCM_CNC_PARAM_PATH_DECELERATION` 减速停车，队列里没走的段全部丢弃。停完通道回 `SCM_CNC_STATE_IDLE` ，要再跑得从 `SCM_CncCycleStart` 重来。

说明 注意： ⚠ 要能续跑请用 `SCM_CncFeedHold` 。

相关指令： `SCM_CncFeedHold` 、 `SCM_CncReset`

5.19.13 `SCM_CncReset`

```
int SCM_CncReset(SCM_CARD_HANDLE handle, int crd)
```

功能： 复位通道：清路径 + 解组 + 清状态。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	卡句柄
<code>crd</code>	<code>int</code>	入	通道号

返回值： `SCM_OK` ； `SCM_INVALID_ARGUMENT` = 该 `crd` 上没有通道。

权限： 需要控制权（要下发解组）。

说明： 程序文本与参数留着，清掉的是编译产物与运行状态——复位后重新 `SCM_CncCycleStart` 会按当前轴位置重编一次。故障后可用本条恢复通道。

相关指令： `SCM_CncStop` 、 `SCM_CncClose`

5.19.14 `SCM_CncSetOverride`

```
int SCM_CncSetOverride(SCM_CARD_HANDLE handle, int crd, double percent)
```

功能： 进给倍率。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	卡句柄
<code>crd</code>	<code>int</code>	入	通道号
<code>percent</code>	<code>double</code>	入	倍率百分比，钳到 <code>0 ~ 150</code>

返回值： `SCM_OK` ； `SCM_REJECTED` = 没有控制权。

权限： 需要控制权。

说明： 在线可调，随时生效，不用停车。程序没在跑时设的值会在下次启动时生效。

说明 注意：⚠ `0` 就是进给保持（与 `SCM_CncFeedHold` 同一个机制），队列一段不少。

相关指令： `SCM_CncFeedHold`、 `SCM_GroupSetOverride`

5.19.15 `SCM_CncSetSingleBlock`

```
int SCM_CncSetSingleBlock(SCM_CARD_HANDLE handle, int crd, int enable)
```

功能：单段模式开关。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	卡句柄
<code>crd</code>	<code>int</code>	入	通道号
<code>enable</code>	<code>int</code>	入	非 <code>0</code> = 开单段

返回值： `SCM_OK`； `SCM_INVALID_STATE` = 程序正在运行（先停）。

权限：只改本地通道配置，不需要控制权。

说明：单段模式下每次 `SCM_CncCycleStart` 只推一块并让它走完，停稳后等你再点。首件试切与对刀时用它。

说明 注意：⚠ 单段是每块一条独立小路径，块与块之间必然停车——所以拐角过渡（ `SCM_CNC_PARAM_CORNER_TOLERANCE` ）在单段下不起作用。

相关指令： `SCM_CncCycleStart`、 `SCM_CncGetStatus`

5.19.16 `SCM_CncGetStatus`

```
int SCM_CncGetStatus(SCM_CARD_HANDLE handle, int crd, SCM_CNC_STATUS* status)
```

功能：读通道状态：状态机、当前行、进度、DRO。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	卡句柄
<code>crd</code>	<code>int</code>	入	通道号
<code>status</code>	<code>SCM_CNC_STATUS*</code>	出	状态快照，字段见本章末「CNC 通道状态」表

返回值： `SCM_OK`； `SCM_INVALID_ARGUMENT` = 该 `crd` 上没有通道。

权限：只读，不需要控制权。

说明：界面上的运行状态显示应轮询本条。判断程序是否运行完毕看 `state` 是否变为 `SCM_CNC_STATE_DONE`；判断是否出错看它是否变为 `SCM_CNC_STATE_ERROR`，详细文案用 `SCM_GetLastMessage` 取。
`currentLine` 是程序里的实际行号，可以直接高亮到你的编辑器上。

说明 注意：⚠️ RTCP（G43.4）会话的 `positionX/Y/Z` 给的是工件系刀尖点，不是机床轴位置——两者不是一回事，拿它去摆机床模型会画错。
⚠️ 程序跑到一半失败时（比如某一行无法下发），状态会变 `SCM_CNC_STATE_ERROR` 而不会有额外的返回码送到你手上——该错误发生在后台逐段下发的过程中。所以轮询状态是必须的，不能只看各条指令的返回码。

相关指令： `SCM_CncCycleStart`、`SCM_GetLastMessage`、`SCM_GroupGetStatus`

5.19.17 CNC 通道状态 `SCM_CNC_STATE`（`SCM_CNC_STATUS.state`）

值	名	含义
0	<code>SCM_CNC_STATE_IDLE</code>	空闲：没载入 / 已停止 / 已复位
1	<code>SCM_CNC_STATE_READY</code>	编译通过，还没启动
2	<code>SCM_CNC_STATE_RUNNING</code>	运行中
3	<code>SCM_CNC_STATE_PAUSED</code>	进给保持中（队列保留）
4	<code>SCM_CNC_STATE_DONE</code>	程序走完（组保持，可直接再启动）
5	<code>SCM_CNC_STATE_ERROR</code>	出错，文案见 <code>SCM_GetLastMessage</code> / <code>SCM_CncGetErrorAt</code>
6	<code>SCM_CNC_STATE_PROGRAM_STOP</code>	<code>M0</code> / <code>M1</code> 程序停：已停在停点，等 <code>SCM_CncCycleStart</code> 从停点续跑。与 <code>PAUSED</code> （进给保持，队列保留）不是一回事——程序停时队列已排空

5.19.18 CNC 通道参数号 `SCM_CNC_PARAM`（`SCM_CncSetParam` 的 `paramId`）

长度/速度/加速度类一律轴用户单位；每行的「单位」列给的是那一行的准确单位（无量纲的写清楚是哪一种号）。这些都是编译期参数——运行中改只对下一次编译生效（例外单独标注）。

号	名	含义	单位	默认
0	<code>SCM_CNC_PARAM_DEFAULT_FEED</code>	<code>G1</code> 默认进给（程序里没写 <code>F</code> 时用它）	用户单位/s	50

1	SCM_CNC_PARM_RAPID_FEED	G0 快移	用户单位/s	200
2	SCM_CNC_PARM_PATH_ACCELERATION	路径加速度	用户单位/s ²	500
3	SCM_CNC_PARM_PATH_DECELERATION	路径减速度 (SCM_CncStop 用)	用户单位/s ²	500
4	SCM_CNC_PARM_JERK	加加速度; 0 = 梯形	用户单位/s ³	0
5	SCM_CNC_PARM_CORNER_TOLERANCE	拐角过渡容差; 0 = 精确停止	用户单位	0
6	SCM_CNC_PARM_LOOKAHEAD_COUNT	前瞻段数; 0 = 默认	段	0
7	SCM_CNC_PARM_DEFAULT_ABSOLUTE	程序默认 G90 (1) / G91 (0)	开关 (0/1)	1
10	SCM_CNC_PARM_SCALE_X	X 轴当量。整组 10~14 默认由 SCM_CncOpen 向主软件问, 要覆盖才设	脉冲/用户单位	1 (SCM_CncOpen 问到即覆盖; 问不到且你没设, 启动时拒绝)
11	SCM_CNC_PARM_SCALE_Y	Y 轴当量 (同 10)	脉冲/用户单位	1 (同 10)
12	SCM_CNC_PARM_SCALE_Z	Z 轴当量 (同 10)	脉冲/用户单位	1 (同 10)
13	SCM_CNC_PARM_SCALE_A	A 轴当量 (同 10)	脉冲/用户单位	1 (同 10)
14	SCM_CNC_PARM_SCALE_C	C 轴当量 (同 10)	脉冲/用户单位	1 (同 10)
20	SCM_CNC_PARM_RTCP_TOOL_LENGTH	五轴 RTCP: 刀长 L	用户单位	0

2 1	SCM_CNC_PA RAM_RTCP_A_ AXIS_LINE	五轴 RTCP: A 轴线 Ta 的 Z 分量	用户 单位	0
2 2	SCM_CNC_PA RAM_RTCP_TA BLE_CENTER	五轴 RTCP: C 台中心 Tc 的 Z 分量	用户 单位	0
2 3	SCM_CNC_PA RAM_RTCP_A_ AXIS_X	五轴 RTCP: Ta 的 X 分量	用户 单位	0
2 4	SCM_CNC_PA RAM_RTCP_A_ AXIS_Y	五轴 RTCP: Ta 的 Y 分量	用户 单位	0
2 5	SCM_CNC_PA RAM_RTCP_TA BLE_CENTER_ X	五轴 RTCP: Tc 的 X 分量	用户 单位	0
2 6	SCM_CNC_PA RAM_RTCP_TA BLE_CENTER_ Y	五轴 RTCP: Tc 的 Y 分量	用户 单位	0
3 0	SCM_CNC_PA RAM_SPINDLE _FWD_SLAVE	M3 主轴正转位: 从站拓扑位置。 -1 = 未配 (合法值)。整组 30~38 默认由 SCM_CncOpen 向主软件问 (对号: crd+1 = 页面通道号)	从站 号	-1
3 1	SCM_CNC_PA RAM_SPINDLE _FWD_BIT	M3 : 站内 0 基位号	位号	0
3 2	SCM_CNC_PA RAM_SPINDLE _REV_SLAVE	M4 主轴反转位 (单向主轴可不配)	从站 号	-1
3 3	SCM_CNC_PA RAM_SPINDLE _REV_BIT	M4 : 站内 0 基位号	位号	0
3 4	SCM_CNC_PA RAM_SPINDLE _ANALOG_CHA NNEL	S 转速: 全局 I/O 通道序号	通道 号	-1
3 5	SCM_CNC_PA RAM_SPINDLE _SPEED_GAI N	模拟量工程值 = S × 增益	工程 值每 S	1

3 6	SCM_CNC_PA RAM_SPINDLE _SPEED_MAX	S 上限 (0 = 不限)	S 值	0
3 7	SCM_CNC_PA RAM_COOLANT _SLAVE	M8 / M9 冷却位: 从站拓扑位置	从站号	-1
3 8	SCM_CNC_PA RAM_COOLANT _BIT	冷却位: 站内 0 基位号	位号	0
3 9	SCM_CNC_PA RAM_OPTIONA L_STOP_ENAB LE	M1 选择停开关: 0 = M1 当注释掠过, 非 0 = 同 M0。△ 运行中即时生效 (每个周期实时读取), 是本表唯一的例外	开关 (0/ 非 0)	0
4 0	SCM_CNC_PA RAM_SPINDLE _AXIS	C 轴主轴: 当主轴用的那根轴号, -1 = 未配。必须与 SCM_CncOpen 的 axisC 是同一根。配了它, 螺纹指令 (G32 与 G92 / G78 / G76 循环) 才允许编译; 同时 M3 / M4 / M5 一律编译报错——IO 主轴与 C 轴主轴不能同时使用	轴号	-1
4 1	SCM_CNC_PA RAM_THREAD_ CHAMFER	螺纹退尾 (倒角) 量, 取值 0~99 的整数, 0 = 不退尾。只管 G92 / G78 螺纹循环; G76 的倒角量写在它第一行 P 的中间两位。退刀方向按起点 X 在螺纹的哪一侧自动判 (外螺纹 +X / 内螺纹 -X)	0.1 倍导 程	0

5.19.19 CNC 通道状态 SCM_CNC_STATUS 字段

字段	类型	含义
state	int	见 14.1
currentLine	int	正在执行的源行号 (1 基); 0 = 还没开始
progressPercent	double	0~100
remainingLength	double	剩余路径长 (用户单位)
pathVelocity	double	当前路径速度 (用户单位/s)
segmentsQueued	int	组队列里还剩几段没走
positionX	double	DRO X。△ RTCP 会话给的是工件系刀尖点
positionY	double	DRO Y
positionZ	double	DRO Z
positionA	double	DRO A (第 4 轴); 本程序不用 A 时恒 0
positionC	double	DRO C (第 5 轴); 本程序不用 C 时恒 0

5.20 机器人

说明 机器人通道（`rob` = 通道号，0~7，共 8 个通道）：把若干根已映射的伺服轴组成一台 SCARA / Delta / 6 轴串联机器人，按关节或笛卡尔位姿驱动（语义对齐 PLCopen Part 4 的通道式机器人指令）。每个通道独立建、独立查状态，互不干扰。

单位与姿态约定

说明 关节量一律 = 该关节轴的轴用户单位（与点位 / 回零同一套）；笛卡尔与几何量一律 `mm / deg`。姿态是 ZYX 欧拉角：`c` = 绕 Z（偏航）、`b` = 绕 Y（俯仰）、`a` = 绕 X（滚转），单位度；位姿参数顺序固定为 `(x, y, z, c, b, a)`。
SCARA 姿态只用 `c`；Delta 姿态恒零。

不适用分量非零即拒

说明 本构型不用的关节分量（`SCM_RobotMoveJ` / `SCM_RobotTableAddPoint`）与位姿分量（`SCM_RobotMoveL`：SCARA 的 `b / a`、Delta 的 `c / b / a`）必须传 0，非零返回 `SCM_INVALID_ARGUMENT` —— 避免误以为不存在的关节（如 J4）会运动。

受理 ≠ 完成

说明 全族异步受理：返回码只答“收下了吗”。是否完成通过轮询 `SCM_RobotGetState` 判断：运动指令等 `state` 回 0（IDLE），收尾条件必须含 2（FAULTED）—— 只等 IDLE 遇到故障会一直等下去。`FAULTED` 是粘滞的：保持到下一次 `SCM_RobotMoveJ` / `SCM_RobotMoveL` / `SCM_RobotRunTable` 受理时清；驱动器本身的故障复位走 `SCM_AxisReset`（按轴）。

通道存在性靠返回码

说明 `rob` 未 `SCM_RobotOpen` 的一切指令返回 `SCM_INVALID_ARGUMENT`（`SCM_RobotGetState` 也拒）—— 通道不存在与“值不可得”用返回码区分，不靠值。

点动与仿真的边界

说明 连接真实总线时，机器人通道支持点动，步进制 —— 每次调用走一步（步距取 `JOG_STEP_JOINT` / `JOG_STEP_CART`），连续调用即连动，`dir=0` 停点动。安全保护与 MoveJ/MoveL 相同：软限位、每周期步长限制、段末到位等待、掉使能停机、FAULTED 粘滞；笛卡尔点动目标位姿逆解不可达时拒该步且不动作。仿真后端仍不做笛卡尔运动（`SCM_RobotMoveL` / `SCM_RobotJogCart` 返回 `SCM_NOT_SUPPORTED`）—— 笛卡尔运动需要运动学层，离线开发请连主软件仿真模式。

5.20.1 SCM_RobotOpen

```
int SCM_RobotOpen(SCM_CARD_HANDLE handle, int rob, int robotType, int j1, int j2, int j3, int j4, int j5, int j6)
```

功能：建机器人通道 —— 指定构型，并把关节逐个映射到轴。关节可以是已映射的硬件伺服轴，也可以是未映射的虚拟轴（虚关节：参与运动学与轨迹推进、不驱动硬件）——全部为硬件关节、虚实混合、全部为虚关节这三种组合都使用同一套指令。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
rob	int	入	机器人通道号，0~7
robotType	int	入	构型：0 = SCARA（4 关节 J1/J2/Z/C）、1 = Delta（3 关节）、2 = 6 轴串联
j1	int	入	关节 J1 映射的硬件伺服轴号（0 基）
j2	int	入	关节 J2 映射的轴号
j3	int	入	关节 J3 映射的轴号
j4	int	入	关节 J4 映射的轴号；本构型不用的关节一律传 -1（Delta 只用 j1~j3）
j5	int	入	关节 J5 映射的轴号；SCARA 只用 j1~j4，j5/j6 传 -1
j6	int	入	关节 J6 映射的轴号；仅 6 轴构型使用

返回值：SCM_OK = 通道已建；SCM_INVALID_ARGUMENT = 通道号越界 / 构型非法 / 不用的关节没传 -1 / 两个关节映射到同一根轴；SCM_INVALID_AXIS = 关节映射的轴号越界；SCM_INVALID_STATE = 该通道已打开（重配先 SCM_RobotClose）；SCM_REJECTED = 无控制权。

权限：需 Owner 或 Maintenance。

说明：用到的关节传轴号（0 基），不用的关节一律传 -1。映射到硬件伺服轴的关节按 CSP 驱动实际伺服；映射到虚拟轴（未映射到从站的轴）的关节为**虚关节**（只参与运动学计算与状态显示，不向驱动器输出——用于离线开发与混合调试）。通道的几何参数（臂长等）与动力学上限在建完通道后用 SCM_RobotSetParam 配。

说明 注意：△ 已打开的 rob 再开被拒 —— 换构型 / 换映射先 SCM_RobotClose。

相关指令：SCM_RobotClose、SCM_RobotSetParam、SCM_RobotGetState

5.20.2 SCM_RobotClose

```
int SCM_RobotClose(SCM_CARD_HANDLE handle, int rob)
```

功能：关机器人通道，释放关节轴。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
rob	int	入	机器人通道号

返回值： SCM_OK = 已关； SCM_INVALID_ARGUMENT = 通道未打开； SCM_REJECTED = 无控制权。

权限：需 Owner 或 Maintenance。

说明：运行中调用会**先停再关**（参与关节停在当前位置）。关掉后该 rob 上的一切指令返回 SCM_INVALID_ARGUMENT，直到再次 SCM_RobotOpen。

相关指令： SCM_RobotOpen、SCM_RobotStop

5.20.3 SCM_RobotSetParam

```
int SCM_RobotSetParam(SCM_CARD_HANDLE handle, int rob, int paramId, double value)
```

功能：设机器人通道参数（几何、动力学上限、TCP / 工件偏移、点动步距、关节软限位）。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
rob	int	入	机器人通道号
paramId	int	入	参数号，取 SCM_ROBOT_PARAM_*，全表见本章末「机器人参数号」表
value	double	入	参数值（单位随参数：关节类 = 关节轴用户单位，笛卡尔 / 几何类 = mm/deg）

返回值： SCM_OK = 已生效 / 已登记； SCM_INVALID_ARGUMENT = 参数号未知 / 只读参数 / 本构型不适用（如非 SCARA 设 TCP_C、不存在关节的软限位、本构型未定义的 KIN 槽） / 值非有限实数； SCM_INVALID_STATE = 几何参数（KIN*）在运动中设； SCM_REJECTED = 无控制权。

权限：需 Owner 或 Maintenance。

说明：全构型一张参数表，不适用的项按返回码拒。生效时机分两类：动力学 / 偏移 / 点动步距 / 软限位类下一动作生效；几何 KIN* 类运行中（state = MOVING）拒，停稳再改。

相关指令： SCM_RobotGetParam、SCM_RobotOpen

5.20.4 SCM_RobotGetParam

```
int SCM_RobotGetParam(SCM_CARD_HANDLE handle, int rob, int paramId, double* value)
```

功能：读机器人通道参数（含只读项：构型、关节数、关节映射）。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
rob	int	入	机器人通道号
paramId	int	入	参数号，取 SCM_ROBOT_PARAM_*
value	double*	出	参数值；失败时出参无意义

返回值： SCM_OK = 读到； SCM_INVALID_ARGUMENT = 通道未打开 / 参数号未知 / 本构型不适用 / 出参指针为空。

权限： 只读，不需要控制权。

说明： ⚠ **不可得看返回码不看值** —— 这一族里 0 是多数项的合法值（没设过的偏移就是 0），返回非 SCM_OK 时出参无意义，别拿它当 0 用。

相关指令： SCM_RobotSetParam

5.20.5 SCM_RobotMoveJ

```
int SCM_RobotMoveJ(SCM_CARD_HANDLE handle, int rob, double j1, double j2, double j3, double j4, double j5,
double j6, double vel)
```

功能： 关节 PTP —— 全部参与关节按限速同时趋近各自的绝对目标。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
rob	int	入	机器人通道号
j1	double	入	关节 J1 绝对目标（关节轴用户单位）
j2	double	入	关节 J2 绝对目标
j3	double	入	关节 J3 绝对目标
j4	double	入	关节 J4 绝对目标； 本构型不用的关节分量必须传 0，非零即拒
j5	double	入	关节 J5 绝对目标；同上
j6	double	入	关节 J6 绝对目标；同上
vel	double	入	关节速度上限（关节轴用户单位/s），对全部参与关节生效。 ≤ 0 会被拒绝，无默认值

返回值： SCM_OK = 已受理（**不是已到位** —— 轮询 SCM_RobotGetState）； SCM_INVALID_ARGUMENT = 通道未打开 / vel ≤ 0 / 不用的关节分量非零 / 目标超关节软限位； SCM_INVALID_STATE = 总线未启动 / 关节轴未使能或故障中； SCM_REJECTED = 无控制权。

权限：需 Owner 或 Maintenance。

说明：受理即开始运动，粘滞的 `FAULTED` 由本条受理时清。目标在受理时查关节软限位（配了软限位窗的关节才查）。

说明 注意：⚠ **异步受理：**轮询 `SCM_RobotGetState` 至 `state` 回 `0` (IDLE)，**收尾条件必须含 `2` (FAULTED)。**

⚠ 参与关节必须**真正使能到位**再发：`SCM_AxisPower` 是异步受理，不等 `powered` 就发本条会被拒绝。

相关指令： `SCM_RobotGetState`、`SCM_RobotStop`、`SCM_RobotMoveL`

5.20.6 SCM_RobotMoveL

```
int SCM_RobotMoveL(SCM_CARD_HANDLE handle, int rob, double x, double y, double z, double c, double b, double a, double vel)
```

功能：笛卡尔直线 —— TCP 沿直线走到目标位姿（内部逆解到关节）。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>rob</code>	<code>int</code>	入	机器人通道号
<code>x</code>	<code>double</code>	入	目标 X (mm)
<code>y</code>	<code>double</code>	入	目标 Y (mm)
<code>z</code>	<code>double</code>	入	目标 Z (mm)
<code>c</code>	<code>double</code>	入	姿态 c = 绕 Z 偏航 (deg)。SCARA 只用它；Delta 传 0
<code>b</code>	<code>double</code>	入	姿态 b = 绕 Y 俯仰 (deg)。 仅 6 轴；其余构型必须传 0，非零即拒
<code>a</code>	<code>double</code>	入	姿态 a = 绕 X 滚转 (deg)。同上
<code>vel</code>	<code>double</code>	入	笛卡尔速度 (mm/s)。 ≤ 0 会被拒绝，无默认值

返回值： `SCM_OK` = 已受理；`SCM_INVALID_ARGUMENT` = 通道未打开 / `vel` ≤ 0 / 不用的位姿分量非零；`SCM_INVALID_STATE` = 总线未启动 / 关节轴未使能；`SCM_NOT_SUPPORTED` = 仿真后端（笛卡尔运动需要运动学层）；`SCM_REJECTED` = 无控制权。

权限：需 Owner 或 Maintenance。

说明：位姿顺序固定 (`x, y, z, c, b, a`)，ZYX 欧拉。逆解失败 / 超工作空间时通道转 `FAULTED`，`faultCode` = `SCM_ROBOT_FAULT_IK`。粘滞的 `FAULTED` 由本条受理时清。

说明 注意：⚠ **异步受理**：判断是否完成的方法同 `SCM_RobotMoveJ`。

⚠ 仿真后端返回 `SCM_NOT_SUPPORTED` —— 笛卡尔运动需要运动学层，离线开发请连主软件仿真模式。

相关指令： `SCM_RobotMoveJ`、`SCM_RobotGetState`、`SCM_RobotStop`

5.20.7 SCM_RobotJogJoint

```
int SCM_RobotJogJoint(SCM_CARD_HANDLE handle, int rob, int joint, int dir)
```

功能：关节点动 —— 指定关节按步距走一步，或停掉正在进行的点动。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>rob</code>	<code>int</code>	入	机器人通道号
<code>joint</code>	<code>int</code>	入	关节号（0 基）
<code>dir</code>	<code>int</code>	入	+1 / -1 = 方向；0 = 停该方向点动

返回值：`SCM_OK` = 已受理；`SCM_INVALID_ARGUMENT` = 通道未打开 / 关节号越界 / `dir` 不是 -1/0/+1 / 点动目标超软限位；`SCM_INVALID_STATE` = 总线未启动 / 关节轴未使能 / 步距或速度参数没设；`SCM_REJECTED` = 无控制权 / 模式不允许 / 通道正跑别的运动（先 `SCM_RobotStop`）。

权限：需 Owner 或 Maintenance。

说明：步距取通道参数 `SCM_ROBOT_PARAM_JOG_STEP_JOINT`，点动速度取 `SCM_ROBOT_PARAM_JOINT_VMAX` —— 两者都必须先设为 > 0，无默认值。

说明 注意：连续调用即连动（同一会话内延伸目标，不重建会话）；`dir=0` 停点动。点动与 MoveJ/MoveL 互斥 —— 会话在跑时发另一类运动会被拒。

相关指令： `SCM_RobotJogCart`、`SCM_RobotSetParam`、`SCM_RobotStop`

5.20.8 SCM_RobotJogCart

```
int SCM_RobotJogCart(SCM_CARD_HANDLE handle, int rob, int axisSel, int dir)
```

功能：笛卡尔点动 —— TCP 沿选定的笛卡尔轴按步距走一步。

参数	类型	方向	说明
<code>handle</code>	<code>SCM_CARD_HANDLE</code>	入	由 <code>SCM_Open*</code> 取得的卡句柄
<code>rob</code>	<code>int</code>	入	机器人通道号

axisSel	int	入	笛卡尔轴选择：0~5 = X/Y/Z/C/B/A。B/A 仅 6 轴，其余构型会被拒绝
dir	int	入	+1 / -1 = 方向；0 = 停该方向点动

返回值： SCM_OK = 已受理； SCM_INVALID_ARGUMENT = 通道未打开 / axisSel 越界或本构型不适用 / dir 非法； SCM_NOT_SUPPORTED = 仿真后端（笛卡尔运动需要运动学层）； SCM_REJECTED = 无控制权 / 模式不允许 / 通道正跑别的运动（先 SCM_RobotStop）。

权限： 需 Owner 或 Maintenance。

说明： 步距取通道参数 SCM_ROBOT_PARAM_JOG_STEP_CART（mm/deg）。

说明 注意： 目标位姿逆解不可达时拒该步且不动作（不会走到一半停）；仿真后端不支持笛卡尔运动——离线开发连主软件仿真模式。

相关指令： SCM_RobotJogJoint、SCM_RobotSetParam

5.20.9 SCM_RobotStop

```
int SCM_RobotStop(SCM_CARD_HANDLE handle, int rob)
```

功能： 停机器人通道 —— 中止当前运动（MoveJ / MoveL / 点动 / 表运行），参与关节停住。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
rob	int	入	机器人通道号

返回值： SCM_OK = 已受理（未在运动时也返回 SCM_OK，幂等）； SCM_INVALID_ARGUMENT = 通道未打开； SCM_REJECTED = 无控制权。

权限： 需 Owner 或 Maintenance。

说明： 不清 FAULTED —— 故障是粘滞的，由下一次 SCM_RobotMoveJ / SCM_RobotMoveL / SCM_RobotRunTable 受理时清。对单根关节轴发 SCM_AxisStop 是接管式中止：通道转 FAULTED（faultCode = SCM_ROBOT_FAULT_STOPPED）。

相关指令： SCM_RobotGetState、SCM_AxisStop

5.20.10 SCM_RobotGetState

```
int SCM_RobotGetState(SCM_CARD_HANDLE handle, int rob, SCM_ROBOT_STATE* state)
```

功能： 读机器人通道状态（状态机 / 当前运动种类 / 故障码 / 示教表状态 / 关节与笛卡尔 DRO）。

参数	类型	方向	说明
----	----	----	----

handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
rob	int	入	机器人通道号
state	SCM_ROBOT_STATE*	出	通道状态，字段见本章末「机器人状态结构」表；失败时出参无意义

返回值： SCM_OK = 读到； SCM_INVALID_ARGUMENT = 通道未打开也拒（通道不存在与"值不可得"用返回码区分，不靠值） / 出参指针为空。

权限： 只读，不需要控制权。

说明： 本族所有异步受理指令都通过本条判断是否完成：运动指令等 state 回 0（IDLE），**收尾条件必须含 2（FAULTED）**。FAULTED 粘滞：faultCode 一直保持，直到下一次 MoveJ / MoveL / RunTable 受理。

说明 注意： ⚠ 关节 DRO 按关节轴用户单位，不用的关节恒 0；笛卡尔 DRO 是正解出的 TCP 位姿（mm/deg），不用的分量恒 0。

相关指令： SCM_RobotMoveJ、SCM_RobotRunTable、SCM_RobotGetParam

5.20.11 SCM_RobotTableBegin

```
int SCM_RobotTableBegin(SCM_CARD_HANDLE handle, int rob, int pointCount)
```

功能： 开始定义示教表 —— 清暂存区并声明点数。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
rob	int	入	机器人通道号
pointCount	int	入	点数，1~64

返回值： SCM_OK = 暂存区已开； SCM_INVALID_ARGUMENT = 通道未打开 / 点数越界； SCM_REJECTED = 表正被 SCM_RobotRunTable 使用（先 SCM_RobotStop） / 无控制权。

权限： 需 Owner 或 Maintenance。

说明： 示教表只在运行期间有效，不会保存到文件——程序重新连接后需要重新下发。表按三段式逐点下发：Begin → N 次 AddPoint → End；本条只动暂存区，推到一半放弃，已生效的旧表照常在。

相关指令： SCM_RobotTableAddPoint、SCM_RobotTableEnd、SCM_RobotRunTable

5.20.12 SCM_RobotTableAddPoint

```
int SCM_RobotTableAddPoint(SCM_CARD_HANDLE handle, int rob, int index, double j1, double j2, double j3, double j4, double j5, double j6)
```

功能：向示教表暂存区写一个关节位姿点。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
rob	int	入	机器人通道号
index	int	入	点序号，0 ~ pointCount-1。 可乱序下发，重复 index = 覆盖
j1	double	入	关节 J1 位置（关节轴用户单位）
j2	double	入	关节 J2 位置
j3	double	入	关节 J3 位置
j4	double	入	关节 J4 位置； 本构型不用的关节分量必须传 0，非零即拒
j5	double	入	关节 J5 位置；同上
j6	double	入	关节 J6 位置；同上

返回值： SCM_OK = 已收； SCM_INVALID_STATE = 表未在定义中（先 SCM_RobotTableBegin）；
SCM_INVALID_ARGUMENT = 通道未打开 / 点序号越界 / 不用的关节分量非零； SCM_REJECTED = 无控制权。

权限：需 Owner 或 Maintenance。

说明：逐点合法性只查分量形态，**整表校验在 SCM_RobotTableEnd 一次性做**（含逐点关节软限位）。

相关指令： SCM_RobotTableBegin 、 SCM_RobotTableEnd

5.20.13 SCM_RobotTableEnd

```
int SCM_RobotTableEnd(SCM_CARD_HANDLE handle, int rob)
```

功能：收表 —— 整表校验通过后新表生效。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
rob	int	入	机器人通道号

返回值： SCM_OK = 新表已生效； SCM_INVALID_STATE = 表未在定义中； SCM_INVALID_ARGUMENT = 通道未打开 / 整表校验不过（有 index 没下发 / 某点超关节软限位）； SCM_REJECTED = 无控制权。

权限：需 Owner 或 Maintenance。

说明：校验条件：**所有 index 都已下发，且每个点都在关节软限位内**。**校验失败保留旧表，暂存区一律丢弃**（半张表绝不生效） —— 失败后要重来请重新 SCM_RobotTableBegin 。

相关指令：SCM_RobotTableBegin、SCM_RobotTableAddPoint、SCM_RobotRunTable

5.20.14 SCM_RobotRunTable

```
int SCM_RobotRunTable(SCM_CARD_HANDLE handle, int rob, int loop)
```

功能：跑示教表 —— 按点序逐点 PTP（关节插补链）。

参数	类型	方向	说明
handle	SCM_CARD_HANDLE	入	由 SCM_Open* 取得的卡句柄
rob	int	入	机器人通道号
loop	int	入	0 = 跑一遍到末点停；1 = 循环（末点走完回点 0 续跑）

返回值：SCM_OK = 已受理；SCM_INVALID_ARGUMENT = 通道未打开 / loop 不是 0/1；SCM_INVALID_STATE = 总线未启动 / 关节轴未使能 / 示教表为空 / 关节速度参数没设；SCM_REJECTED = 无控制权。

权限：需 Owner 或 Maintenance。

说明：关节速度取通道参数 SCM_ROBOT_PARAM_JOINT_VMAX（必须先设为 > 0，无默认值）。运行中 SCM_RobotGetState 的 tableIndex 回报当前点序（0 基）；非表运行时为 -1。粘滞的 FAULTED 由本条受理时清。

说明 注意：⚠ 异步受理：轮询 SCM_RobotGetState 至 state 回 0（IDLE）；loop = 1 时不会自行结束，用 SCM_RobotStop 收。收尾条件必须含 2（FAULTED）。

相关指令：SCM_RobotTableEnd、SCM_RobotStop、SCM_RobotGetState

5.20.15 机器人参数号 SCM_ROBOT_PARAM_*（SCM_RobotSetParam / SCM_RobotGetParam 的 paramId）

全构型一张表；本构型不适用的项按返回码拒（不可得看返回码不看值）。ro = 只读（Set 拒）。生效时机：动力学 / 偏移 / 点动步距 / 软限位类下一动作生效；几何 KIN* 类运行中（MOVING）拒。

常量	值	读写	说明
SCM_ROBOT_PARAM_TYP E	0	ro	构型：0/1/2 = SCARA/Delta/6 轴
SCM_ROBOT_PARAM_JOIN T_COUNT	1	ro	关节数：4/3/6
SCM_ROBOT_PARAM_JOIN T_AXIS_J1 ~ J6	2~7	ro	各关节映射的轴号；-1 = 本构型无此关节

SCM_ROBOT_PARAM_CART_VMAX	8	rw	笛卡尔速度上限 (mm/s)
SCM_ROBOT_PARAM_CART_AMAX	9	rw	笛卡尔加速度上限
SCM_ROBOT_PARAM_CART_JMAX	10	rw	笛卡尔加加速度上限
SCM_ROBOT_PARAM_JOIN_T_VMAX	11	rw	关节速度上限 (关节轴用户单位/s)。点动与表运行的速度取它
SCM_ROBOT_PARAM_JOIN_T_AMAX	12	rw	关节加速度上限
SCM_ROBOT_PARAM_JOIN_T_JMAX	13	rw	关节加加速度上限
SCM_ROBOT_PARAM_TCP_X / _Y / _Z	14~16	rw	TCP 工具偏移 (mm)
SCM_ROBOT_PARAM_TCP_C	17	rw	TCP 姿态偏移。 SCARA 专属，其余构型拒
SCM_ROBOT_PARAM_WORK_X / _Y / _Z / _C	18~21	rw	工件坐标偏移
SCM_ROBOT_PARAM_JOG_STEP_JOINT	22	rw	关节点动步距 (关节轴用户单位)
SCM_ROBOT_PARAM_JOG_STEP_CART	23	rw	笛卡尔点动步距 (mm/deg)
SCM_ROBOT_PARAM_J1_MIN ~ SCM_ROBOT_PARAM_J6_MAX	24~35	rw	关节软限位：J*n* 的下限 = 24+2(n-1)，上限 = 25+2(n-1)。 不存在的关节拒
SCM_ROBOT_PARAM_KIN0 ~ _KIN7	40~47	rw	构型几何 (槽位按构型解释，见下)。 本构型未定义的槽拒
SCM_ROBOT_PARAM_ELBO_W_POSITIVE	48	rw	SCARA 逆解肘分支： 1 = 肘正 ($J2 \geq 0$) / 0 = 肘负 ($J2 \leq 0$)。 仅 SCARA 构型，其余构型 Set/Get 均拒 ；值只收 0/1，其余拒；与几何参数一样，运行中 (MOVING) 修改会被拒。默认肘正

KIN 槽位按构型解释：

构型	KIN0	KIN1	KIN2	KIN3	KIN4
Delta	基座半径	动平台半径	主臂长	从臂长	—
SCARA	L1 (大臂)	L2 (小臂)	—	—	—

6 轴	D1	A1	A2	D4	D6
-----	----	----	----	----	----

5.20.16 机器人状态结构 `SCM_ROBOT_STATE` (`SCM_RobotGetState` 的出参)

字段	类型	含义
<code>state</code>	<code>int</code>	<code>0</code> = IDLE / <code>1</code> = MOVING / <code>2</code> = FAULTED (粘滞：下一次 MoveJ / MoveL / RunTable 受理时清)
<code>moveKind</code>	<code>int</code>	当前运动种类： <code>0</code> = 无 / <code>1</code> = MoveJ / <code>2</code> = MoveL / <code>3</code> = JogJoint / <code>4</code> = JogCart / <code>5</code> = Table
<code>faultCode</code>	<code>int</code>	FAULTED 时的原因码 (<code>SCM_ROBOT_FAULT_*</code> ，见 15.3)，其余 0
<code>tableCount</code>	<code>int</code>	已生效示教表点数
<code>tableIndex</code>	<code>int</code>	RunTable 当前点序 (0 基)；非表运行时 <code>-1</code>
<code>loop</code>	<code>int</code>	当前 RunTable 使用的 <code>loop</code> 参数值
<code>j1 ~ j6</code>	<code>double</code>	关节 DRO (关节轴用户单位)；不用的关节恒 0
<code>x / y / z</code>	<code>double</code>	笛卡尔 DRO：TCP 位置 (mm，正解)
<code>c / b / a</code>	<code>double</code>	笛卡尔 DRO：TCP 姿态 (deg，ZYX 欧拉：c = 绕 Z / b = 绕 Y / a = 绕 X)；不用的分量恒 0

5.20.17 机器人故障码 `SCM_ROBOT_FAULT_*` (`SCM_ROBOT_STATE.faultCode`)

常量	值	含义
<code>SCM_ROBOT_FAULT_NONE</code>	0	无故障
<code>SCM_ROBOT_FAULT_IK</code>	1	逆解失败 / 超工作空间
<code>SCM_ROBOT_FAULT_SOFT_LIMIT</code>	2	关节软限位
<code>SCM_ROBOT_FAULT_DRIVE</code>	3	驱动器掉使能 / 故障 (由状态字检测)
<code>SCM_ROBOT_FAULT_STOPPED</code>	4	外部接管停止 (对关节轴发 <code>SCM_AxisStop</code> 等)

第6章 软卡排障与版本匹配

客户程序连不上、被拒、轴不动时按一条固定顺序查；末节说明主软件、SDK 包、实时驱动和工程之间哪些必须匹配。

6.1 排查顺序

不要先猜代码。按这条链走，每一环看到正常再往下：

1. 返回码：每一步的 `int` 返回值是不是 `SCM_OK`。是 `SCM_REJECTED` 就读 `SCM_GetLastReason`。
2. 主软件在不在、模式对不对：真机要「实时运行」且「激活配置参数」进入 OP；离线要在「仿真」。「配置调试」下写命令按设计被拒。
3. 诊断：`SCM_GetDiagnostics` 的 `cycleCounter` 在不在走、`workCounter` 稳不稳、`missedCycleCount` 涨不涨。
4. 控制权：`ownerClientId` 是不是自己的 `clientId`。
5. 轴状态：`SCM_AxisGetStatus` 的 `powered`、`faulted`、`actualPosition`、`statusWord`、`errorCode`。
6. 最后回主软件：轴页「在线」、从站过程数据、消息栏和导出的日志，看真实驱动器状态。

6.2 连不上

返回码	含义	处理
<code>SCM_INVALID_ARGUMENT</code>	连接配置文件不存在或无法读取。	多半是主软件还没进 OP（或没切进「仿真」），先去主软件确认。路径固定 <code>C:\ProgramData\SoftSC\softcard.runtime.json</code> 。
<code>SCM_BACKEND_UNAVAILABLE</code>	软卡 SDK 服务没有运行，或共享内存不存在，或规格 / 布局不一致。	读 <code>SCM_GetOpenErrorText</code> ，它逐项列出双方的轴数、I/O 点数、周期。主软件退出 OP 后运行时会停，这时连不上是正常的。

连接配置文件由主软件自动写出，不要复制旧文件，不要手填共享内存名和轴数。

6.3 被拒

收到 `SCM_REJECTED` 先读 `SCM_GetLastReason`，按原因码分支。完整表在「软卡 SDK 指令说明」末尾，最常见的几条：

原因码	意思	处理
NoControl	没拿控制权。	先 <code>SCM_RequestOwner</code> 。
NotOwner	控制权在别的程序手上。	让占用者 <code>SCM_ReleaseOwner</code> 或关掉它；主软件「软卡设置」页的「控制权」一行能看到是谁。占用者崩溃时，约 5 秒后判定其失联，下一次 <code>SCM_RequestOwner</code> 会自动收回。

BusNotStarted	本程序没武装。	重新 <code>SCM_RequestOwner</code> ，成功即武装。
HostNotReady	主软件未就绪，或轴没使能完就发了运动 / 建组。	确认进 OP；轮询 <code>powered</code> 再发。
AxisClaimedByOther	轴被插补页、CNC、凸轮、齿轮、机器人或别的组占着。	在主软件里停掉那个通道，或换轴。
SoftRealtimeNotAllowed	当前运行的实时核不提供连续伺服运动（插补 / 凸轮 / 齿轮 / 机器人）。	确认主软件在「实时运行」下激活。

6.4 诊断值不对

现象	先查
<code>cycleCounter</code> 不走	运行时没在跑：主软件没进 OP 或没在「仿真」。真机再看主站在线页的总线状态。
<code>workCounter</code> 为 0	过程数据没有有效响应：从站没进 OP、网线、供电。
<code>workCounter</code> 跳变	从站掉线或响应不稳：从站状态、拓扑、干扰。WKC 不是站数，站数看 <code>sSlaveCount</code> 。
<code>workCounter</code> 与预期不符	工程改过没重新激活：从站数、PDO 映射变了。重新「激活配置参数」。
<code>missedCycleCount</code> 持续增长	周期过小或系统负载高。把 <code>lastCycleTimeUs</code> 、 <code>maxCycleTimeUs</code> 和 <code>missedCycleCount</code> 一起记下来，判断是偶发尖峰还是持续超时；判断方法见《用户使用手册》「实时形态与性能」。

6.5 轴不动

- 返回码是不是 `SCM_OK`。被拒看上面「被拒」一节。
- `ownerClientId` 是不是自己。
- `powered` 为真、`faulted` 为假。`faulted` 为真先记下 `errorCode`，再 `SCM_AxisReset`，等 `faulted` 转假。
- `cycleCounter`、`workCounter` 正常。
- `actualPosition`、`actualVelocity` 有没有变化。有变化但方向反了，回主软件轴页和驱动器参数查方向；没变化，查驱动器控制模式、目标值 PDO 和反馈 PDO。
- 回主软件轴页「在线」看状态字、限位、报警、目标位置。

6.6 症状速查

现象	原因	处理
----	----	----

打开连接失败	主软件没进 OP / 仿真；或 SDK 与主软件版本不匹配。	看「连不上」一节；读 <code>SCM_GetOpenErrorText</code> 。
能读不能写	没有控制权，并非通信问题。	<code>SCM_GetDiagnostics</code> 看 <code>ownerClientId</code> → <code>SCM_RequestOwner</code> → 再写。
<code>SCM_INVALID_AXIS</code>	轴号超出工程轴数，或把轴号当成 1 起。	轴号从 0 开始； <code>axisCount</code> 看 <code>SCM_GetDiagnostics</code> 。
使能后立刻发运动被拒	使能是异步受理，还没完成。	轮询 <code>powered</code> ，超时不短于 15 秒。
建组每次都被拒	成员轴没使能完，或轴被别的通道占着。	全部 <code>powered</code> 后再等约 1 秒；读原因码。
写输出返回 OK 但现场没动作	该点没映射，或另一个程序也在发命令互相覆盖。	核对从站位置和位号；同一时刻只让一个程序驱动软卡。
归还控制权后轴还在动	<code>SCM_ReleaseOwner</code> 不停运动。	归还前先 <code>SCM_AxisStop</code> ，或用 <code>SCM_EtherCATStop</code> 。
SDK 拿到的轴数 / 点数和工程不一样	工程改了没重新激活。	重新「激活配置参数」，客户程序重连。

6.7 版本与配置匹配

对象	决定什么	不匹配时
主软件工程	轴数、I/O 点数、周期、从站拓扑、映射。	轴号无效、点号错位、WKC 不对。工程改了要重新「激活配置参数」，SDK 拿到的轴数、I/O 点数以激活那一刻为准。
主软件与 SDK 包	共享内存布局、指令集、结构体定义。	共享内存布局不一致的版本在连接时会被直接拒绝并给出说明，不会读错数据。主软件和 SDK 包应配套使用、一起升级，不要只换一侧。
主软件与实时驱动	驱动接口版本。	不匹配时主软件拒绝启动总线并说明，先按主软件提示处理，再连客户程序。
头文件与 DLL	C 结构体布局。	字段读数离谱。用同一个包里的头文件和 DLL 重新编译。

- 升级后至少验证一遍：连接、读诊断、拿控制权、读轴状态、一根轴小距离运动、归还。
- 升级注意：没有说明文件的 I/O 模块也计入全局序号。用 `SCM_WriteOutput` / `SCM_ReadInput` 按全局序号的老程序要逐个核对序号；新程序用 `SCM_WriteOutputBit` / `SCM_ReadInputBit`。
- 客户程序启动时把 SDK 版本（`SCM_GetVersion`）、clientId、诊断里的轴数和点数写进日志，现场回溯有据可查。

第7章 案例工程说明

20 个案例的清单、目录结构、运行前提与代码约定。

7.1 案例清单

SCMotion 软卡 SDK 的 20 个功能上位机示例，**一个案例一个独立工程**，提供 C# WinForms、C# WPF、C++ Qt 三个版本（Qt 版目录名为 `SCmotionQt\ScNN`，例如 `Sc07`）。代码是「一个按钮 = 一个函数，函数体里直接调 `SCM_*`」，整段复制到别的项目就能用。

#	目录	演示内容	主要指令（省略 SCM_ 前缀）
01	案例01_连接与生命周期	打开契约 / 读版本 / 读诊断 / 关句柄	<code>OpenProjectConfig</code> , <code>GetVersion</code> , <code>GetDiagnostics</code> , <code>GetLastMessage</code> , <code>Close</code>
02	案例02_EtherCAT总线与控制权	总线启停、WKC、从站数、控制权	<code>EtherCATStart/Stop</code> , <code>EtherCATGetWkc</code> , <code>EtherCATGetSlaveCount</code> , <code>RequestOwner/ReleaseOwner</code>
03	案例03_轴使能与状态监视	使能、复位、状态表实时监视	<code>AxisPower</code> , <code>AxisReset</code> , <code>AxisGetStatus</code> , <code>AxisReadStatus</code> , <code>AxisGetPosition/Velosity/Param</code>
04	案例04_点位运动	绝对、相对、停止、急停、受控暂停	<code>AxisMoveAbs</code> , <code>AxisMoveRel</code> , <code>AxisStop</code> , <code>AxisEStop</code> , <code>AxisHalt</code> , <code>AxisSetPosition</code>
05	案例05_速度运动与点动	连续速度 + 按住点动（松手收手）	<code>AxisMoveVel</code> , <code>AxisJog</code> , <code>AxisStop</code> , <code>AxisHalt</code> , <code>AxisEStop</code>
06	案例06_全回零方式总案例	驱动器回零 / 控制器回零 / 当前位置置零	<code>AxisSetHomeParam</code> , <code>AxisGetHomeParam</code> , <code>AxisHome</code> , <code>AxisGetHomingState/Result</code> , <code>AxisAbortHoming</code>
07	案例07_数字量IO	位/字/全局序号读写 + 输入位扫描	<code>WriteOutputBit</code> , <code>ReadInputBit</code> , <code>WriteOutputWord</code> , <code>ReadInputWord</code> , <code>WriteOutput</code> , <code>ReadInput</code>
08	案例08_模拟量IO	工程值与原始值双向读写	<code>WriteAnalogOutput</code> , <code>ReadAnalogInput</code> , <code>WriteChannelRaw</code> , <code>ReadChannelRaw</code>
09	案例09_坐标组建组与配置	建组、逐轴当量、组限制、路径前瞻	<code>GroupEnable</code> , <code>GroupConfigAxis</code> , <code>GroupSetLimits</code> , <code>GroupConfigPath</code> , <code>GroupGetStatus/Space</code> , <code>GroupDisable</code>
10	案例10_直线插补	绝对/相对直线、冲刷、倍率、清路径	<code>GroupMoveLinearAbs/Rel</code> , <code>GroupFlush</code> , <code>GroupSetOverride</code> , <code>GroupStop</code> , <code>GroupResetPath</code>
11	案例11_圆弧插补	圆心偏移式 / 半径式 / 相对圆弧 + 平面选择	<code>GroupMoveCircularAbs/Rad/Rel</code> , <code>GroupFlush</code>

12	案例12_连续路径与前瞻	一次推方框轨迹不停 + 前瞻与拐角压缩	GroupConfigPath, GroupMoveLinearAbs, GroupGetSpace, GroupFlush
13	案例13_路径同步IO与模拟量	输出位/延时/模拟量挂到路径段边界	GroupBufIO, GroupBufDelay, GroupBufAnalog, GroupFlush
14	案例14_电子齿轮	啮合、状态、脱开、驱动主轴	GearIn, GearGetState, GearOut, AxisMoveVel
15	案例15_电子凸轮	建表、整表校验、啮合、换表、脱开	CamTableBegin/AddPoint/End, CamIn, CamGetState, CamSwitchTable, CamOut
16	案例16_CNC与G代码通道	加载、编译（看错误行）、启动、保持、单段、倍率	CncOpen, CncLoadText, CncCompile, CncGetErrorCount/At, CncCycleStart, CncFeedHold, CncStop/Reset, CncSetSingleBlock/Override, CncGetStatus, CncClose
17	案例17_诊断与错误处理	诊断、错误码、结构化原因码、错误文本	GetDiagnostics, GetLastError, GetLastErrorReason, GetLastMessage, GetErrorText, GetOpenErrorText
18	案例18_Delta并联机器人	3 关节通道、几何参数、PTP/LIN、点动	RobotOpen, RobotSetParam, RobotMoveJ/L, RobotGetState, RobotJogJoint/Cart, RobotStop, RobotClose
19	案例19_SCARA机器人	4 关节 + 肘型参数 + 示教表	同上 + RobotTableBegin/AddPoint/End, RobotRunTable
20	案例20_六轴串联机器人	6 关节通道、几何 KIN0~KIN4、PTP/LIN	同上 (robotType 2)

7.2 每个案例目录里有什么

```

案例NN_xxx\
|—— 案例NN_xxx.sln          单案例解决方案，双击即开
|—— 案例NN_xxx.csproj       net8.0-windows + UseWindowsForms
|—— Program.cs              入口
|—— MainForm.Designer.cs    ★ 设计器文件：控件全部有名字，可在 VS 里拖
|—— MainForm.Common.cs     连接 / 断开 / 日志（20 个案例一样）
|—— MainForm.cs             ★ 本案例业务函数：一个按钮 = 一个函数
|—— 使用说明.md            本案例的编译运行与复制说明
|—— sdk\                   SCSmotionSoftCard.Core.dll + interp_core.dll
|—— 运行程序\              ★ 已编译好，双击 exe 就能跑

```

根目录的 `SCMOTION案例20_WinForm版.sln` 一次打开全部 20 个工程；`预览图\` 里是代表性案例的界面截图。

7.3 运行前提

- 1 Windows x64 + .NET 8 桌面运行时（只跑程序装运行时就够；要改代码再装 VS 2022「.NET 桌面开发」）。
- 2 先让 SCMotion 主软件进 OP（或切到「仿真」）。它会写出连接契约 `C:\ProgramData\SoftSC\softcard.runtime.json`；找不到这个文件通常是主软件没进 OP，不是路径写错。
- 3 双击 `案例NN_xxx\运行程序\案例NN_xxx.exe`，按界面顺序：**连接** → **使能** → **功能按钮**。想改代码就双击 `.sln` 按 **F5**。

注意 每个 exe 只认自己那个 dll，不要把一个案例的 exe 复制到别的案例目录改名用；要挪就重新编译。

7.4 代码约定

- 一个按钮 = 一个函数，函数名 `OnXxx`，在 `MainForm.Designer.cs` 里绑到按钮的 `Click`。
- 函数体内直接调 SDK，没有中间层；卫语句一律单行：`if (_card == null) { Log("还没连接"); return; }`。
- 点动按钮绑 `MouseDown` / `MouseUp`（不是 `Click`），松手发 `direction = 0` 收手，否则轴会一直走。
- 轴号 **0 基**，必须和主软件轴页里的一致。
- SDK 调用处参数 ≥ 2 个的拆成一行一个参数，行尾写「参数名 + 含义」，参数名照本手册第 5 章的签名：

```
var code = ScMotionApi.SCM_AxisMoveAbs(
    _card,    //handle 软卡句柄
    axis,     //axis 轴号 0 基
    target,   //position 目标位置 用户单位
    vel,      //velocity 速度 用户单位/s
    acc,      //acceleration 加速度 0=轴页默认
    dec,      //deceleration 减速度 0=轴页默认
    jerk);    //jerk 加加速度 0=梯形
```

7.5 复制一个函数到别的项目要带走什么

要带走的	在哪
<code>ConfigPath</code> / <code>ClientId</code> / <code>_card</code> / <code>Log()</code>	<code>MainForm.Common.cs</code>
日志框 <code>rtbLog</code> ，以及该函数引用的输入框 <code>txtXxx</code> / <code>cboXxx</code>	<code>MainForm.Designer.cs</code>
<code>SCmotionSoftCard.Core.dll</code> 引用 + <code>interp_core.dll</code> （必须和 exe 同目录）	<code>sdk\</code>

7.6 常见拒绝原因

读 `SCM_GetLastReason` 的原因码分支，不要解析文案（案例 17 有演示）：

码	含义	码	含义
1	没有控制权	7	宿主未就绪
2	不是属主	8	未启用回零引擎
3	总线未启动（本程序未武装）	9	回零开关未配
4	轴号非法	10	轴在运动中
5	软实时禁用	11	回零方式不支持
6	轴被其他客户端占用		

注意 一次只允许一个进程驱动软卡：命令槽全局一份，两个客户端同时写会互踩且返回码仍是 `0`。写未映射的输出点同样返回 `0`（不报错也无动作），必须现场确认接线。

附录A 案例教程索引

教程	内容	对应案例
教程 01 数字量 IO 控制	主软件「I/O 测试」页点检 + SDK 按从站读写输入输出位	案例07_数字量IO
教程 02 轴使能与点动	主软件轴「在线」页使能点动 + SDK 使能、状态轮询、按住点动	案例03、案例05

每篇教程配有讲解视频（MP4）和配音稿，在资料包「03_案例教程」目录下。