工业控制文档

KRMOTION

KRmotion 编程手册

版权说明

本手册内容可能因产品迭代而调整，具体功能与界面请以当前交付版本为准。

安全提示

设备调试、联机和运动前，用户必须完成急停、限位、联锁等安全保护设计，并在软件中加入必要的异常处理和风险隔离逻辑。

目录

第 1 章

KRmotion 产品说明书

1.1 发布说明

KRmotion 是面向客户开放的 C 风格运动控制接口层，客户应用可通过统一的 Ez* 接口完成控制器启动、设备发现、运动控制与状态读取等核心任务。

与上一代产品相比，当前版本的 API 调用模型已经重构，但产品使用路径保持连续：仍可按“准备工程 -> 启动控制器 -> 执行运动任务 -> 停止与退出”的方式开展项目交付与现场调试。

本说明书以当前公开头文件中的 Ez* 接口为编写依据，面向客户提供可直接落地的使用说明与调用建议。

1.2 支持规范

本产品以“动态库导出接口 + 头文件声明”的方式对外提供能力：

- 动态库导出 C 风格 API（Ez*）
- 头文件通过 #include <ez_motion.h> 引入

运行时依赖控制器运行环境和启动配置文件。应用在调用 EzStart 时，需要提供当前 KRM 项目使用的工程/配置文件路径（如.krm 文件），以完成控制器启动和工程加载。

控制器工作模式支持：

- 仿真模式

- 真机模式

当前公开接口覆盖以下能力域：

- 主站/从站发现与基础信息读取
- 引擎状态与运行模式查询
- 单轴控制
- 电子齿轮耦合
- 轴组插补
- 终端 IO 访问
- PDO/SDO 访问
- 通用运动控制与状态监控

1.3 安装指南

首次使用前，建议先完成以下准备：

- 将 `ez_motion` 动态库部署到应用可加载的位置
- 在工程中包含`ez_motion.h`
- 准备控制器启动所需的配置文件（供 EzStart 使用）

在客户应用中，这三类文件的关系可理解为：

- 动态库：提供运行时可调用的接口实现
- 头文件：提供编译期函数声明与类型定义
- 配置文件：提供启动控制器所需的工程与设备配置

调用 EzStart(const char* path) 时，`path` 表示当前 KRM 项目使用的工程/配置文件路径。建议在交付工程中使用稳定、可追踪的配置文件目录，并确保应用进程对该路径具有读取权限。

实现层会将相对路径转换为绝对路径后再执行启动流程。因此，您既可以传入绝对路径，也可以在确认当前工作目录可控的前提下传入相对路径。

1.4 通用编程指南

建议按以下生命周期组织调用顺序：

1. EzInit
2. EzSetControllerWorkMode
3. EzStart
4. 业务接口调用
5. EzStop
6. 按退出目标选择：仅释放客户端资源时调用 EzUnInit；需要关闭整个引擎时调用 EzClose

请特别注意，EzSetControllerWorkMode 必须在 EzStart 之前调用，才能使工作模式设置生效。

所有业务调用（如扫描、轴控制、轴组插补、IO 访问、PDO/SDO 访问等）应在初始化与启动成功后执行。建议每次调用后立即检查返回码，并在失败时通过 EzGetErrorText 获取可读错误信息，便于日志记录和现场排查。

在工程结束或应用退出前，建议先调用 EzStop 停止控制器；随后根据退出目标选择调用 EzUnInit 或 EzClose。

1.5 运动编程指南

1.5.1 系统与控制器管理

1.5.1.1 功能概述

本章主要介绍控制器生命周期相关接口。典型应用会先建立通信、设置工作模式，再加载工程启动控制器；业务结束后再按顺序停止并释放资源。对现场调试而言，EzGetControllerWorkMode、EzGetEngineStatus 和 EzGetDebugMode 也常用于确认当前运行方式和引擎状态。

1.5.1.2 相关接口

- EzInit
- EzUnInit
- EzSetControllerWorkMode
- EzGetControllerWorkMode
- EzGetEngineStatus
- EzGetDebugMode
- EzStart
- EzGetConfigFile
- EzStop
- EzClose

1.5.1.3 使用说明

EzInit 用于建立与控制器的会话，未初始化前不应调用其他业务接口。EzSetControllerWorkMode 用于指定仿真模式或真机模式，该设置只有在 EzStart 之前调用时才会生效；EzGetControllerWorkMode 可用于回读当前模式设置，便于在界面、日志和启动检查流程中做双向确认。

EzStart 用于加载当前 KRM 项目使用的工程/配置文件并启动控制器。启动成功后，应用即可继续读取主站、轴和终端信息，或者执行运动与 IO 操作。EzGetEngineStatus 可用于读取当前引擎状态、停止原因和总线致命故障原因，适合在上电自检、运行监控和异常停机诊断中使用。EzGetDebugMode 可用于确认当前是否处于调试模式。EzGetConfigFile 适合导出当前工程配置文件，便于备份、转存和问题复现。

业务结束后，建议先调用 EzStop 停止控制器，再根据退出目标选择资源释放方式：通常调用 EzUnInit 即可； EzClose 用于关闭整个引擎，常规应用无需每次都调用，只有明确需要关闭引擎时再使用。

1.5.1.4 调用流程

1. 调用 EzInit 建立连接。
2. 根据项目要求调用 EzSetControllerWorkMode 设置仿真或真机模式。
3. 调用 EzStart 传入当前 KRM 项目使用的工程/配置文件路径。
4. 必要时调用 EzGetControllerWorkMode、EzGetEngineStatus、EzGetDebugMode 记录当前运行环境。
5. 执行业务接口。
6. 结束时调用 EzStop。
7. 调用 EzUnInit 释放会话资源；仅在需要关闭整个引擎时调用 EzClose。

1.5.1.5 示例代码

```
int rc = EzInit();
if (rc != 0) {
    printf("EzInit failed: %s\n", EzGetErrorText(rc));
    return;
}

rc = EzSetControllerWorkMode(true);   // true: 仿真模式，false: 真机模式
if (rc == 0) {
    rc = EzStart("config\\project.krm");
}
if (rc != 0) {
    printf("Start failed: %s\n", EzGetErrorText(rc));
    EzUnInit();
    return;
}

bool debug_mode = false;
bool simulation = false;
EzEngineStatus engine_status;
char config_path[260] = {0};
EzGetControllerWorkMode(&simulation);
EzGetEngineStatus(&engine_status);
EzGetDebugMode(&debug_mode);
EzGetConfigFile(config_path);
printf("simulation=%d, engine_state=%d, debug_mode=%d\n",
       simulation, engine_status.state, debug_mode);

EzStop();
EzUnInit();
```

1.5.1.6 注意事项

- EzSetControllerWorkMode 必须在 EzStart 之前调用，否则设置不会生效。
- EzStop 用于停止控制器，EzUnInit 用于释放客户端会话资源，两者职责不同，不建议互相替代。
- EzClose 会关闭整个引擎，通常只在明确需要彻底退出引擎时使用。
- 所有返回值都应立即检查，失败时可通过 EzGetErrorText 输出可读错误信息。

1.5.2 总线扫描与 EtherCAT 访问

1.5.2.1 功能概述

本章主要介绍如何发现总线设备、读取系统拓扑信息，以及如何直接访问 EtherCAT 从站的 PDO/SDO 数据。对于设备联机确认、参数读取、现场诊断和少量定点配置，这一组接口通常是最先使用的基础能力。

1.5.2.2 相关接口

- EzScan
- EzGetMasterCycleTime
- EzGetMasterInfo
- EzGetAxesInfo
- EzGetTerminalInfo
- EzWriteEcatPDOChar / EzReadEcatPDOChar
- EzWriteEcatPDOBool / EzReadEcatPDOBool
- EzWriteEcatPDOShort / EzReadEcatPDOShort
- EzWriteEcatPDOInt / EzReadEcatPDOInt
- EzWriteEcatPDODouble / EzReadEcatPDODouble
- EzWriteEcatSDOChar / EzReadEcatSDOChar
- EzWriteEcatSDOBool / EzReadEcatSDOBool
- EzWriteEcatSDOShort / EzReadEcatSDOShort
- EzWriteEcatSDOInt / EzReadEcatSDOInt
- EzWriteEcatSDODouble / EzReadEcatSDODouble

1.5.2.3 使用说明

启动控制器后，可通过 EzGetMasterInfo、EzGetAxesInfo、EzGetTerminalInfo 读取当前工程中的主站、轴和终端列表；通过 EzScan 可以进一步扫描指定主站上的从站设备信息，适合用于现场核对从站数量、厂商信息和产品代码。EzGetMasterCycleTime 可用于读取总线循环周期，便于评估刷新节拍和访问频率。

PDO 接口通常用于周期性过程数据访问，适合状态字、控制字、实时数值等在线数据读写。SDO 接口更适合参数类、配置类或调试类访问。选择 PDO 还是 SDO，取决于目标数据在从站对象字典中的用途和访问时机。

对于不同数据类型，接口按 Char、Bool、Short、Int、Double 进行区分。其中 Bool 适合位量或开关对象，Short 适合 16 位参数，Int 适合 32 位整型对象。实际调用前，建议先确认主站索引、从站索引、对象地址和子索引都与目标设备手册一致，再进行读写。

1.5.2.4 调用流程

1. 在 EzStart 成功后读取主站、轴、终端信息。
2. 调用 EzScan 核对目标主站上的从站清单。
3. 调用 EzGetMasterCycleTime 确认总线周期。
4. 根据目标对象选择 PDO 或 SDO 接口，并按数据类型选择对应函数。
5. 读取或写入后立即检查返回值，并在必要时重新读取确认结果。

1.5.2.5 示例代码

```
EzSlaveInfo* slaves = 0;
int slave_count = 0;
int cycle_time = 0;

int rc = EzScan(ETHERCAT, 0, &slaves, &slave_count);
if (rc != 0) {
    printf("EzScan failed: %s\n", EzGetErrorText(rc));
    return;
}

EzGetMasterCycleTime(ETHERCAT, 0, &cycle_time);
printf("slave_count=%d, cycle_time=%d\n", slave_count, cycle_time);

int status_word = 0;
```

```
rc = EzReadEcatPDOInt(0, 0, 0x6041, 0, &status_word);
if (rc == 0) {
    printf("PDO status word = 0x%X\n", status_word);
}

double max_profile_velocity = 0.0;
rc = EzReadEcatSDODouble(0, 0, 0x607F, 0, &max_profile_velocity);
if (rc == 0) {
    printf("SDO 0x607F = %.3f\n", max_profile_velocity);
}
```

1.5.2.6 注意事项

- EzScan、PDO/SDO 访问都应在控制器已成功启动后进行。
- PDO 更适合周期过程数据，SDO 更适合参数访问；不要将二者简单混用。
- 从站索引、对象地址、子索引和数据类型必须与设备对象字典一致。
- 对驱动参数执行写操作前，建议先在设备手册中确认是否允许在线修改，以及修改后是否需要重新使能或重新启动。

1.5.3 单轴基础运动

1.5.3.1 功能概述

本章主要介绍单轴最常用的基础动作，包括上电使能、点位运动、点动、停止和紧急停止等。完成这些接口的掌握后，即可搭建最基本的单轴调试流程。

1.5.3.2 相关接口

- EzPower
- EzMove
- EzJog
- EzAxisStop
- EzEmergencyStop
- EzSetPosition

1.5.3.3 使用说明

EzPower 用于轴使能或去使能。通常在执行运动前先上电使能，在确认运动结束或需要退出时再去使能。EzMove 用于执行点位运动，运动类型由`EzMoveType` 决定：

- `MOVE_TYPE_ABSOLUTE`：绝对运动，目标值相对于工件坐标或机械坐标原点。
- `MOVE_TYPE_RELATIVE`：相对运动，目标值相对于当前轴位置。

EzJog 用于点动运行，运动方向由`EzMoveDirection` 指定：

- `MOVE_DIR_POSITIVE`：正方向点动。
- `MOVE_DIR_NEGATIVE`：负方向点动。

停止相关接口需要重点区分。EzAxisStop 用于指定轴停止，其中`STOP_MODE_CONTROLLED` 表示按减速过程受控停止，适合常规收停；`STOP_MODE_IMMEDIATE` 表示立即停止，适合异常处理。EzEmergencyStop 则用于整机范围内的紧急停车，会让所有正在运动的对象尽快停止。

EzSetPosition 用于强制设置轴当前位置。它常用于对位、坐标校正或仿真调试，但不应在不明确后果的情况下随意修改，以免破坏当前坐标基准。

1.5.3.4 调用流程

1. 确认控制器已启动，并已具备可用的运动参数。
2. 调用 EzPower 对目标轴上电使能。
3. 如需修正坐标基准，可先调用 EzSetPosition。
4. 调用 EzMove 或 EzJog 执行基础运动。
5. 需要收停时调用 EzAxisStop；遇到危险状态时调用 EzEmergencyStop。
6. 任务结束后按工艺要求决定是否去使能。

1.5.3.5 示例代码

```
int axis_id = 0;
int rc = EzPower(axis_id, true);
if (rc != 0) {
    printf("EzPower failed: %s\n", EzGetErrorText(rc));
    return;
}

EzSetPosition(axis_id, 0.0);

rc = EzMove(axis_id, 100.0, MOVE_TYPE_ABSOLUTE);
if (rc != 0) {
    printf("EzMove absolute failed: %s\n", EzGetErrorText(rc));
}

rc = EzMove(axis_id, -10.0, MOVE_TYPE_RELATIVE);
if (rc != 0) {
    printf("EzMove relative failed: %s\n", EzGetErrorText(rc));
}

rc = EzJog(axis_id, MOVE_DIR_POSITIVE);
if (rc == 0) {
    EzAxisStop(axis_id, STOP_MODE_CONTROLLED);
}
```

1.5.3.6 注意事项

- 在调用 EzMove 之前，应确认轴已经使能，并且运动参数已正确设置。
- 绝对运动适合按固定坐标点运行，相对运动更适合重复步进或循环轨迹。
- 点动通常用于手动找位或调试，结束后应显式调用停止接口。
- EzSetPosition 会改变位置基准，建议仅在确认工艺允许时使用。
- EzEmergencyStop 适合紧急场景，不应替代常规的受控停止流程。

1.5.4 单轴高级运动

1.5.4.1 功能概述

本章主要介绍单轴参数化运动和在线修调能力，用于实现更平滑的速度曲线、更高的节拍适应性，以及更贴近工艺需求的特殊运动过程。

1.5.4.2 相关接口

- EzSetProfile
- EzGetProfile
- EzSetSProfile
- EzGetSProfile
- EzSetGearProfile
- EzGetGearProfile
- EzGearIn
- EzGearOut
- EzGetGearState
- EzChangeSpeed
- EzUpdateTargetPosition
- EzMoveEx
- EzSoftEndMove
- EzSoftStartMove
- EzUpdateTargetPositionEx

1.5.4.3 使用说明

EzSetProfile / EzGetProfile 用于设置和读取梯形速度曲线参数，包括起始速度、设定速度、加速度、减速度和末速度。梯形曲线适合一般的点位运动，参数直观，适用面广。

EzSetSProfile / EzGetSProfile 用于设置和读取 S 型曲线参数，其中 `J` 和 `Jd` 分别表示加加速度和减加速度。对首次使用的项目，可先通过 EzSetProfile 完成基础参数整定；当需要降低机械冲击、振动或噪声时，再增加 EzSetSProfile 形成更平滑的 S 型过渡。

当轴已经进入运行过程后，可通过在线修调接口进行微调：

- EzChangeSpeed：在线修调运行速度。
- EzUpdateTargetPosition：在线修调目标位置。
- EzUpdateTargetPositionEx：在线修调目标位置，并按软着陆思路处理末段接近。

运行时修调适合用于工艺补偿、节拍微调、追位或来料偏差修正，但前提是机械系统允许在线改变轨迹。如果现场对安全窗口或工艺节拍控制较严，建议先在仿真模式或低速条件下验证，再转入正式生产。

EzMoveEx 用于在一次调用中同时下发目标位置和全套运动参数，适合高速节拍或需要“参数与动作同步生效”的场景。EzSoftEndMove 强调末段柔和接近目标，常用于接触、贴合、压合前的减冲击控制；EzSoftStartMove 强调起步阶段更柔和，适合希望降低起动冲击或配合前段工艺等待的场景。

本章新增的另一项能力是电子齿轮耦合。电子齿轮适合把一根从轴按比例跟随另一根主轴运动，常用于跟随送料、同步搬运、相位跟随或简化版电子凸轮之前的比例联动场景。其典型接口包括：

- EzSetGearProfile：设置从轴的电子齿轮配置
- EzGetGearProfile：回读电子齿轮配置
- EzGearIn：让从轴进入齿轮耦合
- EzGearOut：让从轴退出齿轮耦合
- EzGetGearState：读取当前是否已进入齿轮耦合

其中`EzGearProfile` 的关键字段包括：

- `master_axis`：主轴 ID
- `ratio`：从轴相对主轴的齿轮比
- `protect.enable`：是否开启齿轮误差保护
- `protect.decel_stop_error`：误差超过该阈值时执行减速停止
- `protect.emg_stop_error`：误差超过该阈值时执行急停

对于首次接入的项目，建议先完成单轴调试与主从轴方向确认，再启用电子齿轮。否则即使齿轮比设置正确，也可能因为方向、零点或机械传动关系错误而导致跟随方向相反或同步误差异常。

1.5.4.4 调用流程

1. 先调用 EzSetProfile 配置基础运动参数。
2. 若需要更平滑的加减速过渡，再调用 EzSetSProfile 增加 S 型曲线参数；若无此需求，可直接按梯形曲线使用。
3. 需要时调用 EzGetProfile 回读基础参数；若已启用 S 型平滑，再调用 EzGetSProfile 回读确认。
4. 通过 EzMoveEx、EzSoftEndMove 或 EzSoftStartMove 发起运动。
5. 在运动过程中,按工艺要求调用 EzChangeSpeed、EzUpdateTargetPosition 或 EzUpdateTargetPositionEx 进行在线修调。
6. 若项目需要主从比例跟随，则先调用 EzSetGearProfile 写入齿轮配置，再调用 EzGearIn 进入耦合；退出时调用 EzGearOut，并通过 EzGetGearState 确认状态。

1.5.4.5 示例代码

```
int axis_id = 0;
int rc = EzSetProfile(axis_id, 0.0, 200.0, 800.0, 800.0, 0.0);
if (rc != 0) {
    printf("EzSetProfile failed: %s\n", EzGetErrorText(rc));
    return;
}

// 需要 S 型平滑时再增加该组参数
rc = EzSetSProfile(axis_id, 5000.0, 5000.0);
if (rc != 0) {
    printf("EzSetSProfile failed: %s\n", EzGetErrorText(rc));
    return;
}

rc = EzMoveEx(axis_id, 300.0, 0.0, 200.0, 800.0, 800.0, 0.0, 5000.0, 5000.0,
    MOVE_TYPE_ABSOLUTE);
if (rc != 0) {
    printf("EzMoveEx failed: %s\n", EzGetErrorText(rc));
    return;
}

EzChangeSpeed(axis_id, 150.0, 600.0);
EzUpdateTargetPosition(axis_id, 320.0);
EzUpdateTargetPositionEx(axis_id, 310.0, 325.0);
```

```
EzGearProfile gear = {0};
gear.master_axis = 0;
gear.ratio = 2.0;
gear.protect.enable = true;
gear.protect.decel_stop_error = 1.0;
gear.protect.emg_stop_error = 3.0;

EzSetGearProfile(1, &gear);
EzGearIn(1);

bool in_gear = false;
EzGetGearState(1, &in_gear);

EzGearOut(1);
```

1.5.4.6 注意事项

- 梯形曲线参数由 EzSetProfile 控制，S 型过渡参数由 EzSetSProfile 控制，两者应结合工艺一起整定。

- 在线修调应建立在“运动已开始、机械允许、工艺允许”的前提上，不建议把它当作常规容错手段。
- EzMoveEx 适合在同一时刻同时确定参数和目标的场景，避免先设参、后运动之间的时间差。
- EzSoftEndMove 更强调终点前的柔和接近，EzSoftStartMove 更强调起步阶段的柔和进入，应按工艺意图选择。
- 对高速、高惯量或接触式机构，建议先在低速条件下验证软启动、软着陆参数。
- 电子齿轮耦合前，应先确认主轴与从轴都处于稳定、可运行状态，且比例与方向已经验证。
- 开启齿轮误差保护后，`decel_stop_error` 与 `emg_stop_error` 应按机械允许误差窗口设置，不宜直接套用经验值。

1.5.5 回零

1.5.5.1 功能概述

回零用于建立轴的参考原点，是后续绝对位置控制、工件坐标建立和安全联机的基础。本章仅介绍控制器回零主流程。

当前版本的控制器回零主流程,建议统一采用 EzSetAxisHomeProfile + EzAxisHome + EzGetHomeState 这一套接口。这样可以把回零策略、速度、偏移量和现场开关信号统一收敛在控制器侧进行管理。

1.5.5.2 相关接口

- EzPower
- EzSetAxisHomeProfile
- EzGetAxisHomeProfile
- EzAxisHome
- EzGetHomeState

1.5.5.3 使用说明

控制器回零的核心入口是 EzSetAxisHomeProfile。其中：

- home_mode 用于选择控制器回零方式，对应 6098h 回原方式。
- high_velocity 用于搜索阶段的高速运行。
- low_velocity 用于最终接近原点阶段的低速运行。
- acceleration 用于控制回零过程的加速过程。
- offset 用于回零完成后建立用户坐标偏移。

本节涉及的现场信号定义如下：

- N-OT：负向超程开关
- P-OT：正向超程开关
- HW：原点开关
- Z：编码器 Z 信号

回零前应先对目标轴执行 EzPower(axis_id, true) 使能。实际投运前，建议先在仿真或低速条件下确认方向、开关逻辑、偏移量和模式编号设置。

1.5.5.3.1 控制器回零模式 17-30

17-30 模式省去了最后一步“寻找 Z 信号”的动作，改为在指定的原点信号边沿到达后立即停机。因此，这一组模式适合“不依赖编码器 Z 信号，而是直接以开关边沿作为最终停机依据”的控制器回零场景。

为便于现场使用，可以按以下几组理解：

- 模式 17：最终在 N-OT 下降沿停机。
- 模式 18：最终在 P-OT 下降沿停机。
- 模式 19 / 20：最终在 HW 变化边沿停机。
- 模式 21 / 22：使用负向原点开关类轨迹，最终在 HW 变化边沿停机。
- 模式 23 / 24 / 25 / 26：带正向限位避让逻辑，最终在 HW 边沿停机。
- 模式 27 / 28 / 29 / 30：带负向限位避让逻辑，最终在 HW 边沿停机。

1.5.5.3.1.1 模式 17

- 原点检出位置不是 Z 信号，而是 N-OT 变化位置。
- NOT 未分配时，驱动器会产生 Homing error。

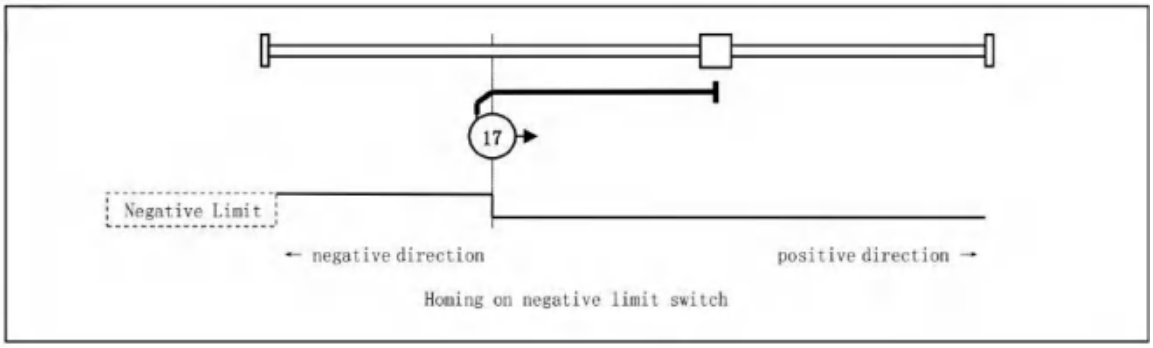

图 1.1 回零方式 17

1.5.5.3.1.2 模式 18

- 原点检出位置不是 Z 信号，而是 P-OT 变化位置。
- POT / 正向限位输入逻辑需要与工程配置一致。

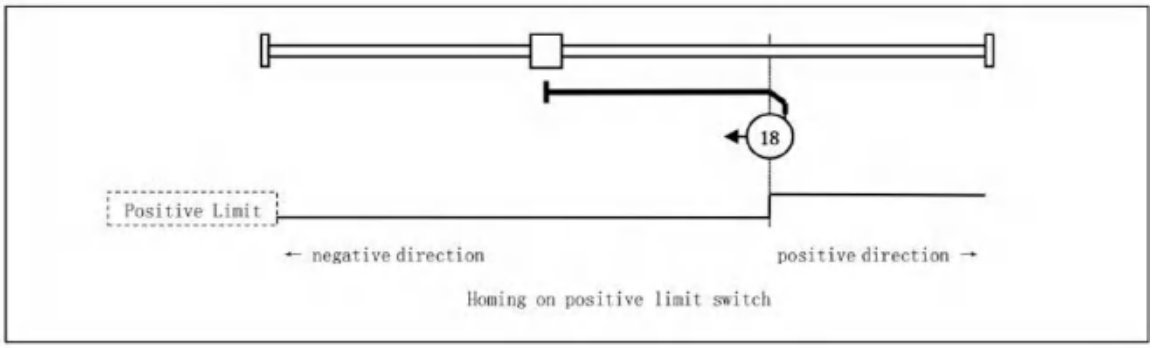

图 1.2 回零方式 18

1.5.5.3.1.3 模式 19 / 20

- 原点检出位置不是 Z 信号，而是 HOME 开关变化位置。
- 两者差异在于停机边沿不同，因此现场必须确认 HOME 的有效边沿和安装方向。

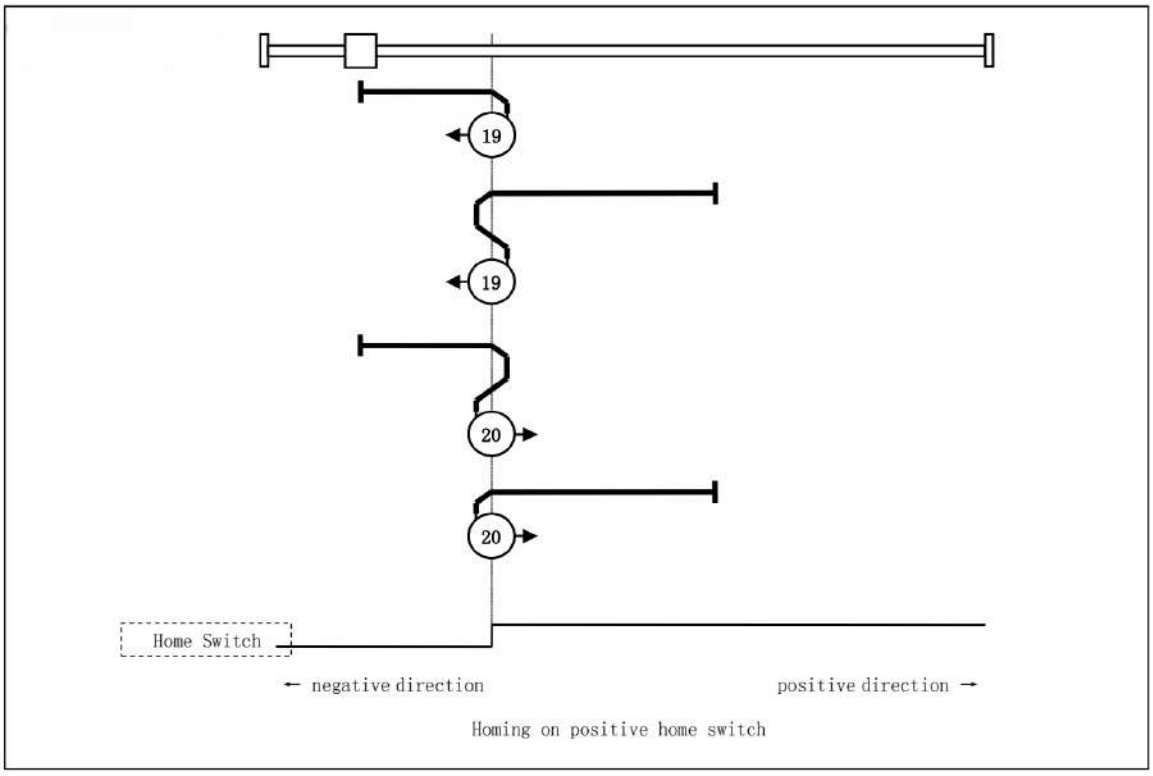

图 1.3 回零方式 19,20

1.5.5.3.1.4 模式 21 / 22

- 使用负向原点开关类轨迹。
- 原点检出位置不是 Z 信号，而是 HOME 开关变化位置。

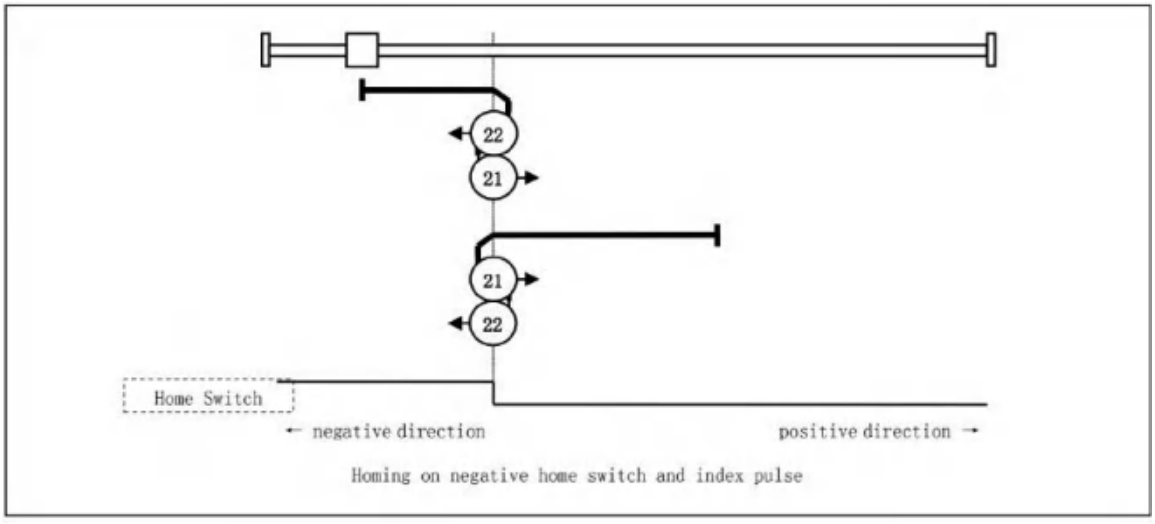

图 1.4 回零方式 21,22

1.5.5.3.1.5 模式 23 / 24 / 25 / 26

- 含正向限位避让逻辑。
- 原点检出位置不是 Z 信号，而是 HOME 开关变化位置。
- HOME、POT 的接线和边沿逻辑必须与现场完全一致。

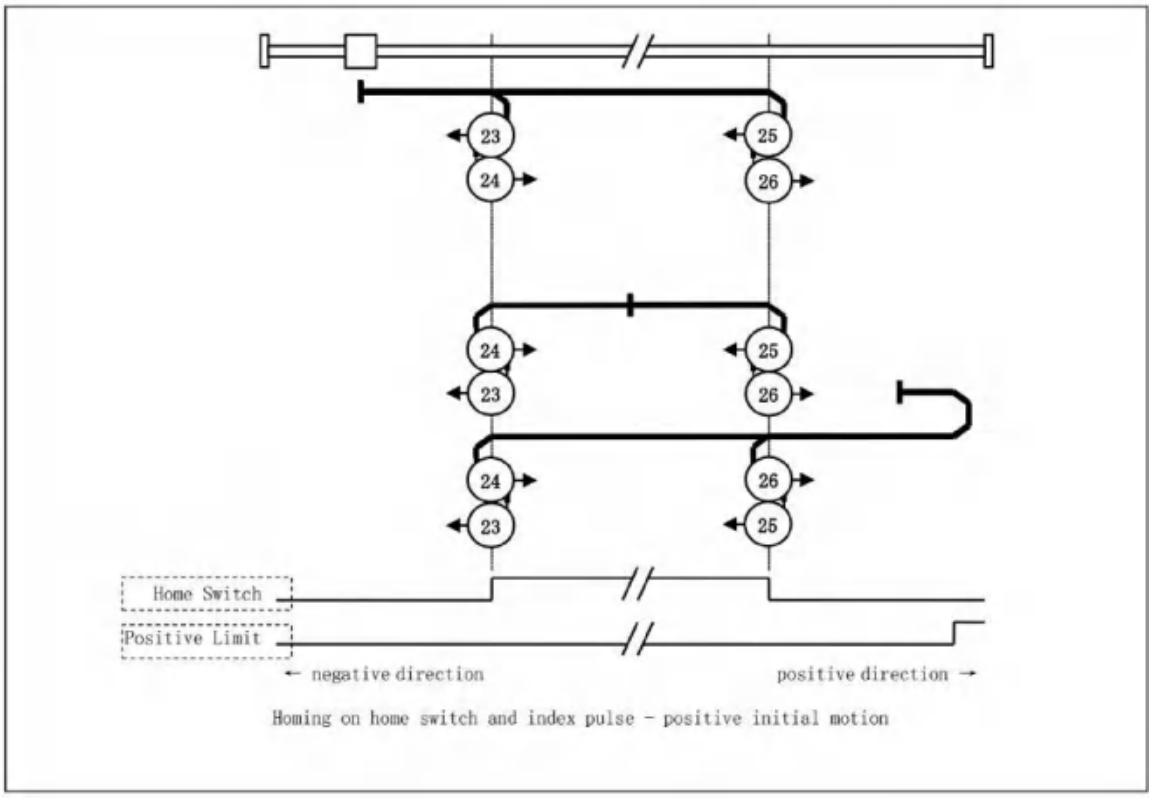

图 1.5 回零方式 23-26

1.5.5.3.1.6 模式 27 / 28 / 29 / 30

- 含负向限位避让逻辑。
- 原点检出位置不是 Z 信号，而是 HOME 开关变化位置。
- HOME、NOT 的接线和方向定义必须先确认。

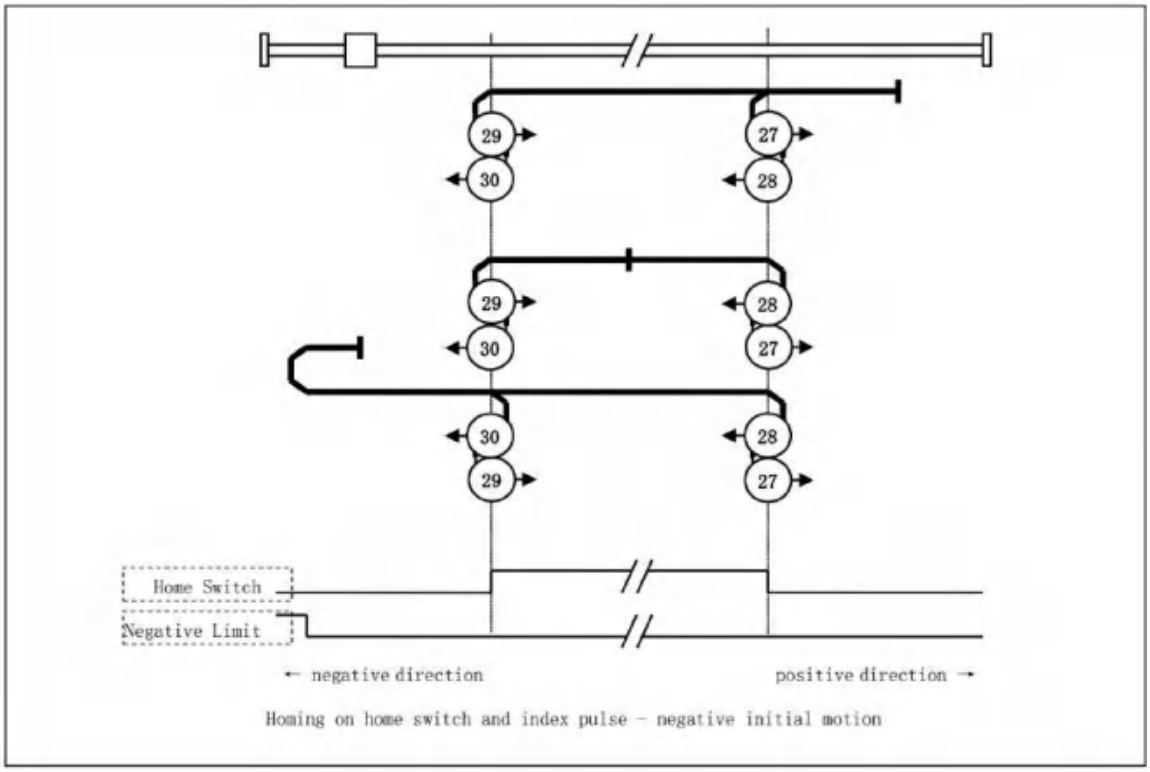

图 1.6 回零方式 27-30

1.5.5.3.2 控制器回零模式 35

模式 35 不执行寻点运动。触发控制器回零后，系统直接把当前位置当作机械原点使用，并将用户当前位置设置为 0。因此，模式 35 的关键不是“去找开关”，而是“确认当前停留位置已经可靠地代表原点位置”。

它适用于以下场景：

- 设备已经由人工对准工艺原点
- 设备已经由治具、挡块或外部工装固定在参考位置
- 不希望再执行任何回零运动，只希望把当前坐标确认成原点

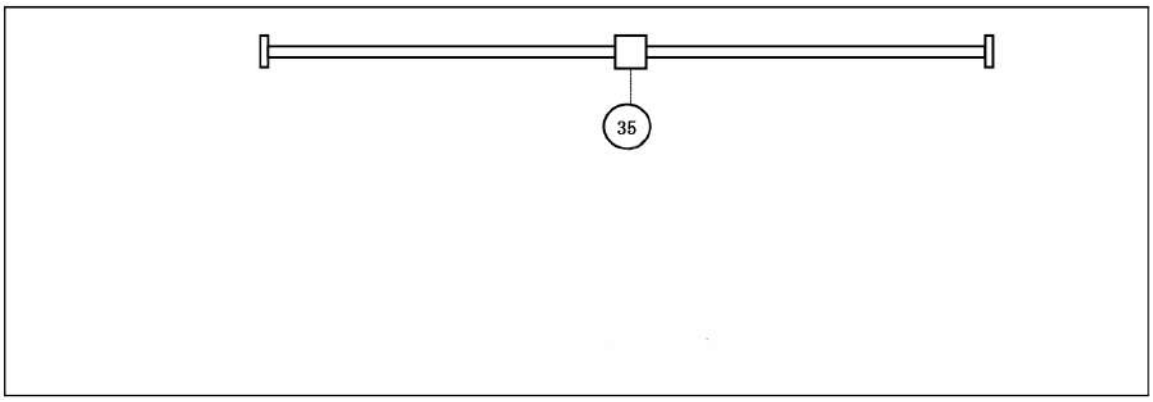

图 1.7 回零方式 35

1.5.5.4 调用流程

1. 确认控制器已启动，调用 EzPower(axis_id, true) 使能目标轴。
2. 调用 EzSetAxisHomeProfile 写入控制器回零模式、低速/高速、加速度和偏移量。
3. 调用 EzAxisHome 开始控制器回零。
4. 周期调用 EzGetHomeState，直到状态变为AXIS_HOME_HOMED。
5. 回零完成后，再执行绝对运动或工艺动作。

1.5.5.5 示例代码

```
int axis_id = 0;
EzHomeState home_state = AXIS_HOME_UNHOMED;

int rc = EzPower(axis_id, true);
if (rc != 0) {
    printf("EzPower failed: %s\n", EzGetErrorText(rc));
    return;
}

// 以下以控制器回零模式 19（HW 下降沿停机）为例
rc = EzSetAxisHomeProfile(axis_id, 19, 5.0, 20.0, 50.0, 0.0);
if (rc != 0) {
    printf("EzSetAxisHomeProfile failed: %s\n", EzGetErrorText(rc));
    return;
}

rc = EzAxisHome(axis_id);
if (rc != 0) {
    printf("EzAxisHome failed: %s\n", EzGetErrorText(rc));
    return;
}

do {
    rc = EzGetHomeState(axis_id, &home_state);
} while (rc == 0 && home_state == AXIS_HOME_HOMING);

printf("home_state=%d\n", home_state);
```

1.5.5.6 注意事项

- 回零前应先确认机械零点方向、限位信号和原点开关逻辑正确。
- 控制器回零流程建议固定为：EzPower 使能 -> EzSetAxisHomeProfile -> EzAxisHome -> EzGetHomeState。
- 17-30 模式的核心特征是“轨迹沿用 1-14，但省去最终找 Z 步骤”，因此应重点确认最终停机边沿是否符合工艺要求。
- 模式 35 不执行寻点运动，只适用于“当前位置已经可以被确认成原点”的场景。
- 回零过程中不建议再调用 EzSetPosition 改写坐标基准。
- 当前手册仅展开控制器回零主流程；伺服回零接口保留在函数参考中，不作为本章推荐主路径展开。

1.5.6 轴组插补

1.5.6.1 功能概述

轴组插补用于协调多根轴完成直线、圆弧等联动轨迹。本接口既支持“先构建轨迹，再统一启动”的缓存式路径，也支持“直接下发单段轨迹”的即时执行路径，便于分别适配连续加工和单次定位两类任务。

1.5.6.2 相关接口

- EzSetAxisGroupProfile
- EzSetAxisGroupSProfile
- EzGetAxisGroupProfile
- EzGetAxisGroupSProfile
- EzAxisGroupOpen
- EzAxisGroupStart
- EzAxisGroupStop
- EzAxisGroupPause
- EzAxisGroupClose
- EzGetAxisGroupCycleData

- EzAxisGroupChangeFeedRate
- EzAxisGroupAddLine
- EzAxisGroupAddLineEx
- EzAxisGroupAddArcCenter
- EzAxisGroupAddArcCenterEx
- EzAxisGroupAddArcRadius
- EzAxisGroupAddArcRadiusEx
- EzAxisGroupAddArc3Point
- EzAxisGroupAddArc3PointEx
- EzAxisGroupAddDelay
- EzAxisGroupWaitInput
- EzAxisGroupWriteOutBit
- EzAxisGroupAddEnd
- EzAxisGroupMoveLine
- EzAxisGroupMoveArcCenter
- EzAxisGroupMoveArcRadius
- EzAxisGroupMoveArc3Point
- EzUpdateCyclicPosition

1.5.6.3 使用说明

轴组曲线参数分为两层：EzSetAxisGroupProfile 用于设置最大速度、加速度和减速度，适合普通梯形节拍；EzSetAxisGroupSProfile 用于设置 S 型过渡参数，适合对轨迹平顺性要求更高的联动场景。设置后可通过 EzGetAxisGroupProfile 和 EzGetAxisGroupSProfile 回读确认。

对于需要单段差异化节拍的连续轨迹，可使用EzAxisGroupSegmentProfileMask 和EzAxisGroupSegmentProfile
通过 EzAxisGroupAddLineEx、EzAxisGroupAddArc∗Ex 在单段级别覆盖速度或加速度等参数。当前版本建议按以下口径理解：vs 生效；a/d 当前仅使用 a；j/jd 当前仅使用 j，且仅在全局 S 型曲线下有效；ve 当前仅支持 0。

轴组使用上建议明确区分两条路径：

路径	适用场景	典型接口
先构建再启动	多段连续轨迹、需要暂停/继续、需要统一起跑的加工任务	EzAxisGroupOpen -> EzAxisGroupAddLine / EzAxisGroupAddArcCenter / EzAxisGroupAddArcRadius / EzAxisGroupAddArc3Point / EzAxisGroupAddLineEx / EzAxisGroupAddArcCenterEx / EzAxisGroupAddArcRadiusEx / EzAxisGroupAddArc3PointEx / 工艺段接口 -> EzAxisGroupAddEnd -> EzAxisGroupStart
直接运动	单段即发即走、无需预先缓存多段轨迹的任务	EzAxisGroupMoveLine、EzAxisGroupMoveArcCenter、EzAxisGroupMoveArcRadius、EzAxisGroupMoveArc3Point

在轨迹运行过程中，可通过 EzGetAxisGroupCycleData 监控当前行号、轴数、各轴指令位置和路径速度；通过 EzAxisGroupChangeFeedRate 在线修调倍率；通过 EzAxisGroupPause 和 EzAxisGroupStop 实现暂停和收停。若加工节拍中需要非几何动作，还可在缓存式路径中插入 EzAxisGroupAddDelay、EzAxisGroupWaitInput、EzAxisGroupWriteOutBit 等工艺段。任务结束后，调用 EzAxisGroupClose 释放轴组。

除上述两条常规轴组插补路径外，EzUpdateCyclicPosition 还可用于按轨迹文件导入时间-位置序列的周期位置更新。它更适合作为点表型轨迹的补充能力，不属于常规的“先构建再启动”或“直接运动”插补流程，项目设计时应单独规划其文件来源和调用时序。

1.5.6.4 调用流程

1. 选定参与联动的轴，并准备轴 ID 列表。
2. 调用 EzAxisGroupOpen 建立轴组。
3. 调用 EzSetAxisGroupProfile / EzSetAxisGroupSProfile 设置轨迹参数。
4. 按需求在“先构建再启动”和“直接运动”之间选择其一。
5. 若采用“先构建再启动”，依次调用 EzAxisGroupAddLine、EzAxisGroupAddLineEx、EzAxisGroupAddArc*、EzAxisGroupAddArc*Ex、EzAxisGroupAddDelay、EzAxisGroupWaitInput、EzAxisGroupWriteOutBit 等接口构建轨迹，最后调用 EzAxisGroupAddEnd 和 EzAxisGroupStart。
6. 若采用“直接运动”，直接调用 EzAxisGroupMoveLine 或圆弧类接口执行单段轨迹。

7. 运行过程中使用 EzGetAxisGroupCycleData、EzAxisGroupChangeFeedRate、EzAxisGroupPause、EzAxisGroupStop 进行监控和控制。
8. 完成后调用 EzAxisGroupClose。

1.5.6.5 示例代码

```
int group_id = 0;
unsigned short axis_ids[2] = {0, 1};
double line_target[2] = {100.0, 50.0};
double arc_target[2] = {150.0, 100.0};
double arc_center[2] = {120.0, 80.0};

int rc = EzAxisGroupOpen(group_id, 2, axis_ids);
if (rc != 0) {
    printf("EzAxisGroupOpen failed: %s\n", EzGetErrorText(rc));
    return;
}

EzSetAxisGroupProfile(group_id, 300.0, 1000.0, 1000.0);
EzSetAxisGroupSProfile(group_id, 8000.0, 8000.0);

EzAxisGroupAddLine(group_id, 2, line_target, MOVE_TYPE_ABSOLUTE);
EzAxisGroupAddArcCenter(group_id, 2, arc_target, arc_center, ARC_CW, MOVE_TYPE_ABSOLUTE);
EzAxisGroupAddEnd(group_id);
EzAxisGroupStart(group_id);

EzAxisGroupChangeFeedRate(group_id, 0.8);
EzAxisGroupPause(group_id);
EzAxisGroupStop(group_id);
EzAxisGroupClose(group_id);
```

```
int group_id = 1;
unsigned short axis_ids[2] = {0, 1};
double direct_target[2] = {200.0, 0.0};

EzAxisGroupMoveLine(group_id, 2, axis_ids, direct_target, MOVE_TYPE_ABSOLUTE);
```

补充能力示例：当项目需要给单段轨迹覆盖节拍，并在路径中插入等待输入、写出输出和延时动作时，可使用 Ex 版本和工艺段接口。

```
EzAxisGroupSegmentProfileEx seg = {};
seg.mask = EZ_AXIS_GROUP_SEGMENT_PROFILE_VS | EZ_AXIS_GROUP_SEGMENT_PROFILE_A;
```

```
seg.vs = 120.0;
seg.a = 600.0;

EzAxisGroupAddLineEx(group_id, 2, line_target, MOVE_TYPE_ABSOLUTE, &seg);
EzAxisGroupWaitInput(group_id, 0, 1, true, 2.0);
EzAxisGroupWriteOutBit(group_id, 0, 0, true, 0.05);
EzAxisGroupAddDelay(group_id, 50.0);
EzAxisGroupAddEnd(group_id);
```

补充能力示例：当项目采用点表文件驱动单轴周期位置更新时，可单独使用 EzUpdateCyclicPosition。

```
EzUpdateCyclicPosition(0, "traj\\axis0.csv");
```

1.5.6.6 注意事项

- 使用轴组前，应先确认组内各轴已经完成使能、回零和坐标准备。
- “先构建再启动”更适合多段连续加工；“直接运动”更适合单段即时动作，两种路径应按任务特性选择。
- EzAxisGroupChangeFeedRate 适合节拍微调，但倍率调整范围应结合工艺允许值确定。
- EzGetAxisGroupCycleData 中的 `line_no` 可用于判断当前运行到哪一段轨迹。
- EzUpdateCyclicPosition 属于点表型周期更新能力，使用时应与普通插补轨迹分别管理文件和调用时序。
- 使用 `EzAxisGroupAdd*Ex` 时，段级参数覆盖应显式设置 `mask`；未置位的字段会继承轴组全局参数。
- EzAxisGroupWaitInput 和 EzAxisGroupWriteOutBit 中使用的终端 ID、DI/DO 索引应先通过 EzGetTerminalInfo 核对。

1.5.7 终端 IO 控制

1.5.7.1 功能概述

本章主要介绍终端输入输出、模拟量、脉冲与编码器读写能力，以及轴侧数字输入读取方式。这些接口常用于工位触发、气缸控制、模拟量采集、计数器调试和仿真联调。

1.5.7.2 相关接口

- EzGetDI
- EzSetDO
- EzGetDO
- EzSetAO
- EzGetAO
- EzGetAI
- EzGetPulseCount
- EzGetEncoder
- EzSetPulseFrequency
- EzGetPulseFrequency
- EzSetDI
- EzSetAI
- EzSetPulseCount
- EzSetEncoder
- EzGetAxisDI

1.5.7.3 使用说明

终端 IO 一般以“终端 ID + 通道索引”的方式访问。建议先通过 EzGetTerminalInfo 确认终端编号以及 DI、DO、AI、AO、脉冲计数、编码器和脉冲输出通道数量，再进行具体读写。

常见调用方式如下：

- EzGetDI / EzSetDO / EzGetDO：数字量输入输出。
- EzSetAO / EzGetAO / EzGetAI：模拟量输出与输入。
- EzGetPulseCount / EzGetEncoder：读取脉冲计数和编码器值。
- EzSetPulseFrequency / EzGetPulseFrequency：设置和读取脉冲输出频率。
- EzGetAxisDI：读取轴侧数字输入，适合与轴状态联动使用。

EzSetDI、EzSetAI、EzSetPulseCount、EzSetEncoder 用于在仿真模式下向输入类数据注入测试值，适合上位机联调、界面联调和逻辑验证。

1.5.7.4 调用流程

1. 启动控制器并读取终端信息。
2. 按终端 ID 和索引读取 DI、AI、计数器或编码器数据。
3. 根据工艺逻辑设置 DO、AO 或脉冲输出频率。
4. 需要仿真联调时，在仿真模式下使用 EzSetDI、EzSetAI、EzSetPulseCount、EzSetEncoder 注入测试数据。
5. 需要读取轴侧输入时调用 EzGetAxisDI。

1.5.7.5 示例代码

```
int terminal_id = 0;
bool di_value = false;
bool do_value = false;
short ai_value = 0;
short ao_value = 0;
int pulse_count = 0;
int encoder_value = 0;
int pulse_frequency = 0;

EzGetDI(terminal_id, 0, &di_value);
EzSetDO(terminal_id, 0, di_value);
EzGetDO(terminal_id, 0, &do_value);

EzSetAO(terminal_id, 0, 1200);
EzGetAO(terminal_id, 0, &ao_value);
EzGetAI(terminal_id, 0, &ai_value);

EzGetPulseCount(terminal_id, 0, &pulse_count);
EzGetEncoder(terminal_id, 0, &encoder_value);
EzSetPulseFrequency(terminal_id, 0, 5000);
EzGetPulseFrequency(terminal_id, 0, &pulse_frequency);

// 以下接口仅适用于仿真模式
EzSetDI(terminal_id, 0, true);
EzSetAI(terminal_id, 0, 800);
EzSetPulseCount(terminal_id, 0, 10000);
EzSetEncoder(terminal_id, 0, 2048);

bool axis_di = false;
EzGetAxisDI(0, 0, &axis_di);
```

1.5.7.6 注意事项

- 访问前应先确认终端 ID 和通道索引是否存在，避免超范围操作。
- 输入类模拟写入接口仅适用于仿真模式，切换到真机模式时不应依赖这些接口。
- 对 DO、AO 和脉冲输出的修改会直接影响现场执行元件，建议先确认联锁条件。
- EzGetAxisDI 读取的是轴侧输入，应与通用终端 IO 区分管理。

1.5.8 状态监控与错误处理

1.5.8.1 功能概述

本章主要介绍运行状态采集、限位使能管理以及错误信息转换。建议将这部分能力纳入应用的常驻监控逻辑，用于及时发现轴异常、限位触发、轨迹停机和接口调用失败。

1.5.8.2 相关接口

- EzGetAxisCycleData
- EzGetAxesCycleData
- EzSetSoftLimitEnable
- EzGetSoftLimitEnable
- EzSetHardLimitEnable
- EzGetHardLimitEnable
- EzGetAxisGroupCycleData
- EzGetErrorText
- EzGetEngineStatus
- 相关枚举与结构：`EzEngineState`、`EzEngineStopReason`、`EzBusFatalReason`、`EzAxisState`、`EzAxisLimitStopReason`、`EzAxisGroupState`、`EzAxisCycleData`、`EzAxisGroupCycleData`、`EzEngineStatus`

1.5.8.3 使用说明

EzGetAxisCycleData 用于读取单轴周期数据。EzAxisCycleData 中包含轴状态、指令位置、反馈位置、速度、力矩以及正负限位标记。其中：

- state 对应EzAxisState，可用于区分去使能、静止、回零中、停止中、离散运动、连续运动、同步运动和错误停止等状态。
- limit_stop_reason 对应EzAxisLimitStopReason，可用于判断本次停机是否由正/负软限位或正/负硬限位引起。
- limit_pos、limit_neg 可用于快速判断当前正向或负向限位输入状态。

EzGetAxesCycleData 用于一次读取当前工程内所有轴的周期数据。返回数组顺序按轴 id 排列，适合做多轴界面刷新、主从轴联动监控和“同一轮采样”下的状态比对；相比上层逐轴循环调用 EzGetAxisCycleData，它更适合作为全轴监控入口。

EzSetSoftLimitEnable / EzGetSoftLimitEnable 和 EzSetHardLimitEnable / EzGetHardLimitEnable 用于在运行时管理限位保护开关。一般建议在项目初始化时明确限位策略，只有在调试或特殊工艺确认后才临时调整。

EzGetAxisGroupCycleData 用于读取轴组周期数据。EzAxisGroupCycleData 中包含轴组状态、当前轨迹行号、组内轴数、各轴指令/实际位置以及路径速度，适合在插补监控界面中周期刷新。

EzGetEngineStatus 用于读取当前引擎整体状态。通过EzEngineStatus 中的 state、stopReason 和 fatalReason，可以快速判断当前引擎处于未初始化、已初始化、已启动、停止中还是已停止状态，并进一步区分用户主动停机还是总线致命故障导致的停机。

所有 Ez* 接口失败时都会返回错误码，可用 EzGetErrorText 转换为可读文本。建议在取得错误码后立即读取并输出错误文本，不要长期保存该指针，日志或告警中如需长期使用应及时复制成应用自己的字符串。

1.5.8.4 调用流程

1. 仅关注单轴时，周期调用 EzGetAxisCycleData 监控指定轴状态。
2. 需要全轴同步监控时，调用 EzGetAxesCycleData 一次读取所有轴周期数据。
3. 若发现限位触发或状态异常，读取软限位/硬限位使能状态确认当前保护策略。
4. 对插补任务同时调用 EzGetAxisGroupCycleData 监控轴组运行段号和路径速度。
5. 对整机状态监控同时调用 EzGetEngineStatus，记录引擎状态、停止原因和总线致命故障原因。
6. 任何接口返回非零错误码时，立即调用 EzGetErrorText 输出错误文本。
7. 根据状态枚举、限位原因和引擎停止原因决定复位、停机或人工干预流程。

1.5.8.5 示例代码

```
EzAxisCycleData axis_data;
EzAxisGroupCycleData group_data;
bool soft_limit_enable = false;
bool hard_limit_enable = false;

int rc = EzGetAxisCycleData(0, &axis_data);
if (rc != 0) {
    printf("EzGetAxisCycleData failed: %s\n", EzGetErrorText(rc));
    return;
}

EzGetSoftLimitEnable(0, &soft_limit_enable);
EzGetHardLimitEnable(0, &hard_limit_enable);
EzGetAxisGroupCycleData(0, &group_data);

printf("axis_state=%d, cmd_pos=%.3f, fb_pos=%.3f, limit_reason=%d\n",
       axis_data.state,
       axis_data.cmd_position,
       axis_data.feedback_position,
       axis_data.limit_stop_reason);

printf("group_state=%d, line_no=%u, path_vel=%.3f\n",
       group_data.state,
       group_data.line_no,
       group_data.actual_path_velocity);

if (!soft_limit_enable) {
    EzSetSoftLimitEnable(0, true);
}
if (!hard_limit_enable) {
    EzSetHardLimitEnable(0, true);
}
```

1.5.8.6 注意事项

- 周期数据适合做持续监控，不建议只在报错后再临时读取。
- 运行时修改限位使能前，应先确认机械位置和现场安全状态。
- `EzAxisState` 与`EzAxisGroupState` 都属于运行态判断基础，建议在界面和日志中统一映射为可读状态文本。
- 当 `limit_stop_reason` 不为`EZ_AXIS_LIMIT_STOP_REASON_NONE` 时，应先排查限位来源，再决定是否复位或重启运动。

- EzGetErrorText 返回的是英文错误文本，建议在客户应用中配合中文业务提示一起记录，便于现场排查。

1.6 教程

1.6.1 教程 1：启动一个工程

1.6.1.1 目标

完成一次最小可运行工程启动：初始化客户端、设置工作模式、启动控制器，并读取主站与轴信息确认工程已上线。

1.6.1.2 前置条件

- 已准备当前 KRM 项目使用的工程/配置文件路径（如 `config\\project.krm`）。
- 控制器运行环境可用；若仅做联调，建议先使用仿真模式。

1.6.1.3 操作步骤

1. 调用 EzInit 建立会话。
2. 调用 EzSetControllerWorkMode(true)（仿真）或 EzSetControllerWorkMode(false)（真机）。
3. 调用 EzStart 传入当前 KRM 项目使用的工程/配置文件路径。
4. 调用 EzGetMasterInfo 读取主站数量与索引。
5. 调用 EzGetAxesInfo 读取轴列表，确认轴 ID 与名称。

```
EzMasterInfo* masters = 0;
EzAxisInfo* axes = 0;
int master_count = 0;
int axis_count = 0;

int rc = EzInit();
if (rc != 0) return;
rc = EzSetControllerWorkMode(true);
if (rc != 0) return;
rc = EzStart(”config\\project.krm”);
if (rc != 0) return;

EzGetMasterInfo(&masters, &master_count);
EzGetAxesInfo(&axes, &axis_count);
```

1.6.1.4 结果确认

- EzStart 返回 0。
- `master_count >= 1`。
- `axis_count` 与项目配置一致，轴名称可正常读取。

1.6.1.5 常见问题

- 启动失败：先检查 EzSetControllerWorkMode 是否在 EzStart 之前调用。
- 主站或轴数量为 0：检查启动配置文件是否与当前工程一致，或是否加载了错误路径。

1.6.2 教程 2：完成一次单轴绝对运动

1.6.2.1 目标

让一根轴从当前位置运行到指定绝对坐标，并通过“到位完成或超时中止”的方式形成可交付的基础动作闭环。

1.6.2.2 前置条件

- 教程 1 已完成，控制器已启动。
- 目标轴已完成基础调试，机械行程与限位安全可用。

1.6.2.3 操作步骤

1. 调用 EzPower(axis_id, true) 使能目标轴。
2. 调用 EzSetProfile 设置基础梯形参数。
3. 调用 EzMove(axis_id, target, MOVE_TYPE_ABSOLUTE) 发起绝对运动。
4. 循环调用 EzGetAxisCycleData 观察 `feedback_position` 与 `state`,若进入`AXIS_STATE_STANDSTILL` 则按成功完成处理。
5. 若超过设定超时时间仍未完成，调用 EzAxisStop(axis_id, STOP_MODE_CONTROLLED) 做超时中止。

```
int axis_id = 0;
EzAxisCycleData data;
bool reached = false;
int elapsed_ms = 0;
const int poll_interval_ms = 10;
const int timeout_ms = 5000;

EzPower(axis_id, true);
EzSetProfile(axis_id, 0.0, 100.0, 300.0, 300.0, 0.0);
EzMove(axis_id, 120.0, MOVE_TYPE_ABSOLUTE);

while (elapsed_ms < timeout_ms) {
    if (EzGetAxisCycleData(axis_id, &data) != 0) {
        break;
    }
    if (data.state == AXIS_STATE_STANDSTILL) {
        reached = true;
        break;
    }
    // 等待一个采样周期后再轮询，例如 Sleep(poll_interval_ms)
    elapsed_ms += poll_interval_ms;
}

if (!reached) {
    EzAxisStop(axis_id, STOP_MODE_CONTROLLED);
}
```

1.6.2.4 结果确认

- 正常完成时：轴状态在超时前进入AXIS_STATE_STANDSTILL，且 feedback_position 接近目标值（允许工艺误差）。
- 超时中止时：执行 EzAxisStop 后轴安全停下，并输出超时日志用于排查。

1.6.2.5 常见问题

- 轴不动：先确认 EzPower 是否成功，以及轴是否处于错误停止状态。
- 到位误差大：检查 EzSetProfile 参数是否过于激进，或机械传动存在背隙/跟随偏差。
- 经常超时：检查目标位置是否超出可达范围、速度参数是否过小，或现场存在限位/互锁约束。

1.6.3 教程 3：执行一次回零

1.6.3.1 目标

使用控制器回零模式 19 完成一次标准回零流程，并读取回零结果。

1.6.3.2 前置条件

- 控制器已启动。
- 已确认回零方向、回零开关与限位逻辑。

1.6.3.3 操作步骤

1. 调用 EzPower(axis_id, true) 使能目标轴。
2. 调用 EzSetAxisHomeProfile 设置控制器回零模式、低速/高速、加速度和偏移。
3. 调用 EzAxisHome 启动回零。
4. 循环调用 EzGetHomeState 直到状态不再是AXIS_HOME_HOMING。

```
int axis_id = 0;
EzHomeState home_state = AXIS_HOME_UNHOMED;

int rc = EzPower(axis_id, true);
if (rc != 0) return;

EzSetAxisHomeProfile(axis_id, 19, 3.0, 20.0, 50.0, 0.0);
EzAxisHome(axis_id);

do {
    EzGetHomeState(axis_id, &home_state);
} while (home_state == AXIS_HOME_HOMING);
```

1.6.3.4 结果确认

- EzAxisHome 返回 0。
- EzGetHomeState 最终为AXIS_HOME_HOMED。
- 回零后可继续执行绝对运动命令。

1.6.3.5 常见问题

- 长时间停在回零中：检查回零开关信号是否到位，以及 `home_mode` 选用的停机边沿是否与现场信号逻辑一致。
- 回零后坐标不符合预期：检查 `offset` 设置值与工艺原点定义。
- 若现场不希望轴再执行寻点运动，可重新评估是否应改用控制器回零模式 `35`。

1.6.4 教程 4：创建轴组并执行直线插补

1.6.4.1 目标

创建一个两轴轴组，添加一段直线轨迹并启动执行，再通过周期数据确认轨迹运行状态。

1.6.4.2 前置条件

- 参与插补的各轴已使能并完成回零。
- 已明确轴组内轴 ID 与坐标单位。

1.6.4.3 操作步骤

1. 调用 EzAxisGroupOpen 创建轴组并绑定轴 ID 列表。
2. 调用 EzSetAxisGroupProfile 设置轴组最大速度、加速度、减速度。
3. 调用 EzAxisGroupAddLine 添加直线段。
4. 调用 EzAxisGroupAddEnd 写入轨迹结束标记。
5. 调用 EzAxisGroupStart 启动插补。
6. 循环调用 EzGetAxisGroupCycleData 观察 `line_no`、`state` 和路径速度，若进入 `AXIS_GROUP_STATE_STANDSTI` 则按成功完成处理。
7. 若超时未完成，调用 EzAxisGroupStop 中止轨迹执行。
8. 调用 EzAxisGroupClose(group_id) 关闭轴组，完成本次轴组生命周期。

```
int group_id = 0;
unsigned short axis_ids[2] = {0, 1};
double p1[2] = {80.0, 40.0};
EzAxisGroupCycleData group_data;
bool finished = false;
int elapsed_ms = 0;
const int poll_interval_ms = 10;
const int timeout_ms = 8000;

EzAxisGroupOpen(group_id, 2, axis_ids);
EzSetAxisGroupProfile(group_id, 200.0, 600.0, 600.0);
EzAxisGroupAddLine(group_id, 2, p1, MOVE_TYPE_ABSOLUTE);
EzAxisGroupAddEnd(group_id);
EzAxisGroupStart(group_id);

while (elapsed_ms < timeout_ms) {
    if (EzGetAxisGroupCycleData(group_id, &group_data) != 0) {
        break;
    }
    if (group_data.state == AXIS_GROUP_STATE_STANDSTILL) {
        finished = true;
        break;
    }
    // 等待一个采样周期后再轮询，例如 Sleep(poll_interval_ms)
    elapsed_ms += poll_interval_ms;
}

if (!finished) {
    EzAxisGroupStop(group_id);
}
EzAxisGroupClose(group_id);
```

1.6.4.4 结果确认

- 正常完成时：轴组状态从AXIS_GROUP_STATE_MOVING 回到AXIS_GROUP_STATE_STANDSTILL，且末端位置与期望一致。
- 超时中止时：EzAxisGroupStop 成功执行，轴组停止后可安全关闭。
- 收尾阶段：EzAxisGroupClose(group_id) 执行成功。

1.6.4.5 常见问题

- 启动后不动：检查轴组内是否存在未使能或未回零轴。
- 轨迹中断：检查目标点是否越过软/硬限位，或轨迹参数是否超过机械能力。
- 关闭失败：确认轴组已停止并且没有其他任务占用同一轴组 ID。

1.6.5 教程 5：读取终端输入并控制输出

1.6.5.1 目标

完成一次“单次快照复制”：读取一个 DI 通道当前值，写入到对应 DO 通道，并回读确认。

1.6.5.2 前置条件

- 控制器已启动，终端 ID 与通道号已确认。
- 现场已具备输入信号源，或仿真模式可提供测试信号。

1.6.5.3 操作步骤

1. 调用 EzGetDI 读取 DI 通道当前状态（快照值）。
2. 调用 EzSetDO 将该快照值写入 DO 通道。
3. 调用 EzGetDO 回读 DO，确认本次写入生效。

```
int terminal_id = 0;
bool di0 = false;
bool do0 = false;

EzGetDI(terminal_id, 0, &di0);
EzSetDO(terminal_id, 0, di0);
EzGetDO(terminal_id, 0, &do0);
```

可选补充：如果项目同时需要模拟量联调，可在上述快照复制之外单独增加 EzGetAI / EzSetAO 调试步骤。

1.6.5.4 结果确认

- 本次读取到的 `di0` 已成功写入 DO。
- EzGetDO 回读值与本次下发值一致。

1.6.5.5 常见问题

- DO 无变化：先确认终端 ID、通道号和接线是否正确。
- 结果不一致：检查是否有其他线程/任务同时写同一 DO 通道。
- （可选模拟量联调）AI 波动大：检查采样滤波与现场干扰，必要时降低控制逻辑灵敏度。

1.6.6 教程 6：执行一次 PDO/SDO 访问

1.6.6.1 目标

完成一次 EtherCAT PDO 读写与一次 SDO 读写，建立最小通信验证闭环。

1.6.6.2 前置条件

- 控制器已启动，主站与从站索引已确认。
- 已确认目标对象地址和子索引可读写。

1.6.6.3 操作步骤

1. 使用一组 PDO 接口完成“写入 + 读取”：
   - EzWriteEcatPDOInt
   - EzReadEcatPDOInt
2. 使用一组 SDO 接口完成“写入 + 读取”：
   - EzWriteEcatSDOInt
   - EzReadEcatSDOInt
3. 对比写入值与回读值，确认访问成功。

```
unsigned char master = 0;
unsigned char slave = 0;
int pdo_readback = 0;
int sdo_readback = 0;

EzWriteEcatPDOInt(master, slave, 0x7000, 0x01, 123);
EzReadEcatPDOInt(master, slave, 0x7000, 0x01, &pdo_readback);

EzWriteEcatSDOInt(master, slave, 0x2300, 0x00, 456);
EzReadEcatSDOInt(master, slave, 0x2300, 0x00, &sdo_readback);
```

1.6.6.4 结果确认

- 四次访问均返回 0。
- `pdo_readback` 与 PDO 写入值一致。
- `sdo_readback` 与 SDO 写入值一致。

1.6.6.5 常见问题

- 返回访问失败：优先核对主站号、从站号、对象地址、子索引和数据类型。
- 写入成功但读值异常：确认目标对象是否允许在线写入，或是否需切换设备状态后生效。

1.7 附录

1.7.1 推荐调用时序

以下时序用于帮助客户建立稳定、可复用的调用习惯。实际项目可根据现场联锁、界面流程和安全要求做适配，但建议不要打乱核心生命周期。

1.7.1.1 1. 启动流程

1. EzInit
2. EzSetControllerWorkMode
3. EzStart
4. EzGetMasterInfo / EzGetAxesInfo / EzGetTerminalInfo
5. 进入业务监控或动作准备

适用场景：工程上线、联机调试、程序启动自检。

1.7.1.2 2. 单轴运动流程

1. EzPower
2. EzSetProfile
3. 可选：EzSetSProfile
4. EzMove 或 EzMoveEx
5. 运动中按需调用 EzChangeSpeed、EzUpdateTargetPosition 或 EzUpdateTargetPositionEx
6. EzGetAxisCycleData 监控状态与位置
7. 结束时调用 EzAxisStop

适用场景：点位移动、调试找位、工艺补偿。

1.7.1.3 3. 控制器回零流程

1. EzPower
2. EzSetAxisHomeProfile
3. EzAxisHome
4. EzGetHomeState
5. 回零成功后再进入绝对运动流程

适用场景：工程启动后建立绝对坐标基准。当前版本建议优先使用控制器回零模式，现场可根据信号条件选用 17-30 或 35 等模式。

1.7.1.4 4. 轴组插补流程

缓存式路径：

1. EzAxisGroupOpen
2. EzSetAxisGroupProfile
3. 可选：EzSetAxisGroupSProfile
4. EzAxisGroupAddLine / EzAxisGroupAddArcCenter / EzAxisGroupAddArcRadius / EzAxisGroupAddArc3Point

5. EzAxisGroupAddEnd
6. EzAxisGroupStart
7. EzGetAxisGroupCycleData
8. 按需 EzAxisGroupChangeFeedRate / EzAxisGroupPause / EzAxisGroupStop
9. EzAxisGroupClose

直接运动路径：

1. 准备轴 ID 列表和目标位置
2. EzSetAxisGroupProfile
3. EzAxisGroupMoveLine 或圆弧类 `EzAxisGroupMove*`
4. EzGetAxisGroupCycleData
5. EzAxisGroupStop / EzAxisGroupClose

1.7.1.5 5. 停止与退出流程

1. 结束业务动作并做必要的 EzAxisStop 或 EzAxisGroupStop
2. EzStop
3. 常规退出时调用 EzUnInit
4. 仅在明确需要关闭整个引擎时调用 EzClose

适用场景：程序退出、切换工程、停机维护。

1.7.2 常见错误与处理建议

下表列出现场最常见、且可直接从公开 API 表面看出的使用问题。建议在客户程序中把返回码、错误文本和上下文参数一起记录下来。

常见情况	典型表现	处理建议
未先调用 EzInit 或 EzStart 就直接执行业务接口	扫描、运动、IO、PDO/SDO 接口直接失败	先检查生命周期是否完整执行：EzInit -> EzSetControllerWorkMode -> EzStart。
在 EzStart 之后才调用 EzSetControllerWorkMode	工作模式与预期不一致	将模式设置前移到启动之前，并在日志中记录模式选择。
EzStart 路径错误或配置文件不可读	启动失败	核对配置文件路径、访问权限和部署目录，必要时先打印绝对路径。
轴未使能就调用 EzMove、EzJog 或回零接口	运动不执行或直接失败	先调用 EzPower(id, true)，再检查轴状态是否允许运动。
轴未建立原点就直接做绝对运动	绝对坐标动作与现场位置不一致	先执行回零流程，或确认项目允许在当前阶段使用绝对坐标。
软限位 / 硬限位触发	轴停机，limit_stop_reason 非EZ_AXIS_LIMIT_STOP_↩REASON_NONE	通过 EzGetAxisCycleData、EzGetSoftLimitEnable、EzGetHardLimitEnable 联合排查，再决定是否复位。
直接关闭软限位或硬限位后继续生产	现场存在越界风险	除调试验证外，不建议关闭限位保护；恢复生产前应重新打开。
轴组数组参数长度与 axis_↩number 不一致	轨迹创建失败或执行异常	统一以 axis_number 作为数组长度基准，调用前做参数校验。
缓存式轴组轨迹未写 EzAxisGroupAddEnd 就 EzAxisGroupStart	轨迹执行不符合预期	在缓存式路径中始终把 EzAxisGroupAddEnd 作为启动前的固定步骤。
轴组任务结束后未 EzAxisGroupClose	轴组 ID 长时间占用	在任务结束收尾流程中固定加入 EzAxisGroupClose。
PDO/SDO 地址、子索引或数据类型不匹配	EtherCAT 访问失败或读写值异常	逐项核对主站号、从站号、对象地址、子索引和类型，并与从站对象字典保持一致。
在真机模式下使用 EzSetDI、EzSetAI、EzSetPulseCount、EzSetEncoder	输入注入不生效	这些接口仅用于仿真模式，真机应读取真实硬件输入。
长期保存 EzGetErrorText 返回的指针	后续日志文本异常或被覆盖	调用后立即复制到应用自己的字符串缓存。

1.7.3 术语表

术语	说明
主站	管理现场总线通信的控制对象。公开接口中常通过`EzMasterInfo`、EzScan、EzGetMasterCycleTime 访问。
从站	挂接在主站下的现场设备，如驱动器、IO 模块或带扩展模块的终端。
轴	可执行单轴控制的运动对象。通常通过轴 ID 在 EzPower、EzMove、EzGetAxisCycleData 等接口中访问。
轴组	由多根轴组成的联动对象，用于执行直线、圆弧等插补轨迹。
PDO	Process Data Object，面向过程数据的在线交换对象，适合实时或周期性数据读写。
SDO	Service Data Object，面向参数或配置类数据的访问对象，适合读写参数对象。
软限位	基于软件参数或逻辑判断建立的位置边界保护，可由 EzSetSoftLimitEnable 控制使能状态。
硬限位	基于硬件输入信号建立的位置边界保护，可由 EzSetHardLimitEnable 控制使能状态。
仿真模式	不依赖真实现场输入即可做接口联调的运行模式，通过 EzSetControllerWorkMode(true) 设置。
真机模式	面向真实设备和现场总线的运行模式，通过 EzSetControllerWorkMode(false) 设置。
控制器回零	当前文档推荐使用的回零主流程，通过 EzSetAxisHomeProfile、EzAxisHome、EzGetHomeState 组织。
伺服回零接口	通过 EzSetHomeProfile、EzHome、EzGetHomeState 组织的伺服回零路径，当前文档不作为主流程展开。

第 2 章

.krm 文件说明

.krm 是当前 KRM 项目使用的工程/配置文件，通常由配置调试工具生成并维护。

在编程接口场景中，应用调用EzStart 时应传入可访问、可追踪的.krm 文件路径，以完成控制器启动和工程加载。

交付时建议将.krm 文件与应用版本、设备配置和现场环境保持一致，并确保部署目录具备稳定的读取权限。详细创建和配置方法请参考KRmotion 软件操作手册。

第 3 章

结构体说明

3.1 EzAxisCycleData 结构体参考

```
#include <ez_motion.h>
```

成员变量

- EzAxisState state
  轴状态
- double cmd_position
  轴指令位置
- double cmd_velocity
  轴指令速度
- short cmd_torque
  轴指令力矩
- double feedback_position
  轴反馈位置
- double feedback_velocity
  轴反馈速度
- short feedback_torque
  轴反馈力矩
- bool limit_pos
  轴正限位
- bool limit_neg
  轴负限位
- EzAxisLimitStopReason limit_stop_reason
  轴限位停车原因

3.1.1 结构体成员变量说明

3.1.1.1 cmd_position

```
double cmd_position
```

轴指令位置

3.1.1.2 cmd_torque

```
short cmd_torque
```

轴指令力矩

3.1.1.3 cmd_velocity

```
double cmd_velocity
```

轴指令速度

3.1.1.4 feedback_position

```
double feedback_position
```

轴反馈位置

3.1.1.5 feedback_torque

```
short feedback_torque
```

轴反馈力矩

3.1.1.6 feedback_velocity

```
double feedback_velocity
```

轴反馈速度

3.1.1.7 limit_neg

```
bool limit_neg
```

轴负限位

3.1.1.8 limit_pos

```
bool limit_pos
```

轴正限位

3.1.1.9 limit_stop_reason

```
EzAxisLimitStopReason limit_stop_reason
```

轴限位停车原因

3.1.1.10 state

```
EzAxisState state
```

轴状态

该结构体的文档由以下文件生成:

- ez_motion.h

3.2 EzAxisGroupCycleData 结构体参考

```
#include <ez_motion.h>
```

成员变量

- EzAxisGroupState state
  轴组状态
- unsigned int line_no
  轴组当前行号
- unsigned short axis_count
  轴组轴数
- double cmd_position [EZ_AXIS_GROUP_MAX_AXIS]
  轴组指令位置
- double actual_position [EZ_AXIS_GROUP_MAX_AXIS]
  轴组反馈位置
- double cmd_path_velocity
  轴组指令路径速度
- double actual_path_velocity
  轴组反馈路径速度

3.2.1 结构体成员变量说明

3.2.1.1 actual_path_velocity

```
double actual_path_velocity
```

轴组反馈路径速度

3.2.1.2 actual_position

```
double actual_position[EZ_AXIS_GROUP_MAX_AXIS]
```

轴组反馈位置

3.2.1.3 axis_count

```
unsigned short axis_count
```

轴组轴数

3.2.1.4 cmd_path_velocity

```
double cmd_path_velocity
```

轴组指令路径速度

3.2.1.5 cmd_position

```
double cmd_position[EZ_AXIS_GROUP_MAX_AXIS]
```

轴组指令位置

3.2.1.6 line_no

```
unsigned int line_no
```

轴组当前行号

3.2.1.7 state

```
EzAxisGroupState state
```

轴组状态

该结构体的文档由以下文件生成:

- ez_motion.h

3.3 EzAxisGroupSegmentProfileEx 结构体参考

```
#include <ez_motion.h>
```

成员变量

- unsigned int mask
  覆盖字段掩码
- double vs
  轴组段级速度
- double a
  轴组段级加速度
- double d
  轴组段级减速度, 暂时无效
- double ve
  轴组段级末速度, 暂时只支持 *0*
- double j
  轴组段级加加速度
- double jd
  轴组段级减加速度, 暂时无效

3.3.1 结构体成员变量说明

3.3.1.1 a

```
double a
```

轴组段级加速度

3.3.1.2 d

```
double d
```

轴组段级减速度, 暂时无效

3.3.1.3 j

```
double j
```

轴组段级加加速度

3.3.1.4 jd

```
double jd
```

轴组段级减加速度, 暂时无效

3.3.1.5 mask

```
unsigned int mask
```

覆盖字段掩码

3.3.1.6 ve

```
double ve
```

轴组段级末速度, 暂时只支持 0

3.3.1.7 vs

```
double vs
```

轴组段级速度

该结构体的文档由以下文件生成:

- ez_motion.h

3.4 EzAxisInfo 结构体参考

```
#include <ez_motion.h>
```

成员变量

- unsigned short id
  轴 *id*
- char name [64]
  轴名称

3.4.1 结构体成员变量说明

3.4.1.1 id

```
unsigned short id
```

轴 id

3.4.1.2 name

```
char name[64]
```

轴名称

该结构体的文档由以下文件生成:

- ez_motion.h

3.5 EzDirectionalEnable 结构体参考

```
#include <ez_motion.h>
```

成员变量

- bool positive
  正向使能
- bool negative
  负向使能

3.5.1 结构体成员变量说明

3.5.1.1 negative

```
bool negative
```

负向使能

3.5.1.2 positive

```
bool positive
```

正向使能

该结构体的文档由以下文件生成:

- ez_motion.h

3.6 EzEngineStatus 结构体参考

```
#include <ez_motion.h>
```

成员变量

- EzEngineState state
  引擎状态
- EzEngineStopReason stopReason
  停止原因
- EzBusFatalReason fatalReason
  总线故障

3.6.1 结构体成员变量说明

3.6.1.1 fatalReason

```
EzBusFatalReason fatalReason
```

总线故障

3.6.1.2 state

```
EzEngineState state
```

引擎状态

3.6.1.3 stopReason

```
EzEngineStopReason stopReason
```

停止原因

该结构体的文档由以下文件生成:

- ez_motion.h

3.7 EzGearErrorProtect 结构体参考

齿轮异常保护配置

```
#include <ez_motion.h>
```

成员变量

- bool enable
  是否开启保护功能
- double decel_stop_error
  错误阈值大于多少时，执行减速停止
- double emg_stop_error
  错误阈值大于多少时，执行急停

3.7.1 详细描述

齿轮异常保护配置

3.7.2 结构体成员变量说明

3.7.2.1 decel_stop_error

```
double decel_stop_error
```

错误阈值大于多少时，执行减速停止

3.7.2.2 emg_stop_error

```
double emg_stop_error
```

错误阈值大于多少时，执行急停

3.7.2.3 enable

```
bool enable
```

是否开启保护功能

该结构体的文档由以下文件生成:

- ez_motion.h

3.8 EzGearProfile 结构体参考

齿轮配置

```
#include <ez_motion.h>
```

成员变量

- int master_axis
  主轴 *id*
- double ratio
  齿轮比
- EzGearErrorProtect protect
  齿轮错误保护

3.8.1 详细描述

齿轮配置

3.8.2 结构体成员变量说明

3.8.2.1 master_axis

```
int master_axis
```

主轴 id

3.8.2.2 protect

```
EzGearErrorProtect protect
```

齿轮错误保护

3.8.2.3 ratio

```
double ratio
```

齿轮比

该结构体的文档由以下文件生成:

- ez_motion.h

3.9 EzMasterInfo 结构体参考

```
#include <ez_motion.h>
```

成员变量

- EzBusType type
  总线类型
- unsigned char index
  总线索引

3.9.1 结构体成员变量说明

3.9.1.1 index

```
unsigned char index
```

总线索引

3.9.1.2 type

```
EzBusType type
```

总线类型

该结构体的文档由以下文件生成:

- ez_motion.h

3.10 EzSlaveInfo 结构体参考

```
#include <ez_motion.h>
```

成员变量

- unsigned int slave_id
  从站 *id*
- unsigned int vendor_id
  厂商 *id*
- unsigned int product_code
  产品代码
- unsigned int revision
  版本号
- unsigned int modules [32]
  模块信息
- unsigned int module_count
  模块数量

3.10.1 结构体成员变量说明

3.10.1.1 module_count

```
unsigned int module_count
```

模块数量

3.10.1.2 modules

```
unsigned int modules[32]
```

模块信息

3.10.1.3 product_code

```
unsigned int product_code
```

产品代码

3.10.1.4 revision

```
unsigned int revision
```

版本号

3.10.1.5 slave_id

```
unsigned int slave_id
```

从站 id

3.10.1.6 vendor_id

```
unsigned int vendor_id
```

厂商 id

该结构体的文档由以下文件生成:

- ez_motion.h

3.11 EzSoftLimitConfig 结构体参考

```
#include <ez_motion.h>
```

成员变量

- bool positive_enable
  正向软限位是否使能
- double positive_position
  正向软限位位置
- bool negative_enable
  负向软限位是否使能
- double negative_position
  负向软限位位置

3.11.1 结构体成员变量说明

3.11.1.1 negative_enable

```
bool negative_enable
```

负向软限位是否使能

3.11.1.2 negative_position

```
double negative_position
```

负向软限位位置

3.11.1.3 positive_enable

```
bool positive_enable
```

正向软限位是否使能

3.11.1.4 positive_position

```
double positive_position
```

正向软限位位置

该结构体的文档由以下文件生成:

- ez_motion.h

3.12 EzTerminalInfo 结构体参考

```
#include <ez_motion.h>
```

成员变量

- unsigned short id
  终端 *id*
- char name [64]
  终端名称
- unsigned short do_number
  轴输出数量
- unsigned short di_number
  轴输入数量
- unsigned short ao_number
  轴输出数量
- unsigned short ai_number
  轴输入数量
- unsigned short pulse_counter_number
  轴脉冲计数器数量
- unsigned short encoder_number
  轴编码器数量
- unsigned short pulse_frequency_number
  轴脉冲频率数量

3.12.1 结构体成员变量说明

3.12.1.1 ai_number

```
unsigned short ai_number
```

轴输入数量

3.12.1.2 ao_number

```
unsigned short ao_number
```

轴输出数量

3.12.1.3 di_number

```
unsigned short di_number
```

轴输入数量

3.12.1.4 do_number

```
unsigned short do_number
```

轴输出数量

3.12.1.5 encoder_number

```
unsigned short encoder_number
```

轴编码器数量

3.12.1.6 id

```
unsigned short id
```

终端 id

3.12.1.7 name

```
char name[64]
```

终端名称

3.12.1.8 pulse_counter_number

```
unsigned short pulse_counter_number
```

轴脉冲计数器数量

3.12.1.9 pulse_frequency_number

```
unsigned short pulse_frequency_number
```

轴脉冲频率数量

该结构体的文档由以下文件生成:

- ez_motion.h

第 4 章

文件说明

4.1 ez_motion.h 文件参考

: None

结构体

- struct EzEngineStatus
- struct EzAxisCycleData
- struct EzDirectionalEnable
- struct EzSoftLimitConfig
- struct EzAxisGroupCycleData
- struct EzAxisGroupSegmentProfileEx
- struct EzGearErrorProtect
  齿轮异常保护配置
- struct EzGearProfile
  齿轮配置
- struct EzSlaveInfo
- struct EzMasterInfo
- struct EzAxisInfo
- struct EzTerminalInfo

宏定义

- #define SOFT_MOTION_LIB_API
- #define EZ_AXIS_GROUP_MAX_AXIS 9

枚举

- enum EzBusType { ETHERCAT = 0 , ETHERMAC = 1 }
- enum EzMoveType { MOVE_TYPE_ABSOLUTE = 0 , MOVE_TYPE_RELATIVE = 1 }
- enum EzMoveDirection { MOVE_DIR_POSITIVE = 0 , MOVE_DIR_NEGATIVE = 1 }
- enum EzEnableMode { ENABLE_CLOSE = 0 , ENABLE_ONE = 1 , ENABLE_MULTI = 2 }
- enum EzTriggerMode { TRIGGER_MODE_FALLING = 0 , TRIGGER_MODE_RISING = 1 }
- enum EzTriggerSource { TRIGGER_SOURCE_CMD = 0 , TRIGGER_SOURCE_ENCODER = 1 }
- enum EzStopMode { STOP_MODE_CONTROLLED = 0 , STOP_MODE_IMMEDIATE = 1 }
- enum EzEngineState {
  ENGINE_STATE_UNINIT = 0 , ENGINE_STATE_INITED , ENGINE_STATE_STARTED , ENGINE_STATE_STOPPING
  ,
  ENGINE_STATE_STOPPED }
- enum EzEngineStopReason { ENGINE_STOP_REASON_NONE = 0 , ENGINE_STOP_REASON_USER_REQUEST
  , ENGINE_STOP_REASON_FIELDBUS_FATAL , ENGINE_STOP_REASON_UNKNOWN }
- enum EzBusFatalReason {
  BUS_FATAL_REASON_NONE = 0 , BUS_FATAL_REASON_MASTER_ERROR_INTERRUPT ,
  BUS_FATAL_REASON_MASTER_DISCONNECTED , BUS_FATAL_REASON_SLAVE_OFFLINE ,
  BUS_FATAL_REASON_SLAVE_COUNT_CHANGED , BUS_FATAL_REASON_DATASET_CONSECUTIVE_FAILURE
  , BUS_FATAL_REASON_HEARTBEAT_TIMEOUT , BUS_FATAL_REASON_UNKNOWN }
- enum EzAxisState {
  AXIS_STATE_ERROR_STOP = 0 , AXIS_STATE_DISABLED , AXIS_STATE_STANDSTILL ,
  AXIS_STATE_HOMING ,
  AXIS_STATE_STOPPING , AXIS_STATE_DISCRETE_MOTION , AXIS_STATE_SYNCHRONIZED_MOTION
  , AXIS_STATE_CONTINUOUS_MOTION }
- enum EzAxisLimitStopReason {
  EZ_AXIS_LIMIT_STOP_REASON_NONE = 0 , EZ_AXIS_LIMIT_STOP_REASON_SOFT_POSITIVE ,
  EZ_AXIS_LIMIT_STOP_REASON_SOFT_NEGATIVE , EZ_AXIS_LIMIT_STOP_REASON_HARD_POSITIVE
  ,
  EZ_AXIS_LIMIT_STOP_REASON_HARD_NEGATIVE }
- enum EzHomeState { AXIS_HOME_UNHOMED = 0 , AXIS_HOME_HOMING = 1 ,
  AXIS_HOME_HOMED = 2 }
- enum EzArcDir { ARC_CW = 0 , ARC_CCW = 1 }
- enum EzAxisGroupState {
  AXIS_GROUP_STATE_DISABLED = 1 , AXIS_GROUP_STATE_STANDSTILL = 2 , AXIS_GROUP_STATE_MOVING
  = 4 , AXIS_GROUP_STATE_STOPPING = 5 ,
  AXIS_GROUP_STATE_ERROR_STOP = 6 }
- enum EzAxisGroupSegmentProfileMask {
  EZ_AXIS_GROUP_SEGMENT_PROFILE_NONE = 0 , EZ_AXIS_GROUP_SEGMENT_PROFILE_VS
  = 1 << 0 , EZ_AXIS_GROUP_SEGMENT_PROFILE_A = 1 << 1 , EZ_AXIS_GROUP_SEGMENT_PROFILE_D
  = 1 << 2 ,
  EZ_AXIS_GROUP_SEGMENT_PROFILE_VE = 1 << 3 , EZ_AXIS_GROUP_SEGMENT_PROFILE_J
  = 1 << 4 , EZ_AXIS_GROUP_SEGMENT_PROFILE_JD = 1 << 5 }

段级参数覆盖掩码

函数

- SOFT_MOTION_LIB_API int __stdcall EzInit (const char *ip="127.0.0.1")

  初始化, 所有调用要在初始化之后调用
- SOFT_MOTION_LIB_API int __stdcall EzUnInit ()

  释放资源
- SOFT_MOTION_LIB_API int __stdcall EzScan (EzBusType bus_type, unsigned char master_index, EzSlaveInfo **slaves, int *slave_count)

  扫描从站设备信息
- SOFT_MOTION_LIB_API int __stdcall EzGetMasterCycleTime (EzBusType bus_type, unsigned char master_index, int *cycle_time)

  获取总线循环周期
- SOFT_MOTION_LIB_API int __stdcall EzWriteEcatPDOChar (unsigned char master_index, unsigned char slave_index, unsigned short address, unsigned short var_index, char value)

  向 *EtherCAT* 主站 *PDO* 地址空间写入字符值
- SOFT_MOTION_LIB_API int __stdcall EzReadEcatPDOChar (unsigned char master_index, unsigned char slave_index, unsigned short address, unsigned short var_index, char *value)

  读取 *EtherCAT* 主站 *PDO* 地址空间中的字符值
- SOFT_MOTION_LIB_API int __stdcall EzWriteEcatPDOBool (unsigned char master_index, unsigned char slave_index, unsigned short address, unsigned short var_index, bool value)

  向 *EtherCAT* 主站 *PDO* 地址空间写入布尔值
- SOFT_MOTION_LIB_API int __stdcall EzReadEcatPDOBool (unsigned char master_index, unsigned char slave_index, unsigned short address, unsigned short var_index, bool *value)

  读取 *EtherCAT* 主站 *PDO* 地址空间中的布尔值
- SOFT_MOTION_LIB_API int __stdcall EzWriteEcatPDOShort (unsigned char master_index, unsigned char slave_index, unsigned short address, unsigned short var_index, short value)

  向 *EtherCAT* 主站 *PDO* 地址空间写入 *short* 值
- SOFT_MOTION_LIB_API int __stdcall EzReadEcatPDOShort (unsigned char master_index, unsigned char slave_index, unsigned short address, unsigned short var_index, short *value)

  读取 *EtherCAT* 主站 *PDO* 地址空间中的 *short* 值
- SOFT_MOTION_LIB_API int __stdcall EzWriteEcatPDOInt (unsigned char master_index, unsigned char slave_index, unsigned short address, unsigned short var_index, int value)

  向 *EtherCAT* 主站 *PDO* 地址空间写入整数值
- SOFT_MOTION_LIB_API int __stdcall EzReadEcatPDOInt (unsigned char master_index, unsigned char slave_index, unsigned short address, unsigned short var_index, int *value)

  读取 *EtherCAT* 主站 *PDO* 地址空间中的整数值
- SOFT_MOTION_LIB_API int __stdcall EzWriteEcatPDODouble (unsigned char master_index, unsigned char slave_index, unsigned short address, unsigned short var_index, double value)

向 *EtherCAT* 主站 *PDO* 地址空间写入浮点值

- SOFT_MOTION_LIB_API int __stdcall EzReadEcatPDODouble (unsigned char master_index, unsigned char slave_index, unsigned short address, unsigned short var_index, double *value)

  读取 *EtherCAT PDO* 中的浮点值

- SOFT_MOTION_LIB_API int __stdcall EzWriteEcatSDOChar (unsigned char master_index, unsigned char slave_index, unsigned short address, unsigned short var_index, char value)

  向 *EtherCAT* 主站 *SDO* 地址空间写入字符值

- SOFT_MOTION_LIB_API int __stdcall EzReadEcatSDOChar (unsigned char master_index, unsigned char slave_index, unsigned short address, unsigned short var_index, char *value)

  读取 *EtherCAT* 主站 *SDO* 地址空间中的字符值

- SOFT_MOTION_LIB_API int __stdcall EzWriteEcatSDOBool (unsigned char master_index, unsigned char slave_index, unsigned short address, unsigned short var_index, bool value)

  向 *EtherCAT* 主站 *SDO* 地址空间写入布尔值

- SOFT_MOTION_LIB_API int __stdcall EzReadEcatSDOBool (unsigned char master_index, unsigned char slave_index, unsigned short address, unsigned short var_index, bool *value)

  读取 *EtherCAT* 主站 *SDO* 地址空间中的布尔值

- SOFT_MOTION_LIB_API int __stdcall EzWriteEcatSDOShort (unsigned char master_index, unsigned char slave_index, unsigned short address, unsigned short var_index, short value)

  向 *EtherCAT* 主站 *SDO* 地址空间写入 *short* 值

- SOFT_MOTION_LIB_API int __stdcall EzReadEcatSDOShort (unsigned char master_index, unsigned char slave_index, unsigned short address, unsigned short var_index, short *value)

  读取 *EtherCAT* 主站 *SDO* 地址空间中的 *short* 值

- SOFT_MOTION_LIB_API int __stdcall EzWriteEcatSDOInt (unsigned char master_index, unsigned char slave_index, unsigned short address, unsigned short var_index, int value)

  向 *EtherCAT* 主站 *SDO* 地址空间写入整数值

- SOFT_MOTION_LIB_API int __stdcall EzReadEcatSDOInt (unsigned char master_index, unsigned char slave_index, unsigned short address, unsigned short var_index, int *value)

  读取 *EtherCAT* 主站 *SDO* 地址空间中的整数值

- SOFT_MOTION_LIB_API int __stdcall EzWriteEcatSDODouble (unsigned char master_index, unsigned char slave_index, unsigned short address, unsigned short var_index, double value)

  向 *EtherCAT* 主站 *SDO* 地址空间写入浮点值

- SOFT_MOTION_LIB_API int __stdcall EzReadEcatSDODouble (unsigned char master_index, unsigned char slave_index, unsigned short address, unsigned short var_index, double *value)

  读取 *EtherCAT SDO* 中的浮点值

- SOFT_MOTION_LIB_API int __stdcall EzSetControllerWorkMode (bool simulation)

  设置控制器工作模式, 只有在*EzStart* 前生效

- SOFT_MOTION_LIB_API int __stdcall EzGetControllerWorkMode (bool *simulation)

  获取控制器工作模式

- SOFT_MOTION_LIB_API int __stdcall EzGetEngineStatus (EzEngineStatus *status)

获取引擎当前状态

- SOFT_MOTION_LIB_API int __stdcall EzGetDebugMode (bool *debug_mode)

  获取引擎当前运行模式

- SOFT_MOTION_LIB_API int __stdcall EzStart (const char *path)

  启动控制器, 成功后总线已经在*OP*

- SOFT_MOTION_LIB_API int __stdcall EzGetConfigFile (const char *path)

  获取当前启动配置文件并覆盖写入到指定路径

- SOFT_MOTION_LIB_API int __stdcall EzGetMasterInfo (EzMasterInfo **masters, int *master_count)

  获取总线数据

- SOFT_MOTION_LIB_API int __stdcall EzGetAxesInfo (EzAxisInfo **axes_info, int *axes_count)

  获取轴信息

- SOFT_MOTION_LIB_API int __stdcall EzGetTerminalInfo (EzTerminalInfo **terminal_info, int *terminal_count)

  获取终端信息

- SOFT_MOTION_LIB_API int __stdcall EzStop ()

  停止控制器

- SOFT_MOTION_LIB_API int __stdcall EzPower (int id, bool power_on)

  轴使能

- SOFT_MOTION_LIB_API int __stdcall EzMove (int id, double target_position, EzMoveType move_type)

  轴运动

- SOFT_MOTION_LIB_API int __stdcall EzSetProfile (int id, double vb, double vs, double a, double d, double ve)

  设置轴参数

- SOFT_MOTION_LIB_API int __stdcall EzGetProfile (int id, double *vb, double *vs, double *a, double *d, double *ve)

  获取轴参数

- SOFT_MOTION_LIB_API int __stdcall EzSetSProfile (int id, double J, double Jd)

  设置轴*S* 型曲线参数

- SOFT_MOTION_LIB_API int __stdcall EzGetSProfile (int id, double *J, double *Jd)

  获取轴*S* 型曲线参数

- SOFT_MOTION_LIB_API int __stdcall EzGetAxisCycleData (int id, EzAxisCycleData *cycle_data)

  获取轴状态

- SOFT_MOTION_LIB_API int __stdcall EzGetAxesCycleData (EzAxisCycleData **cycle_data, int *axis_count)

  获取所有轴周期数据

- SOFT_MOTION_LIB_API int __stdcall EzSetSoftLimitConfig (int id, const EzSoftLimitConfig *config)

  设置软限位运行时配置

- SOFT_MOTION_LIB_API int __stdcall EzGetSoftLimitConfig (int id, EzSoftLimitConfig *config)

  获取软限位运行时配置
- SOFT_MOTION_LIB_API int __stdcall EzSetHardLimitEnable (int id, const EzDirectionalEnable *enable)

  设置硬限位运行时使能状态
- SOFT_MOTION_LIB_API int __stdcall EzGetHardLimitEnable (int id, EzDirectionalEnable *enable)

  获取硬限位运行时使能状态
- SOFT_MOTION_LIB_API int __stdcall EzSetGearProfile (int slave_axis, const EzGearProfile *profile)

  设置从轴的 *gear* 配置
- SOFT_MOTION_LIB_API int __stdcall EzGetGearProfile (int slave_axis, EzGearProfile *profile)

  获取从轴的 *gear* 配置
- SOFT_MOTION_LIB_API int __stdcall EzGearIn (int slave_axis)

  让从轴进入 *gear* 耦合
- SOFT_MOTION_LIB_API int __stdcall EzGearOut (int slave_axis)

  让从轴退出 *gear* 耦合
- SOFT_MOTION_LIB_API int __stdcall EzGetGearState (int slave_axis, bool *in_gear)

  获取从轴当前是否处于 *gear* 耦合状态
- SOFT_MOTION_LIB_API int __stdcall EzJog (int id, EzMoveDirection direction)

  轴 *jog*
- SOFT_MOTION_LIB_API int __stdcall EzUpdateTargetPosition (int id, double position)

  轴在线更新目标位置
- SOFT_MOTION_LIB_API int __stdcall EzSetPosition (int id, double position)

  强制设置轴位置
- SOFT_MOTION_LIB_API int __stdcall EzChangeSpeed (int id, double velocity, double acceleration)

  轴在线更新速度
- SOFT_MOTION_LIB_API int __stdcall EzMoveEx (int id, double position, double vb, double vs, double a, double d, double ve, double J, double Jd, EzMoveType move_type)

  轴运动, 带参数, 用于高速场景
- SOFT_MOTION_LIB_API int __stdcall EzSoftEndMove (int id, double mid_position, double position, double vb, double vs, double a, double d, double ve, double J, double Jd, EzMoveType move_type)

  轴软着陆移动
- SOFT_MOTION_LIB_API int __stdcall EzSoftStartMove (int id, double mid_position, double position, double vb, double vs1, double ve1, double delay_time, double vs2, double a, double d, double ve2, double J, double Jd, EzMoveType move_type)

  轴软启动移动

- SOFT_MOTION_LIB_API int __stdcall EzUpdateTargetPositionEx (int id, double mid_position, double target_position)

  在线修调目标位置并实现软着陆
- SOFT_MOTION_LIB_API int __stdcall EzGetAxisDI (int id, int index, bool *value)

  获取轴*DI*
- SOFT_MOTION_LIB_API int __stdcall EzAxisStop (int id, EzStopMode stop_mode)

  轴停止
- SOFT_MOTION_LIB_API int __stdcall EzEmergencyStop ()

  紧急停止, 所有的运动停止
- SOFT_MOTION_LIB_API int __stdcall EzSetHomeProfile (int id, int home_mode, double low_velocity, double high_velocity, double acceleration, double deceleration, double offset)

  设置轴*Home* 参数, 此回零为控制器控制回零, 回零开关接在远程*I/O* 上
- SOFT_MOTION_LIB_API int __stdcall EzGetHomeProfile (int id, int *home_mode, double *low_velocity, double *high_velocity, double *acceleration, double *deceleration, double *offset)

  获取轴*Home* 参数
- SOFT_MOTION_LIB_API int __stdcall EzHome (int id)

  轴回零
- SOFT_MOTION_LIB_API int __stdcall EzGetHomeState (int id, EzHomeState *home_state)

  获取轴当前回零状态
- SOFT_MOTION_LIB_API int __stdcall EzSetAxisHomeProfile (int id, int home_mode, double low_velocity, double high_velocity, double acceleration, double offset)

  设置轴控制器回零参数
- SOFT_MOTION_LIB_API int __stdcall EzGetAxisHomeProfile (int id, int *home_mode, double *low_velocity, double *high_velocity, double *acceleration, double *offset)

  获取轴控制器回零参数
- SOFT_MOTION_LIB_API int __stdcall EzAxisHome (int id)

  轴控制器回零
- SOFT_MOTION_LIB_API int __stdcall EzAxisReset (int id)

  轴复位
- SOFT_MOTION_LIB_API int __stdcall EzSetAxisGroupProfile (int id, double vs, double a, double d)

  设定轴组运动参数
- SOFT_MOTION_LIB_API int __stdcall EzSetAxisGroupSProfile (int id, double J, double Jd)

  获取轴组运动参数
- SOFT_MOTION_LIB_API int __stdcall EzGetAxisGroupProfile (int id, double *vs, double *a, double *d)

  获取轴组运动参数
- SOFT_MOTION_LIB_API int __stdcall EzGetAxisGroupSProfile (int id, double *J, double *Jd)

  获取轴组运动参数

- SOFT_MOTION_LIB_API int __stdcall EzAxisGroupOpen (int id, int axis_number, const unsigned short *axis_id_list)

  打开轴组
- SOFT_MOTION_LIB_API int __stdcall EzAxisGroupStart (int id)

  开始插补运动
- SOFT_MOTION_LIB_API int __stdcall EzAxisGroupStop (int id)

  停止轴组插补运动
- SOFT_MOTION_LIB_API int __stdcall EzAxisGroupPause (int id)

  暂停轴组插补运动
- SOFT_MOTION_LIB_API int __stdcall EzAxisGroupClose (int id)

  关闭轴组
- SOFT_MOTION_LIB_API int __stdcall EzGetAxisGroupCycleData (int id, EzAxisGroupCycleData *cycle_data)

  获取轴组周期数据
- SOFT_MOTION_LIB_API int __stdcall EzAxisGroupChangeFeedRate (int id, double feed_rate)

  修调连续插补倍率
- SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddLine (int id, int axis_number, const double *target_position, EzMoveType move_type)

  添加轴组直线运动
- SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddLineEx (int id, int axis_number, const double *target_position, EzMoveType move_type, const EzAxisGroupSegmentProfileEx *profile)

  添加带单段参数覆盖的轴组直线运动
- SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddArcCenter (int id, int axis_number, const double *target_position, const double *center_position, EzArcDir dir, EzMoveType move_type)

  添加轴组圆弧运动, 基于圆心和目标位置的圆弧
- SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddArcCenterEx (int id, int axis_number, const double *target_position, const double *center_position, EzArcDir dir, EzMoveType move_type, const EzAxisGroupSegmentProfileEx *profile)

  添加带单段参数覆盖的轴组圆弧运动, 基于圆心和目标位置的圆弧
- SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddArcRadius (int id, int axis_number, const double *target_position, double radius, EzArcDir dir, EzMoveType move_type)

  添加轴组圆弧运动, 基于半径和目标位置的圆弧
- SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddArcRadiusEx (int id, int axis_number, const double *target_position, double radius, EzArcDir dir, EzMoveType move_type, const EzAxisGroupSegmentProfileEx *profile)

  添加带单段参数覆盖的轴组圆弧运动, 基于半径和目标位置的圆弧
- SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddArc3Point (int id, int axis_number, const double *target_position, const double *mid_position, EzArcDir dir, EzMoveType move_type)

添加轴组圆弧运动, 基于 *3* 点位置的圆弧

- SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddArc3PointEx (int id, int axis_number, const double *target_position, const double *mid_position, EzArcDir dir, EzMoveType move_type, const EzAxisGroupSegmentProfileEx *profile)

  添加带单段参数覆盖的轴组圆弧运动, 基于 *3* 点位置的圆弧
- SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddDelay (int id, double delay_time_ms)

  添加轴组延时段
- SOFT_MOTION_LIB_API int __stdcall EzAxisGroupWaitInput (int id, unsigned short terminal_id, unsigned short di_index, bool on_off, double timeout_s)

  添加轴组等待输入段
- SOFT_MOTION_LIB_API int __stdcall EzAxisGroupWriteOutBit (int id, unsigned short terminal_id, unsigned short do_index, bool on_off, double reverse_time_s)

  添加轴组不中断写出段
- SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddEnd (int id)

  添加轴组前瞻结束标志
- SOFT_MOTION_LIB_API int __stdcall EzAxisGroupMoveLine (int id, int axis_number, const unsigned short *axis_id_list, const double *target_position, EzMoveType move_type)

  轴组直线运动
- SOFT_MOTION_LIB_API int __stdcall EzAxisGroupMoveArcCenter (int id, int axis_number, const unsigned short *axis_id_list, const double *target_position, const double *center_position, EzArcDir dir, EzMoveType move_type)

  轴组圆弧运动, 基于圆心和目标位置的圆弧
- SOFT_MOTION_LIB_API int __stdcall EzAxisGroupMoveArcRadius (int id, int axis_number, const unsigned short *axis_id_list, const double *target_position, double radius, EzArcDir dir, EzMoveType move_type)

  轴组圆弧运动, 基于半径和目标位置的圆弧
- SOFT_MOTION_LIB_API int __stdcall EzAxisGroupMoveArc3Point (int id, int axis_number, const unsigned short *axis_id_list, const double *target_position, const double *mid_position, EzArcDir dir, EzMoveType move_type)

  轴组圆弧运动, 基于 *3* 点位置的圆弧
- SOFT_MOTION_LIB_API int __stdcall EzUpdateCyclicPosition (int id, const char *path)

  导入点表移动, 根据传入文件中时间和位置的序列进行移动
- SOFT_MOTION_LIB_API int __stdcall EzGetDI (int id, unsigned char index, bool *value)

  获取数字量输入
- SOFT_MOTION_LIB_API int __stdcall EzSetDO (int id, unsigned char index, bool value)

  设定数字量输出
- SOFT_MOTION_LIB_API int __stdcall EzGetDO (int id, unsigned char index, bool *value)

  获取数字量输出
- SOFT_MOTION_LIB_API int __stdcall EzSetAO (int id, unsigned char index, short value)

  设定模拟量输出
- SOFT_MOTION_LIB_API int __stdcall EzGetAO (int id, unsigned char index, short *value)

获取模拟量输出

- SOFT_MOTION_LIB_API int __stdcall EzGetAI (int id, unsigned char index, short *value)

  获取模拟量输入

- SOFT_MOTION_LIB_API int __stdcall EzGetPulseCount (int id, unsigned char index, int *value)

  获取脉冲计数

- SOFT_MOTION_LIB_API int __stdcall EzGetEncoder (int id, unsigned char index, int *value)

  获取编码器

- SOFT_MOTION_LIB_API int __stdcall EzSetPulseFrequency (int id, unsigned char index, int value)

  设置脉冲输出

- SOFT_MOTION_LIB_API int __stdcall EzGetPulseFrequency (int id, unsigned char index, int *value)

  获取脉冲输出

- SOFT_MOTION_LIB_API int __stdcall EzSetDI (int id, unsigned char index, bool value)

  设置数字量输入

- SOFT_MOTION_LIB_API int __stdcall EzSetAI (int id, unsigned char index, short value)

  设定模拟量输入

- SOFT_MOTION_LIB_API int __stdcall EzSetPulseCount (int id, unsigned char index, int value)

  设置脉冲计数

- SOFT_MOTION_LIB_API int __stdcall EzSetEncoder (int id, unsigned char index, int value)

  设定编码器

- SOFT_MOTION_LIB_API const char *__stdcall EzGetErrorText (int error_code)

  将错误代码转换为英文文本

- SOFT_MOTION_LIB_API int __stdcall EzClose ()

  关闭引擎，正常无需调用

4.1.1 详细描述

：None

作者

：MJ

注意

：None

日期

：2025/11/28

4.1.2 宏定义说明

4.1.2.1 EZ_AXIS_GROUP_MAX_AXIS

```
#define EZ_AXIS_GROUP_MAX_AXIS 9
```

4.1.2.2 SOFT_MOTION_LIB_API

```
#define SOFT_MOTION_LIB_API
```

4.1.3 枚举类型说明

4.1.3.1 EzArcDir

```
enum EzArcDir
```

枚举值

ARC_CW	顺时针
ARC_CCW	逆时针

4.1.3.2 EzAxisGroupSegmentProfileMask

```
enum EzAxisGroupSegmentProfileMask
```

段级参数覆盖掩码

枚举值

EZ_AXIS_GROUP_SEGMENT_PROFILE_NONE	不覆盖任何字段，整段继承全局参数
EZ_AXIS_GROUP_SEGMENT_PROFILE_VS	覆盖段级速度
EZ_AXIS_GROUP_SEGMENT_PROFILE_A	覆盖段级加速度
EZ_AXIS_GROUP_SEGMENT_PROFILE_D	覆盖段级减速度（当前版本仅开放字段）
EZ_AXIS_GROUP_SEGMENT_PROFILE_VE	覆盖段级末速度（当前版本仅支持 ve == 0）

EZ_AXIS_GROUP_SEGMENT_PROFILE_J	覆盖段级加加速度（仅 S 型全局曲线有效）
EZ_AXIS_GROUP_SEGMENT_PROFILE_JD	覆盖段级减加速度（当前版本仅开放字段）

4.1.3.3 EzAxisGroupState

```
enum EzAxisGroupState
```

枚举值

AXIS_GROUP_STATE_DISABLED	禁用
AXIS_GROUP_STATE_STANDSTILL	待机
AXIS_GROUP_STATE_MOVING	运行中
AXIS_GROUP_STATE_STOPPING	停止中
AXIS_GROUP_STATE_ERROR_STOP	错误停止

4.1.3.4 EzAxisLimitStopReason

```
enum EzAxisLimitStopReason
```

枚举值

EZ_AXIS_LIMIT_STOP_REASON_NONE	无限位停车原因
EZ_AXIS_LIMIT_STOP_REASON_SOFT_POSITIVE	正软限位
EZ_AXIS_LIMIT_STOP_REASON_SOFT_NEGATIVE	负软限位
EZ_AXIS_LIMIT_STOP_REASON_HARD_POSITIVE	正硬限位
EZ_AXIS_LIMIT_STOP_REASON_HARD_NEGATIVE	负硬限位

4.1.3.5 EzAxisState

```
enum EzAxisState
```

枚举值

AXIS_STATE_ERROR_STOP	错误停止
AXIS_STATE_DISABLED	去使能
AXIS_STATE_STANDSTILL	使能
AXIS_STATE_HOMING	回零中
AXIS_STATE_STOPPING	停止中
AXIS_STATE_DISCRETE_MOTION	离散运动
AXIS_STATE_SYNCHRONIZED_MOTION	同步运动
AXIS_STATE_CONTINUOUS_MOTION	连续运动

4.1.3.6 EzBusFatalReason

`enum EzBusFatalReason`

枚举值

BUS_FATAL_REASON_NONE	无
BUS_FATAL_REASON_MASTER_ERROR_INTERRUPT	总线故障
BUS_FATAL_REASON_MASTER_DISCONNECTED	总线断开
BUS_FATAL_REASON_SLAVE_OFFLINE	从站离线
BUS_FATAL_REASON_SLAVE_COUNT_CHANGED	从站数量改变
BUS_FATAL_REASON_DATASET_CONSECUTIVE_FAILURE	数据集连续性失败
BUS_FATAL_REASON_HEARTBEAT_TIMEOUT	总线心跳超时
BUS_FATAL_REASON_UNKNOWN	未知总线故障

4.1.3.7 EzBusType

`enum EzBusType`

枚举值

ETHERCAT	ethercat
ETHERMAC	ethermac

4.1.3.8 EzEnableMode

enum EzEnableMode

枚举值

ENABLE_CLOSE	关闭
ENABLE_ONE	单次触发
ENABLE_MULTI	多次触发

4.1.3.9 EzEngineState

enum EzEngineState

枚举值

ENGINE_STATE_UNINIT	未初始化
ENGINE_STATE_INITED	已初始化
ENGINE_STATE_STARTED	已启动
ENGINE_STATE_STOPPING	停止中
ENGINE_STATE_STOPPED	已停止

4.1.3.10 EzEngineStopReason

enum EzEngineStopReason

枚举值

ENGINE_STOP_REASON_NONE	停止原因无
ENGINE_STOP_REASON_USER_REQUEST	用户请求停止
ENGINE_STOP_REASON_FIELDBUS_FATAL	总线故障停止
ENGINE_STOP_REASON_UNKNOWN	未知停止

4.1.3.11 EzHomeState

enum EzHomeState

枚举值

AXIS_HOME_UNHOMED	未使能
AXIS_HOME_HOMING	回零中
AXIS_HOME_HOMED	回零完成

4.1.3.12 EzMoveDirection

enum EzMoveDirection

枚举值

MOVE_DIR_POSITIVE	正向移动
MOVE_DIR_NEGATIVE	负向移动

4.1.3.13 EzMoveType

enum EzMoveType

枚举值

MOVE_TYPE_ABSOLUTE	绝对移动
MOVE_TYPE_RELATIVE	相对移动

4.1.3.14 EzStopMode

enum EzStopMode

枚举值

STOP_MODE_CONTROLLED	减速停止
STOP_MODE_IMMEDIATE	直接停止

4.1.3.15 EzTriggerMode

```
enum EzTriggerMode
```

枚举值

TRIGGER_MODE_FALLING	下降沿触发
TRIGGER_MODE_RISING	上升沿触发

4.1.3.16 EzTriggerSource

```
enum EzTriggerSource
```

枚举值

TRIGGER_SOURCE_CMD	命令位置触发
TRIGGER_SOURCE_ENCODER	编码器位置触发

4.1.4 函数说明

4.1.4.1 EzAxisGroupAddArc3Point()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddArc3Point (
                int id,
                int axis_number,
                const double * target_position,
                const double * mid_position,
                EzArcDir dir,
                EzMoveType move_type)
```

添加轴组圆弧运动, 基于 3 点位置的圆弧

参数

id	轴组 id
axis_number	轴数量
target_position	目标位置
mid_position	中点位置

dir	圆弧方向
move_type	运动类型

返回

错误码

4.1.4.2 EzAxisGroupAddArc3PointEx()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddArc3PointEx (
                int id,
                int axis_number,
                const double * target_position,
                const double * mid_position,
                EzArcDir dir,
                EzMoveType move_type,
                const EzAxisGroupSegmentProfileEx * profile)
```

添加带单段参数覆盖的轴组圆弧运动, 基于 3 点位置的圆弧

参数

id	轴组 id
axis_number	轴数量
target_position	目标位置
mid_position	中间位置
dir	圆弧方向
move_type	运动类型
profile	段级参数覆盖；传空指针或 `mask == 0` 时整段继承全局参数

返回

错误码

4.1.4.3 EzAxisGroupAddArcCenter()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddArcCenter (
                int id,
                int axis_number,
                const double * target_position,
                const double * center_position,
                EzArcDir dir,
                EzMoveType move_type)
```

添加轴组圆弧运动, 基于圆心和目标位置的圆弧

参数

id	轴组 id
axis_number	轴数量
target_position	目标位置
center_position	中点位置
dir	圆弧方向
move_type	运动类型

返回

错误码

4.1.4.4 EzAxisGroupAddArcCenterEx()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddArcCenterEx (
                int id,
                int axis_number,
                const double * target_position,
                const double * center_position,
                EzArcDir dir,
                EzMoveType move_type,
                const EzAxisGroupSegmentProfileEx * profile)
```

添加带单段参数覆盖的轴组圆弧运动, 基于圆心和目标位置的圆弧

参数

id	轴组 id
axis_number	轴数量
target_position	目标位置
center_position	中点位置
dir	圆弧方向
move_type	运动类型
profile	段级参数覆盖；传空指针或 `mask == 0` 时整段继承全局参数

返回

错误码

4.1.4.5 EzAxisGroupAddArcRadius()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddArcRadius (
                int id,
                int axis_number,
                const double * target_position,
                double radius,
                EzArcDir dir,
                EzMoveType move_type)
```

添加轴组圆弧运动, 基于半径和目标位置的圆弧

参数

id	轴组 id
axis_number	轴数量
target_position	目标位置
radius	半径
dir	圆弧方向
move_type	运动类型

返回

错误码

4.1.4.6 EzAxisGroupAddArcRadiusEx()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddArcRadiusEx (
                int id,
                int axis_number,
                const double * target_position,
                double radius,
                EzArcDir dir,
                EzMoveType move_type,
                const EzAxisGroupSegmentProfileEx * profile)
```

添加带单段参数覆盖的轴组圆弧运动, 基于半径和目标位置的圆弧

参数

id	轴组 id
axis_number	轴数量
target_position	目标位置
radius	半径
dir	圆弧方向
move_type	运动类型
profile	段级参数覆盖；传空指针或 `mask == 0` 时整段继承全局参数

返回

错误码

4.1.4.7 EzAxisGroupAddDelay()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddDelay (
                int id,
                double delay_time_ms)
```

添加轴组延时段

参数

id	轴组 id

delay_time_ms	延时时间，单位毫秒

返回

错误码

4.1.4.8 EzAxisGroupAddEnd()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddEnd (
                int id)
```

添加轴组前瞻结束标志

参数

id	轴组 id

返回

错误码

4.1.4.9 EzAxisGroupAddLine()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddLine (
                int id,
                int axis_number,
                const double * target_position,
                EzMoveType move_type)
```

添加轴组直线运动

参数

id	轴组 id
axis_number	轴数量
target_position	目标位置

move_type	运动类型

返回

错误码

4.1.4.10 EzAxisGroupAddLineEx()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddLineEx (
                int id,
                int axis_number,
                const double * target_position,
                EzMoveType move_type,
                const EzAxisGroupSegmentProfileEx * profile)
```

添加带单段参数覆盖的轴组直线运动

参数

id	轴组 id
axis_number	轴数量
target_position	目标位置
move_type	运动类型
profile	段级参数覆盖；传空指针或 `mask == 0` 时整段继承全局参数

返回

错误码

4.1.4.11 EzAxisGroupChangeFeedRate()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisGroupChangeFeedRate (
                int id,
                double feed_rate)
```

修调连续插补倍率

参数

id	轴组 id
feed_rate	倍率

返回

错误码

4.1.4.12 EzAxisGroupClose()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisGroupClose (
            int id)
```

关闭轴组

参数

id	轴组 id

返回

错误码

4.1.4.13 EzAxisGroupMoveArc3Point()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisGroupMoveArc3Point (
            int id,
            int axis_number,
            const unsigned short * axis_id_list,
            const double * target_position,
            const double * mid_position,
            EzArcDir dir,
            EzMoveType move_type)
```

轴组圆弧运动, 基于 3 点位置的圆弧

参数

id	轴组 id
axis_number	轴数量
axis_id_list	轴 id 列表
target_position	目标位置
mid_position	中点位置
dir	圆弧方向
move_type	运动类型

返回

错误码

4.1.4.14 EzAxisGroupMoveArcCenter()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisGroupMoveArcCenter (
                int id,
                int axis_number,
                const unsigned short * axis_id_list,
                const double * target_position,
                const double * center_position,
                EzArcDir dir,
                EzMoveType move_type)
```

轴组圆弧运动, 基于圆心和目标位置的圆弧

参数

id	轴组 id
axis_number	轴数量
axis_id_list	轴 id 列表
target_position	目标位置
center_position	中点位置
dir	圆弧方向
move_type	运动类型

返回

错误码

4.1.4.15 EzAxisGroupMoveArcRadius()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisGroupMoveArcRadius (
                int id,
                int axis_number,
                const unsigned short * axis_id_list,
                const double * target_position,
                double radius,
                EzArcDir dir,
                EzMoveType move_type)
```

轴组圆弧运动, 基于半径和目标位置的圆弧

参数

id	轴组 id
axis_number	轴数量
axis_id_list	轴 id 列表
target_position	目标位置
radius	半径
dir	圆弧方向
move_type	运动类型

返回

错误码

4.1.4.16 EzAxisGroupMoveLine()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisGroupMoveLine (
                int id,
                int axis_number,
                const unsigned short * axis_id_list,
```

```
                const double * target_position,
                EzMoveType move_type)
```

轴组直线运动

参数

id	轴组 id
axis_number	轴数量
axis_id_list	轴 id 列表
target_position	目标位置
move_type	运动类型

返回

错误码

4.1.4.17 EzAxisGroupOpen()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisGroupOpen (
                int id,
                int axis_number,
                const unsigned short * axis_id_list)
```

打开轴组

参数

id	轴组 id
axis_number	轴数量
axis_id_list	轴 id 列表

返回

错误码

4.1.4.18 EzAxisGroupPause()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisGroupPause (
            int id)
```

暂停轴组插补运动

参数

id	轴组 id

返回

错误码

4.1.4.19 EzAxisGroupStart()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisGroupStart (
            int id)
```

开始插补运动

参数

id	轴组 id

返回

错误码

4.1.4.20 EzAxisGroupStop()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisGroupStop (
            int id)
```

停止轴组插补运动

参数

id	轴组 id

返回

错误码

4.1.4.21 EzAxisGroupWaitInput()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisGroupWaitInput (
                int id,
                unsigned short terminal_id,
                unsigned short di_index,
                bool on_off,
                double timeout_s)
```

添加轴组等待输入段

参数

id	轴组 id
terminal_id	终端 id
di_index	DI 索引
on_off	目标输入电平
timeout_s	超时时间，单位秒；0 表示无限等待

返回

错误码

4.1.4.22 EzAxisGroupWriteOutBit()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisGroupWriteOutBit (
                int id,
                unsigned short terminal_id,
```

```
                unsigned short do_index,
                bool on_off,
                double reverse_time_s)
```

添加轴组不中断写出段

参数

id	轴组 id
terminal_id	终端 id
do_index	DO 索引
on_off	输出电平
reverse_time_s	反转时间，单位秒；0 表示不恢复

返回

错误码

4.1.4.23 EzAxisHome()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisHome (
                int id)
```

轴控制器回零

参数

id	轴 id

返回

错误码

4.1.4.24 EzAxisReset()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisReset (
                int id)
```

轴复位

参数

id	轴 id

返回

错误码

4.1.4.25 EzAxisStop()

```
SOFT_MOTION_LIB_API int __stdcall EzAxisStop (
                int id,
                EzStopMode stop_mode)
```

轴停止

参数

id	轴 id
stop_mode	停止模式

返回

错误码

4.1.4.26 EzChangeSpeed()

```
SOFT_MOTION_LIB_API int __stdcall EzChangeSpeed (
                int id,
                double velocity,
                double acceleration)
```

轴在线更新速度

参数

id	轴 id
velocity	速度
acceleration	加速度

返回

错误码

4.1.4.27 EzClose()

```
SOFT_MOTION_LIB_API int __stdcall EzClose ()
```

关闭引擎，正常无需调用

返回

错误码

4.1.4.28 EzEmergencyStop()

```
SOFT_MOTION_LIB_API int __stdcall EzEmergencyStop ()
```

紧急停止, 所有的运动停止

返回

错误码

4.1.4.29 EzGearIn()

```
SOFT_MOTION_LIB_API int __stdcall EzGearIn (
            int slave_axis)
```

让从轴进入 gear 耦合

参数

slave_axis	从轴号

返回

错误码

4.1.4.30 EzGearOut()

```
SOFT_MOTION_LIB_API int __stdcall EzGearOut (
            int slave_axis)
```

让从轴退出 gear 耦合

参数

slave_axis	从轴号

返回

错误码

4.1.4.31 EzGetAI()

```
SOFT_MOTION_LIB_API int __stdcall EzGetAI (
            int id,
            unsigned char index,
            short * value)
```

获取模拟量输入

参数

id	终端 id
index	AI 索引
value	AI 值

返回

错误码

4.1.4.32 EzGetAO()

```
SOFT_MOTION_LIB_API int __stdcall EzGetAO (
                int id,
                unsigned char index,
                short * value)
```

获取模拟量输出

参数

id	终端 id
index	AO 索引
value	AO 值

返回

错误码

4.1.4.33 EzGetAxesCycleData()

```
SOFT_MOTION_LIB_API int __stdcall EzGetAxesCycleData (
                EzAxisCycleData ** cycle_data,
                int * axis_count)
```

获取所有轴周期数据

参数

cycle_data	所有轴周期数据数组，按轴 id 顺序返回
axis_count	轴数量

返回

错误码

4.1.4.34 EzGetAxesInfo()

```
SOFT_MOTION_LIB_API int __stdcall EzGetAxesInfo (
                EzAxisInfo ** axes_info,
                int * axes_count)
```

获取轴信息

参数

axes_info	轴信息
axes_count	轴数量

返回

错误码

4.1.4.35 EzGetAxisCycleData()

```
SOFT_MOTION_LIB_API int __stdcall EzGetAxisCycleData (
                int id,
                EzAxisCycleData * cycle_data)
```

获取轴状态

参数

id	轴 id

cycle_data	轴状态

返回

错误码

4.1.4.36 EzGetAxisDI()

```
SOFT_MOTION_LIB_API int __stdcall EzGetAxisDI (
            int id,
            int index,
            bool * value)
```

获取轴DI

参数

id	轴 id
index	输入 index
value	DI 状态

返回

错误码

4.1.4.37 EzGetAxisGroupCycleData()

```
SOFT_MOTION_LIB_API int __stdcall EzGetAxisGroupCycleData (
            int id,
            EzAxisGroupCycleData * cycle_data)
```

获取轴组周期数据

参数

id	轴组 id

cycle_data	周期数据

返回

错误码

4.1.4.38 EzGetAxisGroupProfile()

```
SOFT_MOTION_LIB_API int __stdcall EzGetAxisGroupProfile (
                int id,
                double * vs,
                double * a,
                double * d)
```

获取轴组运动参数

参数

id	轴组 id
vs	最大速度
a	加速度
d	减速度

返回

错误码

4.1.4.39 EzGetAxisGroupSProfile()

```
SOFT_MOTION_LIB_API int __stdcall EzGetAxisGroupSProfile (
                int id,
                double * J,
                double * Jd)
```

获取轴组运动参数

参数

id	轴组 id
J	加加速度
Jd	减加速度

返回

错误码

4.1.4.40 EzGetAxisHomeProfile()

```
SOFT_MOTION_LIB_API int __stdcall EzGetAxisHomeProfile (
                int id,
                int * home_mode,
                double * low_velocity,
                double * high_velocity,
                double * acceleration,
                double * offset)
```

获取轴控制器回零参数

参数

id	轴 id
home_mode	回零模式
low_velocity	回零低速速度
high_velocity	回零高速速度
acceleration	回零加速度
offset	回零偏置

返回

错误码

4.1.4.41 EzGetConfigFile()

```
SOFT_MOTION_LIB_API int __stdcall EzGetConfigFile (
                const char * path)
```

获取当前启动配置文件并覆盖写入到指定路径

参数

path	目标文件路径

返回

错误码

4.1.4.42 EzGetControllerWorkMode()

```
SOFT_MOTION_LIB_API int __stdcall EzGetControllerWorkMode (
                bool * simulation)
```

获取控制器工作模式

参数

simulation	模拟模式

返回

错误码

4.1.4.43 EzGetDebugMode()

```
SOFT_MOTION_LIB_API int __stdcall EzGetDebugMode (
                bool * debug_mode)
```

获取引擎当前运行模式

参数

debug_mode	true 表示调试模式

返回

错误码

4.1.4.44 EzGetDI()

```
SOFT_MOTION_LIB_API int __stdcall EzGetDI (
                int id,
                unsigned char index,
                bool * value)
```

获取数字量输入

参数

id	终端 id
index	DI 索引
value	DI 值

返回

错误码

4.1.4.45 EzGetDO()

```
SOFT_MOTION_LIB_API int __stdcall EzGetDO (
                int id,
                unsigned char index,
                bool * value)
```

获取数字量输出

参数

id	终端 id
index	DO 索引
value	DO 值

返回

错误码

4.1.4.46 EzGetEncoder()

```
SOFT_MOTION_LIB_API int __stdcall EzGetEncoder (
                int id,
                unsigned char index,
                int * value)
```

获取编码器

参数

id	终端 id
index	encoder 索引
value	编码器数值

返回

错误码

4.1.4.47 EzGetEngineStatus()

```
SOFT_MOTION_LIB_API int __stdcall EzGetEngineStatus (
                EzEngineStatus * status)
```

获取引擎当前状态

参数

status	当前引擎状态

返回

错误码

4.1.4.48 EzGetErrorText()

```
SOFT_MOTION_LIB_API const char *__stdcall EzGetErrorText (
            int error_code)
```

将错误代码转换为英文文本

参数

error_code	API 返回的错误代码

返回

英文错误文本; 返回的指针有效直到同一线程的下一次调用

4.1.4.49 EzGetGearProfile()

```
SOFT_MOTION_LIB_API int __stdcall EzGetGearProfile (
            int slave_axis,
            EzGearProfile * profile)
```

获取从轴的 gear 配置

参数

slave_axis	从轴号
profile	gear 配置输出

返回

错误码

4.1.4.50 EzGetGearState()

```
SOFT_MOTION_LIB_API int __stdcall EzGetGearState (
                int slave_axis,
                bool * in_gear)
```

获取从轴当前是否处于 gear 耦合状态

参数

slave_axis	从轴号
in_gear	是否处于 gear 状态

返回

错误码

4.1.4.51 EzGetHardLimitEnable()

```
SOFT_MOTION_LIB_API int __stdcall EzGetHardLimitEnable (
                int id,
                EzDirectionalEnable * enable)
```

获取硬限位运行时使能状态

参数

id	轴 id
enable	硬限位使能状态

返回

错误码

4.1.4.52 EzGetHomeProfile()

```
SOFT_MOTION_LIB_API int __stdcall EzGetHomeProfile (
                int id,
                int * home_mode,
                double * low_velocity,
                double * high_velocity,
                double * acceleration,
                double * deceleration,
                double * offset)
```

获取轴Home 参数

参数

id	轴 id
home_mode	轴Home 模式
low_velocity	回零低速速度
high_velocity	回零高速速度
acceleration	回零加速度
deceleration	回零减速度
offset	回零偏移量

返回

错误码

4.1.4.53 EzGetHomeState()

```
SOFT_MOTION_LIB_API int __stdcall EzGetHomeState (
                int id,
                EzHomeState * home_state)
```

获取轴当前回零状态

参数

id	轴 id

home_state	回零状态

返回

错误码

4.1.4.54 EzGetMasterCycleTime()

```
SOFT_MOTION_LIB_API int __stdcall EzGetMasterCycleTime (
                EzBusType bus_type,
                unsigned char master_index,
                int * cycle_time)
```

获取总线循环周期

参数

bus_type	总线类型
master_index	主站号
cycle_time	循环周期

返回

错误码

4.1.4.55 EzGetMasterInfo()

```
SOFT_MOTION_LIB_API int __stdcall EzGetMasterInfo (
                EzMasterInfo ** masters,
                int * master_count)
```

获取总线数据

参数

masters	总线信息

master_count	总线数量

返回

错误码

4.1.4.56 EzGetProfile()

```
SOFT_MOTION_LIB_API int __stdcall EzGetProfile (
                int id,
                double * vb,
                double * vs,
                double * a,
                double * d,
                double * ve)
```

获取轴参数

参数

id	轴 id
vb	速度
vs	最大速度
a	加速度
d	减速度
ve	末速度

返回

错误码

4.1.4.57 EzGetPulseCount()

```
SOFT_MOTION_LIB_API int __stdcall EzGetPulseCount (
                int id,
```

```
                unsigned char index,
                int * value)
```

获取脉冲计数

参数

id	终端 id
index	脉冲索引
value	脉冲计数值

返回

错误码

4.1.4.58 EzGetPulseFrequency()

```
SOFT_MOTION_LIB_API int __stdcall EzGetPulseFrequency (
                int id,
                unsigned char index,
                int * value)
```

获取脉冲输出

参数

id	终端 id
index	encoder 索引
value	脉冲输出值

返回

错误码

4.1.4.59 EzGetSoftLimitConfig()

```
SOFT_MOTION_LIB_API int __stdcall EzGetSoftLimitConfig (
                int id,
                EzSoftLimitConfig * config)
```

获取软限位运行时配置

参数

id	轴 id
config	软限位配置

返回

错误码

4.1.4.60 EzGetSProfile()

```
SOFT_MOTION_LIB_API int __stdcall EzGetSProfile (
                int id,
                double * J,
                double * Jd)
```

获取轴S 型曲线参数

参数

id	轴 id
J	加加速度
Jd	减加速度

返回

错误码

4.1.4.61 EzGetTerminalInfo()

```
SOFT_MOTION_LIB_API int __stdcall EzGetTerminalInfo (
                EzTerminalInfo ** terminal_info,
                int * terminal_count)
```

获取终端信息

参数

terminal_info	终端信息
terminal_count	终端数量

返回

错误码

4.1.4.62 EzHome()

```
SOFT_MOTION_LIB_API int __stdcall EzHome (
                int id)
```

轴回零

参数

id	轴 id

返回

错误码

4.1.4.63 EzInit()

```
SOFT_MOTION_LIB_API int __stdcall EzInit (
                const char * ip = ”127.0.0.1”)
```

初始化, 所有调用要在初始化之后调用

参数

ip	本机链接可以缺省, 远程链接时请填写控制器 ip

返回

错误码

4.1.4.64 EzJog()

```
SOFT_MOTION_LIB_API int __stdcall EzJog (
                int id,
                EzMoveDirection direction)
```

轴 jog

参数

id	轴 id
direction	运动方向

返回

错误码

4.1.4.65 EzMove()

```
SOFT_MOTION_LIB_API int __stdcall EzMove (
                int id,
                double target_position,
                EzMoveType move_type)
```

轴运动

参数

id	轴 id

target_position	目标位置
move_type	运动类型

返回

错误码

4.1.4.66 EzMoveEx()

```
SOFT_MOTION_LIB_API int __stdcall EzMoveEx (
                int id,
                double position,
                double vb,
                double vs,
                double a,
                double d,
                double ve,
                double J,
                double Jd,
                EzMoveType move_type)
```

轴运动, 带参数, 用于高速场景

参数

id	轴 id
position	目标位置
vb	速度
vs	最大速度
a	加速度
d	减速度
ve	末速度
J	加加速度
Jd	减加速度
move_type	运动类型

返回

错误码

4.1.4.67 EzPower()

```
SOFT_MOTION_LIB_API int __stdcall EzPower (
                int id,
                bool power_on)
```

轴使能

参数

id	轴 id
power_on	使能

返回

错误码

4.1.4.68 EzReadEcatPDOBool()

```
SOFT_MOTION_LIB_API int __stdcall EzReadEcatPDOBool (
                unsigned char master_index,
                unsigned char slave_index,
                unsigned short address,
                unsigned short var_index,
                bool * value)
```

读取 EtherCAT 主站 PDO 地址空间中的布尔值

参数

master_index	主站索引
slave_index	从站索引
address	PDO 地址
var_index	PDO 子索引
value	读取到的布尔值

返回

错误码

4.1.4.69 EzReadEcatPDOChar()

```
SOFT_MOTION_LIB_API int __stdcall EzReadEcatPDOChar (
                unsigned char master_index,
                unsigned char slave_index,
                unsigned short address,
                unsigned short var_index,
                char * value)
```

读取 EtherCAT 主站 PDO 地址空间中的字符值

参数

master_index	主站索引
slave_index	从站索引
address	PDO 地址
var_index	PDO 子索引
value	读取到的字符值

返回

错误码

4.1.4.70 EzReadEcatPDODouble()

```
SOFT_MOTION_LIB_API int __stdcall EzReadEcatPDODouble (
                unsigned char master_index,
                unsigned char slave_index,
                unsigned short address,
                unsigned short var_index,
                double * value)
```

读取 EtherCAT PDO 中的浮点值

参数

master_index	主站索引
slave_index	从站索引

address	PDO 地址
var_index	PDO 子索引
value	读取到的浮点值

返回

错误码

4.1.4.71 EzReadEcatPDOInt()

```
SOFT_MOTION_LIB_API int __stdcall EzReadEcatPDOInt (
                unsigned char master_index,
                unsigned char slave_index,
                unsigned short address,
                unsigned short var_index,
                int * value)
```

读取 EtherCAT 主站 PDO 地址空间中的整数值

参数

master_index	主站索引
slave_index	从站索引
address	PDO 地址
var_index	PDO 子索引
value	读取到的整数值

返回

错误码

4.1.4.72 EzReadEcatPDOShort()

```
SOFT_MOTION_LIB_API int __stdcall EzReadEcatPDOShort (
                unsigned char master_index,
                unsigned char slave_index,
```

```
                unsigned short address,
                unsigned short var_index,
                short * value)
```

读取 EtherCAT 主站 PDO 地址空间中的 short 值

参数

master_index	主站索引
slave_index	从站索引
address	PDO 地址
var_index	PDO 子索引
value	读取到的 short 值

返回

错误码

4.1.4.73 EzReadEcatSDOBool()

```
SOFT_MOTION_LIB_API int __stdcall EzReadEcatSDOBool (
                unsigned char master_index,
                unsigned char slave_index,
                unsigned short address,
                unsigned short var_index,
                bool * value)
```

读取 EtherCAT 主站 SDO 地址空间中的布尔值

参数

master_index	主站索引
slave_index	从站索引
address	SDO 地址
var_index	SDO 子索引
value	读取到的布尔值

返回

错误码

4.1.4.74 EzReadEcatSDOChar()

```
SOFT_MOTION_LIB_API int __stdcall EzReadEcatSDOChar (
                unsigned char master_index,
                unsigned char slave_index,
                unsigned short address,
                unsigned short var_index,
                char * value)
```

读取 EtherCAT 主站 SDO 地址空间中的字符值

参数

master_index	主站索引
slave_index	从站索引
address	SDO 地址
var_index	SDO 子索引
value	读取到的字符值

返回

错误码

4.1.4.75 EzReadEcatSDODouble()

```
SOFT_MOTION_LIB_API int __stdcall EzReadEcatSDODouble (
                unsigned char master_index,
                unsigned char slave_index,
                unsigned short address,
                unsigned short var_index,
                double * value)
```

读取 EtherCAT SDO 中的浮点值

参数

master_index	主站索引
slave_index	从站索引

address	SDO 地址
var_index	SDO 子索引
value	读取到的浮点值

返回

错误码

4.1.4.76 EzReadEcatSDOInt()

```
SOFT_MOTION_LIB_API int __stdcall EzReadEcatSDOInt (
                unsigned char master_index,
                unsigned char slave_index,
                unsigned short address,
                unsigned short var_index,
                int * value)
```

读取 EtherCAT 主站 SDO 地址空间中的整数值

参数

master_index	主站索引
slave_index	从站索引
address	SDO 地址
var_index	SDO 子索引
value	读取到的整数值

返回

错误码

4.1.4.77 EzReadEcatSDOShort()

```
SOFT_MOTION_LIB_API int __stdcall EzReadEcatSDOShort (
                unsigned char master_index,
                unsigned char slave_index,
```

```
                unsigned short address,
                unsigned short var_index,
                short * value)
```

读取 EtherCAT 主站 SDO 地址空间中的 short 值

参数

master_index	主站索引
slave_index	从站索引
address	SDO 地址
var_index	SDO 子索引
value	读取到的 short 值

返回

错误码

4.1.4.78 EzScan()

```
SOFT_MOTION_LIB_API int __stdcall EzScan (
                EzBusType bus_type,
                unsigned char master_index,
                EzSlaveInfo ** slaves,
                int * slave_count)
```

扫描从站设备信息

参数

bus_type	总线类型
master_index	主站号
slaves	从站信息
slave_count	从站数量

返回

错误码

4.1.4.79 EzSetAI()

```
SOFT_MOTION_LIB_API int __stdcall EzSetAI (
                int id,
                unsigned char index,
                short value)
```

设定模拟量输入

参数

id	终端 id
index	AI 索引
value	AI 值

返回

错误码

注解

该函数只适用于仿真状态

4.1.4.80 EzSetAO()

```
SOFT_MOTION_LIB_API int __stdcall EzSetAO (
                int id,
                unsigned char index,
                short value)
```

设定模拟量输出

参数

id	终端 id
index	AO 索引
value	AO 值

返回

错误码

4.1.4.81 EzSetAxisGroupProfile()

```
SOFT_MOTION_LIB_API int __stdcall EzSetAxisGroupProfile (
                int id,
                double vs,
                double a,
                double d)
```

设定轴组运动参数

参数

id	轴组 id
vs	最大速度
a	加速度
d	减速度

返回

错误码

注解

调用以后轴组运动曲线为梯形

4.1.4.82 EzSetAxisGroupSProfile()

```
SOFT_MOTION_LIB_API int __stdcall EzSetAxisGroupSProfile (
                int id,
                double J,
                double Jd)
```

获取轴组运动参数

参数

id	轴组 id
J	加加速度

Jd	减加速度

返回

错误码

注解

调用以后轴组运动曲线为S 型

4.1.4.83 EzSetAxisHomeProfile()

```
SOFT_MOTION_LIB_API int __stdcall EzSetAxisHomeProfile (
                int id,
                int home_mode,
                double low_velocity,
                double high_velocity,
                double acceleration,
                double offset)
```

设置轴控制器回零参数

参数

id	轴 id
home_mode	回零模式
low_velocity	回零低速速度
high_velocity	回零高速速度
acceleration	回零加速度
offset	回零偏置

返回

错误码

4.1.4.84 EzSetControllerWorkMode()

```
SOFT_MOTION_LIB_API int __stdcall EzSetControllerWorkMode (
                bool simulation)
```

设置控制器工作模式, 只有在EzStart 前生效

参数

simulation	模拟模式

返回

错误码

4.1.4.85 EzSetDI()

```
SOFT_MOTION_LIB_API int __stdcall EzSetDI (
                int id,
                unsigned char index,
                bool value)
```

设置数字量输入

参数

id	终端 id
index	DI 索引
value	DI 值

返回

错误码

注解

该函数只适用于仿真状态

4.1.4.86 EzSetDO()

```
SOFT_MOTION_LIB_API int __stdcall EzSetDO (
                int id,
                unsigned char index,
                bool value)
```

设定数字量输出

参数

id	终端 id
index	DO 索引
value	DO 值

返回

错误码

4.1.4.87 EzSetEncoder()

```
SOFT_MOTION_LIB_API int __stdcall EzSetEncoder (
                int id,
                unsigned char index,
                int value)
```

设定编码器

参数

id	终端 id
index	encoder 索引
value	编码器数值

返回

错误码

注解

该函数只适用于仿真状态

4.1.4.88 EzSetGearProfile()

```
SOFT_MOTION_LIB_API int __stdcall EzSetGearProfile (
                int slave_axis,
                const EzGearProfile * profile)
```

设置从轴的 gear 配置

参数

slave_axis	从轴号
profile	gear 配置，master_axis = -1 表示清除配置

返回

错误码

4.1.4.89 EzSetHardLimitEnable()

```
SOFT_MOTION_LIB_API int __stdcall EzSetHardLimitEnable (
                int id,
                const EzDirectionalEnable * enable)
```

设置硬限位运行时使能状态

参数

id	轴 id
enable	正负方向使能配置

返回

错误码

4.1.4.90 EzSetHomeProfile()

```
SOFT_MOTION_LIB_API int __stdcall EzSetHomeProfile (
                int id,
                int home_mode,
                double low_velocity,
                double high_velocity,
                double acceleration,
                double deceleration,
                double offset)
```

设置轴Home 参数, 此回零为控制器控制回零, 回零开关接在远程I/O 上

参数

id	轴 id
home_mode	轴Home 模式
low_velocity	回零低速速度
high_velocity	回零高速速度
acceleration	回零加速度
deceleration	回零减速度
offset	回零偏移量

返回

错误码

4.1.4.91 EzSetPosition()

```
SOFT_MOTION_LIB_API int __stdcall EzSetPosition (
                int id,
                double position)
```

强制设置轴位置

参数

id	轴 id

position	位置

返回

错误码

4.1.4.92 EzSetProfile()

```
SOFT_MOTION_LIB_API int __stdcall EzSetProfile (
                int id,
                double vb,
                double vs,
                double a,
                double d,
                double ve)
```

设置轴参数

参数

id	轴 id
vb	开始速度
vs	设定速度, 运动可达最大速度
a	加速度
d	减速度
ve	末速度

返回

错误码

4.1.4.93 EzSetPulseCount()

```
SOFT_MOTION_LIB_API int __stdcall EzSetPulseCount (
                int id,
```

```
                unsigned char index,
                int value)
```

设置脉冲计数

参数

id	终端 id
index	脉冲索引
value	脉冲计数数值

返回

错误码

注解

该函数只适用于仿真状态

4.1.4.94 EzSetPulseFrequency()

```
SOFT_MOTION_LIB_API int __stdcall EzSetPulseFrequency (
                int id,
                unsigned char index,
                int value)
```

设置脉冲输出

参数

id	终端 id
index	脉冲输出索引
value	脉冲频率

返回

错误码

4.1.4.95 EzSetSoftLimitConfig()

```
SOFT_MOTION_LIB_API int __stdcall EzSetSoftLimitConfig (
                int id,
                const EzSoftLimitConfig * config)
```

设置软限位运行时配置

参数

id	轴 id
config	软限位配置

返回

错误码

4.1.4.96 EzSetSProfile()

```
SOFT_MOTION_LIB_API int __stdcall EzSetSProfile (
                int id,
                double J,
                double Jd)
```

设置轴S 型曲线参数

参数

id	轴 id
J	加加速度
Jd	减加速度

返回

错误码

4.1.4.97 EzSoftEndMove()

```
SOFT_MOTION_LIB_API int __stdcall EzSoftEndMove (
                int id,
                double mid_position,
                double position,
                double vb,
                double vs,
                double a,
                double d,
                double ve,
                double J,
                double Jd,
                EzMoveType move_type)
```

轴软着陆移动

参数

id	轴 id
mid_position	中点位置
position	终点位置
vb	速度
vs	最大速度
a	加速度
d	减速度
ve	末速度
J	加加速度
Jd	减加速度
move_type	运动类型

返回

错误码

4.1.4.98 EzSoftStartMove()

```
SOFT_MOTION_LIB_API int __stdcall EzSoftStartMove (
                int id,
```

```
                double mid_position,
                double position,
                double vb,
                double vs1,
                double ve1,
                double delay_time,
                double vs2,
                double a,
                double d,
                double ve2,
                double J,
                double Jd,
                EzMoveType move_type)
```

轴软启动移动

参数

id	轴 id
mid_position	中点位置
position	终点位置
vb	速度
vs1	启动速度
ve1	末速度
delay_time	启动延时
vs2	减速速度
a	加速度
d	减速度
ve2	末减速速度
J	加加速度
Jd	减加速度
move_type	运动类型

返回

错误码

4.1.4.99 EzStart()

```
SOFT_MOTION_LIB_API int __stdcall EzStart (
            const char * path)
```

启动控制器, 成功后总线已经在OP

参数

path	启动文件路径

返回

错误码

4.1.4.100 EzStop()

```
SOFT_MOTION_LIB_API int __stdcall EzStop ()
```

停止控制器

返回

错误码

4.1.4.101 EzUnInit()

```
SOFT_MOTION_LIB_API int __stdcall EzUnInit ()
```

释放资源

返回

错误码

4.1.4.102 EzUpdateCyclicPosition()

```
SOFT_MOTION_LIB_API int __stdcall EzUpdateCyclicPosition (
                int id,
                const char * path)
```

导入点表移动, 根据传入文件中时间和位置的序列进行移动

参数

id	轴 id
path	轨迹文件路径

返回

错误码

4.1.4.103 EzUpdateTargetPosition()

```
SOFT_MOTION_LIB_API int __stdcall EzUpdateTargetPosition (
                int id,
                double position)
```

轴在线更新目标位置

参数

id	轴 id
position	目标位置

返回

错误码

4.1.4.104 EzUpdateTargetPositionEx()

```
SOFT_MOTION_LIB_API int __stdcall EzUpdateTargetPositionEx (
                int id,
                double mid_position,
                double target_position)
```

在线修调目标位置并实现软着陆

参数

id	轴 id
mid_position	中点位置
target_position	目标位置

返回

错误码

4.1.4.105 EzWriteEcatPDOBool()

```
SOFT_MOTION_LIB_API int __stdcall EzWriteEcatPDOBool (
                unsigned char master_index,
                unsigned char slave_index,
                unsigned short address,
                unsigned short var_index,
                bool value)
```

向 EtherCAT 主站 PDO 地址空间写入布尔值

参数

master_index	主站索引
slave_index	从站索引
address	PDO 地址
var_index	PDO 子索引
value	要写入的布尔值

返回

错误码

4.1.4.106 EzWriteEcatPDOChar()

```
SOFT_MOTION_LIB_API int __stdcall EzWriteEcatPDOChar (
                unsigned char master_index,
                unsigned char slave_index,
                unsigned short address,
                unsigned short var_index,
                char value)
```

向 EtherCAT 主站 PDO 地址空间写入字符值

参数

master_index	主站索引
slave_index	从站索引
address	PDO 地址
var_index	PDO 子索引
value	要写入的字符值

返回

错误码

4.1.4.107 EzWriteEcatPDODouble()

```
SOFT_MOTION_LIB_API int __stdcall EzWriteEcatPDODouble (
                unsigned char master_index,
                unsigned char slave_index,
                unsigned short address,
                unsigned short var_index,
                double value)
```

向 EtherCAT 主站 PDO 地址空间写入浮点值

参数

master_index	主站索引
slave_index	从站索引

address	PDO 地址
var_index	PDO 子索引
value	要写入的浮点值

返回

错误码

4.1.4.108 EzWriteEcatPDOInt()

```
SOFT_MOTION_LIB_API int __stdcall EzWriteEcatPDOInt (
                unsigned char master_index,
                unsigned char slave_index,
                unsigned short address,
                unsigned short var_index,
                int value)
```

向 EtherCAT 主站 PDO 地址空间写入整数值

参数

master_index	主站索引
slave_index	从站索引
address	PDO 地址
var_index	PDO 子索引
value	要写入的整数值

返回

错误码

4.1.4.109 EzWriteEcatPDOShort()

```
SOFT_MOTION_LIB_API int __stdcall EzWriteEcatPDOShort (
                unsigned char master_index,
                unsigned char slave_index,
```

```
                unsigned short address,
                unsigned short var_index,
                short value)
```

向 EtherCAT 主站 PDO 地址空间写入 short 值

参数

master_index	主站索引
slave_index	从站索引
address	PDO 地址
var_index	PDO 子索引
value	要写入的 short 值

返回

错误码

4.1.4.110 EzWriteEcatSDOBool()

```
SOFT_MOTION_LIB_API int __stdcall EzWriteEcatSDOBool (
                unsigned char master_index,
                unsigned char slave_index,
                unsigned short address,
                unsigned short var_index,
                bool value)
```

向 EtherCAT 主站 SDO 地址空间写入布尔值

参数

master_index	主站索引
slave_index	从站索引
address	SDO 地址
var_index	SDO 子索引
value	要写入的布尔值

返回

错误码

4.1.4.111 EzWriteEcatSDOChar()

```
SOFT_MOTION_LIB_API int __stdcall EzWriteEcatSDOChar (
                unsigned char master_index,
                unsigned char slave_index,
                unsigned short address,
                unsigned short var_index,
                char value)
```

向 EtherCAT 主站 SDO 地址空间写入字符值

参数

master_index	主站索引
slave_index	从站索引
address	SDO 地址
var_index	SDO 子索引
value	要写入的字符值

返回

错误码

4.1.4.112 EzWriteEcatSDODouble()

```
SOFT_MOTION_LIB_API int __stdcall EzWriteEcatSDODouble (
                unsigned char master_index,
                unsigned char slave_index,
                unsigned short address,
                unsigned short var_index,
                double value)
```

向 EtherCAT 主站 SDO 地址空间写入浮点值

参数

master_index	主站索引
slave_index	从站索引

address	SDO 地址
var_index	SDO 子索引
value	要写入的浮点值

返回

错误码

4.1.4.113 EzWriteEcatSDOInt()

```
SOFT_MOTION_LIB_API int __stdcall EzWriteEcatSDOInt (
                unsigned char master_index,
                unsigned char slave_index,
                unsigned short address,
                unsigned short var_index,
                int value)
```

向 EtherCAT 主站 SDO 地址空间写入整数值

参数

master_index	主站索引
slave_index	从站索引
address	SDO 地址
var_index	SDO 子索引
value	要写入的整数值

返回

错误码

4.1.4.114 EzWriteEcatSDOShort()

```
SOFT_MOTION_LIB_API int __stdcall EzWriteEcatSDOShort (
                unsigned char master_index,
                unsigned char slave_index,
```

```
unsigned short address,
unsigned short var_index,
short value)
```

向 EtherCAT 主站 SDO 地址空间写入 short 值

参数

master_index	主站索引
slave_index	从站索引
address	SDO 地址
var_index	SDO 子索引
value	要写入的 short 数据值

返回

错误码

4.2 ez_motion.h

浏览该文件的文档.

```
00001
00010 #ifndef EZ_MOTION_H
00011 #define EZ_MOTION_H
00012
00013 #ifndef SOFT_MOTION_LIB_API
00014 #ifdef _WIN32
00015 #ifdef SOFT_MOTION_EXPORTS
00016 #define SOFT_MOTION_LIB_API __declspec(dllexport)
00017 #else
00018 #define SOFT_MOTION_LIB_API
00019 #endif
00020 #else
00021 #define SOFT_MOTION_LIB_API
00022 #endif
00023 #endif
00024
00025 #define EZ_AXIS_GROUP_MAX_AXIS 9
00026
00027 extern "C" {
00028 enum EzBusType{
```

```
00029     ETHERCAT = 0,
00030     ETHERMAC = 1,
00031 };
00032
00033 enum EzMoveType {
00034     MOVE_TYPE_ABSOLUTE = 0,
00035     MOVE_TYPE_RELATIVE = 1,
00036 };
00037
00038 enum EzMoveDirection {
00039     MOVE_DIR_POSITIVE = 0,
00040     MOVE_DIR_NEGATIVE = 1,
00041 };
00042
00043 enum EzEnableMode {
00044     ENABLE_CLOSE = 0,
00045     ENABLE_ONE = 1,
00046     ENABLE_MULTI = 2,
00047 };
00048
00049 enum EzTriggerMode {
00050     TRIGGER_MODE_FALLING = 0,
00051     TRIGGER_MODE_RISING = 1,
00052 };
00053
00054 enum EzTriggerSource {
00055     TRIGGER_SOURCE_CMD = 0,
00056     TRIGGER_SOURCE_ENCODER = 1,
00057 };
00058
00059 enum EzStopMode {
00060     STOP_MODE_CONTROLLED = 0,
00061     STOP_MODE_IMMEDIATE = 1,
00062 };
00063
00064 enum EzEngineState {
00065     ENGINE_STATE_UNINIT = 0,
00066     ENGINE_STATE_INITED,
00067     ENGINE_STATE_STARTED,
00068     ENGINE_STATE_STOPPING,
00069     ENGINE_STATE_STOPPED,
00070 };
00071
00072 enum EzEngineStopReason {
00073     ENGINE_STOP_REASON_NONE = 0,
00074     ENGINE_STOP_REASON_USER_REQUEST,
00075     ENGINE_STOP_REASON_FIELDBUS_FATAL,
00076     ENGINE_STOP_REASON_UNKNOWN,
00077 };
00078
```

```
00079 enum EzBusFatalReason {
00080     BUS_FATAL_REASON_NONE = 0,
00081     BUS_FATAL_REASON_MASTER_ERROR_INTERRUPT,
00082     BUS_FATAL_REASON_MASTER_DISCONNECTED,
00083     BUS_FATAL_REASON_SLAVE_OFFLINE,
00084     BUS_FATAL_REASON_SLAVE_COUNT_CHANGED,
00085     BUS_FATAL_REASON_DATASET_CONSECUTIVE_FAILURE,
00086     BUS_FATAL_REASON_HEARTBEAT_TIMEOUT,
00087     BUS_FATAL_REASON_UNKNOWN,
00088 };
00089
00090 struct EzEngineStatus {
00091     EzEngineState state;
00092     EzEngineStopReason stopReason;
00093     EzBusFatalReason fatalReason;
00094 };
00095
00096 enum EzAxisState {
00097     AXIS_STATE_ERROR_STOP = 0,
00098     AXIS_STATE_DISABLED,
00099     AXIS_STATE_STANDSTILL,
00100     AXIS_STATE_HOMING,
00101     AXIS_STATE_STOPPING,
00102     AXIS_STATE_DISCRETE_MOTION,
00103     AXIS_STATE_SYNCHRONIZED_MOTION,
00104     AXIS_STATE_CONTINUOUS_MOTION,
00105 };
00106
00107 enum EzAxisLimitStopReason {
00108     EZ_AXIS_LIMIT_STOP_REASON_NONE = 0,
00109     EZ_AXIS_LIMIT_STOP_REASON_SOFT_POSITIVE,
00110     EZ_AXIS_LIMIT_STOP_REASON_SOFT_NEGATIVE,
00111     EZ_AXIS_LIMIT_STOP_REASON_HARD_POSITIVE,
00112     EZ_AXIS_LIMIT_STOP_REASON_HARD_NEGATIVE,
00113 };
00114
00115 enum EzHomeState {
00116     AXIS_HOME_UNHOMED = 0,
00117     AXIS_HOME_HOMING = 1,
00118     AXIS_HOME_HOMED = 2,
00119 };
00120
00121 enum EzArcDir {
00122     ARC_CW = 0,
00123     ARC_CCW = 1,
00124 };
00125
00126 enum EzAxisGroupState {
00127     AXIS_GROUP_STATE_DISABLED = 1,
00128     AXIS_GROUP_STATE_STANDSTILL = 2,
```

```
00129     AXIS_GROUP_STATE_MOVING = 4,
00130     AXIS_GROUP_STATE_STOPPING = 5,
00131     AXIS_GROUP_STATE_ERROR_STOP = 6,
00132 };
00133
00135 enum EzAxisGroupSegmentProfileMask {
00136     EZ_AXIS_GROUP_SEGMENT_PROFILE_NONE = 0,
00137     EZ_AXIS_GROUP_SEGMENT_PROFILE_VS = 1 « 0,
00138     EZ_AXIS_GROUP_SEGMENT_PROFILE_A = 1 « 1,
00139     EZ_AXIS_GROUP_SEGMENT_PROFILE_D = 1 « 2,
00140     EZ_AXIS_GROUP_SEGMENT_PROFILE_VE = 1 « 3,
00141     EZ_AXIS_GROUP_SEGMENT_PROFILE_J = 1 « 4,
00142     EZ_AXIS_GROUP_SEGMENT_PROFILE_JD = 1 « 5,
00143 };
00144
00145 struct EzAxisCycleData {
00146     EzAxisState state;
00147     double cmd_position;
00148     double cmd_velocity;
00149     short cmd_torque;
00150     double feedback_position;
00151     double feedback_velocity;
00152     short feedback_torque;
00153     bool limit_pos;
00154     bool limit_neg;
00155     EzAxisLimitStopReason limit_stop_reason;
00156 };
00157
00158 struct EzDirectionalEnable {
00159     bool positive;
00160     bool negative;
00161 };
00162
00163 struct EzSoftLimitConfig {
00164     bool positive_enable;
00165     double positive_position;
00166     bool negative_enable;
00167     double negative_position;
00168 };
00169
00170 struct EzAxisGroupCycleData {
00171     EzAxisGroupState state;
00172     unsigned int line_no;
00173     unsigned short axis_count;
00174     double cmd_position[EZ_AXIS_GROUP_MAX_AXIS];
00175     double actual_position[EZ_AXIS_GROUP_MAX_AXIS];
00176     double cmd_path_velocity;
00177     double actual_path_velocity;
00178 };
00179
```

```
00180 struct EzAxisGroupSegmentProfileEx {
00181     unsigned int mask;
00182     double vs;
00183     double a;
00184     double d;
00185     double ve;
00186     double j;
00187     double jd;
00188 };
00189
00191 struct EzGearErrorProtect {
00192     bool enable;
00193     double decel_stop_error;
00194     double emg_stop_error;
00195 };
00196
00198 struct EzGearProfile {
00199     int master_axis;
00200     double ratio;
00201     EzGearErrorProtect protect;
00202 };
00203
00204 struct EzSlaveInfo {
00205     unsigned int slave_id;
00206     unsigned int vendor_id;
00207     unsigned int product_code;
00208     unsigned int revision;
00209     unsigned int modules[32];
00210     unsigned int module_count;
00211 };
00212
00213 struct EzMasterInfo {
00214     EzBusType type;
00215     unsigned char index;
00216 };
00217
00218 struct EzAxisInfo {
00219     unsigned short id;
00220     char name[64];
00221 };
00222
00223 struct EzTerminalInfo {
00224     unsigned short id;
00225     char name[64];
00226     unsigned short do_number;
00227     unsigned short di_number;
00228     unsigned short ao_number;
00229     unsigned short ai_number;
00230     unsigned short pulse_counter_number;
00231     unsigned short encoder_number;
```

```
00232     unsigned short pulse_frequency_number;
00233 };
00234
00240 SOFT_MOTION_LIB_API int __stdcall EzInit(const char* ip = "127.0.0.1");
00245 SOFT_MOTION_LIB_API int __stdcall EzUnInit();
00254  SOFT_MOTION_LIB_API  int  __stdcall  EzScan(EzBusType  bus_type,  unsigned  char  mas-
     ter_index, EzSlaveInfo** slaves, int* slave_count);
00262 SOFT_MOTION_LIB_API int __stdcall EzGetMasterCycleTime(EzBusType bus_type, unsigned
     char master_index, int* cycle_time);
00272 SOFT_MOTION_LIB_API int __stdcall EzWriteEcatPDOChar(unsigned char master_index, un-
     signed char slave_index, unsigned short address, unsigned short var_index, char value);
00282 SOFT_MOTION_LIB_API int __stdcall EzReadEcatPDOChar(unsigned char master_index, un-
     signed char slave_index, unsigned short address, unsigned short var_index, char* value);
00292 SOFT_MOTION_LIB_API int __stdcall EzWriteEcatPDOBool(unsigned char master_index, un-
     signed char slave_index, unsigned short address, unsigned short var_index, bool value);
00302 SOFT_MOTION_LIB_API int __stdcall EzReadEcatPDOBool(unsigned char master_index, un-
     signed char slave_index, unsigned short address, unsigned short var_index, bool* value);
00312 SOFT_MOTION_LIB_API int __stdcall EzWriteEcatPDOShort(unsigned char master_index, un-
     signed char slave_index, unsigned short address, unsigned short var_index, short value);
00322 SOFT_MOTION_LIB_API int __stdcall EzReadEcatPDOShort(unsigned char master_index, un-
     signed  char  slave_index,  unsigned  short  address,  unsigned  short  var_index,  short*
     value);
00332 SOFT_MOTION_LIB_API int __stdcall EzWriteEcatPDOInt(unsigned char master_index, un-
     signed char slave_index, unsigned short address, unsigned short var_index, int value);
00342 SOFT_MOTION_LIB_API int __stdcall EzReadEcatPDOInt(unsigned char master_index, un-
     signed char slave_index, unsigned short address, unsigned short var_index, int* value);
00352 SOFT_MOTION_LIB_API int __stdcall EzWriteEcatPDODouble(unsigned char master_index,
     unsigned char slave_index, unsigned short address, unsigned short var_index, double
     value);
00362 SOFT_MOTION_LIB_API int __stdcall EzReadEcatPDODouble(unsigned char master_index, un-
     signed  char  slave_index,  unsigned  short  address,  unsigned  short  var_index,  double*
     value);
00372 SOFT_MOTION_LIB_API int __stdcall EzWriteEcatSDOChar(unsigned char master_index, un-
     signed char slave_index, unsigned short address, unsigned short var_index, char value);
00382 SOFT_MOTION_LIB_API int __stdcall EzReadEcatSDOChar(unsigned char master_index, un-
     signed char slave_index, unsigned short address, unsigned short var_index, char* value);
00392 SOFT_MOTION_LIB_API int __stdcall EzWriteEcatSDOBool(unsigned char master_index, un-
     signed char slave_index, unsigned short address, unsigned short var_index, bool value);
00402 SOFT_MOTION_LIB_API int __stdcall EzReadEcatSDOBool(unsigned char master_index, un-
     signed char slave_index, unsigned short address, unsigned short var_index, bool* value);
00412 SOFT_MOTION_LIB_API int __stdcall EzWriteEcatSDOShort(unsigned char master_index, un-
     signed char slave_index, unsigned short address, unsigned short var_index, short value);
00422 SOFT_MOTION_LIB_API int __stdcall EzReadEcatSDOShort(unsigned char master_index, un-
     signed  char  slave_index,  unsigned  short  address,  unsigned  short  var_index,  short*
     value);
00432 SOFT_MOTION_LIB_API int __stdcall EzWriteEcatSDOInt(unsigned char master_index, un-
     signed char slave_index, unsigned short address, unsigned short var_index, int value);
00442 SOFT_MOTION_LIB_API int __stdcall EzReadEcatSDOInt(unsigned char master_index, un-
     signed char slave_index, unsigned short address, unsigned short var_index, int* value);
00452 SOFT_MOTION_LIB_API int __stdcall EzWriteEcatSDODouble(unsigned char master_index,
     unsigned char slave_index, unsigned short address, unsigned short var_index, double
     value);
```

```
00462 SOFT_MOTION_LIB_API int __stdcall EzReadEcatSDODouble(unsigned char master_index, unsigned char slave_index, unsigned short address, unsigned short var_index, double* value);
00468 SOFT_MOTION_LIB_API int __stdcall EzSetControllerWorkMode(bool simulation);
00474 SOFT_MOTION_LIB_API int __stdcall EzGetControllerWorkMode(bool* simulation);
00480 SOFT_MOTION_LIB_API int __stdcall EzGetEngineStatus(EzEngineStatus* status);
00486 SOFT_MOTION_LIB_API int __stdcall EzGetDebugMode(bool* debug_mode);
00492 SOFT_MOTION_LIB_API int __stdcall EzStart(const char* path);
00498 SOFT_MOTION_LIB_API int __stdcall EzGetConfigFile(const char* path);
00505 SOFT_MOTION_LIB_API int __stdcall EzGetMasterInfo(EzMasterInfo** masters, int* master_count);
00512 SOFT_MOTION_LIB_API int __stdcall EzGetAxesInfo(EzAxisInfo** axes_info, int* axes_count);
00519 SOFT_MOTION_LIB_API int __stdcall EzGetTerminalInfo(EzTerminalInfo** terminal_info, int* terminal_count);
00524 SOFT_MOTION_LIB_API int __stdcall EzStop();
00531 SOFT_MOTION_LIB_API int __stdcall EzPower(int id, bool power_on);
00539 SOFT_MOTION_LIB_API int __stdcall EzMove(int id, double target_position, EzMoveType move_type);
00550 SOFT_MOTION_LIB_API int __stdcall EzSetProfile(int id, double vb, double vs, double a, double d, double ve);
00561 SOFT_MOTION_LIB_API int __stdcall EzGetProfile(int id, double* vb, double* vs, double* a, double* d, double* ve);
00569 SOFT_MOTION_LIB_API int __stdcall EzSetSProfile(int id, double J, double Jd);
00577 SOFT_MOTION_LIB_API int __stdcall EzGetSProfile(int id, double* J, double* Jd);
00584 SOFT_MOTION_LIB_API int __stdcall EzGetAxisCycleData(int id, EzAxisCycleData* cycle_data);
00591 SOFT_MOTION_LIB_API int __stdcall EzGetAxesCycleData(EzAxisCycleData** cycle_data, int* axis_count);
00598 SOFT_MOTION_LIB_API int __stdcall EzSetSoftLimitConfig(int id, const EzSoftLimitConfig* config);
00605 SOFT_MOTION_LIB_API int __stdcall EzGetSoftLimitConfig(int id, EzSoftLimitConfig* config);
00612 SOFT_MOTION_LIB_API int __stdcall EzSetHardLimitEnable(int id, const EzDirectionalEnable* enable);
00619 SOFT_MOTION_LIB_API int __stdcall EzGetHardLimitEnable(int id, EzDirectionalEnable* enable);
00626 SOFT_MOTION_LIB_API int __stdcall EzSetGearProfile(int slave_axis, const EzGearProfile* profile);
00633 SOFT_MOTION_LIB_API int __stdcall EzGetGearProfile(int slave_axis, EzGearProfile* profile);
00639 SOFT_MOTION_LIB_API int __stdcall EzGearIn(int slave_axis);
00645 SOFT_MOTION_LIB_API int __stdcall EzGearOut(int slave_axis);
00652 SOFT_MOTION_LIB_API int __stdcall EzGetGearState(int slave_axis, bool* in_gear);
00659 SOFT_MOTION_LIB_API int __stdcall EzJog(int id, EzMoveDirection direction);
00666 SOFT_MOTION_LIB_API int __stdcall EzUpdateTargetPosition(int id, double position);
00673 SOFT_MOTION_LIB_API int __stdcall EzSetPosition(int id, double position);
00681 SOFT_MOTION_LIB_API int __stdcall EzChangeSpeed(int id, double velocity, double acceleration);
00696 SOFT_MOTION_LIB_API int __stdcall EzMoveEx(int id, double position, double vb, double vs, double a, double d, double ve, double J, double Jd, EzMoveType move_type);
```

```
00712 SOFT_MOTION_LIB_API int __stdcall EzSoftEndMove(int id, double mid_position, double
    position, double vb, double vs, double a, double d, double ve, double J, double Jd,
    EzMoveType move_type);
00731 SOFT_MOTION_LIB_API int __stdcall EzSoftStartMove(
00732     int id, double mid_position, double position, double vb, double vs1, double ve1,
    double delay_time, double vs2, double a, double d, double ve2, double J, double Jd,
    EzMoveType move_type);
00740   SOFT_MOTION_LIB_API   int   __stdcall   EzUpdateTargetPositionEx(int   id,   double
    mid_position, double target_position);
00748 SOFT_MOTION_LIB_API int __stdcall EzGetAxisDI(int id, int index, bool* value);
00755 SOFT_MOTION_LIB_API int __stdcall EzAxisStop(int id, EzStopMode stop_mode);
00760 SOFT_MOTION_LIB_API int __stdcall EzEmergencyStop();
00761
00773  SOFT_MOTION_LIB_API  int  __stdcall  EzSetHomeProfile(int  id,  int  home_mode,  double
    low_velocity, double high_velocity, double acceleration, double deceleration, double
    offset);
00785 SOFT_MOTION_LIB_API int __stdcall EzGetHomeProfile(int id, int* home_mode, double*
    low_velocity, double* high_velocity, double* acceleration, double* deceleration, dou-
    ble* offset);
00791 SOFT_MOTION_LIB_API int __stdcall EzHome(int id);
00798 SOFT_MOTION_LIB_API int __stdcall EzGetHomeState(int id, EzHomeState* home_state);
00809 SOFT_MOTION_LIB_API int __stdcall EzSetAxisHomeProfile(int id, int home_mode, double
    low_velocity, double high_velocity, double acceleration, double offset);
00820 SOFT_MOTION_LIB_API int __stdcall EzGetAxisHomeProfile(int id, int* home_mode, double*
    low_velocity, double* high_velocity, double* acceleration, double* offset);
00826 SOFT_MOTION_LIB_API int __stdcall EzAxisHome(int id);
00832 SOFT_MOTION_LIB_API int __stdcall EzAxisReset(int id);
00842 SOFT_MOTION_LIB_API int __stdcall EzSetAxisGroupProfile(int id, double vs, double a,
    double d);
00851 SOFT_MOTION_LIB_API int __stdcall EzSetAxisGroupSProfile(int id, double J, double Jd);
00860 SOFT_MOTION_LIB_API int __stdcall EzGetAxisGroupProfile(int id, double* vs, double*
    a, double* d);
00868 SOFT_MOTION_LIB_API int __stdcall EzGetAxisGroupSProfile(int id, double* J, double*
    Jd);
00876 SOFT_MOTION_LIB_API int __stdcall EzAxisGroupOpen(int id, int axis_number, const un-
    signed short* axis_id_list);
00882 SOFT_MOTION_LIB_API int __stdcall EzAxisGroupStart(int id);
00888 SOFT_MOTION_LIB_API int __stdcall EzAxisGroupStop(int id);
00894 SOFT_MOTION_LIB_API int __stdcall EzAxisGroupPause(int id);
00900 SOFT_MOTION_LIB_API int __stdcall EzAxisGroupClose(int id);
00907     SOFT_MOTION_LIB_API     int     __stdcall     EzGetAxisGroupCycleData(int     id,
    EzAxisGroupCycleData* cycle_data);
00914 SOFT_MOTION_LIB_API int __stdcall EzAxisGroupChangeFeedRate(int id, double feed_rate);
00923 SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddLine(int id, int axis_number, const
    double* target_position, EzMoveType move_type);
00933 SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddLineEx(
00934      int id, int axis_number, const double* target_position, EzMoveType move_type,
    const EzAxisGroupSegmentProfileEx* profile);
00945 SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddArcCenter(int id, int axis_number,
    const double* target_position, const double* center_position, EzArcDir dir,
00946     EzMoveType move_type);
00958 SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddArcCenterEx(
```

```
00959       int id, int axis_number, const double* target_position, const double* cen-
    ter_position, EzArcDir dir, EzMoveType move_type,
00960     const EzAxisGroupSegmentProfileEx* profile);
00971 SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddArcRadius(int id, int axis_number,
    const double* target_position, double radius, EzArcDir dir, EzMoveType move_type);
00983 SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddArcRadiusEx(
00984     int id, int axis_number, const double* target_position, double radius, EzArcDir
    dir, EzMoveType move_type, const EzAxisGroupSegmentProfileEx* profile);
00995 SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddArc3Point(int id, int axis_number,
    const double* target_position, const double* mid_position, EzArcDir dir,
00996     EzMoveType move_type);
01008 SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddArc3PointEx(
01009    int id, int axis_number, const double* target_position, const double* mid_position,
    EzArcDir dir, EzMoveType move_type, const EzAxisGroupSegmentProfileEx* profile);
01016 SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddDelay(int id, double delay_time_ms);
01026 SOFT_MOTION_LIB_API int __stdcall EzAxisGroupWaitInput(int id, unsigned short termi-
    nal_id, unsigned short di_index, bool on_off, double timeout_s);
01036 SOFT_MOTION_LIB_API int __stdcall EzAxisGroupWriteOutBit(int id, unsigned short ter-
    minal_id, unsigned short do_index, bool on_off, double reverse_time_s);
01042 SOFT_MOTION_LIB_API int __stdcall EzAxisGroupAddEnd(int id);
01052 SOFT_MOTION_LIB_API int __stdcall EzAxisGroupMoveLine(int id, int axis_number, const
    unsigned short* axis_id_list, const double* target_position, EzMoveType move_type);
01064 SOFT_MOTION_LIB_API int __stdcall EzAxisGroupMoveArcCenter(
01065       int id, int axis_number, const unsigned short* axis_id_list, const double*
    target_position, const double* center_position, EzArcDir dir, EzMoveType move_type);
01077 SOFT_MOTION_LIB_API int __stdcall EzAxisGroupMoveArcRadius(
01078       int id, int axis_number, const unsigned short* axis_id_list, const double*
    target_position, double radius, EzArcDir dir, EzMoveType move_type);
01090 SOFT_MOTION_LIB_API int __stdcall EzAxisGroupMoveArc3Point(
01091       int id, int axis_number, const unsigned short* axis_id_list, const double*
    target_position, const double* mid_position, EzArcDir dir, EzMoveType move_type);
01098 SOFT_MOTION_LIB_API int __stdcall EzUpdateCyclicPosition(int id, const char* path);
01106 SOFT_MOTION_LIB_API int __stdcall EzGetDI(int id, unsigned char index, bool* value);
01107
01115 SOFT_MOTION_LIB_API int __stdcall EzSetDO(int id, unsigned char index, bool value);
01123 SOFT_MOTION_LIB_API int __stdcall EzGetDO(int id, unsigned char index, bool* value);
01131 SOFT_MOTION_LIB_API int __stdcall EzSetAO(int id, unsigned char index, short value);
01139 SOFT_MOTION_LIB_API int __stdcall EzGetAO(int id, unsigned char index, short* value);
01147 SOFT_MOTION_LIB_API int __stdcall EzGetAI(int id, unsigned char index, short* value);
01155 SOFT_MOTION_LIB_API int __stdcall EzGetPulseCount(int id, unsigned char index, int*
    value);
01163 SOFT_MOTION_LIB_API int __stdcall EzGetEncoder(int id, unsigned char index, int*
    value);
01171 SOFT_MOTION_LIB_API int __stdcall EzSetPulseFrequency(int id, unsigned char index, int
    value);
01179 SOFT_MOTION_LIB_API int __stdcall EzGetPulseFrequency(int id, unsigned char index,
    int* value);
01188 SOFT_MOTION_LIB_API int __stdcall EzSetDI(int id, unsigned char index, bool value);
01197 SOFT_MOTION_LIB_API int __stdcall EzSetAI(int id, unsigned char index, short value);
01206 SOFT_MOTION_LIB_API int __stdcall EzSetPulseCount(int id, unsigned char index, int
    value);
01215 SOFT_MOTION_LIB_API int __stdcall EzSetEncoder(int id, unsigned char index, int value);
```

```
01221 SOFT_MOTION_LIB_API const char* __stdcall EzGetErrorText(int error_code);
01226 SOFT_MOTION_LIB_API int __stdcall EzClose();
01227 }
01228
01229 #endif //EZ_MOTION_H
```