Agilebot Robot SDK
捷勃特机器人 SDK

Python SDK

基于 Python 的高性能机器人控制开发套件，提供简洁优雅的 API 设计，助力快速构建智能机器人应用。

了解更多 →

C# SDK

面向 .NET 生态的企业级机器人控制解决方案，提供类型安全的强类型 API，轻松集成至工业自动化系统。

了解更多 →

C# SDK

序章

版本记录

文档版本	SDK 版本号	版本时间
V3.3	2.1.0.*	2026.04.13

更新说明

机器人版本兼容性

SDK 支持捷勃特 Scara，Puma 及协作机器人系列。须对已安装机器人软件的设备使用。部分功能因版本不同，返回结果有所差异。 SDK 连接到机械臂时会对机械臂运动控制软件的版本进行检查，如果低于版本最低要求将无法连接，低于推荐版本要求将提示版本过低，请及时更新兼容的机器人软件版本 SDK 某些接口只支持对应的版本控制器，请注意查看具体接口兼容性

SDK 版本	兼容的机器人软件版本	支持状态
0.1.1.X	Copper v7.5.X.X、Bronze v7.5.X.X	不再支持
0.1.2.X	Copper v7.5.X.X、Bronze v7.5.X.X	不再支持
0.2.0.X	Copper v7.5.X.X、Bronze v7.5.X.X	不再支持
1.0.0.X	Copper v7.6.X.X、Bronze v7.5.X.X	支持中
2.0.X.X	Copper v7.7.X.X、Bronze v7.7.X.X	支持中
2.1.X.X	Copper v7.7.I.X	支持中

1 简介与部署

1.1 环境要求

系统：

- Windows 10 及以上
  - x86_64 架构
- .NET 版本
  - 6.0 以上
- .NET Framework 版本
  - 4.7 以上

1.2 安装

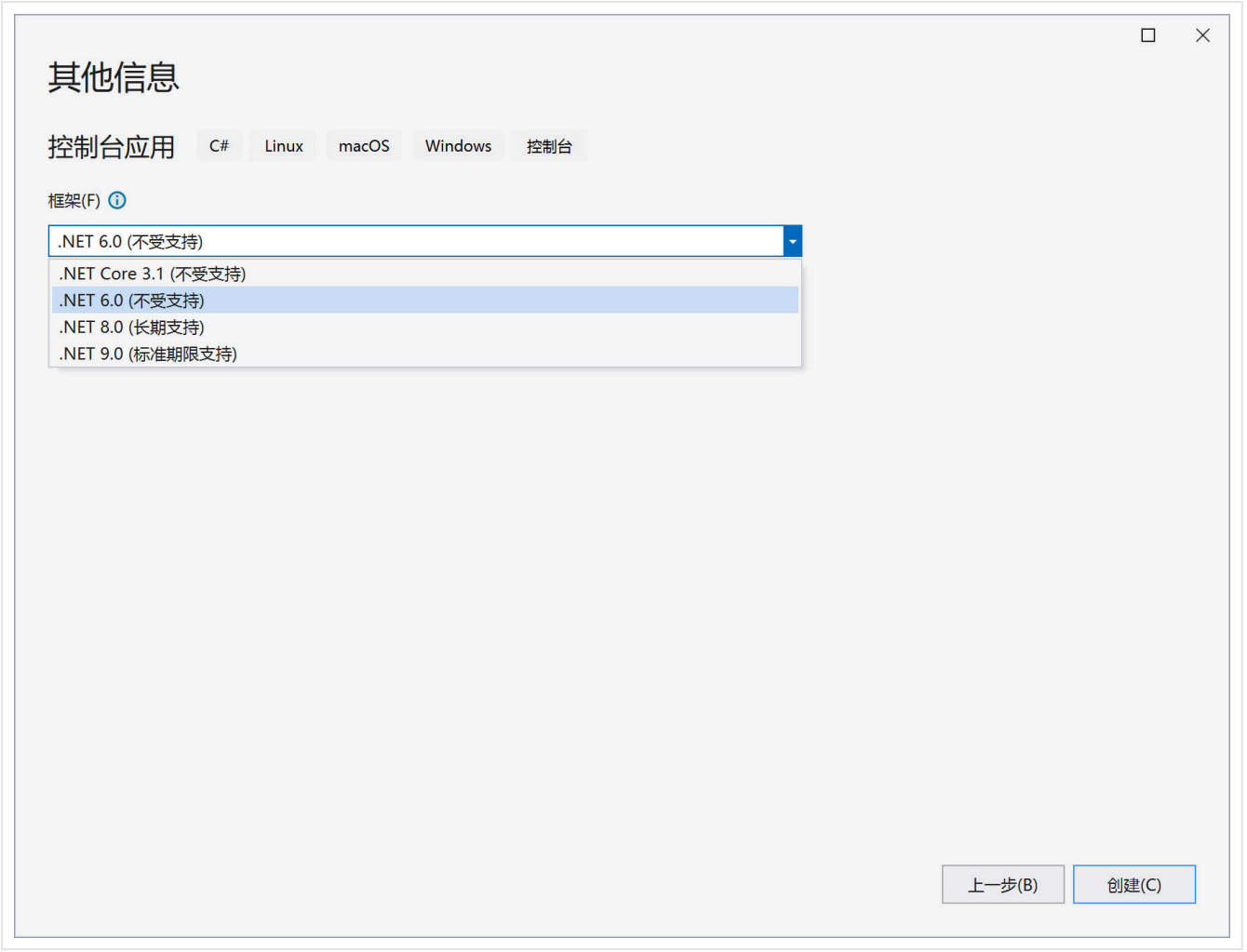
本节介绍开发环境准备、SDK 获取与常见运行注意事项，确保在最短时间内完成 Agilebot SDK 的本地调试。

IDE 安装与配置

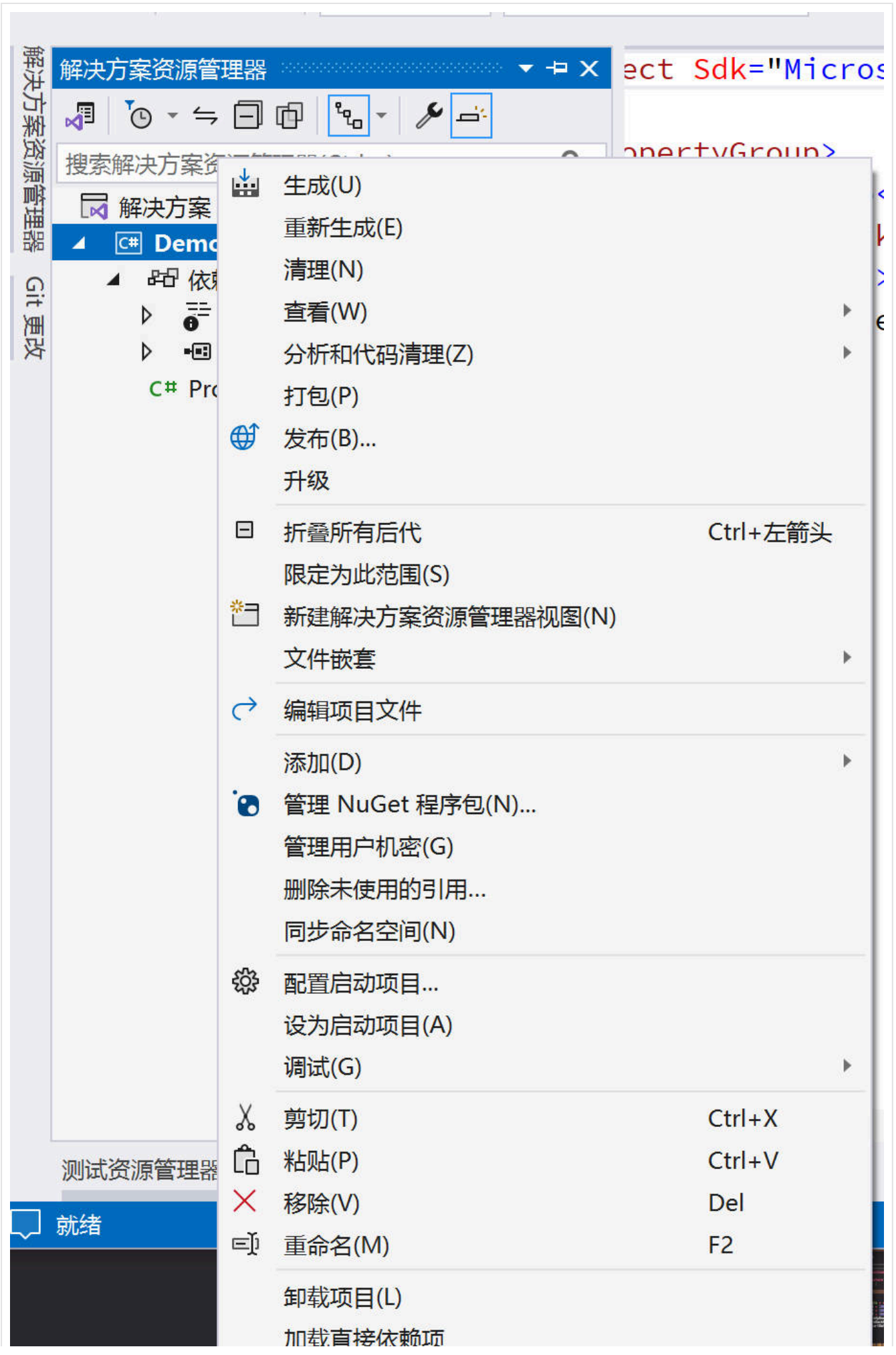
1. 推荐使用 Visual Studio 作为 C# 开发环境，可在 下载 Visual Studio Tools - 免费安装 Windows、Mac、Linux 获取最新版本。
2. 安装完成后启动 Visual Studio，并根据提示完成初始配置（例如登录、安装必要的工作负载）。

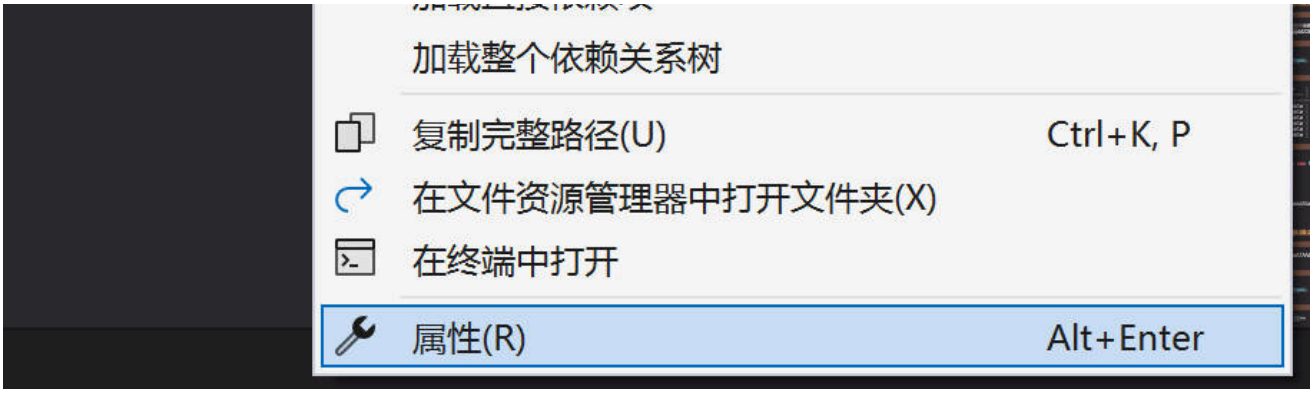
SDK 获取与项目创建

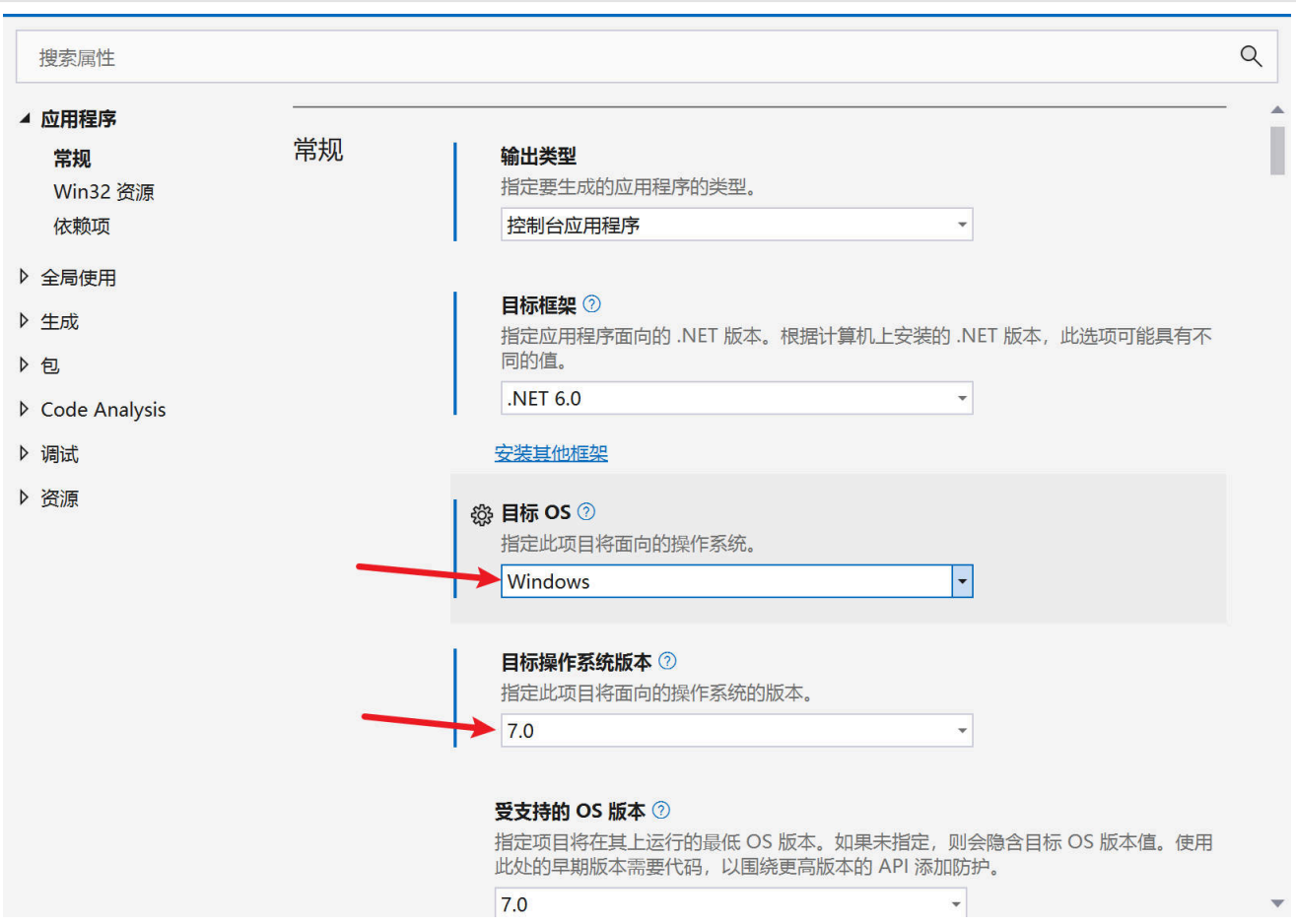
1. 在 Visual Studio 中新建 **C# 控制台应用**，目标框架选择 `.NET 6.0 及以上` 或 `NET Framework 4.7 及以上` 。



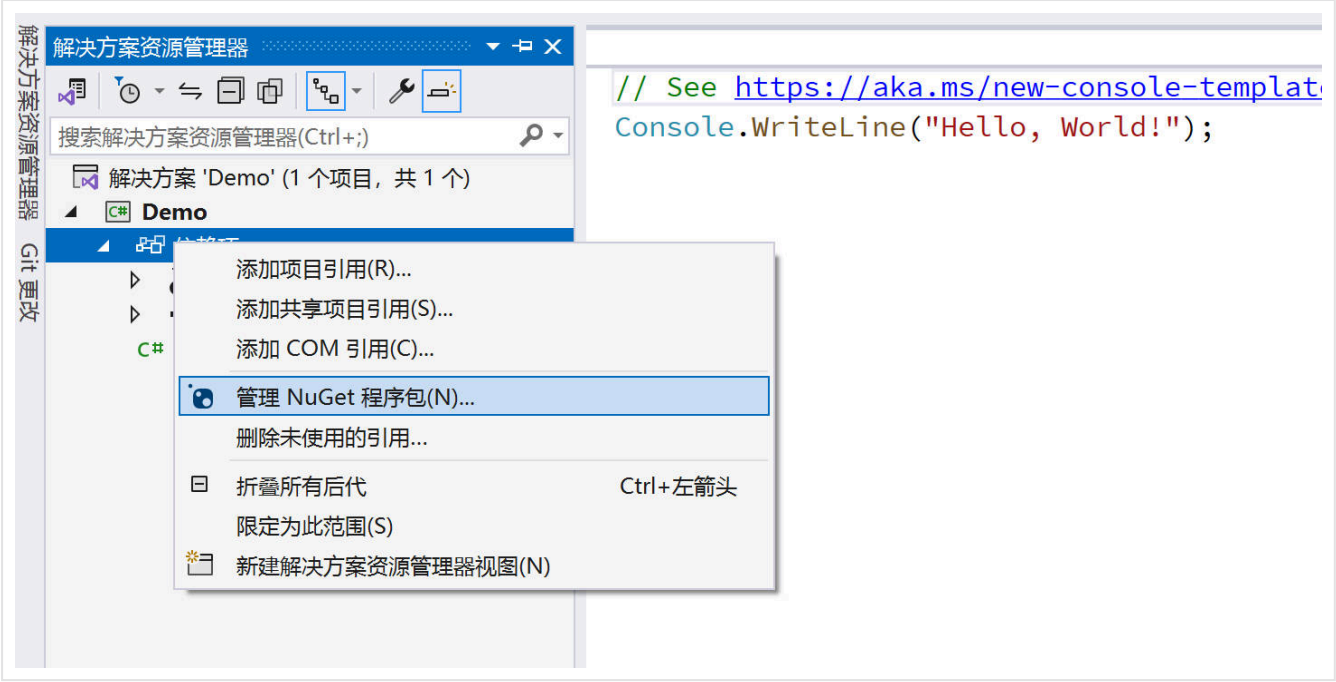
2. 进入项目属性，将目标操作系统设置为 **Windows**，目标版本选择 **7.0 或更高**，确保能够使用最新的 WinApp SDK 功能。

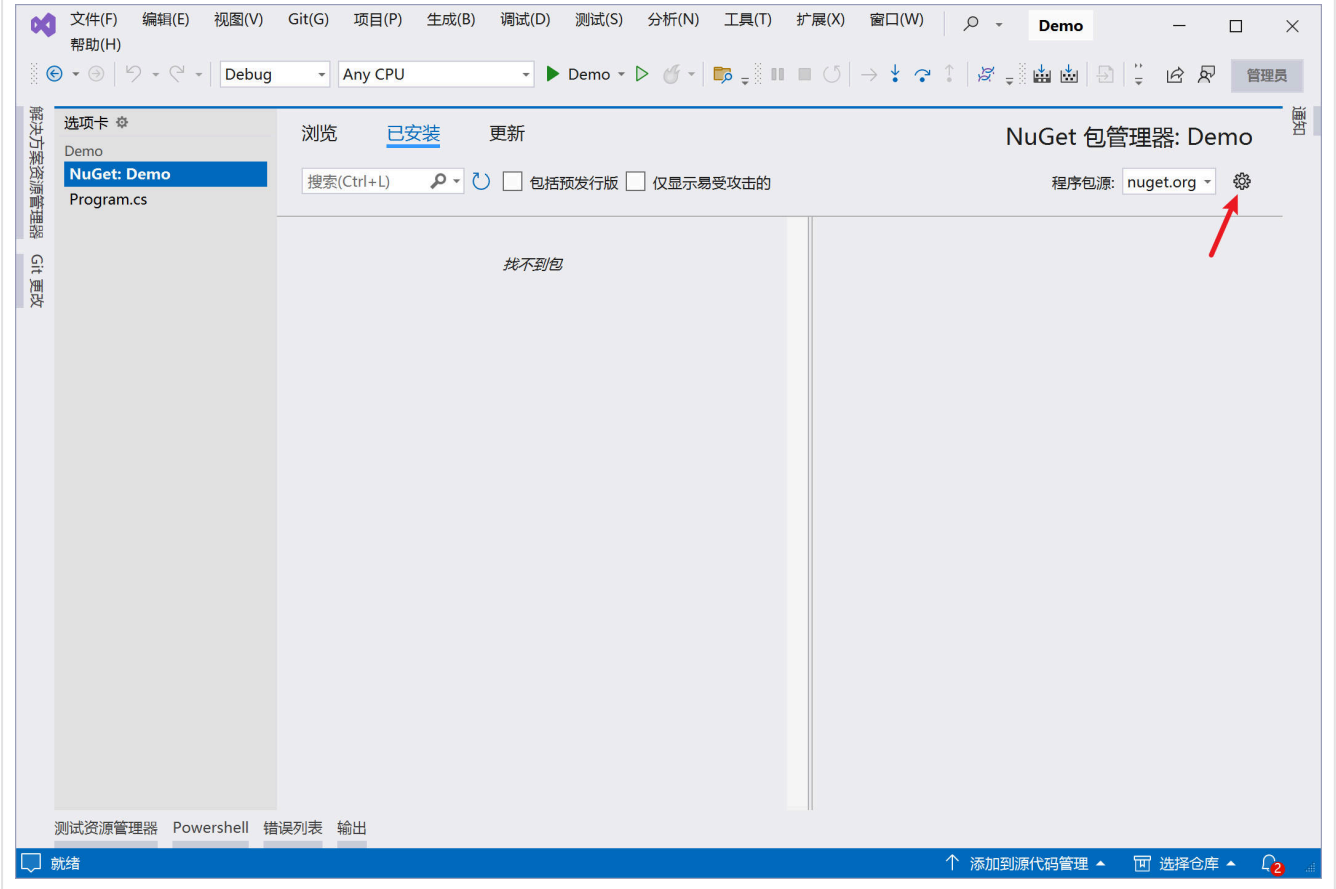


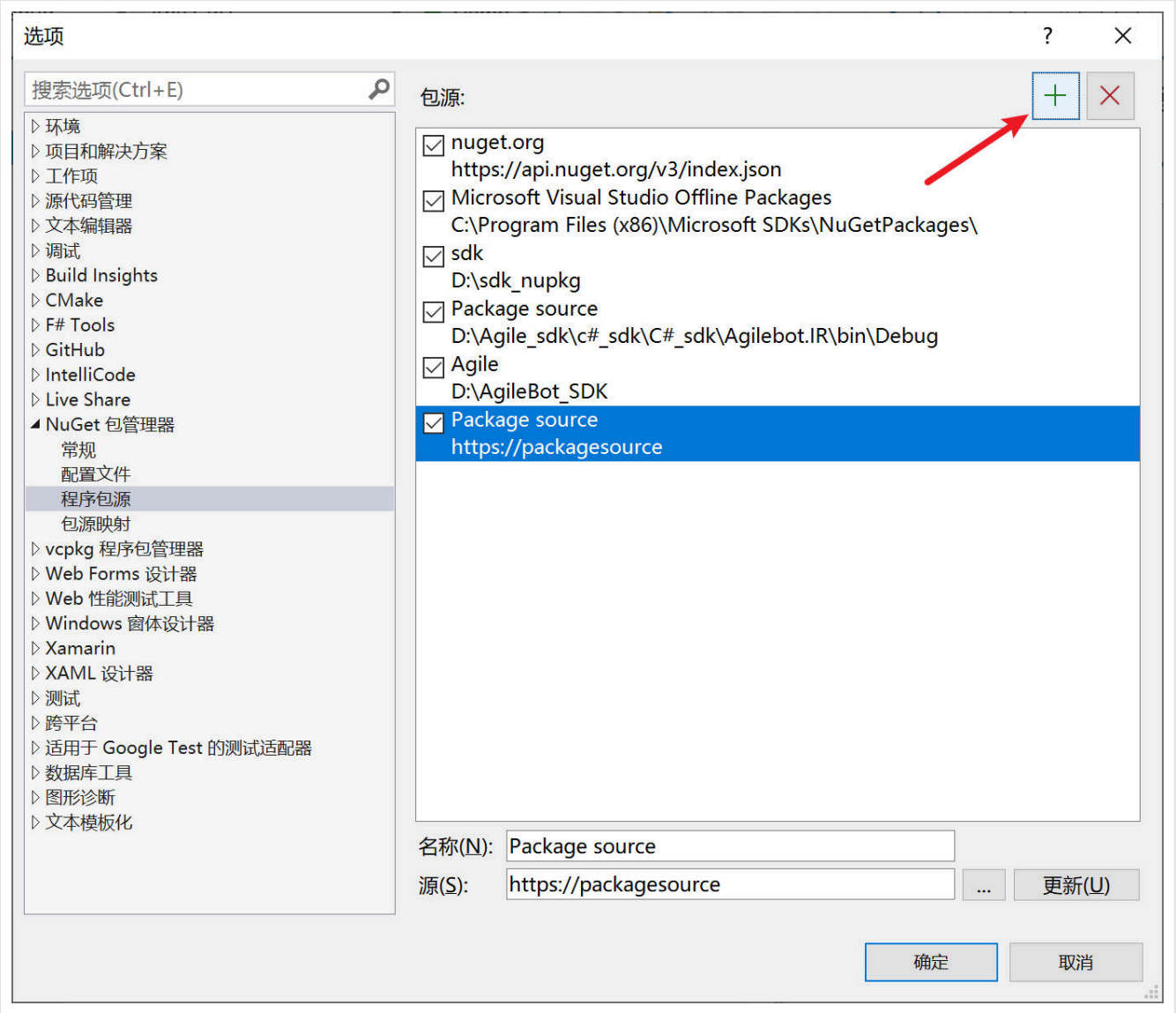


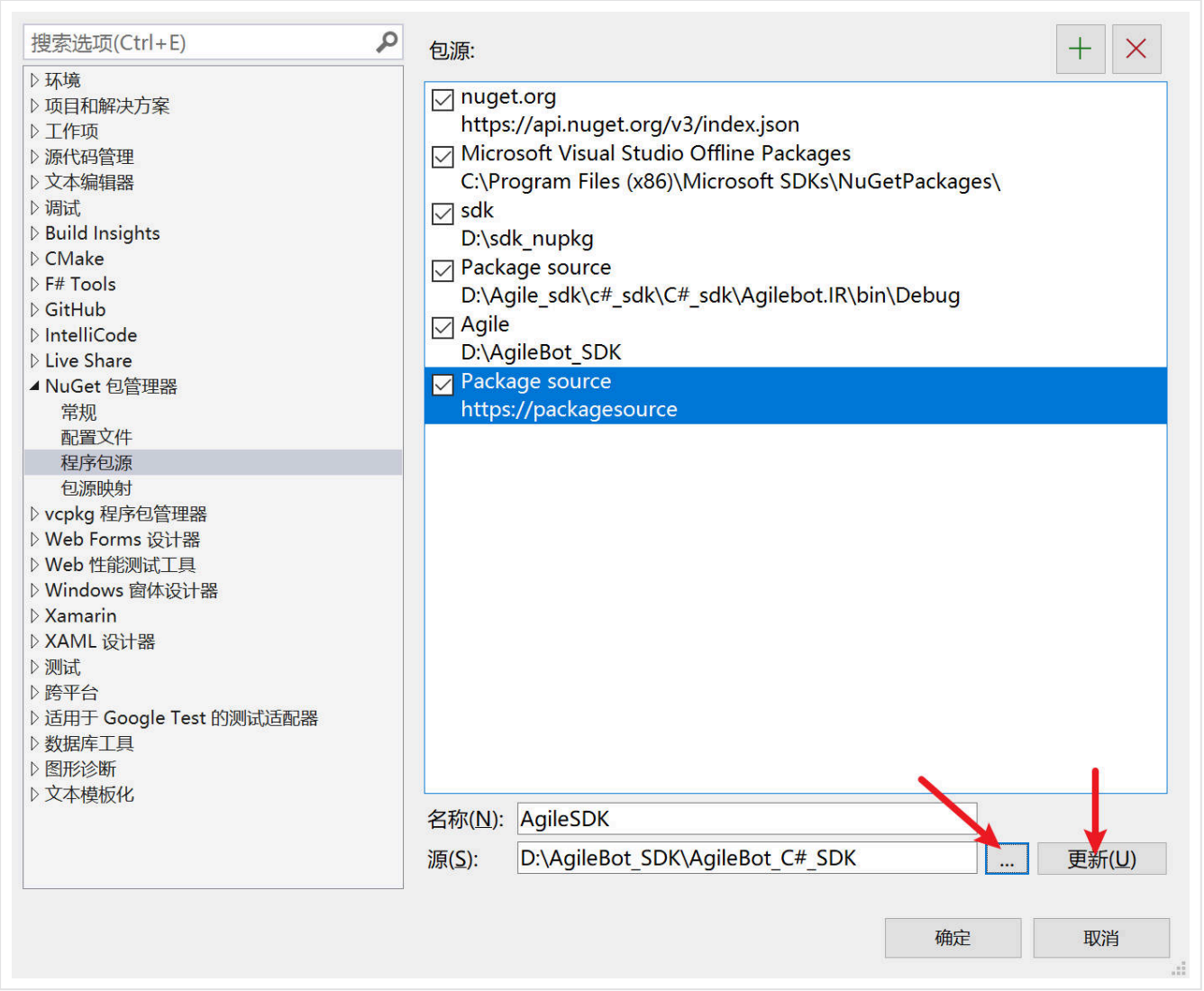


3. 打开 工具 > NuGet 包管理器 > 程序包管理器设置 ，在右上角的程序包源中添加 SDK 包所在目录。

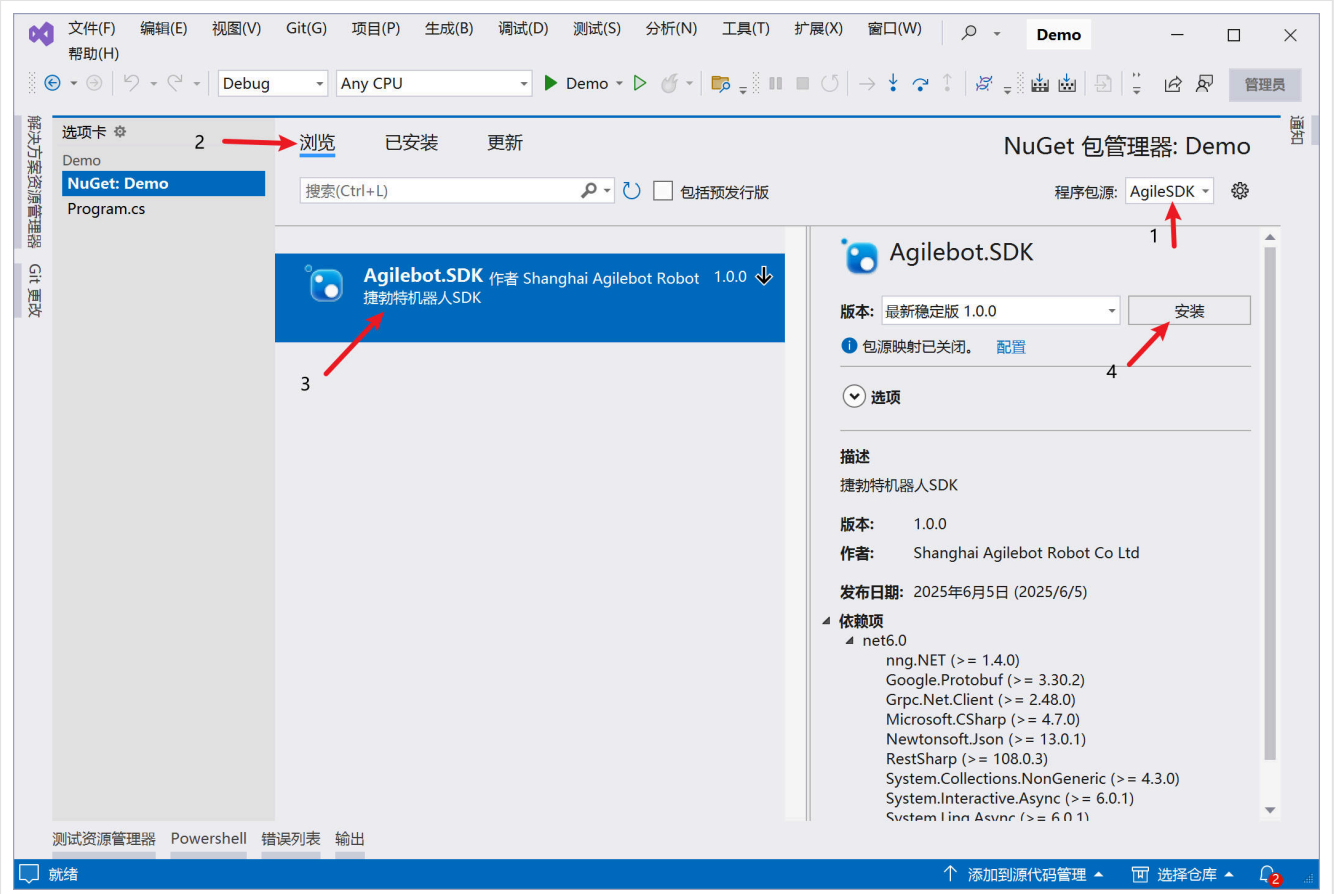








4. 返回 NuGet 程序包管理器，将程序包源切换为刚添加的目录，搜索并安装 Agilebot.SDK 包。

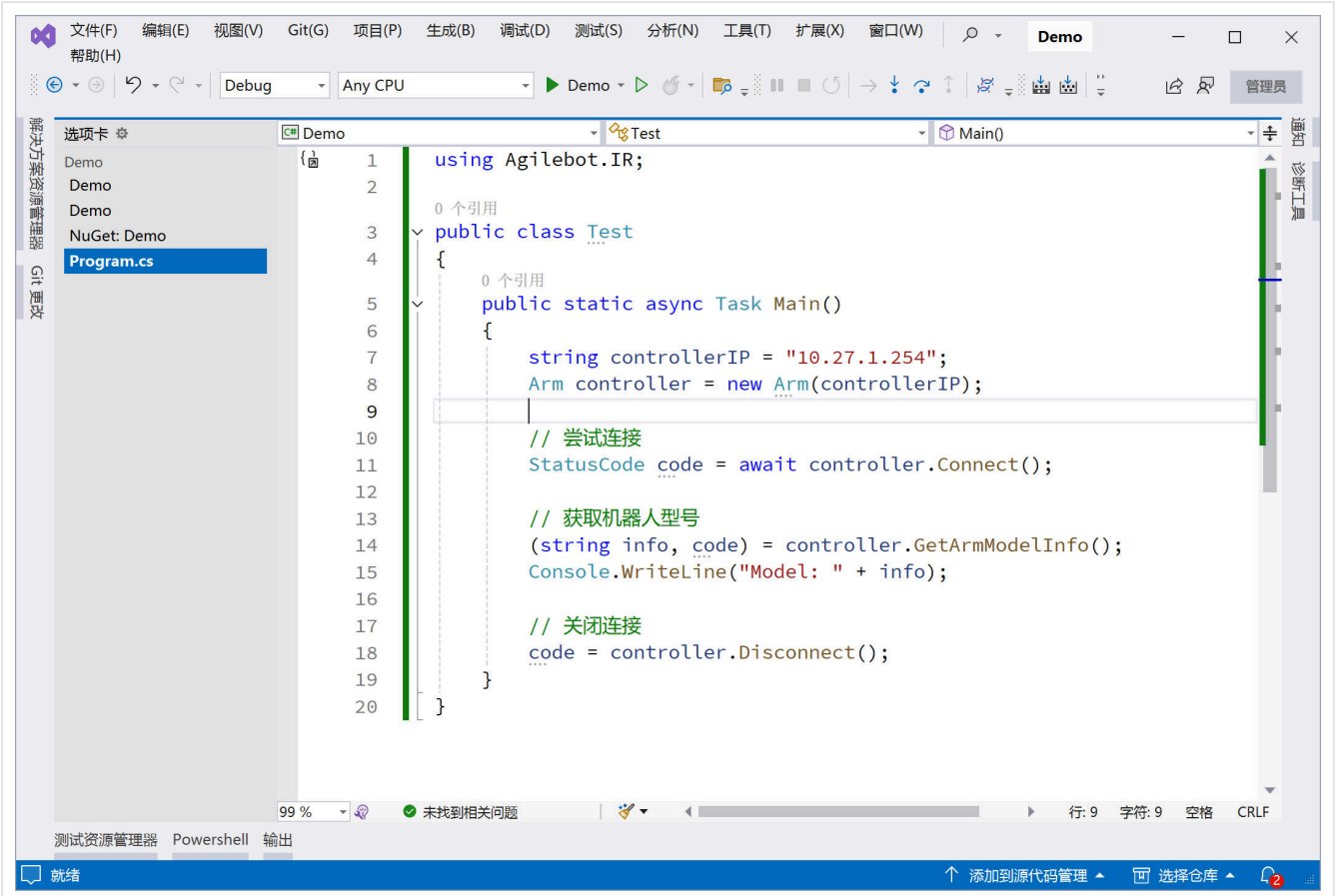


代理服务与常见问题

- 安装 SDK 后，项目会自动新增 `Tools` 文件夹，包含本地控制器代理服务所需的 `controller_proxy_service_windows_amd64.exe` 。若该文件未自动复制，可手动将其放入项目根目录与生成输出目录。
- 如程序异常退出导致代理服务残留，可在 Windows 任务管理器中终止 `controller_proxy_service_windows_amd64` 进程。
- 代理服务运行时，请勿将代理服务所在目录移动到其他位置。

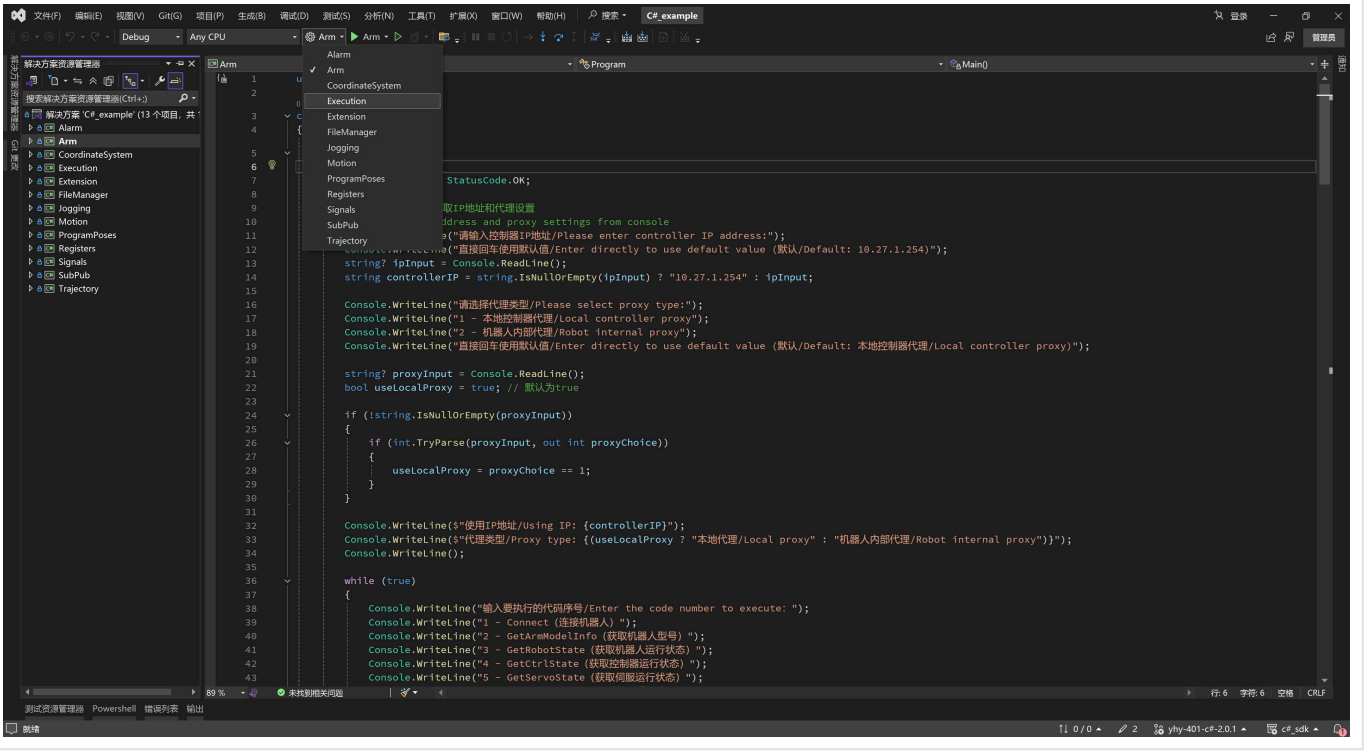
联网要求与调试

1. 开始编写并运行示例代码前，确保上位机已接入机器人网络，或与机器人位于同一局域网。
2. 调试过程中请保持网络连接稳定，避免代理服务因断网而退出。



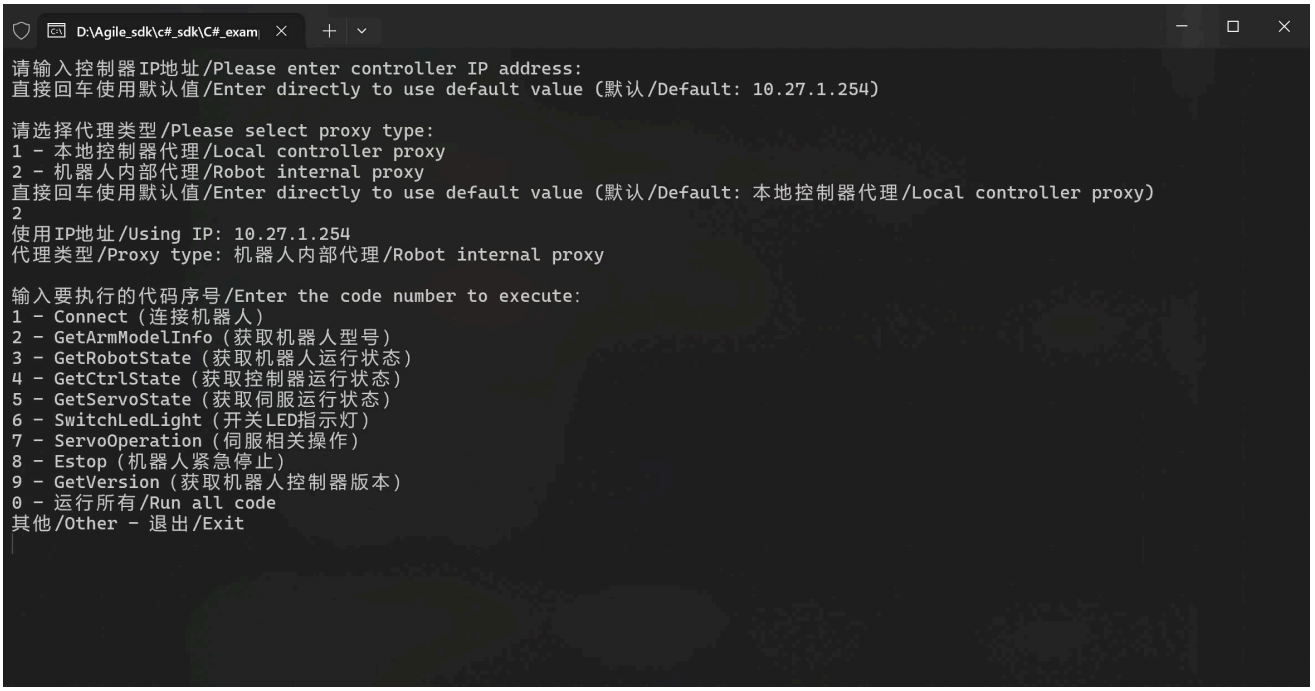
1.3 示例程序使用方法

本章示范如何使用 SDK 附带的 `C#_example` 项目，通过切换不同的启动项来快速体验主要功能类。



运行步骤

1. 打开 `C#_example` 并运行，程序会自动弹出终端窗口。
2. 按提示输入机器人 IP 地址。
3. 选择代理服务部署位置（机器人内部或本地）。
4. 点击 **“开始运行”** 即可加载对应示例。



代理类型说明

- **机器人内部代理**：使用机器人控制柜自带的代理服务，代理运行在机器人控制柜上，适合机器人软件版本不低于 v7.7.0.0 的场景。
- **本地控制器代理**：使用 SDK 自带的代理服务，代理运行在上位机，占用资源极少，适用于所有版本，机器人软件版本低于 v7.7.0.0 时只能使用本地控制器代理。
- 在 **Airbot** 上仅支持机器人内部代理（本地代理无法与 AirBot 通信），请选择对应选项以确保连接成功。

2 名词解释

名词	描述
示教器	连接在机器人上的手持设备，用于对机器人进行示教和控制
SDK	软件开发工具包，用于对机器人进行编程和控制
机器人网络	机器人与外部计算机之间的网络连接
控制器	机器人的控制单元，负责执行运动指令、处理传感器数据和管理机器人状态
机械臂	机器人的主要运动部分，由多个关节和连杆组成
伺服系统	控制机器人关节运动的电机驱动系统，提供精确的位置和速度控制
示教	通过手动操作机器人或示教器来记录机器人运动轨迹和动作的过程
关节	机器人机械臂中连接各个连杆的可动部件，每个关节对应一个自由度
笛卡尔坐标	以 X、Y、Z 三个相互垂直的轴为基准的三维坐标系统，用于描述机器人在空间中的位置和姿态
位姿	机器人在空间中的位置和姿态的组合，包括位置坐标和旋转角度
轨迹	机器人末端执行器在空间中移动的路径，通常由一系列位姿点组成
负载	机器人末端执行器所承载的重量和物体，影响机器人的运动性能和精度
坐标系	用于描述机器人位置和姿态的参考系统，包括基坐标系、工具坐标系、用户坐标系等
OVC	Overall Velocity Control，全局速度控制，用于设置机器人整体运动速度的倍率
OAC	Overall Acceleration Control，全局加速度控制，用于设置机器人整体加速度的倍率
TF	Tool Frame，工具坐标系，以机器人末端工具为原点的坐标系
UF	User Frame，用户坐标系，用户自定义的坐标系，便于编程和定位
TCS	Teach Coordinate System，示教坐标系，用于示教时的坐标参考系统
DH 参数	Denavit-Hartenberg 参数，用于描述机器人连杆几何关系的标准参数
PR 寄存器	Pose Register，位姿寄存器，用于存储机器人位姿信息的寄存器

名词	描述
MR 寄存器	Motion Register，运动寄存器，用于存储运动相关参数的寄存器
SR 寄存器	String Register，字符串寄存器，用于存储字符串信息的寄存器
R 寄存器	Real Register，实数寄存器，用于存储数值信息的寄存器
MH 寄存器	Modbus Holding Register，Modbus 保持寄存器，用于 Modbus 通信的保持寄存器
MI 寄存器	Modbus Input Register，Modbus 输入寄存器，用于 Modbus 通信的输入寄存器
DI	Digital Input，数字信号输入，用于接收外部数字信号
DO	Digital Output，数字信号输出，用于控制外部设备或执行器
BAS	Basic Script，基础脚本语言，用于编写机器人控制程序的高级编程语言
Scara	Selective Compliance Assembly Robot Arm，选择性柔顺装配机器人手臂，一种四轴工业机器人类型
协作机器人	能够与人类安全协作的机器人，通常具有力感知和碰撞检测功能
工业机器人	用于工业自动化生产的机器人，通常具有高精度、高速度和高负载能力
Copper	捷勃特协作机器人产品线的代号
Bronze	捷勃特工业机器人产品线的代号

3 数据结构

3.1 StatusCode

说明

接口返回状态码

导入

```c#
using Agilebot.IR;
```

字段

名称	枚举值	描述
OK	0	执行成功
INCOMPATIBLE_VERSION	-1	版本不兼容
TIMEOUT	-3	连接超时
INTERFACE_NOT_IMPLEMENTED	-4	接口未实现
INDEX_OUT_OF_RANGE	-5	索引越界
UNSUPPORTED_FILETYPE	-6	不支持的文件类型
UNSUPPORTED_PARAMETER	-7	不支持的机器人参数
UNSUPPORTED_SIGNALTYPE	-8	不支持的 IO 信号类型
PROGRAM_NOT_FOUND	-9	找不到程序
PROGRAM_POSE_NOT_FOUND	-10	找不到程序位姿信息

名称	枚举值	描述
WRITE_PROGRAM_FAILED	-11	更新程序位姿信息失败
GET_ALARM_CODE_FAILED	-12	访问报警服务获取报警码失败
WRONG_POSITION_INFO	-13	控制器返回错误的点位信息
UNSUPPORTED_TRA_TYPE	-14	不支持的运动类型
FILE_NOT_FOUND	-15	文件或文件夹未找到
FILE_ALREADY_EXIST	-16	文件已存在
INVALID_PARAMETER	-27	参数有误
GET_ALARM_DESC_FAILED	-17	根据报警码获取报警信息失败
RESET_ALARM_ERRORS_FAILED	-18	重置报警信息失败
GET_ALL_ALARMS_FAILED	-19	获取所有报警信息失败
WRONG_DATA_FORMAT	-20	接收的数据格式有误
CONNECT_FAILED	-21	初始化连接失败，请检查 ip 地址或控制柜服务
POSE_INDEX_DUPLICATED	-23	位姿序号重复
CONTROLLER_ERROR	-254	控制器错误，请联系开发人员
OTHER_REASON	-255	其他原因

3.2 RobotState

说明

机器人运行状态

导入

```c#
using Agilebot.IR.Types;
```

字段

名称	枚举值	描述
WRONG_DATA	-1	未知状态
ROBOT_IDLE	0	机器人空闲
ROBOT_RUNNING	1	机器人运行中
ROBOT_TEACHING	2	机器人示教中
ROBOT_IDLE_TO_RUNNING	101	机器人中间状态 空闲转换为运行
ROBOT_IDLE_TO_TEACHING	102	机器人中间状态 空闲转换为示教
ROBOT_RUNNING_TO_IDLE	103	机器人中间状态 运行转换为空闲
ROBOT_TEACHING_TO_IDLE	104	机器人中间状态 示教转换为空闲

3.3 CtrlState

说明

控制器运行状态

导入

```c#
using Agilebot.IR.Types;
```

字段

名称	枚举值	描述
WRONG_DATA	-1	未知的控制器状态
CTRL_INIT	0	控制器初始化
CTRL_ENGAGED	1	控制器使能
CTRL_ESTOP	2	控制器急停
CTRL_TERMINATED	3	控制器中止
CTRL_ANY_TO_ESTOP	101	控制器中间状态 其他转换为急停
CTRL_ESTOP_TO_ENGAGED	102	控制器中间状态 急停到使能
CTRL_ESTOP_TO_TERMINATED	103	控制器中间状态 急停到中止

3.4 ServoState

说明

伺服控制器状态

导入

```c#
using Agilebot.IR.Types;
```

字段

名称	枚举值	描述
WRONG_DATA	-1	未知的伺服控制器状态
SERVO_IDLE	1	伺服控制器空闲
SERVO_RUNNING	2	伺服控制器运行中

名称	枚举值	描述
SERVO_DISABLE	3	伺服控制器关闭
SERVO_WAIT_READY	4	伺服控制器等待就绪
SERVO_WAIT_DOWN	5	伺服控制器等待关闭
SERVO_INIT	10	伺服控制器初始化

3.5 TransformStatusEnum

说明

离线轨迹文件转换状态的枚举

导入

```c#
using Agilebot.IR.Types;
```

字段

名称	枚举值	描述
TRANSFORM_START	0	转换任务开始
TRANSFORM_RUNNING	1	转换任务执行中
TRANSFORM_SUCCESS	2	转换任务已完成
TRANSFORM_FAILED	3	转换任务失败
TRANSFORM_NOT_FOUND	4	转换任务没找到
TRANSFORM_UNKNOWN	-1	未知的转换任务状态

以下是为 PayloadInfo 、 MassCenter 和 InertiaMoment 类生成的详细说明文档：

3.6 PayloadInfo

说明

`PayloadInfo` 类用于存储机器人的负载信息，包括负载编号、重量、质心和惯性矩等参数。这些信息对于机器人在负载条件下的运动学和动力学分析至关重要，尤其是在进行路径规划和力矩计算时。

导入

```c#
using Agilebot.IR.Motion;
```

属性

属性	类型	描述
`Id`	`uint`	负载编号，用于唯一标识不同的负载配置
`Comment`	`string`	注释，用于描述负载的额外信息
`Weight`	`double`	负载的重量（单位：千克）
`MassCenter`	`MassCenter`	负载的质心位置（X、Y、Z 坐标）
`InertiaMoment`	`InertiaMoment`	负载的惯性矩（LX、LY、LZ）

示例

```c#
PayloadInfo payload = new PayloadInfo
{
    Id = 1,
    Comment = "Sample Payload",
    Weight = 5.0,
    MassCenter = new MassCenter { X = 10.0, Y = 20.0, Z = 30.0 },
    InertiaMoment = new InertiaMoment { LX = 0.1, LY = 0.2, LZ = 0.3 }
};
```

3.6.1 MassCenter

说明

`MassCenter` 类用于表示负载的质心位置，包含 X、Y 和 Z 三个坐标轴的值。质心位置是负载在空间中的几何中心，对于机器人运动控制和力矩计算非常重要。

导入

```c#
using Agilebot.IR.Motion;
```

属性

属性	类型	描述
`X`	`double`	质心的 X 坐标（单位：毫米）
`Y`	`double`	质心的 Y 坐标（单位：毫米）
`Z`	`double`	质心的 Z 坐标（单位：毫米）

3.6.2 InertiaMoment

说明

`InertiaMoment` 类用于表示负载的惯性矩，包含 LX、LY 和 LZ 三个方向的值。惯性矩是负载在旋转运动中抵抗变化的能力，对于机器人动力学分析和控制非常重要。

导入

```c#
using Agilebot.IR.Motion;
```

属性

属性	类型	描述
`LX`	`double`	惯性矩的 X 分量（单位：千克·毫米 ^{2}）

属性	类型	描述
LY	double	惯性矩的 Y 分量（单位：千克・毫米 ^{2}）
LZ	double	惯性矩的 Z 分量（单位：千克・毫米 ^{2}）

3.7 TransformState

说明

离线轨迹文件转换状态的枚举

导入

```c#
using Agilebot.IR.Types;
```

字段

枚举值	值	描述
TRANSFORM_START	0	转换任务开始
TRANSFORM_RUNNING	1	转换任务执行中
TRANSFORM_SUCCESS	2	转换任务已完成
TRANSFORM_FAILED	3	转换任务失败
TRANSFORM_NOT_FOUND	4	转换任务未找到
TRANSFORM_UNKNOWN	-1	数据错误，未知状态

3.8 TCSType

说明

TCS 坐标系类型

导入

```c#
using Agilebot.IR.Types;
```

字段

名称	枚举值	描述
WRONG_TYPE	-1	错误类型
JOINT	0	关节空间
BASE	1	基坐标系
WORLD	2	世界坐标系
USER	3	用户坐标系
TOOL	4	工具坐标系
RTCP_USER	5	RTCP 用户坐标系
RTCP_TOOL	6	RTCP 工具坐标系

3.9 MotionPose

说明

描述机器人点位的结构 坐标数据中，XYZ 方向距离数据单位为毫米 （mm），角度数据单位为 度（°），部分版本角度信息为弧度，详见功能列表返回结果说明。

导入

```c#
using Agilebot.IR.Motion;
```

属性

属性	类型	描述
CartData	BaseCartData	笛卡尔数据
Joint	Joint	关节数据
Pt	PoseType	点位类型，默认为 Unknown

示例

```c#
MotionPose motionPose = new MotionPose();
motionPose.Pt = PoseType.Cart;
motionPose.CartData.Position = new Position{
    X  = 300,
    Y = 300,
    Z = 300,
    A = 0,
    B = 0,
    C = 0
};
motionPose.CartData.Posture = new Posture{
    WristFlip = 1,
    ArmUpDown = 1,
    ArmBackFront = 1,
    ArmLeftRight = 1,
    TurnCircle = new List<int>(9){0,0,0,0,0,0,0,0,0}
};

MotionPose motionPose2 = new MotionPose();
motionPose2.Pt = PoseType.Joint;
motionPose2.Joint = new Joint{
    J1 = 0,
    J2 = 0,
    J3 = 60,
    J4 = 60,
```

```
    J5 = 0,
    J6 = 0
};
```

3.10 BaseCartData

说明

描述机器人在笛卡尔坐标系中的空间位置和姿态信息。其中，空间坐标使用毫米（mm）为单位，姿态信息包括腕部、臂部的姿态以及各个轴的回转数。

导入

```c#
using Agilebot.IR.Types;
```

属性

属性	类型	描述
Position	Position	机器人的空间坐标（X, Y, Z, A, B, C）
Posture	Posture	机器人的姿态信息（腕部、臂部姿态及轴的回转数）

示例

```c#
BaseCartData cartData = new BaseCartData();
cartData.Position.X = 100.0;
cartData.Position.Y = 200.0;
cartData.Position.Z = 300.0;
cartData.Posture.ArmUpDown = 1;
cartData.Posture.ArmBackFront = -1;
Console.WriteLine(cartData.ToString());
```

3.10.1 Position

说明

描述机器人操作点的笛卡尔坐标系空间位置及旋转角度坐标。坐标数据中，X、Y、Z 方向的距离单位为毫米（mm），A、B、C 方向的角度单位为度（°）。

导入

```c#
using Agilebot.IR.Types;
```

属性

属性	类型	描述
X	double	笛卡尔坐标系 X 方向的距离（单位：毫米）
Y	double	笛卡尔坐标系 Y 方向的距离（单位：毫米）
Z	double	笛卡尔坐标系 Z 方向的距离（单位：毫米）
A	double	笛卡尔坐标系 A 方向的角度（单位：度）
B	double	笛卡尔坐标系 B 方向的角度（单位：度）
C	double	笛卡尔坐标系 C 方向的角度（单位：度）

示例

```c#
Position position = new Position();
position.X = 100.0;
position.Y = 200.0;
position.Z = 300.0;
position.A = 45.0;
position.B = 30.0;
position.C = 60.0;
Console.WriteLine(position.ToString());
```

3.10.2 Posture

说明

描述机器人的姿态信息，包括腕部、臂部的姿态以及各个轴的回转数。姿态信息用于定义机器人在空间中的具体姿态。

导入

```c#
using Agilebot.IR.Types;
```

属性

属性	类型	描述
WristFlip	int	腕部翻转姿态，取值范围为 - 1、1。在 6 轴机器人 J5 关节配置中，值 = 1 表示腕向下翻转，值 =-1 表示腕向上翻转
ArmUpDown	int	臂部上下姿态，取值范围为 - 1、1。在 6 轴机器人 J3 关节配置中，值 = 1 表示手臂在上（前向条件下，3 轴在 4 轴到 2 轴连线上方，3 轴关节角 <0），值 =-1 表示手臂在下（前向条件下，3 轴在 4 轴到 2 轴连线下方，3 轴关节角> 0）
ArmBackFront	int	臂部前后姿态，取值范围为 - 1、1。在 6 轴机器人 J1 关节配置中，值 = 1 表示手臂在前（协作面向前方，2 轴在 1 轴左侧的状态下），值 =-1 表示手臂在后（协作面向前方，2 轴在 1 轴右侧的状态下）
ArmLeftRight	int	臂部左右姿态，取值范围为 - 1、1。在 4 轴 Scara 机器人 J2 关节配置中，值 = 1 表示 Scara 手臂在右，值 =-1 表示 Scara 手臂在左
TurnCircle	List<int>	各个轴的回转数，取值范围为 - 1、1。各轴处在 0° 姿势下，回转数为 0。执行直线、圆弧动作时，目标点回转数自动选定，可能与示教位置资料不同。各轴回转数 >=180 对应值 = 1 或更大，-179.99~179.99 对应值 = 0，<=-180 对应值 =-1 或更小

示例

```c#
Posture posture = new Posture();
posture.TurnCircle = new List<int>(9){0,0,0,0,0,0,0,0,0}
posture.WristFlip = 1;
```

```
posture.ArmUpDown = 1;
posture.ArmBackFront = -1;
posture.ArmLeftRight = 1;
Console.WriteLine(posture.ToString());
```

以下是为 `Joint` 类生成的说明文档：

3.11 Joint

说明

描述机器人各个关节的角度数据。每个关节的角度值用于定义机器人在关节空间中的具体位置。角度单位通常为度（°），但具体单位需根据实际机器人系统确认。

导入

C#

```
using Agilebot.IR.Types;
```

属性

属性	类型	描述
J1	double	机器人一轴的关节角度
J2	double	机器人二轴的关节角度
J3	double	机器人三轴的关节角度
J4	double	机器人四轴的关节角度
J5	double	机器人五轴的关节角度
J6	double	机器人六轴的关节角度
J7	double	机器人七轴的关节角度

属性	类型	描述
J8	double	机器人八轴的关节角度
J9	double	机器人九轴的关节角度

示例

```c#
Joint joint = new Joint();
joint.J1 = 45.0;
joint.J2 = 30.0;
joint.J3 = 60.0;
joint.J4 = 90.0;
joint.J5 = 120.0;
joint.J6 = 135.0;
joint.J7 = 150.0;
joint.J8 = 180.0;
joint.J9 = 225.0;
Console.WriteLine(joint.ToString());
```

注意事项

- 关节角度的单位通常为度（°），但某些机器人系统可能使用弧度（rad）。请根据具体机器人系统的文档确认单位。
- 关节角度的取值范围通常由机器人硬件限制，超出范围可能导致错误或损坏设备。

以下是为 `PoseType` 枚举生成的说明文档：

3.12 PoseType

说明

定义了机器人位姿数据的类型，用于区分数据是关节角度、笛卡尔空间坐标还是未知类型。该枚举用于标识机器人位姿数据的格式，以便在程序中正确处理不同类型的数据。

导入

```c#
using Agilebot.IR.Types;
```

枚举值

枚举值	描述
`Unknown`	未知类型，表示位姿数据类型未定义
`Joint`	关节角度数据类型，表示数据为关节角度
`Cart`	笛卡尔空间坐标数据类型，表示数据为笛卡尔坐标

以下是为 `DHparam` 类生成的说明文档：

3.13 DHparam

说明

`DHparam` 类用于描述机器人连杆的参数，基于 Denavit-Hartenberg 参数（D-H 参数）。这些参数用于定义机器人关节之间的几何关系，是机器人运动学和动力学分析的基础。

导入

```c#
using Agilebot.IR.Types;
```

属性

属性	类型	描述
`id`	`uint`	杆件的唯一标识符，用于区分不同的连杆

属性	类型	描述
a	double	杆件长度，表示相邻关节的轴向距离（单位：毫米）
alpha	double	杆件扭角，表示相邻关节轴之间的夹角（单位：度或弧度）
d	double	关节距离，表示沿当前关节轴到下一个关节的距离（单位：毫米）
offset	double	关节转角偏移量，表示关节的初始角度偏移（单位：度或弧度）

构造函数

```c#
public DHparam(uint id, double d, double a, double alpha, double offset)
```

注意事项

- **单位一致性**： `a` 和 `d` 的单位应保持一致（通常为毫米），而 `alpha` 和 `offset` 的单位也应保持一致（通常为度或弧度）。
- **角度单位**：在某些机器人系统中，角度单位可能为弧度而非度。请根据实际需求确认并统一单位。
- **D-H 参数的定义**：D-H 参数的定义依赖于具体的机器人模型和坐标系约定。在使用 `DHparam` 类时，确保参数的定义与机器人的实际几何结构一致。

以下是为 `CartStatus` 、 `JointStatus` 和 `DragStatus` 类生成的详细说明文档：

3.14 CartStatus

说明

`CartStatus` 类用于表示笛卡尔坐标系中各轴的状态。每个轴的状态用布尔值表示， `true` 表示轴可用， `false` 表示轴不可用。此状态类通常用于机器人运动控制中，以判断某个轴是否可以正常工作。

导入

```csharp
using Agilebot.IR.Types;
```

属性

属性	类型	描述
`X`	`bool`	X 方向状态，默认为 `true` （可用）
`Y`	`bool`	Y 方向状态，默认为 `true` （可用）
`Z`	`bool`	Z 方向状态，默认为 `true` （可用）
`A`	`bool`	A 方向状态，默认为 `true` （可用）
`B`	`bool`	B 方向状态，默认为 `true` （可用）
`C`	`bool`	C 方向状态，默认为 `true` （可用）

3.15 JointStatus

说明

`JointStatus` 类用于表示机械臂各关节的状态。每个关节的状态用布尔值表示， `true` 表示关节可用， `false` 表示关节不可用。此状态类通常用于机器人运动控制中，以判断某个关节是否可以正常工作。

导入

```csharp
using Agilebot.IR.Types;
```

属性

属性	类型	描述
J1	bool	关节 1 状态，默认为 true （可用）
J2	bool	关节 2 状态，默认为 true （可用）
J3	bool	关节 3 状态，默认为 true （可用）
J4	bool	关节 4 状态，默认为 true （可用）
J5	bool	关节 5 状态，默认为 true （可用）
J6	bool	关节 6 状态，默认为 true （可用）
J7	bool	关节 7 状态，默认为 true （可用）
J8	bool	关节 8 状态，默认为 true （可用）
J9	bool	关节 9 状态，默认为 true （可用）

3.16 DragStatus

说明

DragStatus 类用于表示机械臂的拖动状态，包含笛卡尔坐标系状态和关节状态。此外，还包含一个标志位 IsContinuousDrag ，用于表示是否处于连续拖动模式。此状态类通常用于机器人拖动控制中，以判断当前的拖动模式和各轴 / 关节的状态。

导入

```c#
using Agilebot.IR.Types;
```

属性

属性	类型	描述
CartStatus	CartStatus	笛卡尔坐标系状态
JointStatus	JointStatus	关节状态
IsContinuousDrag	bool	是否处于连续拖动模式，默认为 false

构造函数

```c#
public DragStatus()
```

- 初始化 CartStatus 和 JointStatus ，并设置 IsContinuousDrag 为 false 。

示例

```c#
DragStatus dragStatus = new DragStatus();
dragStatus.CartStatus.X = false; // X轴不可用
dragStatus.JointStatus.J3 = false; // 关节3不可用
dragStatus.IsContinuousDrag = true; // 设置为连续拖动模式
Console.WriteLine($"X轴状态: {dragStatus.CartStatus.X}, 关节3状态: {dragStatu
s.JointStatus.J3}, 是否连续拖动: {dragStatus.IsContinuousDrag}");
```

3.17 ProgramPose

说明

ProgramPose 类用于表示程序中的一个位姿（姿态），可以是关节坐标或笛卡尔坐标。该类包含位姿的唯一标识符、数据（关节或笛卡尔坐标信息）、名称和注释。通过此类，可以方便地管理和操作机器人程序中的位姿信息。

导入

```c#
using Agilebot.IR.Types;
```

属性

属性	类型	描述
Id	int	位姿的唯一标识符
PoseData	ProgramPoseData	位姿的数据，包括关节或笛卡尔坐标信息
Name	string	位姿的名称
Comment	string	位姿的注释

构造函数

```c#
public ProgramPose()
```

- 初始化 Id 、 PoseData 、 Name 和 Comment 。

示例

```c#
ProgramPose programPose = new ProgramPose();
programPose.Id = 1; // 设置位姿的唯一标识符
programPose.PoseData = new ProgramPoseData(); // 创建位姿数据
programPose.Name = "Pose1"; // 设置位姿名称
programPose.Comment = "这是一个示例位姿"; // 设置位姿注释
Console.WriteLine($"位姿ID: {programPose.Id}, 名称: {programPose.Name}, 注释:
{programPose.Comment}");
```

3.17.1 ProgramPoseData

说明

ProgramPoseData 类用于表示程序中的位姿数据，包含笛卡尔空间坐标和姿态信息、关节角度信息以及位姿类型。通过此类，可以方便地存储和管理位姿的具体数据。

导入

```c#
using Agilebot.IR.Types;
```

属性

属性	类型	描述
CartData	ProgramCartData	笛卡尔数据
Joint	Joint	关节数据
Pt	PoseType	点位类型，默认为 Unknown

3.17.2 ProgramCartData

说明

ProgramCartData 类用于表示程序的笛卡尔坐标系数据。它通过引用 BaseCartData 类来包含空间坐标和姿态信息，并通过 Uf 和 Tf 的值来确定采用的坐标系类型。 Uf 表示用户坐标系（User Frame）， Tf 表示工具坐标系（Tool Frame）。如果 Uf 和 Tf 的值为 -1 ，则表示使用系统的默认坐标系。此类在机器人编程中用于定义和管理笛卡尔空间中的位姿信息。

导入

```c#
using Agilebot.IR.Types;
```

属性

属性	类型	描述
BaseCart	BaseCartData	机器人的笛卡尔点位和姿态信息
Uf	int	用户坐标系（User Frame）， -1 表示使用系统的坐标系
Tf	int	工具坐标系（Tool Frame）， -1 表示使用系统的坐标系

3.18 FileType

说明

`FileType` 枚举用于定义文件上传时允许的文件类型。它通过不同的枚举值来区分机器人程序文件的来源和格式。此枚举在机器人编程环境中用于文件管理、上传和程序解析等场景，帮助系统正确识别和处理不同类型的程序文件。

导入

```c#
using Agilebot.IR.Types;
```

枚举值

枚举值	描述
`UserProgram`	用户通过点选生成的程序文件，每个程序包含 `.xml` 和 `.json` 两个文件。
`BlockProgram`	用户通过积木块生成的程序文件，每个程序包含 `.block` 、 `.xml` 和 `.json` 文件。
`TrajectoryProgram`	离线轨迹程序文件，通常用于离线编程生成的路径规划文件。

3.19 SignalType

说明

`SignalType` 枚举用于定义机器人系统中支持的信号类型。它通过不同的枚举值来区分各种数字信号和模拟信号的用途和来源。此枚举在机器人控制系统中用于信号配置、信号处理和逻辑判断等场景，帮助系统准确识别和管理不同类型的信号。

导入

```C#
using Agilebot.IR.Types;
```

枚举值

枚举值	描述
DI	数字信号输入（Digital Input），用于接收外部数字信号。
DO	数字信号输出（Digital Output），用于控制外部设备或执行器。
RI	机器人手臂数字信号输入（Robot Input），用于接收机器人手腕部分的数字信号。
RO	机器人手臂数字信号输出（Robot Output），用于控制机器人手腕部分的执行器。
UI	用户数字信号输入（User Input），用于接收用户自定义的数字信号。
UO	用户数字信号输出（User Output），用于输出用户自定义的数字信号。
TDI	工具数字信号输入（Tool Digital Input），用于接收工具端的数字信号。
TDO	工具数字信号输出（Tool Digital Output），用于控制工具端的执行器。
GI	组输入（Group Input），用于接收一组数字信号的组合输入。
GO	组输出（Group Output），用于输出一组数字信号的组合输出。
AI	模拟输入（Analog Input），用于接收连续变化的模拟信号。
AO	模拟输出（Analog Output），用于输出连续变化的模拟信号。
TAI	手腕模拟输入（Tool Analog Input），用于接收工具端的模拟信号。

3.20 PoseRegister

说明

PoseRegister 类用于表示 PR 寄存器中的位姿（姿态），可以是关节坐标或笛卡尔坐标。该类包含位姿的唯一标识符、数据（关节或笛卡尔坐标信息）、名称和注释。通过此类，可以方便地管理和操作机器人程序中的位姿信息。

导入

```c#
using Agilebot.IR.Types;
```

属性

属性	类型	描述
Id	int	位姿的唯一标识符
PoseData	PoseRegisterData	位姿的数据，包括关节或笛卡尔坐标信息
Name	string	位姿的名称
Comment	string	位姿的注释

构造函数

```c#
public PoseRegister()
```

- 初始化 Id 、 PoseData 、 Name 和 Comment 。

示例

```c#
PoseRegister pose = new PoseRegister();
pose.Id = 1; // 设置位姿的唯一标识符
pose.PoseData = new PoseRegisterData(); // 创建位姿数据
pose.Name = "Pose1"; // 设置位姿名称
pose.Comment = "这是一个示例位姿"; // 设置位姿注释
Console.WriteLine($"位姿ID: {pose.Id}, 名称: {pose.Name}, 注释: {pose.Commen
t}");
```

3.20.1 PoseRegisterData

说明

PoseRegisterData 类用于表示 PR 寄存器中的位姿数据，包含笛卡尔空间坐标和姿态信息、关节角度信息以及位姿类型。通过此类，可以方便地存储和管理位姿的具体数据。

导入

```c#
using Agilebot.IR.Types;
```

属性

属性	类型	描述
CartData	BaseCartData	笛卡尔数据
Joint	Joint	关节数据
Pt	PoseType	点位类型，默认为 Unknown

3.22 Coordinate

说明

Coordinate 类用于表示机器人系统中的一个坐标系。它包含了坐标系的基本信息，如唯一标识符（ID）、名称、备注、运动组编号以及具体的位姿数据。此类在机器人编程和控制系统中用于定义和管理坐标系的具体位置和姿态信息，便于在程序中进行运动规划和路径控制。

导入

```c#
using Agilebot.IR.Types;
```

属性

属性	类型	描述
Id	int	坐标系的唯一标识符。

属性	类型	描述
Name	string	坐标系的名称，用于标识和描述坐标系。
Comment	string	坐标系的备注信息，用于进一步说明坐标系的用途或特点。
GroupId	int	坐标系所属的运动组编号，用于分类管理坐标系。
Data	Position	坐标系的具体位姿数据，包含位置和姿态信息。

示例

```c#
// 创建一个 Coordinate 实例
Coordinate coordinate = new Coordinate
{
    Id = 1, // 设置唯一标识符
    Name = "UserCoordinate1", // 设置名称
    Comment = "这是一个用户自定义坐标系", // 设置备注
    GroupId = 1, // 设置运动组编号
    Data = new Position { X = 100, Y = 200, Z = 300, A = 45, B = 30, C = 60
} // 设置位姿数据
};
```

3.22.1 CoordinateType

说明

CoordinateType 枚举用于定义坐标系的类型。它通过不同的枚举值来区分用户坐标系和工具坐标系。此枚举在机器人编程和控制系统中用于明确指定坐标系的用途，帮助系统正确处理坐标系相关的操作。

导入

```c#
using Agilebot.IR.Types;
```

枚举值

枚举值	描述
UserCoordinate	用户坐标系，用于定义用户自定义的坐标系。
ToolCoordinate	工具坐标系，用于定义工具（如末端执行器）的坐标系。

3.22.2 CoordSummary

说明

CoordSummary 类用于表示坐标系的概要信息。它包含坐标系的类型、唯一标识符、名称、注释和组 ID 等信息。此类在机器人编程环境中用于管理和存储坐标系的元数据，便于在程序中快速访问和操作坐标系。

导入

```c#
using Agilebot.IR.Types;
```

属性

属性	类型	描述
Type	CoordinateType	坐标系类型，可以是用户坐标系或工具坐标系。
Id	int	坐标系的唯一标识符。
Name	string	坐标系的名称。
Comment	string	坐标系的注释，用于描述坐标系的用途或特点。
GroupId	int	坐标系所属的组 ID，用于分类管理坐标系。

示例

```c#
// 创建一个 CoordSummary 实例
CoordSummary coordSummary = new CoordSummary
{
    Type = CoordinateType.UserCoordinate, // 设置为用户坐标系
    Id = 1, // 设置唯一标识符
```

```
    Name = "UserCoord1", // 设置名称
    Comment = "这是一个用户自定义坐标系", // 设置注释
    GroupId = 0 // 设置组ID
};
```

3.23 RobotTopicType

说明

RobotTopicType 枚举用于 SubPub.SubscribeStatus 指定机器人状态订阅主题。

导入

```c#
using Agilebot.IR.Types;
```

属性

名称	描述
TopicCurrentJoint	发布机械臂关节状态反馈
TopicCurrentCartesian	发布 TCP 当前笛卡尔坐标
TopicUF	发布当前用户坐标系信息
TopicTF	发布当前工具坐标系信息
TopicVelocity	发布全局速度比率
TopicRunningStatus	发布控制器运行状态
TopicInterpreterStatus	发布解释器状态
TopicRobotStatus	发布机器人状态
TopicCtrlStatus	发布控制器状态

名称	描述
TopicServoStatus	发布伺服控制器状态
TopicTrajectoryRecordsStatus	轨迹复现记录状态
UserOpMode	用户操作模式

3.24 RegTopicType

说明

RegTopicType 枚举用于 SubPub.SubscribeRegister 指定寄存器订阅类型。

导入

```c#
using Agilebot.IR.Types;
```

属性

名称	描述
R	R 寄存器主题
MR	MR 寄存器主题
SR	SR 寄存器主题
PR	PR 寄存器主题

3.25 IOTopicType

说明

`IOTopicType` 枚举用于 `SubPub.SubscribeIO` 指定 IO 订阅类型。

导入

```c#
using Agilebot.IR.Types;
```

属性

名称	描述
DI	数字量输入主题
DO	数字量输出主题
GI	组数字量输入主题
GO	组数字量输出主题
RI	机器人输入主题
RO	机器人输出主题
UI	用户输入主题
UO	用户输出主题
TDI	工具数字量输入主题
TDO	工具数字量输出主题
TAI	工具模拟量输入主题
AI	模拟量输入主题
AO	模拟量输出主题

4 方法与示例

4.1 机器人基础操作

概述

Arm 类封装了绝大多数与捷勃特机器人相关的高频接口，负责完成连接管理、状态查询、控制指令发送等核心能力。典型使用流程：

1. 实例化 `Arm(controllerIP, teachPanelIP, localProxy)` 构造函数
2. 调用 `ConnectSync()` 建立与控制器 / 示教器的通信
3. 根据业务调用运动、状态、I/O 等接口
4. 最后调用 `DisconnectSync()` 释放资源

实例化后无需手动加载配置，类会在内部完成：

- SDK 版本检查
- 控制器类型识别
- 代理服务的自动选择
- 在日志中提示所使用的通信链路

注意事项

- 当机器人软件版本低于 7.7.0 时，请确保 `Arm` 类实例化时 `localProxy=true` ，且 PC 具备本地通信能力。
- 与工业机器人配合时，若需要保持 PC 模式：
  - 连接后需要确保已获得示教器控制权限（具体方式请参考机器人操作手册）
  - 操作完成后及时释放权限
  - 避免示教器控制权限被长时间占用

类构造函数

方法名	Arm(string `controllerIP` , string `teachPanelIP` = null, bool `localProxy` = true)
描述	捷勃特机器人类构造函数，包含所有可用的机器人控制接口。需要先初始化并连接机器人后才能使用其他功能
请求参数	`controllerIP` : string 机器人控制器 IP 地址 `teachPanelIP` : string 可选，示教器侧 IP；不提供时回退到 `controllerIP` `localProxy` : bool 是否使用本地控制器代理服务，默认为 true。当为 true 时将在本地启动控制器代理服务；当为 false 时则需要机器人控制器中已安装代理服务（需要机器人软件版本 7.7 及以上）
返回值	StatusCode: 构造函数执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.1.1 连接机器人

方法名	ConnectSync()
描述	建立与捷勃特机器人的网络连接。必须先调用 `Arm` 构造函数初始化机器人实例。该方法会根据构造函数中指定的代理模式启动相应的代理服务。
请求参数	无参数
返回值	StatusCode: 连接操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.1.2 判断与机械臂的连接是否有效

方法名	IsConnected()
描述	检查与机器人的网络连接状态是否有效。返回 `true` 表示连接有效，可以正常通信；返回 `false` 表示连接已断开或未建立。

方法名	IsConnected()
请求参数	无参数
返回值	bool: 连接状态，true 表示连接有效，false 表示连接失效或未连接
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.1.3 与机器人断开连接

方法名	DisconnectSync()
描述	断开与捷勃特机器人的网络连接，释放相关资源。断开后需要重新调用 `ConnectSync()` 才能再次通信。
请求参数	无参数
返回值	StatusCode: 断开连接操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Arm/Connect.cs

```cs
using Agilebot.IR;

public class Connect
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
```

```
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接到捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        if (code != StatusCode.OK)
        {
            Console.WriteLine(
                "Connect Robot Failed: "
                    + code.GetDescription()
            );
            return code;
        }

        try
        {
            // [ZH] 检查连接状态
            // [EN] Check the connection status
            var state = controller.IsConnected();
            Console.WriteLine("Connected: " + state);
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
```

```
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.1.4 获取当前机器人型号

方法名	GetArmModelInfo()
描述	获取当前连接的捷勃特机器人型号信息。返回机器人型号字符串（例如 “GBT-C5A”）和操作执行状态。
请求参数	无参数
返回值	string: 机器人型号字符串，例如 "GBT-C5A" StatusCode: 获取操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Arm/GetArmModelInfo.cs

```cs
using Agilebot.IR;

public class GetArmModelInfo
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
```

```
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接到捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        if (code != StatusCode.OK)
        {
            Console.WriteLine(
                "Connect Robot Failed: "
                    + code.GetDescription()
            );
            return code;
        }

        try
        {
            // [ZH] 获取机器人型号信息
            // [EN] Get the robot model information
            (string info, code) =
                controller.GetArmModelInfo();
            if (code != StatusCode.OK)
            {
                Console.WriteLine(
                    "Get Robot Model Failed: "
                        + code.GetDescription()
                );
            }
            else
            {
                Console.WriteLine("Model: " + info);
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
```

```
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.1.5 获取机器人运行状态

方法名	GetRobotState()
描述	获取捷勃特机器人的当前运行状态。返回机器人运行状态枚举值和操作执行状态。
请求参数	无参数
返回值	RobotState: 机器人运行状态枚举值 StatusCode: 获取操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Arm/GetRobotState.cs

cs

```cs
using Agilebot.IR;
using Agilebot.IR.Types;

public class GetRobotState
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        if (code != StatusCode.OK)
        {
            Console.WriteLine(
                "Connect Robot Failed: "
                    + code.GetDescription()
            );
            return code;
        }

        try
        {
            // [ZH] 获取机器人运行状态
            // [EN] Get the robot running state
            (RobotState state, code) =
                controller.GetRobotState();
            if (code != StatusCode.OK)
            {
                Console.WriteLine(
                    "Get RobotState Failed: "
```

```
                        + code.GetDescription()
                );
            }
            else
            {
                Console.WriteLine("RobotState: " + state);
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.1.6 获取当前控制器运行状态

方法名	GetCtrlState()
描述	获取捷勃特机器人控制器的当前运行状态。返回控制器运行状态枚举值和操作执行状态。
请求参数	无参数
返回值	CtrlState: 控制器运行状态枚举值 StatusCode: 获取操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Arm/GetCtrlState.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Types;

public class GetCtrlState
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        if (code != StatusCode.OK)
        {
            Console.WriteLine(
                "Connect Robot Failed: "
```

```
                    + code.GetDescription()
            );
            return code;
        }

        try
        {
            // [ZH] 获取控制器运行状态
            // [EN] Get the controller running state
            (CtrlState state, code) =
                controller.GetCtrlState();
            if (code != StatusCode.OK)
            {
                Console.WriteLine(
                    "Get CtrlState Failed: "
                        + code.GetDescription()
                );
            }
            else
            {
                Console.WriteLine("CtrlState: " + state);
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
```

```
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.1.7 获取当前伺服状态

方法名	GetServoState()
描述	获取捷勃特机器人伺服系统的当前状态。返回伺服系统状态枚举值和操作执行状态。
请求参数	无参数
返回值	ServoState: 伺服系统状态枚举值 StatusCode: 获取操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Arm/GetServoState.cs

cs

```cs
using Agilebot.IR;
using Agilebot.IR.Types;

public class GetServoState
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
```

```
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        if (code != StatusCode.OK)
        {
            Console.WriteLine(
                "Connect Robot Failed: "
                    + code.GetDescription()
            );
            return code;
        }

        try
        {
            // [ZH] 获取伺服运行状态
            // [EN] Get the servo operating state
            (ServoState state, code) =
                controller.GetServoState();
            if (code != StatusCode.OK)
            {
                Console.WriteLine(
                    "Get ServoState Failed: "
                        + code.GetDescription()
                );
            }
            else
            {
                Console.WriteLine("ServoState: " + state);
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
```

```
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.1.8 获取机器人控制器版本

方法名	GetVersion()
描述	获取捷勃特机器人控制器的软件版本信息。返回控制器软件版本字符串和操作执行状态。
请求参数	无参数
返回值	string: 控制器软件版本字符串 StatusCode: 获取操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Arm/GetVersion.cs

cs

```cs
using Agilebot.IR;

public class GetVersion
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        if (code != StatusCode.OK)
        {
            Console.WriteLine(
                "Connect Robot Failed: "
                    + code.GetDescription()
            );
            return code;
        }

        try
        {
            // [ZH] 获取机器人控制器版本
            // [EN] Get the robot controller version
            string version;
            (version, code) = controller.GetVersion();
            if (code != StatusCode.OK)
            {
                Console.WriteLine(
                    "Get version Failed: "
                        + code.GetDescription()
```

```
                );
            }
            else
            {
                Console.WriteLine("Version: " + version);
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.1.9 设置机器人的 LED 指示灯

方法名	SwitchLedLight(bool `mode`)
描述	控制捷勃特机器人 LED 指示灯的开关状态。 `true` 开启指示灯， `false` 关闭指示灯。
请求参数	`mode` : bool LED 指示灯控制模式，true 表示开启，false 表示关闭
返回值	StatusCode: 操作执行结果
兼容性	仅支持协作机器人，需要控制器版本 1.3.6 及以上，工业机器人不支持
兼容的机器人软件版本	协作 (Copper): v7.5.1.3+ 工业 (Bronze): 不支持

示例代码

Arm/SwitchLedLight.cs

```cs
using Agilebot.IR;

public class SwitchLedLight
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        if (code != StatusCode.OK)
        {
            Console.WriteLine(
                "Connect Robot Failed: "
                    + code.GetDescription()
```

```
        );
        return code;
    }

    try
    {
        // [ZH] 关闭灯光
        // [EN] Turn off the LED light
        code = controller.SwitchLedLight(false);
        if (code != StatusCode.OK)
        {
            Console.WriteLine(
                "Switch Led Failed: "
                    + code.GetDescription()
            );
        }
        else
        {
            Console.WriteLine("Switch Led Light Off.");
        }

        Thread.Sleep(2000);

        // [ZH] 打开灯光
        // [EN] Turn on the LED light
        code = controller.SwitchLedLight(true);
        if (code != StatusCode.OK)
        {
            Console.WriteLine(
                "Switch Led Failed: "
                    + code.GetDescription()
            );
        }
        else
        {
            Console.WriteLine("Switch Led Light On.");
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(
            $"执行过程中发生异常/Exception occurred during execution: {ex.M
```

```
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Disconnect from the robot
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.1.10 机器人伺服启动

方法名	ServoOn()
描述	启动捷勃特机器人的伺服系统，使机器人进入可控制状态。伺服启动后，机器人可以接受运动指令。
请求参数	无参数
返回值	StatusCode: 伺服启动操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.1.11 机器人伺服关闭

方法名	ServoOff()
描述	关闭捷勃特机器人的伺服系统，使机器人进入安全停止状态。伺服关闭后，机器人将无法运动。
请求参数	无参数
返回值	StatusCode: 伺服关闭操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.1.12 让机器人伺服重置

方法名	ServoReset()
描述	重置捷勃特机器人的伺服系统，清除错误状态并准备重新启动。通常在伺服故障后调用此方法以恢复系统。
请求参数	无参数
返回值	StatusCode: 伺服重置操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Arm/ServoOperation.cs

```cs
using Agilebot.IR;

public class ServoOperation
{
    public static StatusCode Run(
        string controllerIP,
```

```
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        if (code != StatusCode.OK)
        {
            Console.WriteLine(
                "Connect Robot Failed: "
                    + code.GetDescription()
            );
            return code;
        }

        try
        {
            // [ZH] 机械臂伺服重置
            // [EN] Reset the robot arm servo
            code = controller.ServoReset();
            if (code != StatusCode.OK)
            {
                Console.WriteLine(
                    "Servo Reset Failed: "
                        + code.GetDescription()
                );
            }
            else
            {
                Console.WriteLine("Servo Reset Success.");
            }

            Thread.Sleep(3000);

            // [ZH] 机械臂伺服关闭
```

```
            // [EN] Turn off the robot arm servo
            code = controller.ServoOff();
            if (code != StatusCode.OK)
            {
                Console.WriteLine(
                    "Servo Off Failed: "
                        + code.GetDescription()
                );
            }
            else
            {
                Console.WriteLine("Servo Off Success.");
            }

            Thread.Sleep(3000);

            // [ZH] 机械臂伺服打开
            // [EN] Turn on the robot arm servo
            code = controller.ServoOn();
            if (code != StatusCode.OK)
            {
                Console.WriteLine(
                    "Servo On Failed: "
                        + code.GetDescription()
                );
            }
            else
            {
                Console.WriteLine("Servo On Success.");
            }

            Thread.Sleep(3000);
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
```

```
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.1.13 机器人紧急停止

方法名	Estop()
描述	执行捷勃特机器人的紧急停止，立即停止所有运动并进入安全状态。紧急停止后，需要重新启动伺服系统才能恢复运动。
请求参数	无参数
返回值	StatusCode: 紧急停止操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Arm/Estop.cs

```cs
using Agilebot.IR;

public class Estop
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接到捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        if (code != StatusCode.OK)
        {
            Console.WriteLine(
                "Connect Robot Failed: "
                    + code.GetDescription()
            );
            return code;
        }

        try
        {
            // [ZH] 触发机器人急停
            // [EN] Trigger the robot emergency stop
            code = controller.Estop();
            if (code != StatusCode.OK)
            {
                Console.WriteLine(
                    "Emergency Stop Failed: "
                        + code.GetDescription()
                );
            }
            else
```

```
            {
                Console.WriteLine("Emergency Stop Success");
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.2 机器人运动控制和状态

概述

Motion 类是捷勃特机器人运动控制的核心对象，负责封装以下核心功能：

- 速度 / 加速度参数管理
- 坐标系管理
- 点位转换
- 轨迹运动控制
- 拖动示教
- 实时控制
- 负载管理

常见使用流程

在 `Arm` 完成连接后，通过 `arm.Motion` 获取 Motion 实例，无需单独初始化。

4.2.1 获取机器人参数

4.2.1.1 获取 OVC 全局速度比率

方法名	Motion.GetOVC()
描述	获取当前机器人的 OVC（Overall Velocity Control）全局速度比率，比率范围为 0~1
请求参数	无
返回值	double: 速度比率，结果范围为 0~1 StatusCode: 获取操作执行结果

方法名	Motion.GetOVC()
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.2.1.2 获取 OAC 全局加速度比率

方法名	Motion.GetOAC()
描述	获取当前机器人的 OAC（Overall Acceleration Control）全局加速度比率，比率范围为 0~1.2
请求参数	无
返回值	double: 加速度比率，结果范围为 0~1.2 StatusCode: 获取操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.2.1.3 获取当前使用的 TF 工具坐标系编号

方法名	Motion.GetTF()
描述	获取当前机器人使用的 TF（Tool Frame）工具坐标系编号，序号范围为 0~10
请求参数	无
返回值	int: 工具坐标系编号 StatusCode: 获取操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.2.1.4 获取当前使用的 UF 用户坐标系编号

方法名	Motion.GetUF()
描述	获取当前机器人使用的 UF（User Frame）用户坐标系编号，序号范围为 0~10
请求参数	无

方法名	Motion.GetUF()
返回值	int: 用户坐标系编号 StatusCode: 获取操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.2.1.5 获取当前使用的 TCS 示教坐标系

方法名	Motion.GetTCS()
描述	获取当前机器人使用的 TCS（Teach Coordinate System）示教坐标系，具体参见 TCSType
请求参数	无
返回值	TCSType: 示教坐标系类型 StatusCode: 获取操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Motion/GetMotionParameters.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Types;

public class GetMotionParameters
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
```

```
    useLocalProxy
);

// [ZH] 连接捷勃特机器人
// [EN] Connect to the Agilebot robot
StatusCode code = controller.ConnectSync();
Console.WriteLine(
    code != StatusCode.OK
        ? code.GetDescription()
        : "连接成功/Successfully connected."
);

if (code != StatusCode.OK)
{
    return code;
}

try
{
    // [ZH] 获取 OVC 全局速度比率
    // [EN] Get OVC global speed ratio
    double ovc;
    (ovc, code) = controller.Motion.GetOVC();
    if (code == StatusCode.OK)
    {
        Console.WriteLine($"OVC = {ovc}");
    }
    else
    {
        Console.WriteLine(
            $"获取OVC失败: {code.GetDescription()}"
        );
    }

    // [ZH] 获取 OAC 全局加速度比率
    // [EN] Get OAC global acceleration ratio
    double oac;
    (oac, code) = controller.Motion.GetOAC();
    if (code == StatusCode.OK)
    {
        Console.WriteLine($"OAC = {oac}");
    }
```

```
else
{
    Console.WriteLine(
        $"获取OAC失败: {code.GetDescription()}"
    );
}

// [ZH] 获取当前使用的 TF
// [EN] Get current TF (Tool Frame)
int tf;
(tf, code) = controller.Motion.GetTF();
if (code == StatusCode.OK)
{
    Console.WriteLine($"TF = {tf}");
}
else
{
    Console.WriteLine(
        $"获取TF失败: {code.GetDescription()}"
    );
}

// [ZH] 获取当前使用的 UF
// [EN] Get current UF (User Frame)
int uf;
(uf, code) = controller.Motion.GetUF();
if (code == StatusCode.OK)
{
    Console.WriteLine($"UF = {uf}");
}
else
{
    Console.WriteLine(
        $"获取UF失败: {code.GetDescription()}"
    );
}

// [ZH] 获取当前使用的 TCS 示教坐标系
// [EN] Get current TCS teaching coordinate system
TCSType tcs;
(tcs, code) = controller.Motion.GetTCS();
if (code == StatusCode.OK)
```

```
            {
                Console.WriteLine($"TCSType = {tcs}");
            }
            else
            {
                Console.WriteLine(
                    $"获取TCS失败: {code.GetDescription()}"
                );
            }

            // [ZH] 获取机器人软限位
            // [EN] Get robot soft limits
            List<List<double>> softLimit;
            (softLimit, code) =
                controller.Motion.GetUserSoftLimit();
            if (code == StatusCode.OK)
            {
                Console.WriteLine("软限位信息:");
                for (int i = 0; i < softLimit.Count; i++)
                {
                    Console.WriteLine(
                        $"轴{i + 1}: 下限={softLimit[i][0]}, 上限={softLimit
[i][1]}"
                    );
                }
            }
            else
            {
                Console.WriteLine(
                    $"获取软限位失败: {code.GetDescription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
```

```
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            Console.WriteLine(
                disconnectCode != StatusCode.OK
                    ? disconnectCode.GetDescription()
                    : "Successfully disconnected."
            );
        }

        return code;
    }
}
```

4.2.2 设置机器人参数

4.2.2.1 设置 OVC 全局速度比率

方法名	Motion.SetOVC(double `value`)
描述	设置机器人的 OVC 全局速度比率
请求参数	`value` : double 速度比率，范围为 0~1
返回值	StatusCode: 设置操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.2.2.2 设置 OAC 全局加速度比率

方法名	Motion.SetOAC(double `value`)
描述	设置机器人的 OAC 全局加速度比率
请求参数	`value` : double 加速度比率，范围为 0.01~1.2

方法名	Motion.SetOAC(double `value`)
返回值	StatusCode: 设置操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.2.2.3 设置当前使用的 TF 工具坐标系

方法名	Motion.SetTF(int `value`)
描述	设置机器人当前使用的 TF 工具坐标系
请求参数	`value` : int 工具坐标系编号，范围为 0~10
返回值	StatusCode: 设置操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.2.2.4 设置当前使用的 UF 用户坐标系

方法名	Motion.SetUF(int `value`)
描述	设置机器人当前使用的 UF 用户坐标系
请求参数	`value` : int 用户坐标系编号，范围为 0~10
返回值	StatusCode: 设置操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.2.2.5 设置当前使用的 TCS 示教坐标系

方法名	Motion.SetTCS(TCSType `value`)
描述	设置机器人当前使用的 TCS 示教坐标系，具体参见 TCSType
请求参数	`value` : TCSType TCS 示教坐标系类型

方法名	Motion.SetTCS(TCSType value)
返回值	StatusCode: 设置操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Motion/SetMotionParameters.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Types;

public class SetMotionParameters
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }
```

```
        try
        {
            // [ZH] 设置 OVC 全局速度比率
            // [EN] Set OVC global speed ratio
            code = controller.Motion.SetOVC(0.5);
            if (code == StatusCode.OK)
            {
                Console.WriteLine("设置OVC成功");
            }
            else
            {
                Console.WriteLine(
                    $"设置OVC失败: {code.GetDescription()}"
                );
            }

            // [ZH] 设置 OAC 全局加速度比率
            // [EN] Set OAC global acceleration ratio
            code = controller.Motion.SetOAC(0.8);
            if (code == StatusCode.OK)
            {
                Console.WriteLine("设置OAC成功");
            }
            else
            {
                Console.WriteLine(
                    $"设置OAC失败: {code.GetDescription()}"
                );
            }

            // [ZH] 设置当前使用的 TF 用户坐标系编号
            // [EN] Set current TF (Tool Frame) user coordinate system numbe
r
            code = controller.Motion.SetTF(2);
            if (code == StatusCode.OK)
            {
                Console.WriteLine("设置TF成功");
            }
            else
            {
                Console.WriteLine(
```

```
                $"设置TF失败: {code.GetDescription()}"
            );
        }

        // [ZH] 设置当前使用的 UF 工具坐标系编号
        // [EN] Set current UF (User Frame) tool coordinate system numbe
r
        code = controller.Motion.SetUF(1);
        if (code == StatusCode.OK)
        {
            Console.WriteLine("设置UF成功");
        }
        else
        {
            Console.WriteLine(
                $"设置UF失败: {code.GetDescription()}"
            );
        }

        // [ZH] 设置当前使用的 TCS 示教坐标系
        // [EN] Set current TCS teaching coordinate system
        code = controller.Motion.SetTCS(TCSType.TOOL);
        if (code == StatusCode.OK)
        {
            Console.WriteLine("设置TCS成功");
        }
        else
        {
            Console.WriteLine(
                $"设置TCS失败: {code.GetDescription()}"
            );
        }

        // [ZH] 设置UDP位置控制的相关参数
        // [EN] Set UDP position control related parameters
        code =
            controller.Motion.SetPositionTrajectoryParams(
                10,
                20,
                10,
                10
            );
```

```
            if (code == StatusCode.OK)
            {
                Console.WriteLine("设置位置控制参数成功");
            }
            else
            {
                Console.WriteLine(
                    $"设置位置控制参数失败: {code.GetDescription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            Console.WriteLine(
                disconnectCode != StatusCode.OK
                    ? disconnectCode.GetDescription()
                    : "Successfully disconnected."
            );
        }

        return code;
    }
}
```

4.2.3 将笛卡尔点位转换成关节值点位

方法名	**Motion.ConvertCartToJoint(MotionPose `pose` , int `ufIndex` = 0, int `tfIndex` = 0)**
描述	将位姿数据从笛卡尔点位转换成关节值点位表达
请求参数	`pose` : MotionPose 机器人的笛卡尔位姿 (PoseType.CART；未指定 posture 时 SDK 自动求解可行姿态) `ufIndex` : int 用户坐标系索引，默认为 0 `tfIndex` : int 工具坐标系索引，默认为 0
返回值	MotionPose: 转换后的机器人位姿数据 StatusCode: 转换操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Motion/ConvertCartToJoint.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Motion;
using Agilebot.IR.Types;

public class ConvertCartToJoint
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
```

```
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            // [ZH] 创建笛卡尔位姿
            // [EN] Create Cartesian pose
            MotionPose motionPose = new MotionPose();
            motionPose.Pt = PoseType.Cart;
            motionPose.CartData.Position = new Position
            {
                X = 300,
                Y = 300,
                Z = 300,
                A = 0,
                B = 0,
                C = 0,
            };
            motionPose.CartData.Posture = new Posture
            {
                WristFlip = 1,
                ArmUpDown = 1,
                ArmBackFront = 1,
                ArmLeftRight = 1,
                TurnCircle = new List<int>(9)
                {
                    0,
                    0,
                    0,
                    0,
                    0,
                    0,
                    0,
                    0,
```

```
                    0,
                },
            };

            // [ZH] 将笛卡尔点位转换成关节值点位
            // [EN] Convert Cartesian pose to joint pose
            MotionPose convertPose;
            (convertPose, code) =
                controller.Motion.ConvertCartToJoint(
                    motionPose
                );
            if (code == StatusCode.OK)
            {
                Console.WriteLine("笛卡尔转关节成功:");
                Console.WriteLine(
                    $"关节值: J1={convertPose.Joint.J1}, J2={convertPose.Join
t.J2}, J3={convertPose.Joint.J3}, J4={convertPose.Joint.J4}, J5={convertPos
e.Joint.J5}, J6={convertPose.Joint.J6}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"笛卡尔转关节失败: {code.GetDescription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            Console.WriteLine(
```

```
                disconnectCode != StatusCode.OK
                    ? disconnectCode.GetDescription()
                    : "Successfully disconnected."
            );
        }

        return code;
    }
}
```

4.2.4 将关节值点位转换成笛卡尔点位

方法名	**Motion.ConvertJointToCart(MotionPose `pose` , int `ufIndex` = 0, int `tfIndex` = 0)**
描述	将关节值点位转换成笛卡尔点位表达
请求参数	`pose` : MotionPose 机器人的关节位姿 `ufIndex` : int 用户坐标系索引，默认为 0 `tfIndex` : int 工具坐标系索引，默认为 0
返回值	MotionPose: 转换后的机器人位姿数据 StatusCode: 转换操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Motion/ConvertJointToCart.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Motion;
using Agilebot.IR.Types;

public class ConvertJointToCart
{
    public static StatusCode Run(
```

```
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            // [ZH] 创建关节位姿
            // [EN] Create joint pose
            MotionPose motionPose = new MotionPose();
            motionPose.Pt = PoseType.Joint;
            motionPose.Joint = new Joint
            {
                J1 = 0,
                J2 = 0,
                J3 = 60,
                J4 = 60,
                J5 = 0,
                J6 = 0,
            };

            // [ZH] 将关节值点位转换成笛卡尔点位
```

```
            // [EN] Convert joint pose to Cartesian pose
            MotionPose convertPose;
            (convertPose, code) =
                controller.Motion.ConvertJointToCart(
                    motionPose
                );
            if (code == StatusCode.OK)
            {
                Console.WriteLine("关节转笛卡尔成功:");
                Console.WriteLine(
                    $"位置: X={convertPose.CartData.Position.X}, Y={convertP
ose.CartData.Position.Y}, Z={convertPose.CartData.Position.Z}"
                );
                Console.WriteLine(
                    $"姿态: A={convertPose.CartData.Position.A}, B={convertP
ose.CartData.Position.B}, C={convertPose.CartData.Position.C}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"关节转笛卡尔失败: {code.GetDescription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            Console.WriteLine(
                disconnectCode != StatusCode.OK
                    ? disconnectCode.GetDescription()
```

```
                    : "Successfully disconnected."
            );
        }

        return code;
    }
}
```

4.2.5 关节运动

方法名	Motion.MoveJoint(MotionPose `pose` , double `vel` = 1, double `acc` = 1)
描述	控制机器人末端沿关节空间最快路径移动到指定位置
请求参数	`pose` : MotionPose 笛卡尔空间或关节坐标系点位 `vel` : double 速度比率，范围为 0~1 `acc` : double 加速度比率，范围为 0~1.2
返回值	StatusCode: 运动指令执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Motion/MoveJoint.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Motion;
using Agilebot.IR.Types;

public class MoveJoint
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
```

```
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            // [ZH] 创建关节位姿
            // [EN] Create joint pose
            MotionPose motionPose = new MotionPose();
            motionPose.Pt = PoseType.Joint;
            motionPose.Joint = new Joint
            {
                J1 = 10,
                J2 = 30,
                J3 = 30,
                J4 = 0,
                J5 = 0,
                J6 = 0,
            };

            // [ZH] 让机器人末端移动到指定的位置
            // [EN] Move robot end to specified position
            code = controller.Motion.MoveJoint(
                motionPose,
```

```
                0.5,
                0.8
            );
            if (code == StatusCode.OK)
            {
                Console.WriteLine("关节运动请求成功");
            }
            else
            {
                Console.WriteLine(
                    $"关节运动失败: {code.GetDescription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            Console.WriteLine(
                disconnectCode != StatusCode.OK
                    ? disconnectCode.GetDescription()
                    : "Successfully disconnected."
            );
        }

        return code;
    }
}
```

4.2.6 直线运动

方法名	**Motion.MoveLine(MotionPose `pose` , double `vel` = 100, double `acc` = 1)**
描述	控制机器人末端沿直线移动到指定位置，运动轨迹为两点之间的直线
请求参数	`pose` : MotionPose 笛卡尔空间或关节坐标系点位 `vel` : double 末端速度，范围为 0~5000 mm/s `acc` : double 加速度比率，范围为 0~1.2
返回值	StatusCode: 运动指令执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Motion/MoveLine.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Motion;
using Agilebot.IR.Types;

public class MoveLine
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
```

```
StatusCode code = controller.ConnectSync();
Console.WriteLine(
    code != StatusCode.OK
        ? code.GetDescription()
        : "连接成功/Successfully connected."
);

if (code != StatusCode.OK)
{
    return code;
}

try
{
    // [ZH] 创建关节位姿
    // [EN] Create joint pose
    MotionPose motionPose = new MotionPose();
    motionPose.Pt = PoseType.Joint;
    motionPose.Joint = new Joint
    {
        J1 = 20,
        J2 = 40,
        J3 = 40,
        J4 = 5,
        J5 = 5,
        J6 = 5,
    };

    // [ZH] 让机器人末端沿直线移动到指定的位置
    // [EN] Move robot end in straight line to specified position
    code = controller.Motion.MoveLine(
        motionPose,
        100,
        1.0
    );
    if (code == StatusCode.OK)
    {
        Console.WriteLine("直线运动请求成功");
    }
    else
    {
        Console.WriteLine(
```

```
                    $"直线运动失败: {code.GetDescription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            Console.WriteLine(
                disconnectCode != StatusCode.OK
                    ? disconnectCode.GetDescription()
                    : "Successfully disconnected."
            );
        }

        return code;
    }
}
```

4.2.7 圆弧运动

方法名	**Motion.MoveCircle(MotionPose `pose1` , MotionPose `pose2` , double vel = 100, double acc = 1)**
描述	控制机器人末端沿圆弧轨迹移动到指定位置，通过途经点和终点确定圆弧
请求参数	`pose1` : MotionPose 途经点位姿 `pose2` : MotionPose 终点位姿

方法名	**Motion.MoveCircle(MotionPose `pose1` , MotionPose `pose2` , double vel = 100, double acc = 1)**
	`vel` : double 末端速度，范围为 0~5000 mm/s `acc` : double 加速度比率，范围为 0~1.2
返回值	StatusCode: 运动指令执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Motion/MoveCircle.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Motion;
using Agilebot.IR.Types;

public class MoveCircle
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );
```

```
if (code != StatusCode.OK)
{
    return code;
}

try
{
    // [ZH] 创建第一个位姿（途径点）
    // [EN] Create first pose (waypoint)
    MotionPose motionPose1 = new MotionPose();
    motionPose1.Pt = PoseType.Joint;
    motionPose1.Joint = new Joint
    {
        J1 = 0,
        J2 = 0,
        J3 = 60,
        J4 = 60,
        J5 = 0,
        J6 = 0,
    };

    // [ZH] 创建第二个位姿（终点）
    // [EN] Create second pose (endpoint)
    MotionPose motionPose2 = new MotionPose();
    motionPose2.Pt = PoseType.Joint;
    motionPose2.Joint = new Joint
    {
        J1 = 0,
        J2 = 30,
        J3 = 70,
        J4 = 40,
        J5 = 0,
        J6 = 0,
    };

    // [ZH] 让机器人末端沿弧线移动到指定的位置
    // [EN] Move robot end in arc to specified position
    code = controller.Motion.MoveCircle(
        motionPose1,
        motionPose2,
        100,
```

```
                1.0
            );
            if (code == StatusCode.OK)
            {
                Console.WriteLine("弧线运动请求成功");
            }
            else
            {
                Console.WriteLine(
                    $"弧线运动失败: {code.GetDescription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            Console.WriteLine(
                disconnectCode != StatusCode.OK
                    ? disconnectCode.GetDescription()
                    : "Successfully disconnected."
            );
        }

        return code;
    }
}
```

4.2.8 获取当前位姿

方法名	**Motion.GetCurrentPose(PoseType `pt` , int `ufIndex` = 0, int `tfIndex` = 0)**
描述	获取机器人当前位姿，支持笛卡尔空间或关节坐标系下的位姿信息
请求参数	`pt` : PoseType 位姿类型 `ufIndex` : int 用户坐标系索引 (仅 PoseType.CART 生效；默认 0) `tfIndex` : int 工具坐标系索引 (仅 PoseType.CART 生效；默认 0)
返回值	MotionPose: 机器人位姿数据 StatusCode: 获取操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Motion/GetCurrentPose.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Motion;
using Agilebot.IR.Types;

public class GetCurrentPose
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
```

```
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            // [ZH] 获取机器人的当前位姿（笛卡尔坐标）
            // [EN] Get robot current pose (Cartesian coordinates)
            MotionPose cartPose;
            (cartPose, code) =
                controller.Motion.GetCurrentPose(
                    PoseType.Cart,
                    0,
                    0
                );
            if (code == StatusCode.OK)
            {
                Console.WriteLine("当前笛卡尔位姿:");
                Console.WriteLine(
                    $"位置: X={cartPose.CartData.Position.X}, Y={cartPose.Ca
rtData.Position.Y}, Z={cartPose.CartData.Position.Z}"
                );
                Console.WriteLine(
                    $"姿态: A={cartPose.CartData.Position.A}, B={cartPose.Ca
rtData.Position.B}, C={cartPose.CartData.Position.C}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"获取笛卡尔位姿失败: {code.GetDescription()}"
                );
            }
```

```
            // [ZH] 获取机器人的当前位姿（关节坐标）
            // [EN] Get robot current pose (joint coordinates)
            MotionPose jointPose;
            (jointPose, code) =
                controller.Motion.GetCurrentPose(
                    PoseType.Joint,
                    0,
                    0
                );
            if (code == StatusCode.OK)
            {
                Console.WriteLine("当前关节位姿:");
                Console.WriteLine(
                    $"关节值: J1={jointPose.Joint.J1}, J2={jointPose.Joint.J
2}, J3={jointPose.Joint.J3}, J4={jointPose.Joint.J4}, J5={jointPose.Joint.J
5}, J6={jointPose.Joint.J6}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"获取关节位姿失败: {code.GetDescription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            Console.WriteLine(
                disconnectCode != StatusCode.OK
```

```
                    ? disconnectCode.GetDescription()
                    : "Successfully disconnected."
            );
        }

        return code;
    }
}
```

4.2.9 获取 DH 参数

方法名	Motion.GetDHParam()
描述	获取机器人的 DH 参数
请求参数	无
返回值	List<DHparam>: DH 参数列表 StatusCode: 获取操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): 不支持

示例代码

Motion/GetDHParam.cs

cs

```cs
using Agilebot.IR;
using Agilebot.IR.Types;

public class GetDHParam
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
```

```
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            // [ZH] 获取机器人的DH参数
            // [EN] Get robot DH parameters
            List<DHparam> dhParamsList;
            (dhParamsList, code) =
                controller.Motion.GetDHParam(1);
            if (code == StatusCode.OK)
            {
                Console.WriteLine("获取DH参数成功:");
                for (int i = 0; i < dhParamsList.Count; i++)
                {
                    var dh = dhParamsList[i];
                    Console.WriteLine(
                        $"轴{i + 1}: Alpha={dh.alpha}, A={dh.a}, D={dh.d}, O
ffset={dh.offset}"
                    );
                }
            }
            else
            {
                Console.WriteLine(
```

```
                    $"获取DH参数失败: {code.GetDescription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            Console.WriteLine(
                disconnectCode != StatusCode.OK
                    ? disconnectCode.GetDescription()
                    : "Successfully disconnected."
            );
        }

        return code;
    }
}
```

4.2.10 设置 DH 参数

方法名	Motion.SetDHParam(List<DHparam> dHparams)
描述	设置机器人的 DH 参数
请求参数	dHparams : List<DHparam> DH 参数列表
返回值	StatusCode: 设置操作执行结果

方法名	Motion.SetDHParam(List<DHparam> dHparams)
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): 不支持

示例代码

Motion/SetDHParam.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Types;

public class SetDHParam
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
```

```
        {
            // [ZH] 先获取当前的DH参数
            // [EN] First get current DH parameters
            List<DHparam> dhParamsList;
            (dhParamsList, code) =
                controller.Motion.GetDHParam(1);
            if (code != StatusCode.OK)
            {
                Console.WriteLine(
                    $"获取DH参数失败: {code.GetDescription()}"
                );
                return code;
            }

            Console.WriteLine(
                "获取DH参数成功，准备设置相同的参数..."
            );

            // [ZH] 设置DH参数（这里设置为相同的参数作为示例）
            // [EN] Set DH parameters (set same parameters as example)
            code = controller.Motion.SetDHParam(
                dhParamsList
            );
            if (code == StatusCode.OK)
            {
                Console.WriteLine("设置DH参数成功");
            }
            else
            {
                Console.WriteLine(
                    $"设置DH参数失败: {code.GetDescription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
```

```
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            Console.WriteLine(
                disconnectCode != StatusCode.OK
                    ? disconnectCode.GetDescription()
                    : "Successfully disconnected."
            );
        }

        return code;
    }
}
```

4.2.11 获取轴锁定状态

方法名	Motion.GetDragSet()
描述	获取当前机器人轴锁定状态，轴锁定仅针对示教运动
请求参数	无
返回值	DragStatus: 轴锁定状态，True 表示该轴为可移动状态，False 表示被锁定 StatusCode: 获取操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): 不支持

4.2.12 设定机器人轴锁定状态

方法名	Motion.SetDragSet(DragStatus `dragStatus`)
描述	设定当前机器人轴锁定状态，轴锁定只针对示教运动

方法名	Motion.SetDragSet(DragStatus `dragStatus`)
请求参数	`dragStatus` : DragStatus 各轴锁定状态（默认全部 True：解锁）
返回值	StatusCode: 设置操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): 不支持

4.2.13 设定当前机器人是否启动拖动

方法名	Motion.EnableDrag(bool `dragState`)
描述	设定当前机器人是否启动拖动示教功能
请求参数	`dragState` : bool 机器人拖动开关（true 进入拖动状态，false 退出拖动状态）
返回值	StatusCode: 设置操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): 不支持

示例代码

Motion/DragControl.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Motion;
using Agilebot.IR.Types;

public class DragControl
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
```

```
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            // [ZH] 获取当前机器人的轴锁定状态
            // [EN] Get current robot axis lock status
            DragStatus dragStatus;
            (dragStatus, code) =
                controller.Motion.GetDragSet();
            if (code == StatusCode.OK)
            {
                Console.WriteLine("获取轴锁定状态成功:");
                Console.WriteLine(
                    $"X轴: {dragStatus.CartStatus.X}, Y轴: {dragStatus.CartS
tatus.Y}, Z轴: {dragStatus.CartStatus.Z}"
                );
                Console.WriteLine(
                    $"连续拖动: {dragStatus.IsContinuousDrag}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"获取轴锁定状态失败: {code.GetDescription()}"
```

```
        );
    }

    // [ZH] 修改当前机器人的轴锁定状态
    // [EN] Modify current robot axis lock status
    if (code == StatusCode.OK)
    {
        dragStatus.CartStatus.X = false;
        dragStatus.IsContinuousDrag = true;
        code = controller.Motion.SetDragSet(
            dragStatus
        );
        if (code == StatusCode.OK)
        {
            Console.WriteLine("设置轴锁定状态成功");
        }
        else
        {
            Console.WriteLine(
                $"设置轴锁定状态失败: {code.GetDescription()}"
            );
        }
    }

    // [ZH] 启动拖动（注意：实际使用中需要谨慎）
    // [EN] Enable drag (Note: use with caution in practice)
    if (code == StatusCode.OK)
    {
        Console.WriteLine(
            "注意：启动拖动功能，请确保安全！"
        );
        code = controller.Motion.EnableDrag(true);
        if (code == StatusCode.OK)
        {
            Console.WriteLine("启动拖动成功");

            // [ZH] 等待一段时间后停止拖动
            // [EN] Wait for a while then stop drag
            Console.WriteLine(
                "等待3秒后停止拖动..."
            );
            Thread.Sleep(3000);
```

```
                    code = controller.Motion.EnableDrag(
                        false
                    );
                    if (code == StatusCode.OK)
                    {
                        Console.WriteLine("停止拖动成功");
                    }
                    else
                    {
                        Console.WriteLine(
                            $"停止拖动失败: {code.GetDescription()}"
                        );
                    }
                }
                else
                {
                    Console.WriteLine(
                        $"启动拖动失败: {code.GetDescription()}"
                    );
                }
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            Console.WriteLine(
                disconnectCode != StatusCode.OK
                    ? disconnectCode.GetDescription()
                    : "Successfully disconnected."
            );
```

```
            }

            return code;
        }
    }
```

4.2.14 进入实时位置控制模式

方法名	Motion.EnterPositionControl()
描述	进入实时位置控制模式，允许对机器人进行精确的位置控制
请求参数	无
返回值	StatusCode: 模式切换操作执行结果
备注	在进入实时控制模式后，必须通过 UDP 发送控制指令
兼容的机器人软件版本	协作 (Copper): v7.5.2.0+ 工业 (Bronze): 不支持

4.2.15 退出实时位置控制模式

方法名	Motion.ExitPositionControl()
描述	退出实时位置控制模式，恢复默认的机器人控制状态
请求参数	无
返回值	StatusCode: 模式切换操作执行结果
备注	退出后，机器人将不再接受实时控制指令
兼容的机器人软件版本	协作 (Copper): v7.5.2.0+ 工业 (Bronze): 不支持

4.2.16 获取 UDP 反馈配置

方法名	**Motion.GetUDPFeedbackConfig(int `id`)**
描述	读取 UDP 机器人状态配置文件 rtf_info.json
请求参数	`id` : int 配置 ID，从 1 开始
返回值	UDPFeedbackConfig: 指定 ID 的 UDP 机器人状态配置 StatusCode: 获取操作执行结果
备注	配置约束： 1. 当存在多个配置时，只能有一个配置的 sub_flag 为 true，其他应为 false。 2. 当 use_multicast 为 true 时，client_ip 仅支持一个 239...* 格式的组播地址。 3. 当 use_multicast 为 false 时，client_ip 支持多个局域网单播地址。
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.2.17 设置 UDP 反馈配置

方法名	**Motion.SetUDPFeedbackConfig(UDPFeedbackConfig `config`)**
描述	写入 UDP 机器人状态配置文件 rtf_info.json
请求参数	`config` : UDPFeedbackConfig UDP 机器人状态配置，配置中的 ID 从 1 开始
返回值	StatusCode: 设置操作执行结果
备注	配置约束： 1. 当存在多个配置时，只能有一个配置的 sub_flag 为 true，其他应为 false。 2. 当 use_multicast 为 true 时，client_ip 仅支持一个 239...* 格式的组播地址。 3. 当 use_multicast 为 false 时，client_ip 支持多个局域网单播地址。 此接口仅写入配置文件，不会立即生效。
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.2.18 启用 UDP 反馈

方法名	**Motion.EnableUDPFeedback(int `id`)**
描述	启用指定 ID 的 UDP 机器人状态配置，并同步 rtf_info.json 使其立即生效
请求参数	`id` : int 配置 ID，从 1 开始
返回值	StatusCode: 启用操作执行结果
备注	执行后，只有指定 ID 的 sub_flag 为 true，其他所有配置的 sub_flag 为 false。
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.2.19 禁用 UDP 反馈

方法名	**Motion.DisableUDPFeedback(int `id`)**
描述	禁用指定 ID 的 UDP 机器人状态配置，并同步 rtf_info.json 使其立即生效
请求参数	`id` : int 配置 ID，从 1 开始
返回值	StatusCode: 禁用操作执行结果
备注	执行后，所有配置的 sub_flag 都为 false。
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.2.20 设置订阅参数

方法名	**Motion.SetUDPFeedbackParams(bool** `flag` **, string** `ip` **, int** `interval` **, int** `feedbackType` **, List<int>** `DOList` **= null)**
描述	配置机器人向指定 IP 地址推送数据的 UDP 反馈参数
请求参数	`flag` : bool 是否开启 UDP 数据推送 `ip` : string 接收端 IP 地址 `interval` : int 发送间隔（单位：毫秒） `feedbackType` : int 反馈数据格式（0：XML 格式） `DOList` : List<int> DO 信号列表（最多 10 个，可选）
返回值	StatusCode: 参数设置操作执行结果
备注	参数设置仅在 UDP 数据推送功能启用时有效
兼容的机器人软件版本	协作 (Copper): v7.5.2.0+ 工业 (Bronze): 不支持

示例代码

Motion/PositionControl.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Motion;
using Agilebot.IR.Types;

public class PositionControl
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
```

```
// [EN] Connect to the Agilebot robot
StatusCode code = controller.ConnectSync();
Console.WriteLine(
    code != StatusCode.OK
        ? code.GetDescription()
        : "连接成功/Successfully connected."
);

if (code != StatusCode.OK)
{
    return code;
}

try
{
    // [ZH] 设置UDP反馈参数
    // [EN] Set UDP feedback parameters
    code = controller.Motion.SetUDPFeedbackParams(
        true,
        "192.168.1.1",
        10,
        0
    );
    if (code == StatusCode.OK)
    {
        Console.WriteLine("设置UDP反馈参数成功");
    }
    else
    {
        Console.WriteLine(
            $"设置UDP反馈参数失败: {code.GetDescription()}"
        );
    }

    // [ZH] 进入实时位置控制模式
    // [EN] Enter real-time position control mode
    code = controller.Motion.EnterPositionControl();
    if (code == StatusCode.OK)
    {
        Console.WriteLine(
            "进入实时位置控制模式成功"
        );
```

```
                // [ZH] 在此可以插入发送UDP数据控制机器人的代码
                // [EN] Insert UDP data control code here
                Console.WriteLine(
                    "注意：在实时位置控制模式下，需要通过UDP发送控制指令"
                );
                Console.WriteLine("等待2秒...");
                Thread.Sleep(2000);

                // [ZH] 退出实时位置控制模式
                // [EN] Exit real-time position control mode
                code =
                    controller.Motion.ExitPositionControl();
                if (code == StatusCode.OK)
                {
                    Console.WriteLine(
                        "退出实时位置控制模式成功"
                    );
                }
                else
                {
                    Console.WriteLine(
                        $"退出实时位置控制模式失败: {code.GetDescription()}"
                    );
                }
            }
            else
            {
                Console.WriteLine(
                    $"进入实时位置控制模式失败: {code.GetDescription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
```

```
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            Console.WriteLine(
                disconnectCode != StatusCode.OK
                    ? disconnectCode.GetDescription()
                    : "Successfully disconnected."
            );
        }

        return code;
    }
}
```

推送数据说明

名称	字段	描述
RIst: 笛卡尔位置	X	工具坐标系下 X 方向值，单位为毫米
	Y	工具坐标系下 Y 方向值，单位为毫米
	Z	工具坐标系下 Z 方向值，单位为毫米
	A	工具坐标系下绕 X 方向旋转，单位为度
	B	工具坐标系下绕 Y 方向旋转，单位为度
	C	工具坐标系下绕 Z 方向旋转，单位为度
AIPos: 关节位置	A1-A6	六个关节的值，单位为角度
EIPos: 附加轴数据	EIPos	附加轴数据
WristBtnState: 手腕按键状态	按键状态	1 = 按键按下，0 = 按键抬起
	DragModel	拖拽按键状态
	RecordJoint	示教记录按键状态
	PauseResume	暂停恢复按键状态

名称	字段	描述
Digout: DO 输出	Digout	数字输出（DO）的状态
ProgramStatus: 程序状态	ProgId	程序 ID
	Status	解释器执行状态： 0 = INTERPRETER_IDLE 1 = INTERPRETER_EXECUTE 2 = INTERPRETER_PAUSED
	Xpath	程序片段返回值，格式为 `程序名:行号`
IPOC: 时间戳	IPOC	时间戳

4.2.21 获取机器人软限位

方法名	Motion.GetUserSoftLimit()
描述	获取当前机器人软限位信息
请求参数	无
返回值	List<List<double>>: 机器人软限位信息，列表第一层代表各轴，第二层代表每个轴的下限位和上限位值 StatusCode: 获取操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.2.22 指定 UDP 位置控制的相关参数

方法名	Motion.SetPositionTrajectoryParams(int `maxTimeoutCount` , int `timeout` , int `filterLayer` , double `wristElbowThreshold` , double `shoulderThreshold`)
描述	设置 UDP 位置控制的相关参数

方法名	**Motion.SetPositionTrajectoryParams(int** `maxTimeoutCount` **, int** `timeout` **, int** `filterLayer` **, double** `wristElbowThreshold` **, double** `shoulderThreshold` **)**
请求参数	`maxTimeoutCount` : int 最大超时次数，取值范围 [1,100] `timeout` : int 超时时间（即发送间隔，默认为 20ms），取值范围 [1,100] `filterLayer` : int 滤波层级，取值范围 [1,100] `wristElbowThreshold` : double 腕 / 肘部接近奇异点的阈值，取值范围 [10,100] `shoulderThreshold` : double 接近肩部奇异点的阈值，取值范围 [100,300]
返回值	StatusCode: 参数设置操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.2.23 获取轨迹整形器参数

方法名	**Motion.GetShapingParam()**
描述	获取当前轨迹整形器参数值
请求参数	无参数
返回值	int: 轨迹整形器参数值（范围 5~15，0 代表关闭） StatusCode: 查询操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.7.1.0+ 工业 (Bronze): 不支持

4.2.24 设置轨迹整形器参数

方法名	**Motion.SetShapingParam(int** `value` **)**
描述	设置轨迹整形器参数值
请求参数	`value` : int 轨迹整形器参数（范围 5~15，0 代表关闭）

方法名	Motion.SetShapingParam(int `value`)
返回值	StatusCode: 设置操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.7.I.0+ 工业 (Bronze): 不支持

4.2.25 负载相关接口

4.2.25.1 获取当前激活的负载

方法名	Motion.Payload.GetCurrentPayload()
描述	获取当前激活的负载编号，返回对应的索引
请求参数	无
返回值	int: 负载索引 StatusCode: 获取操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.2.25.2 获取对应的负载

方法名	Motion.Payload.GetPayloadById(int `index`)
描述	根据索引获取对应的负载信息
请求参数	`index` : int 负载索引
返回值	StatusCode: 获取操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.2.25.3 激活对应的负载

方法名	Motion.Payload.SetCurrentPayload(int `index`)
描述	根据索引激活指定负载
请求参数	`index` : int 负载索引
返回值	StatusCode: 激活操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+
备注	负载 ID 必须是当前设备中存在的

4.2.25.4 获取所有负载信息

方法名	Motion.Payload.GetAllPayloadInfo()
描述	获取所有负载的详细信息
请求参数	无
返回值	Dictionary<uint, string>: 负载信息字典 StatusCode: 获取操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.2.25.5 添加负载

方法名	Motion.Payload.AddPayload(PayloadInfo `payload`)
描述	添加新的负载
请求参数	`payload` : PayloadInfo 负载对象
返回值	StatusCode: 添加操作执行结果
备注	新的负载 ID 必须是当前设备中没有的且在 1~10 之间
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.2.25.6 删除指定负载

方法名	Motion.Payload.DeletePayload(int `index`)
描述	删除指定索引的负载
请求参数	`index` : int 负载索引
返回值	StatusCode: 删除操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+
备注	注意：无法删除当前激活的负载，如果要删除激活的负载，请先激活其他负载再删除当前负载

4.2.25.7 更新指定负载

方法名	Motion.Payload.UpdatePayload(PayloadInfo `payload`)
描述	更新指定负载信息
请求参数	`payload` : PayloadInfo 负载对象
返回值	StatusCode: 更新操作执行结果
备注	负载 ID 必须是当前设备中存在的
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Motion/PayloadControl.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Motion;

public class PayloadControl
{
    public static StatusCode Run(
        string controllerIP,
```

```
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            // [ZH] 获取负载列表
            // [EN] Get payload list
            Dictionary<int, string> payloadList;
            (payloadList, code) =
                controller.Motion.Payload.GetAllPayloadInfo();
            if (code == StatusCode.OK)
            {
                Console.WriteLine("获取负载列表成功:");
                foreach (var p in payloadList)
                {
                    Console.WriteLine(
                        $"负载ID: {p.Key}, 描述: {p.Value}"
                    );
                }
            }
            else
```

```
{
    Console.WriteLine(
        $"获取负载列表失败: {code.GetDescription()}"
    );
}

// [ZH] 获取当前激活的负载
// [EN] Get current active payload
int currentPayload;
(currentPayload, code) =
    controller.Motion.Payload.GetCurrentPayload();
if (code == StatusCode.OK)
{
    Console.WriteLine(
        $"当前激活的负载ID: {currentPayload}"
    );
}
else
{
    Console.WriteLine(
        $"获取当前负载失败: {code.GetDescription()}"
    );
}

// [ZH] 添加新负载
// [EN] Add new payload
PayloadInfo payload = new()
{
    Id = 3,
    Comment = "测试负载",
    Weight = 1.0,
    MassCenter = new()
    {
        X = 1,
        Y = 2,
        Z = 3,
    },
    InertiaMoment = new()
    {
        LX = 10,
        LY = 20,
        LZ = 30,
```

```
                },
            };

            code = controller.Motion.Payload.AddPayload(
                payload
            );
            if (code == StatusCode.OK)
            {
                Console.WriteLine("添加负载成功");
            }
            else
            {
                Console.WriteLine(
                    $"添加负载失败: {code.GetDescription()}"
                );
            }

            // [ZH] 设置当前激活的负载
            // [EN] Set current active payload
            if (code == StatusCode.OK)
            {
                code =
                    controller.Motion.Payload.SetCurrentPayload(
                        3
                    );
                if (code == StatusCode.OK)
                {
                    Console.WriteLine("设置当前负载成功");
                }
                else
                {
                    Console.WriteLine(
                        $"设置当前负载失败: {code.GetDescription()}"
                    );
                }
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
```

```
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            Console.WriteLine(
                disconnectCode != StatusCode.OK
                    ? disconnectCode.GetDescription()
                    : "Successfully disconnected."
            );
        }

        return code;
    }
}
```

4.2.25.8 检测 3 轴是否水平

方法名	Motion.Payload.CheckAxisThreeHorizontal()
描述	检测 3 轴是否水平
请求参数	无
返回值	double: 3 轴的水平角度 StatusCode: 检测操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.2.0+ 工业 (Bronze): 不支持
备注	水平的角度必须在 - 1~1 之间才能进行负载测定

4.2.25.9 获取负载测定状态

方法名	Motion.Payload.GetPayloadIdentifyState()
描述	获取负载测定的状态
请求参数	无
返回值	PayloadIdentifyState: 负载测定状态 StatusCode: 获取操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.2.0+ 工业 (Bronze): 不支持

4.2.25.10 开始负载测定

方法名	Motion.Payload.StartPayloadIdentify(double `weight` , double `angle`)
描述	开始负载测定
请求参数	`weight` : double 负载重量（未知重量填 - 1） `angle` : double 6 轴允许转动的角度（30-90 度）
返回值	StatusCode: 负载测定启动操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.2.0+ 工业 (Bronze): 不支持
备注	开始负载测定前必须先进入负载测定状态

4.2.25.11 获取负载测定结果

方法名	Motion.Payload.PayloadIdentifyResult()
描述	获取负载测定的结果
请求参数	无
返回值	PayloadInfo: 负载测定结果 StatusCode: 获取操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.2.0+ 工业 (Bronze): 不支持

4.2.25.12 开始负载辨识干涉检查

方法名	Motion.Payload.InterferenceCheckForPayloadIdentify(double `weight` , double `angle`)
描述	开始负载测定的干涉检查，用于查看是否会发生碰撞
请求参数	`weight` : double 负载重量（未知重量填 - 1） `angle` : double 6 轴允许转动的角度（30-90 度）
返回值	StatusCode: 干涉检查操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.2.0+ 工业 (Bronze): 不支持

4.2.25.13 进入负载测定状态

方法名	Motion.Payload.PayloadIdentifyStart()
描述	进入负载测定状态
请求参数	无
返回值	StatusCode: 状态切换操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.2.0+ 工业 (Bronze): 不支持

4.2.25.14 结束负载测定状态

方法名	Motion.Payload.PayloadIdentifyDone()
描述	结束负载测定状态
请求参数	无
返回值	StatusCode: 状态切换操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.2.0+ 工业 (Bronze): 不支持

4.2.25.15 负载测定全流程

方法名	**Motion.Payload.PayloadIdentify(double `weight` = -1, double `angle` = 90)**
描述	完整的负载测定流程，包含上述负载测定全部接口，无特殊需求负载测定只用该接口即可
请求参数	`weight` : double 负载重量（未知重量填 - 1） `angle` : double 6 轴允许转动的角度（30-90 度）
返回值	PayloadInfo: 负载测定结果 StatusCode: 负载测定操作执行结果
备注	返回的负载可以新增到机器人中或写入机器人中已有的某个负载 全流程步骤： 1. 进入负载测定状态 2. 开始负载测定 3. 获取负载测定结果 4. 结束负载测定状态
兼容的机器人软件版本	协作 (Copper): v7.5.2.0+ 工业 (Bronze): 不支持

4.2.25.16 终止负载测定

方法名	**Motion.Payload.TerminatePayloadIdentify()**
描述	终止负载测定状态
请求参数	无
返回值	StatusCode: 状态切换操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.7.I.0+ 工业 (Bronze): 不支持

示例代码

Motion/PayloadIdentify.cs

CS

```cs
using Agilebot.IR;
using Agilebot.IR.Motion;
using Agilebot.IR.Types;

public class PayloadIdentify
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            // [ZH] 获取机器人模式
            // [EN] Get robot mode
            (UserOpMode opMode, StatusCode opCode) =
                controller.GetOpMode();
            if (opCode == StatusCode.OK)
            {
                Console.WriteLine(
```

```
                    $"当前机器人模式/Current robot mode: {opMode}"
                );
                if (opMode != UserOpMode.AUTO)
                {
                    Console.WriteLine(
                        $"负载测定执行必须在机器人自动模式下/Payload identificati
on execution must be in automatic mode"
                    );
                    return StatusCode.OtherReason;
                }
            }
            else
            {
                Console.WriteLine(
                    $"获取机器人模式失败/Failed to get robot mode: {opCode.GetD
escription()}"
                );
            }

            // [ZH] 检测3轴是否水平
            // [EN] Check if axis 3 is horizontal
            double horizontalAngle;
            (horizontalAngle, code) =
                controller.Motion.Payload.CheckAxisThreeHorizontal();
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    $"3轴水平角度: {horizontalAngle}"
                );
                if (Math.Abs(horizontalAngle) > 1)
                {
                    Console.WriteLine(
                        "警告: 3轴水平角度超出范围(-1~1)，无法进行负载测定"
                    );
                    return StatusCode.OtherReason;
                }
            }
            else
            {
                Console.WriteLine(
                    $"检测3轴水平失败: {code.GetDescription()}"
                );
```

```
            }

            // [ZH] 获取负载测定状态
            // [EN] Get payload identification state
            PayloadIdentifyState identifyState;
            (identifyState, code) =
                controller.Motion.Payload.GetPayloadIdentifyState();
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    $"负载测定状态: {identifyState}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"获取负载测定状态失败: {code.GetDescription()}"
                );
            }

            // [ZH] 执行完整的负载测定流程
            // [EN] Execute complete payload identification process
            PayloadInfo payload;
            (payload, code) =
                controller.Motion.Payload.PayloadIdentify(
                    -1,
                    90
                );
            if (code == StatusCode.OK)
            {
                Console.WriteLine("负载测定成功:");
                Console.WriteLine(
                    $"负载重量: {payload.Weight}"
                );
                Console.WriteLine(
                    $"质心位置: X={payload.MassCenter.X}, Y={payload.MassCent
er.Y}, Z={payload.MassCenter.Z}"
                );
                Console.WriteLine(
                    $"惯性矩: LX={payload.InertiaMoment.LX}, LY={payload.Iner
tiaMoment.LY}, LZ={payload.InertiaMoment.LZ}"
                );
```

```
                // [ZH] 保存负载到机器人中
                // [EN] Save payload to robot
                payload.Id = 6;
                code = controller.Motion.Payload.AddPayload(
                    payload
                );
                if (code == StatusCode.OK)
                {
                    Console.WriteLine(
                        "保存负载到机器人成功"
                    );
                }
                else
                {
                    Console.WriteLine(
                        $"保存负载失败: {code.GetDescription()}"
                    );
                }
            }
            else
            {
                Console.WriteLine(
                    $"负载测定失败: {code.GetDescription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            Console.WriteLine(
```

```
                disconnectCode != StatusCode.OK
                    ? disconnectCode.GetDescription()
                    : "Successfully disconnected."
            );
        }

        return code;
    }
}
```

4.3 机器人程序操作类

概述

Execution 类提供机器人程序与运动任务的统一调度接口，负责以下核心功能：

- 启动 / 停止 / 暂停 / 恢复示教程序
- 管理并发运行的任务列表
- 执行 BAS 脚本等自定义流程

结合 `Arm` 与 `Motion` 的连接和运动能力，Execution 负责在上位机侧触发和管控控制器中的程序流程。

4.3.1 执行指定程序

方法名	**Execution.Start(string `programName`)**
描述	执行指定程序
请求参数	`programName` : string 程序名称
返回值	StatusCode: 操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.3.2 停止正在执行的程序

方法名	**Execution.Stop(string `programName` = null)**
描述	停止正在执行的程序或停止机器人当前执行的运动

方法名	Execution.Stop(string `programName` = null)
请求参数	`programName` : string 程序名称（默认为 null，表示停止当前运行的程序或运动指令）
返回值	StatusCode: 操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.3.3 返回所有正在运行的程序详细信息

方法名	Execution.AllRunningPrograms()
描述	返回所有正在运行程序的详细信息，包含程序 ID 和程序名称
请求参数	无
返回值	Dictionary<string, int>: 程序 ID 到程序名的映射 StatusCode: 获取操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.3.4 暂停程序运行

方法名	Execution.Pause(string `programName` = null)
描述	暂停当前执行的程序或暂停机器人当前执行的运动
请求参数	`programName` : string 程序名称（默认为 null，表示暂停当前运行的程序或运动指令）
返回值	StatusCode: 操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.3.5 恢复程序运行

方法名	Execution.Resume(string programName = null)
描述	继续运行处于暂停状态的程序
请求参数	programName : string 程序名称（默认为 null，表示恢复当前处于暂停状态的程序或运动指令）
返回值	StatusCode: 操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Execution/ProgramExecution.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Types;

public class ProgramExecution
{
    /// <summary>
    /// 测试程序执行完整流程功能
    /// 验证程序的启动、暂停、恢复和停止等完整操作流程
    /// </summary>
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
```

```
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            Console.WriteLine(
                "开始程序执行完整流程/Starting Program Execution Complete Flow"
            );

            // [ZH] 获取测试文件路径
            // [EN] Get test file path
            string file_user_program = GetTestFilePath(
                "test_prog.xml"
            );

            // [ZH] 设置程序名称
            // [EN] Set program name
            string progName = "test_prog";

            // [ZH] 上传用户程序文件
            // [EN] Upload user program file
            code = controller.FileManager.Upload(
                file_user_program,
                FileType.UserProgram,
                true
            );
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    $"用户程序文件上传成功/User Program File Upload Success: {p
rogName}"
                );
```

```
            }
            else
            {
                Console.WriteLine(
                    $"用户程序文件上传失败/User Program File Upload Failed: {co
de.GetDescription()}"
                );
                return code;
            }

            // [ZH] 等待
            // [EN] Wait
            Thread.Sleep(3000);

            // [ZH] 启动程序
            // [EN] Start program
            code = controller.Execution.Start(progName);
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    $"程序启动成功/Program Started Successfully: {progName}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"程序启动失败/Program Start Failed: {code.GetDescription
()}"
                );
                return code;
            }
            Thread.Sleep(2000);

            // [ZH] 获取所有正在运行的程序列表
            // [EN] Get all running programs list
            Dictionary<string, int> progList;
            (progList, code) =
                controller.Execution.AllRunningPrograms();
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "获取运行程序列表成功/Get Running Programs List Success"
```

```
                );
                Console.WriteLine(
                    $"运行程序数量/Running Programs Count: {progList.Count}"
                );
                foreach (var prog in progList)
                {
                    Console.WriteLine(
                        $"  程序/Program: {prog.Key}, 状态/Status: {prog.Value}"
                    );
                }
            }
            else
            {
                Console.WriteLine(
                    $"获取运行程序列表失败/Get Running Programs List Failed: {code.GetDescription()}"
                );
                return code;
            }
            Thread.Sleep(2000);

            // [ZH] 暂停程序
            // [EN] Pause program
            code = controller.Execution.Pause(progName);
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    $"程序暂停成功/Program Paused Successfully: {progName}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"程序暂停失败/Program Pause Failed: {code.GetDescription()}"
                );
                return code;
            }
            Thread.Sleep(2000);

            // [ZH] 恢复程序
```

```
            // [EN] Resume program
            code = controller.Execution.Resume(progName);
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    $"程序恢复成功/Program Resumed Successfully: {progName}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"程序恢复失败/Program Resume Failed: {code.GetDescription
()}"
                );
                return code;
            }
            Thread.Sleep(2000);

            // [ZH] 停止程序
            // [EN] Stop program
            code = controller.Execution.Stop(progName);
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    $"程序停止成功/Program Stopped Successfully: {progName}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"程序停止失败/Program Stop Failed: {code.GetDescription
()}"
                );
                return code;
            }

            // [ZH] 删除用户程序文件
            // [EN] Delete user program file
            code = controller.FileManager.Delete(
                progName,
                FileType.UserProgram
            );
```

```
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    $"用户程序文件删除成功/User Program File Delete Success: {p
rogName}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"用户程序文件删除失败/User Program File Delete Failed: {co
de.GetDescription()}"
                );
                return code;
            }

            Console.WriteLine(
                "程序执行完整流程结束/Program Execution Complete Flow Test Comp
leted"
            );
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
```

```
            }
        }

        return code;
    }

    /// <summary>
    /// 获取test_files文件夹中文件的路径示例方法
    /// 展示如何获取当前程序目录下的test_files文件夹中的文件路径
    /// </summary>
    private static string GetTestFilePath(string fileName)
    {
        // [ZH] 获取当前程序集的目录
        // [EN] Get current assembly directory
        string? codeFilePath =
            new System.Diagnostics.StackTrace(true)
                .GetFrame(0)
                ?.GetFileName();
        if (string.IsNullOrEmpty(codeFilePath))
        {
            throw new InvalidOperationException(
                "无法获取当前文件路径/Cannot get current file path"
            );
        }

        string? codeDirectory = Path.GetDirectoryName(
            codeFilePath
        );
        if (string.IsNullOrEmpty(codeDirectory))
        {
            throw new InvalidOperationException(
                "无法获取当前目录路径/Cannot get current directory path"
            );
        }

        // [ZH] 构建test_files文件夹路径
        // [EN] Build test_files folder path
        string testFilesDirectory = Path.Combine(
            codeDirectory,
            "test_files"
        );
        // [ZH] 构建文件完整路径
```

```
        // [EN] Build complete file path
        string filePath = Path.Combine(
            testFilesDirectory,
            fileName
        );
        return filePath;
    }
}
```

4.3.6 执行 BAS 脚本程序

方法名	Execution.ExecuteBasScript(BasScript `script`)
描述	执行用户自定义的 BAS 脚本程序
请求参数	`script` : BasScript BAS 脚本程序
返回值	StatusCode: 操作执行结果
备注	BAS 脚本程序的暂停、恢复、停止方法同普通程序
兼容的机器人软件版本	协作 (Copper): v7.5.2.0+ 工业 (Bronze): 不支持 工业机器人: 7.6.0.0

示例代码

Execution/ExecuteBasScript.cs

```cs
using Agilebot.IR;
using Agilebot.IR.BasScript;
using Agilebot.IR.Execution;
using Agilebot.IR.Types;

public class ExecuteBasScript
{
    /// <summary>
    /// 测试执行Bas脚本功能
    /// 验证能否成功执行包含条件判断、运动控制和赋值操作的Bas脚本
```

```
    /// </summary>
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            Console.WriteLine(
                "开始执行Bas脚本程序/Starting Execute BasScript"
            );

            // [ZH] 创建BAS脚本程序
            // [EN] Create BAS script program
            BasScript script = new BasScript("test_bas");

            // [ZH] 添加条件判断到脚本
            // [EN] Add conditional statement to script
            code = script.Logical.IF(
                RegisterType.R,
                1,
```

```
                OtherType.VALUE,
                0
            );
            if (code != StatusCode.OK)
            {
                Console.WriteLine(
                    $"添加条件判断失败/Add Conditional Statement Failed: {cod
e.GetDescription()}"
                );
                return code;
            }

            // [ZH] 添加运动控制到脚本
            // [EN] Add motion control to script
            BasScript.ExtraParam param =
                new BasScript.ExtraParam();
            param.Acceleration(80);
            code = script.Motion.MoveJoint(
                MovePoseType.PR,
                1,
                SpeedType.VALUE,
                30,
                SmoothType.SD,
                10,
                extraParam: param
            );
            if (code != StatusCode.OK)
            {
                Console.WriteLine(
                    $"添加运动控制失败/Add Motion Control Failed: {code.GetDes
cription()}"
                );
                return code;
            }

            // [ZH] 添加赋值操作到脚本
            // [EN] Add assignment operation to script
            code = script.AssignValue(AssignType.R, 1, 99);
            if (code != StatusCode.OK)
            {
                Console.WriteLine(
                    $"添加赋值操作失败/Add Assignment Operation Failed: {code.
```

```
GetDescription()}"
                );
                return code;
            }

            // [ZH] 结束条件判断
            // [EN] End conditional statement
            code = script.Logical.END_IF();
            if (code != StatusCode.OK)
            {
                Console.WriteLine(
                    $"结束条件判断失败/End Conditional Statement Failed: {cod
e.GetDescription()}"
                );
                return code;
            }

            // [ZH] 等待上一个测试结束
            // [EN] Wait for previous test to end
            Thread.Sleep(1000);

            // [ZH] 执行BAS脚本程序
            // [EN] Execute BAS script program
            code = controller.Execution.ExecuteBasScript(
                script
            );
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "BAS脚本执行成功/Execute BasScript Success"
                );
                Console.WriteLine(
                    "脚本包含条件判断、运动控制和赋值操作/Script includes conditio
nal statements, motion control and assignment operations"
                );
            }
            else
            {
                Console.WriteLine(
                    $"BAS脚本执行失败/Execute BasScript Failed: {code.GetDescr
iption()}"
                );
```

```
            }

            Console.WriteLine(
                "执行Bas脚本测试完成/Execute BasScript Test Completed"
            );
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.4 程序位姿读写

概述

ProgramPoses 类用于读取、写入和转换机器人示教程序中的位姿点。通过此类可以：

- 定位到指定程序与点位序号
- 执行增删改查操作
- 在笛卡尔 / 关节表示之间进行转换

便于在上位机场景下批量维护程序点位或进行离线编辑。

4.4.1 获取指定程序中指定位姿点值

方法名	**ProgramPoses.Read(string `programName` , int `index` , FileType `ft` = FileType.UserProgram)**
描述	获取指定程序中指定序号的位姿点值
请求参数	`programName` : string 程序名称 `index` : int 位姿点序号 `ft` : FileType 文件类型 (默认 FileType.UserProgram)
返回值	ProgramPose: 位姿信息 StatusCode: 操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

ProgramPoses/ReadProgramPose.cs

```cs
using Agilebot.IR;
using Agilebot.IR.ProgramPoses;
```

```
using Agilebot.IR.Types;

public class ReadProgramPose
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            // [ZH] 设置程序名称和位姿点索引
            // [EN] Set program name and pose index
            string progName = "test_prog";
            int index = 1;

            // [ZH] 读取指定程序中指定位姿点值
            // [EN] Read specified pose value in specified program
            ProgramPose pose;
            (pose, code) = controller.ProgramPoses.Read(
                progName,
```

```
                index
            );
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "读取程序位姿点成功/Read Program Pose Success"
                );
                Console.WriteLine(
                    $"位姿信息/Pose Info: {pose}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"读取程序位姿点失败/Read Program Pose Failed: {code.GetDes
cription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }
```

```
        return code;
    }
}
```

4.4.2 修改指定程序中指定位姿点值

方法名	**ProgramPoses.Write(string** `programName` **, int** `index` **, ProgramPose** `value` **, FileType** `ft` **= FileType.UserProgram)**
描述	修改指定程序中指定序号的位姿点值
请求参数	`programName` : string 程序名称 `index` : int 位姿点序号 `value` : ProgramPose 需更新的位姿点值 `ft` : FileType 文件类型 (默认 FileType.UserProgram)
返回值	StatusCode: 操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

ProgramPoses/WriteProgramPose.cs

cs

```
using Agilebot.IR;
using Agilebot.IR.ProgramPoses;
using Agilebot.IR.Types;

public class WriteProgramPose
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
```

```
// [EN] Initialize the Agilebot robot
Arm controller = new Arm(
    controllerIP,
    useLocalProxy
);

// [ZH] 连接捷勃特机器人
// [EN] Connect to the Agilebot robot
StatusCode code = controller.ConnectSync();
Console.WriteLine(
    code != StatusCode.OK
        ? code.GetDescription()
        : "连接成功/Successfully connected."
);

if (code != StatusCode.OK)
{
    return code;
}

try
{
    // [ZH] 设置程序名称和位姿点索引
    // [EN] Set program name and pose index
    string progName = "test_prog";
    int index = 2;

    // [ZH] 生成随机位姿点
    // [EN] Generate random pose
    ProgramPose rndPose =
        ProgramPose.GenerateRandomPose(index);

    // [ZH] 修改指定程序中指定位姿点值
    // [EN] Write specified pose value in specified program
    code = controller.ProgramPoses.Write(
        progName,
        index,
        rndPose
    );
    if (code == StatusCode.OK)
    {
        Console.WriteLine(
```

```
                    "写入程序位姿点成功/Write Program Pose Success"
                );
            }
            else
            {
                Console.WriteLine(
                    $"写入程序位姿点失败/Write Program Pose Failed: {code.GetDescription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.Message}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.4.3 向程序中添加指定位姿点

方法名	**ProgramPoses.Add(string `programName` , int `index` , ProgramPose `value` , FileType `ft` = FileType.UserProgram)**
描述	在指定程序的指定序号处新增位姿点
请求参数	`programName` : string 程序名称 `index` : int 位姿点序号 `value` : ProgramPose 需新增的位姿点值 `ft` : FileType 文件类型 (默认 FileType.UserProgram)
返回值	StatusCode: 操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

ProgramPoses/AddProgramPose.cs

```cs
using Agilebot.IR;
using Agilebot.IR.ProgramPoses;
using Agilebot.IR.Types;

public class AddProgramPose
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
```

```
// [EN] Connect to the Agilebot robot
StatusCode code = controller.ConnectSync();
Console.WriteLine(
    code != StatusCode.OK
        ? code.GetDescription()
        : "连接成功/Successfully connected."
);

if (code != StatusCode.OK)
{
    return code;
}

try
{
    // [ZH] 设置程序名称和位姿点索引
    // [EN] Set program name and pose index
    string progName = "test_prog";
    int index = 3;

    // [ZH] 生成随机位姿点
    // [EN] Generate random pose
    ProgramPose rndPose =
        ProgramPose.GenerateRandomPose(index);

    // [ZH] 添加指定程序中指定位姿点
    // [EN] Add specified pose in specified program
    code = controller.ProgramPoses.Add(
        progName,
        index,
        rndPose
    );
    if (code == StatusCode.OK)
    {
        Console.WriteLine(
            "添加程序位姿点成功/Add Program Pose Success"
        );
    }
    else
    {
        Console.WriteLine(
            $"添加程序位姿点失败/Add Program Pose Failed: {code.GetDesc
```

```
ription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.4.4 删除指定程序中指定序号的位姿点

方法名	ProgramPoses.Delete(string `programName` , int `index` , FileType `ft` = FileType.UserProgram)
描述	删除指定程序中指定序号的位姿点

方法名	**ProgramPoses.Delete(string** `programName` **, int** `index` **, FileType** `ft` **= FileType.UserProgram)**
请求参数	`programName` : string 程序名称 `index` : int 位姿点序号 `ft` : FileType 文件类型 (默认 FileType.UserProgram)
返回值	StatusCode: 操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

ProgramPoses/DeleteProgramPose.cs

```cs
using Agilebot.IR;
using Agilebot.IR.ProgramPoses;
using Agilebot.IR.Types;

public class DeleteProgramPose
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
```

```
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            // [ZH] 设置程序名称和位姿点索引
            // [EN] Set program name and pose index
            string progName = "test_prog";
            int index = 3;

            // [ZH] 删除指定程序中指定位姿点
            // [EN] Delete specified pose in specified program
            code = controller.ProgramPoses.Delete(
                progName,
                index
            );
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "删除程序位姿点成功/Delete Program Pose Success"
                );
            }
            else
            {
                Console.WriteLine(
                    $"删除程序位姿点失败/Delete Program Pose Failed: {code.GetDescription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.Message}"
            );
            code = StatusCode.OtherReason;
        }
```

```
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.4.5 获取指定程序中所有位姿点

方法名	**ProgramPoses.ReadAllPoses(string `programName` , FileType `ft` = FileType.UserProgram)**
描述	获取指定程序中所有位姿点信息
请求参数	`programName` : string 程序名称 `ft` : FileType 文件类型 (默认 FileType.UserProgram)
返回值	List<ProgramPose>: 位姿信息列表 StatusCode: 操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

ProgramPoses/ReadAllProgramPoses.cs

cs

```cs
using Agilebot.IR;
using Agilebot.IR.ProgramPoses;
using Agilebot.IR.Types;

public class ReadAllProgramPoses
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            // [ZH] 设置程序名称
            // [EN] Set program name
            string progName = "test_prog";

            // [ZH] 读取指定程序中所有位姿点
```

```
            // [EN] Read all poses in specified program
            List<ProgramPose> poses;
            (poses, code) =
                controller.ProgramPoses.ReadAllPoses(
                    progName
                );
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "读取所有程序位姿点成功/Read All Program Poses Success"
                );
                Console.WriteLine(
                    $"位姿点数量/Number of poses: {poses.Count}"
                );

                for (int i = 0; i < poses.Count; i++)
                {
                    Console.WriteLine(
                        $"位姿点 {i + 1}/Pose {i + 1}: {poses[i]}"
                    );
                }
            }
            else
            {
                Console.WriteLine(
                    $"读取所有程序位姿点失败/Read All Program Poses Failed: {co
de.GetDescription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
```

```
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.4.6 机器人程序位姿点类型转换

方法名	ProgramPoses.ConvertPose(ProgramPose `pose` , PoseType `toType`)
描述	将机器人程序位姿点在关节坐标和笛卡尔空间坐标之间转换
请求参数	`pose` : ProgramPose 要转换的位姿点值 `toType` : PoseType 转换后的目标类型
返回值	ProgramPose: 转换后的位姿信息 StatusCode: 操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

ProgramPoses/ConvertProgramPose.cs

cs

```
using Agilebot.IR;
using Agilebot.IR.ProgramPoses;
using Agilebot.IR.Types;
```

```
public class ConvertProgramPose
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            // [ZH] 设置程序名称和位姿点索引
            // [EN] Set program name and pose index
            string progName = "test_prog";
            int cartIndex = 1;

            // [ZH] 先读取一个位姿点
            // [EN] First read a pose
            ProgramPose cartPose;
            (cartPose, code) = controller.ProgramPoses.Read(
                progName,
                cartIndex
```

```
            );
            if (code != StatusCode.OK)
            {
                Console.WriteLine(
                    $"读取位姿点失败/Read Pose Failed: {code.GetDescription
()}"
                );
                return code;
            }

            // [ZH] 转换位姿点类型（从笛卡尔坐标转换为关节坐标）
            // [EN] Convert pose type (from Cartesian to Joint coordinates)
            ProgramPose pose;
            (pose, code) =
                controller.ProgramPoses.ConvertPose(
                    cartPose,
                    PoseType.Joint
                );
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "转换程序位姿点成功/Convert Program Pose Success"
                );
                Console.WriteLine(
                    $"原始位姿/Original Pose: {cartPose}"
                );
                Console.WriteLine(
                    $"转换后位姿/Converted Pose: {pose}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"转换程序位姿点失败/Convert Program Pose Failed: {code.Get
Description()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
```

```
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.5 IO 信号

概述

Signals 类提供控制器 IO 的统一读写接口，封装了以下核心功能：

- 数字 / 模拟输入输出控制
- 多路批量操作

通过 `Signals` 可以：

- 读取当前信号状态
- 批量写入 DO/RO/GO 等端口
- 实现与外部夹爪、传感器或生产线设备的联动。

4.5.1 读取指定类型和端口的 IO 值

方法名	**Signals.Read(SignalType `type` , int `index`)**
描述	读取指定类型和端口的 IO 值（支持 DI/DO/UI/UO/RI/RO/GI/GO/TAI/TDI/TDO/AI/AO）
请求参数	`type` : SignalType 要读取的 IO 类型 `index` : int IO 序号 (从 1 开始)
返回值	int: IO 值，1 代表高电平，0 代表低电平 StatusCode: 操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+
备注	UI/UO 只能读不能写

示例代码

Signals/ReadSignal.cs

cs

```cs
using Agilebot.IR;
using Agilebot.IR.Types;

public class ReadSignal
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            // [ZH] 设置IO信号类型和索引
            // [EN] Set IO signal type and index
            SignalType type = SignalType.DI;
            int index = 1;

            // [ZH] 读取指定类型指定端口IO的值
```

```
            // [EN] Read specified type and port IO value
            int res;
            (res, code) = controller.Signals.Read(
                type,
                index
            );
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "读取IO信号成功/Read Signal Success"
                );
                Console.WriteLine(
                    $"{type}: {index} 的值为/has value {res}"
                );
                Console.WriteLine(
                    $"信号状态/Signal Status: {(res == 1 ? "高电平/High Level"
: "低电平/Low Level")}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"读取IO信号失败/Read Signal Failed: {code.GetDescription
()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
```

```
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.5.2 写入指定类型和端口的 IO 值

方法名	Signals.Write(SignalType `type` , int `index` , double `value`)
描述	写入指定类型和端口的 IO 值，当前仅支持 DO/RO/GO/TDO/AO
请求参数	`type` : SignalType 要写入的 IO 类型 `index` : int IO 序号 (从 1 开始) `value` : double IO 值 (DO/RO/TDO 仅允许 0 或 1；GO 为整型；AO 为浮点模拟量)
返回值	StatusCode: 操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+
备注	UI/UO 只能读不能写

示例代码

Signals/WriteSignal.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Types;
```

```
public class WriteSignal
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            // [ZH] 设置IO信号类型、索引和值
            // [EN] Set IO signal type, index and value
            SignalType type = SignalType.DO;
            int index = 1;
            int value = 1;

            // [ZH] 写指定类型指定端口IO的值
            // [EN] Write specified type and port IO value
            code = controller.Signals.Write(
                type,
                index,
                value
```

```
            );
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "写入IO信号成功/Write Signal Success"
                );
                Console.WriteLine(
                    $"{type}：{index} 设置为/set to value {value}"
                );
                Console.WriteLine(
                    $"信号状态/Signal Status: {(value == 1 ? "高电平/High Level" : "低电平/Low Level")}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"写入IO信号失败/Write Signal Failed: {code.GetDescription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.Message}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
```

```
                    code = disconnectCode;
                }
            }

            return code;
        }
    }
```

4.5.3 批量写入 DO (数字输出) 信号

方法名	Signals.MultiWrite(SignalType `type` , List<int> `ioData`)
描述	批量写入 DO (数字输出) 信号
请求参数	`type` : SignalType 要写入的 IO 类型 (仅支持 DO) `ioData` : List<int> 端口号和端口值列表 (例如 [port1, state1, port2, state2]；长度为偶数且大于 0)
返回值	StatusCode: 操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+
备注	仅支持 DO 批量写入；UI/UO 不支持写入，DI/RI 仅支持单点读取

示例代码

```cs
using Agilebot.IR;
using Agilebot.IR.Types;

public class Test
{
    public static async Task Main()
    {
        string controllerIP = "10.27.1.254";
        Arm controller = new Arm(controllerIP);
        StatusCode code = await controller.Connect();
        Console.WriteLine(code != StatusCode.OK ? code.GetDescription() : "S
uccessfully connected.");
```

```
        // 批量写入 DO1=1, DO2=0
        code = controller.Signals.MultiWrite(SignalType.DO, new List<int> {
1, 1, 2, 0 });
        Console.WriteLine(code != StatusCode.OK ? code.GetDescription() : "M
ultiWrite Success");

        code = controller.Disconnect();
        Console.WriteLine(code != StatusCode.OK ? code.GetDescription() : "S
uccessfully disconnected.");
    }
}
```

4.5.4 批量读取 DO (数字输出) 端口值

方法名	Signals.MultiRead(SignalType `type` , List<int> `indexes`)
描述	批量读取 DO (数字输出) 端口值
请求参数	`type` : SignalType 要读取的 IO 类型 (仅支持 DO) `indexes` : List<int> 端口号列表 (不能为空)
返回值	List<int>: 端口值列表（顺序与输入一致） StatusCode: 操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+
备注	仅支持 DO 批量读取；UI/UO 不支持写入，DI/RI 仅支持单点读取

示例代码

```cs
using Agilebot.IR;
using Agilebot.IR.Types;

public class Test
{
    public static async Task Main()
```

```
    {
        string controllerIP = "10.27.1.254";
        Arm controller = new Arm(controllerIP);
        StatusCode code = await controller.Connect();
        Console.WriteLine(code != StatusCode.OK ? code.GetDescription() : "S
uccessfully connected.");

        // 批量读取 DO1、DO2
        (List<int> values, StatusCode readCode) = controller.Signals.MultiRe
ad(SignalType.DO, new List<int> { 1, 2 });
        if (readCode == StatusCode.OK)
        {
            Console.WriteLine($"MultiRead Success: DO1={values[0]}, DO2={val
ues[1]}");
        }

        code = controller.Disconnect();
        Console.WriteLine(code != StatusCode.OK ? code.GetDescription() : "S
uccessfully disconnected.");
    }
}
```

4.6 寄存器信息

概述

Registers 类提供上位机读写控制器寄存器的统一入口，支持多种寄存器类型的操作。

核心功能

- 支持数值寄存器（R）的读写操作
- 支持运动寄存器（MR）的读写操作
- 支持字符串寄存器（SR）的读写操作
- 支持位姿寄存器（PR）的读写操作
- 支持 Modbus 寄存器（MH 保持寄存器、MI 输入寄存器）的读写操作

使用场景

- 程序运行时传递参数
- 同步机器人状态信息
- 与外部系统共享配置数据
- 实现机器人与外部设备的交互控制

4.6.1 R 数值寄存器操作

4.6.1.1 读取 R 寄存器值

方法名	Registers.Read_R(int `index`)
描述	读取 R 数值寄存器的值
请求参数	`index` : int 要读取的寄存器编号

方法名	Registers.Read_R(int index)
返回值	double: 寄存器值 StatusCode: 操作执行结果
兼容的机器人软件版本	协作机器人: 7.6.0.1 工业机器人: 7.6.0.0

4.6.1.2 读取 R 寄存器值（含元信息）

方法名	Registers.Read_R(int index , bool withMeta)
描述	读取 R 数值寄存器的值，可返回元信息（名称 / 注释）
请求参数	index : int 要读取的寄存器编号 withMeta : bool 是否返回元信息（true 时返回寄存器对象）
返回值	Register: 寄存器对象（包含 id/name/comment/value） StatusCode: 操作执行结果
兼容的机器人软件版本	协作机器人: 7.7.0.0+ 工业机器人: 7.7.0.0+

4.6.1.3 写入 R 寄存器值

方法名	Registers.Write_R(int index , double value)
描述	写入 R 数值寄存器的值
请求参数	index : int 要写入的寄存器编号 value : double 要写入的寄存器数值
返回值	StatusCode: 操作执行结果
兼容的机器人软件版本	协作机器人: 7.6.0.1 工业机器人: 7.6.0.0

4.6.1.4 写入 R 寄存器（含元信息）

方法名	Registers.Write_R(Register `register`)
描述	使用 Register 对象写入 R 寄存器，包含名称与注释
请求参数	`register` : Register 寄存器对象（id/name/comment/value）
返回值	StatusCode: 操作执行结果
兼容的机器人软件版本	协作机器人: 7.7.0.0+ 工业机器人: 7.7.0.0+

4.6.1.5 删除 R 寄存器

方法名	Registers.Delete_R(int `index`)
描述	删除指定的 R 数值寄存器
请求参数	`index` : int 要删除的寄存器编号
返回值	StatusCode: 操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Registers/RRegisterOperations.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Types;

public class RRegisterOperations
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
```

```
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            // [ZH] 设置寄存器索引和值
            // [EN] Set register index and value
            int index = 1;
            double value = 9.9;

            // [ZH] 写入R寄存器
            // [EN] Write R register
            code = controller.Registers.Write_R(
                index,
                value
            );
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "写入R寄存器成功/Write R Register Success"
                );
            }
            else
            {
                Console.WriteLine(
                    $"写入R寄存器失败/Write R Register Failed: {code.GetDescri
ption()}"
```

```
                );
            }

            // [ZH] 读取R寄存器
            // [EN] Read R register
            double readValue;
            (readValue, code) = controller.Registers.Read_R(
                index
            );
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    $"读取R寄存器成功/Read R Register Success: 值/Value = {rea
dValue}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"读取R寄存器失败/Read R Register Failed: {code.GetDescrip
tion()}"
                );
            }

            // [ZH] 删除R寄存器
            // [EN] Delete R register
            code = controller.Registers.Delete_R(index);
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "删除R寄存器成功/Delete R Register Success"
                );
            }
            else
            {
                Console.WriteLine(
                    $"删除R寄存器失败/Delete R Register Failed: {code.GetDescr
iption()}"
                );
            }
        }
        catch (Exception ex)
```

```
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.6.2 MR 运动寄存器操作

4.6.2.1 读取 MR 寄存器值

方法名	Registers.Read_MR(int `index`)
描述	读取 MR 运动寄存器的值
请求参数	`index` : int 要读取的寄存器编号

方法名	Registers.Read_MR(int index)
返回值	int: 寄存器值 StatusCode: 操作执行结果
兼容的机器人软件版本	协作机器人: 7.6.0.1 工业机器人: 7.6.0.0

4.6.2.2 读取 MR 寄存器值（含元信息）

方法名	Registers.Read_MR(int index , bool withMeta)
描述	读取 MR 运动寄存器的值，可返回元信息（名称 / 注释）
请求参数	index : int 要读取的寄存器编号 withMeta : bool 是否返回元信息（true 时返回寄存器对象）
返回值	MotionRegister: 寄存器对象（包含 id/name/comment/value） StatusCode: 操作执行结果
兼容的机器人软件版本	协作机器人: 7.7.0.0+ 工业机器人: 7.7.0.0+

4.6.2.3 写入 MR 寄存器值

方法名	Registers.Write_MR(int index , int value)
描述	写入 MR 运动寄存器的值
请求参数	index : int 要写入的寄存器编号 value : int 要写入的寄存器数值
返回值	StatusCode: 操作执行结果
兼容的机器人软件版本	协作机器人: 7.6.0.1 工业机器人: 7.6.0.0

4.6.2.4 写入 MR 寄存器（含元信息）

方法名	Registers.Write_MR(MotionRegister `register`)
描述	使用 MotionRegister 对象写入 MR 寄存器，包含名称与注释
请求参数	`register` : MotionRegister 寄存器对象（id/name/comment/value）
返回值	StatusCode: 操作执行结果
兼容的机器人软件版本	协作机器人: 7.7.0.0+ 工业机器人: 7.7.0.0+

4.6.2.5 删除 MR 寄存器

方法名	Registers.Delete_MR(int `index`)
描述	删除指定的 MR 运动寄存器
请求参数	`index` : int 要删除的寄存器编号
返回值	StatusCode: 操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Registers/MRRegisterOperations.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Types;

public class MRRegisterOperations
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
```

```
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            // [ZH] 设置寄存器索引和值
            // [EN] Set register index and value
            int index = 1;
            int value = 9;

            // [ZH] 写入MR寄存器
            // [EN] Write MR register
            code = controller.Registers.Write_MR(
                index,
                value
            );
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "写入MR寄存器成功/Write MR Register Success"
                );
            }
            else
            {
                Console.WriteLine(
                    $"写入MR寄存器失败/Write MR Register Failed: {code.GetDesc
ription()}"
```

```
                );
            }

            // [ZH] 读取MR寄存器
            // [EN] Read MR register
            int readValue;
            (readValue, code) =
                controller.Registers.Read_MR(index);
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    $"读取MR寄存器成功/Read MR Register Success: 值/Value = {r
eadValue}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"读取MR寄存器失败/Read MR Register Failed: {code.GetDescr
iption()}"
                );
            }

            // [ZH] 删除MR寄存器
            // [EN] Delete MR register
            code = controller.Registers.Delete_MR(index);
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "删除MR寄存器成功/Delete MR Register Success"
                );
            }
            else
            {
                Console.WriteLine(
                    $"删除MR寄存器失败/Delete MR Register Failed: {code.GetDes
cription()}"
                );
            }
        }
        catch (Exception ex)
        {
```

```
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.6.3 SR 字符串寄存器操作

4.6.3.1 读取 SR 寄存器值

方法名	Registers.Read_SR(int `index`)
描述	读取 SR 字符串寄存器的值
请求参数	`index` : int 要读取的寄存器编号
返回值	string: 寄存器字符串值 StatusCode: 操作执行结果

方法名	Registers.Read_SR(int index)
兼容的机器人软件版本	协作机器人: 7.6.0.1 工业机器人: 7.6.0.0

4.6.3.2 读取 SR 寄存器值（含元信息）

方法名	Registers.Read_SR(int index , bool withMeta)
描述	读取 SR 字符串寄存器的值，可返回元信息（名称 / 注释）
请求参数	index : int 要读取的寄存器编号 withMeta : bool 是否返回元信息（true 时返回寄存器对象）
返回值	StringRegister: 寄存器对象（包含 id/name/comment/value） StatusCode: 操作执行结果
兼容的机器人软件版本	协作机器人: 7.7.0.0+ 工业机器人: 7.7.0.0+

4.6.3.3 写入 SR 寄存器值

方法名	Registers.Write_SR(int index , string value)
描述	写入 SR 字符串寄存器的值
请求参数	index : int 要写入的寄存器编号 value : string 要写入的寄存器字符串值
返回值	StatusCode: 操作执行结果
兼容的机器人软件版本	协作机器人: 7.6.0.1 工业机器人: 7.6.0.0

4.6.3.4 写入 SR 寄存器（含元信息）

方法名	Registers.Write_SR(StringRegister register)
描述	使用 StringRegister 对象写入 SR 寄存器，包含名称与注释

方法名	Registers.Write_SR(StringRegister `register`)
请求参数	`register` : StringRegister 寄存器对象（id/name/comment/value）
返回值	StatusCode: 操作执行结果
兼容的机器人软件版本	协作机器人: 7.7.0.0+ 工业机器人: 7.7.0.0+

4.6.3.5 删除 SR 寄存器

方法名	Registers.Delete_SR(int `index`)
描述	删除指定的 SR 字符串寄存器
请求参数	`index` : int 要删除的寄存器编号
返回值	StatusCode: 操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Registers/SRRegisterOperations.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Types;

public class SRRegisterOperations
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
```

```
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            // [ZH] 设置寄存器索引和值
            // [EN] Set register index and value
            int index = 1;
            string value = "test";

            // [ZH] 写入SR寄存器
            // [EN] Write SR register
            code = controller.Registers.Write_SR(
                index,
                value
            );
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "写入SR寄存器成功/Write SR Register Success"
                );
            }
            else
            {
                Console.WriteLine(
                    $"写入SR寄存器失败/Write SR Register Failed: {code.GetDesc
ription()}"
                );
            }
```

```
            // [ZH] 读取SR寄存器
            // [EN] Read SR register
            string readValue;
            (readValue, code) =
                controller.Registers.Read_SR(index);
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    $"读取SR寄存器成功/Read SR Register Success: 值/Value = {r
eadValue}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"读取SR寄存器失败/Read SR Register Failed: {code.GetDescr
iption()}"
                );
            }

            // [ZH] 删除SR寄存器
            // [EN] Delete SR register
            code = controller.Registers.Delete_SR(index);
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "删除SR寄存器成功/Delete SR Register Success"
                );
            }
            else
            {
                Console.WriteLine(
                    $"删除SR寄存器失败/Delete SR Register Failed: {code.GetDes
cription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
```

```
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.6.4 PR 位姿寄存器操作

4.6.4.1 读取 PR 寄存器值

方法名	Registers.Read_PR(int `index`)
描述	读取 PR 位姿寄存器的值
请求参数	`index` : int 要读取的寄存器编号
返回值	PoseRegister: 位姿数据 StatusCode: 操作执行结果

方法名	Registers.Read_PR(int `index`)
兼容的机器人软件版本	协作机器人: 7.6.0.1 工业机器人: 7.6.0.0

4.6.4.2 读取 PR 寄存器值（含元信息）

方法名	Registers.Read_PR(int `index` , bool `withMeta`)
描述	读取 PR 位姿寄存器的值，可返回元信息（名称 / 注释）
请求参数	`index` : int 要读取的寄存器编号 `withMeta` : bool 是否返回元信息（true 时包含 name/comment）
返回值	PoseRegister: 位姿数据（ `withMeta=true` 时包含 name/comment） StatusCode: 操作执行结果
兼容的机器人软件版本	协作机器人: 7.7.0.0+ 工业机器人: 7.7.0.0+

4.6.4.3 写入 PR 寄存器值

方法名	Registers.Write_PR(PoseRegister `pose`)
描述	写入 PR 位姿寄存器的值
请求参数	`pose` : PoseRegister 要写入的位姿数据
返回值	StatusCode: 操作执行结果
兼容的机器人软件版本	协作机器人: 7.6.0.1 工业机器人: 7.6.0.0

4.6.4.4 写入 PR 寄存器（含元信息）

方法名	Registers.Write_PR(PoseRegister `pose` , bool `withMeta`)
描述	写入 PR 位姿寄存器的值，可控制是否写入元信息（名称 / 注释）

方法名	Registers.Write_PR(PoseRegister `pose` , bool `withMeta`)
请求参数	`pose` : PoseRegister 要写入的位姿数据 `withMeta` : bool 是否写入元信息（true 时写入 name/comment）
返回值	StatusCode: 操作执行结果
兼容的机器人软件版本	协作机器人: 7.7.0.0+ 工业机器人: 7.7.0.0+

4.6.4.5 删除 PR 寄存器

方法名	Registers.Delete_PR(int `index`)
描述	删除指定的 PR 位姿寄存器
请求参数	`index` : int 要删除的寄存器编号
返回值	StatusCode: 操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Registers/PRRegisterOperations.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Registers;
using Agilebot.IR.Types;

public class PRRegisterOperations
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
```

```
        controllerIP,
        useLocalProxy
    );

    // [ZH] 连接捷勃特机器人
    // [EN] Connect to the Agilebot robot
    StatusCode code = controller.ConnectSync();
    Console.WriteLine(
        code != StatusCode.OK
            ? code.GetDescription()
            : "连接成功/Successfully connected."
    );

    if (code != StatusCode.OK)
    {
        return code;
    }

    try
    {
        // [ZH] 设置寄存器索引
        // [EN] Set register index
        int index = 1;

        // [ZH] 生成位姿寄存器
        // [EN] Generate pose register
        var pose = new PoseRegister
        {
            Id = 1,
            Name = "Test",
            Comment = "Test",
            PoseRegisterData = new PoseRegisterData
            {
                Pt = PoseType.Joint,
                Joint = new Joint
                {
                    J1 = 6.6,
                    J2 = 6.6,
                    J3 = 6.6,
                    J4 = 6.6,
                    J5 = 6.6,
                    J6 = 6.6,
```

```
                    },
                    CartData = null,
                },
            };

            // [ZH] 写入PR寄存器
            // [EN] Write PR register
            code = controller.Registers.Write_PR(pose);
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "写入PR寄存器成功/Write PR Register Success"
                );
            }
            else
            {
                Console.WriteLine(
                    $"写入PR寄存器失败/Write PR Register Failed: {code.GetDesc
ription()}"
                );
            }

            // [ZH] 读取PR寄存器
            // [EN] Read PR register
            PoseRegister readValue;
            (readValue, code) =
                controller.Registers.Read_PR(index);
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    $"读取PR寄存器成功/Read PR Register Success: ID = {readVal
ue.Id}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"读取PR寄存器失败/Read PR Register Failed: {code.GetDescr
iption()}"
                );
            }
```

```
            // [ZH] 删除PR寄存器
            // [EN] Delete PR register
            code = controller.Registers.Delete_PR(index);
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "删除PR寄存器成功/Delete PR Register Success"
                );
            }
            else
            {
                Console.WriteLine(
                    $"删除PR寄存器失败/Delete PR Register Failed: {code.GetDes
cription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
```

```
        }
    }
```

4.6.5 Modbus 寄存器（MH 保持寄存器、MI 输入寄存器）

4.6.5.1 读取 MH 寄存器值

方法名	Registers.Read_MH(int `index`)
描述	读取 MH 保持寄存器的值
请求参数	`index` : int 要读取的寄存器编号
返回值	int: 寄存器值 StatusCode: 操作执行结果
兼容的机器人软件版本	协作机器人: 7.6.0.0 工业机器人: 7.6.0.0

4.6.5.2 读取 MI 寄存器值

方法名	Registers.Read_MI(int `index`)
描述	读取 MI 输入寄存器的值
请求参数	`index` : int 要读取的寄存器编号
返回值	int: 寄存器值 StatusCode: 操作执行结果
兼容的机器人软件版本	协作机器人: 7.6.0.0 工业机器人: 7.6.0.0

4.6.5.3 写入 MH 寄存器值

方法名	Registers.Write_MH(int `index` , int `value`)
描述	写入 MH 保持寄存器的值
请求参数	`index` : int 要写入的寄存器编号 `value` : int 要写入的寄存器数值
返回值	StatusCode: 操作执行结果
兼容的机器人软件版本	协作机器人: 7.6.0.0 工业机器人: 7.6.0.0

4.6.5.4 写入 MI 寄存器值

方法名	Registers.Write_MI(int `index` , int `value`)
描述	写入 MI 输入寄存器的值
请求参数	`index` : int 要写入的寄存器编号 `value` : int 要写入的寄存器数值
返回值	StatusCode: 操作执行结果
兼容的机器人软件版本	协作机器人: 7.6.0.0 工业机器人: 7.6.0.0

示例代码

Registers/ModbusRegisterOperations.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Types;

public class ModbusRegisterOperations
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
```

```
// [EN] Initialize the Agilebot robot
Arm controller = new Arm(
    controllerIP,
    useLocalProxy
);

// [ZH] 连接捷勃特机器人
// [EN] Connect to the Agilebot robot
StatusCode code = controller.ConnectSync();
Console.WriteLine(
    code != StatusCode.OK
        ? code.GetDescription()
        : "连接成功/Successfully connected."
);

if (code != StatusCode.OK)
{
    return code;
}

try
{
    // [ZH] 设置寄存器索引和值
    // [EN] Set register index and value
    int index = 1;
    int writeValue = 8;

    // [ZH] 写入MH保持寄存器
    // [EN] Write MH holding register
    code = controller.Registers.Write_MH(
        index,
        writeValue
    );
    if (code == StatusCode.OK)
    {
        Console.WriteLine(
            "写入MH保持寄存器成功/Write MH Holding Register Success"
        );
    }
    else
    {
        Console.WriteLine(
```

```
                    $"写入MH保持寄存器失败/Write MH Holding Register Failed: {c
ode.GetDescription()}"
                );
            }

            // [ZH] 写入MI输入寄存器
            // [EN] Write MI input register
            code = controller.Registers.Write_MI(
                index,
                writeValue + 1
            );
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "写入MI输入寄存器成功/Write MI Input Register Success"
                );
            }
            else
            {
                Console.WriteLine(
                    $"写入MI输入寄存器失败/Write MI Input Register Failed: {cod
e.GetDescription()}"
                );
            }

            // [ZH] 读取MH保持寄存器
            // [EN] Read MH holding register
            int mhValue;
            (mhValue, code) = controller.Registers.Read_MH(
                index
            );
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    $"读取MH保持寄存器成功/Read MH Holding Register Success:
值/Value = {mhValue}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"读取MH保持寄存器失败/Read MH Holding Register Failed: {co
```

```
de.GetDescription()}"
                );
            }

            // [ZH] 读取MI输入寄存器
            // [EN] Read MI input register
            int miValue;
            (miValue, code) = controller.Registers.Read_MI(
                index
            );
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    $"读取MI输入寄存器成功/Read MI Input Register Success: 值/V
alue = {miValue}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"读取MI输入寄存器失败/Read MI Input Register Failed: {cod
e.GetDescription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
```

```
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.7 轨迹控制

概述

Trajectory 模块提供离线轨迹执行、轨迹文件转换的接口，支持复杂轨迹的复现。

核心功能

- 支持设置和执行离线轨迹文件
- 支持以安全速度将机器人移动到离线轨迹起始点
- 支持 CSV 到 trajectory 格式的文件转换
- 支持查询轨迹文件转换状态

使用场景

- 实现复杂轨迹的精确复现
- 转换外部轨迹数据为机器人可执行格式

4.7.1 设置待执行的离线轨迹文件

方法名	**Trajectory.SetOffLineTrajectoryFile(string `path`)**
描述	设置待执行的离线轨迹文件
请求参数	`path` : string 离线轨迹文件程序名 (格式说明见下方备注)
返回值	StatusCode: 函数执行结果
备注	A.trajectory 轨迹文件格式为文本文件： - 第 1 行：6 代表 6 个轴，0.001 代表两点间隔 1 ms，8093 代表轨迹点数。 - 第 2 行：6 个轴的起始位置。 - 第 3-8095 行：轨迹点数据，包含 6 轴的位置 / 速度 / 加速度 / 力矩前馈 /do 端口 /do 端口值。 - do_port 取值范围 1~24；do_port 为 -1 表示该位置不触发 IO；do_port 为 1 且 do_state

方法名	Trajectory.SetOffLineTrajectoryFile(string `path`)
	为 1 表示 do1 端口触发 ON；do_port 为 1 且 do_state 为 0 表示 do1 端口触发 OFF。 用户通过 FileManager.upload 将离线文件发送到机器人控制器根目录，用 4.7.2 和 4.7.3 指令执行轨迹。
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.7.2 让机器人以安全速度移动到离线轨迹中的起始点

方法名	Trajectory.PrepareOfflineTrajectory()
描述	让机器人以安全速度移动到离线轨迹中的起始点
请求参数	无参数
返回值	StatusCode: 函数执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.7.3 让机器人开始执行离线轨迹文件

方法名	Trajectory.ExecuteOfflineTrajectory()
描述	让机器人开始执行离线轨迹程序
请求参数	无参数
返回值	StatusCode: 函数执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.7.4 轨迹文件转换功能

方法名	**Trajectory.TransformCsvToTrajectory(string** `fileName` **)**
描述	将轨迹 CSV 文件转换成 trajectory 格式的轨迹文件并存放在控制柜的轨迹文件目录上
请求参数	`fileName` : string CSV 轨迹文件名
重载方法	**Trajectory.TransformCsvToTrajectory**(string `fileName` , string `separator` , string `ioFlag`) `separator` : string 分隔符，支持空格或逗号 `ioFlag` : string IO 来源标识，"1" 表示使用默认 IO 值，"2" 表示使用用户指定 IO
返回值	string: 成功转换后的 trajectory 文件路径 StatusCode: 转换操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.7.5 查询轨迹文件转换状态

方法名	**Trajectory.CheckTransformStatus(string** `fileName` **)**
描述	查询轨迹文件转换的当前状态
请求参数	`fileName` : string 轨迹文件路径 (`TransformCsvToTrajectory` 接口返回)
返回值	TransformState: 转换状态 StatusCode: 函数执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Trajectory/OfflineTrajectory.cs

CS

```cs
using System.IO;
using Agilebot.IR;
using Agilebot.IR.Trajectory;
using Agilebot.IR.Types;

public class OfflineTrajectory
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            // [ZH] 获取机器人模式
            // [EN] Get robot mode
            (UserOpMode opMode, StatusCode opCode) =
                controller.GetOpMode();
            if (opCode == StatusCode.OK)
            {
```

```
                Console.WriteLine(
                    $"当前机器人模式/Current robot mode: {opMode}"
                );
                if (opMode != UserOpMode.AUTO)
                {
                    Console.WriteLine(
                        $"离线轨迹执行必须在机器人自动模式下/Offline trajectory execution must be in automatic mode"
                    );
                    return StatusCode.OtherReason;
                }
            }
            else
            {
                Console.WriteLine(
                    $"获取机器人模式失败/Failed to get robot mode: {opCode.GetDescription()}"
                );
            }

            // [ZH] 添加程序文件到机器人中
            // [EN] Add program file to robot
            string file_user_program = GetTestFilePath(
                "test.csv"
            );
            StatusCode ret_code =
                controller.FileManager.Upload(
                    file_user_program,
                    FileType.TmpFile,
                    true
                );
            if (ret_code != StatusCode.OK)
            {
                Console.WriteLine(
                    $"上传文件失败/Upload file failed: {ret_code.GetDescription()}"
                );
                return ret_code;
            }
            Console.WriteLine(
                "文件上传成功/File upload success"
            );
```

```
            // [ZH] 测试CSV转换为轨迹文件功能
            // [EN] Test CSV to trajectory file conversion functionality
            string csvFilename = "test.csv";
            (
                string trajFileName,
                StatusCode transformCode
            ) =
                controller.Trajectory.TransformCsvToTrajectory(
                    csvFilename
                );

            if (transformCode != StatusCode.OK)
            {
                Console.WriteLine(
                    $"CSV转换失败/CSV conversion failed: {transformCode.GetDe
scription()}"
                );
                return transformCode;
            }
            Console.WriteLine(
                $"CSV转换成功/CSV conversion success, trajectory file: {trajF
ileName}"
            );

            // [ZH] 检查转换状态
            // [EN] Check conversion status
            var startTime = System.DateTime.Now;
            TransformState state;
            StatusCode statusCode;
            do
            {
                (state, statusCode) =
                    controller.Trajectory.CheckTransformStatus(
                        System.IO.Path.GetFileName(
                            trajFileName
                        )
                    );
                if (statusCode != StatusCode.OK)
                {
                    Console.WriteLine(
                        $"检查转换状态失败/Check transform status failed: {sta
```

```
tusCode.GetDescription()}"
                    );
                    return statusCode;
                }

                Console.WriteLine(
                    $"转换状态/Transform state: {state}"
                );
                Thread.Sleep(2000); // 等待2秒

                if (
                    System.DateTime.Now - startTime
                    > System.TimeSpan.FromSeconds(60)
                )
                {
                    Console.WriteLine(
                        "转换状态检查超时/Transform status check timeout"
                    );
                    break;
                }
            } while (
                state != TransformState.TRANSFORM_SUCCESS
                && state != TransformState.TRANSFORM_FAILED
            );

            if (state == TransformState.TRANSFORM_FAILED)
            {
                Console.WriteLine(
                    "CSV转换失败/CSV conversion failed"
                );
                return StatusCode.OtherReason;
            }

            // [ZH] 转换任务成功并进行了结果查询后 服务端不会继续保存转换任务的状态
            // [EN] After the conversion task is successful and the result i
s queried, the server will not continue to save the conversion task status
            (
                TransformState finalState,
                StatusCode finalCode
            ) = controller.Trajectory.CheckTransformStatus(
                System.IO.Path.GetFileName(trajFileName)
            );
```

```
            if (finalCode != StatusCode.OK)
            {
                Console.WriteLine(
                    $"最终状态检查失败/Final status check failed: {finalCode.G
etDescription()}"
                );
                return finalCode;
            }
            Console.WriteLine(
                $"最终转换状态/Final transform state: {finalState}"
            );

            // [ZH] 设置轨迹文件
            // [EN] Set trajectory file
            code =
                controller.Trajectory.SetOffLineTrajectoryFile(
                    "test_torque.trajectory"
                );
            if (code != StatusCode.OK)
            {
                Console.WriteLine(
                    $"设置轨迹文件失败/Set trajectory file failed: {code.GetDe
scription()}"
                );
                return code;
            }
            Console.WriteLine(
                "设置轨迹文件成功/Set trajectory file success"
            );

            // [ZH] 准备离线轨迹
            // [EN] Prepare offline trajectory
            code =
                controller.Trajectory.PrepareOfflineTrajectory();
            if (code != StatusCode.OK)
            {
                Console.WriteLine(
                    $"准备离线轨迹失败/Prepare offline trajectory failed: {cod
e.GetDescription()}"
                );
                return code;
            }
```

```
            Console.WriteLine(
                "准备离线轨迹成功/Prepare offline trajectory success"
            );

            // [ZH] 等待机器人和伺服器空闲
            // [EN] Wait for robot and servo to be idle
            startTime = System.DateTime.Now;
            RobotState robotStatus;
            ServoState servoStatus;
            StatusCode robotStatusCode;
            StatusCode servoStatusCode;

            do
            {
                (robotStatus, robotStatusCode) =
                    controller.GetRobotState();
                if (robotStatusCode != StatusCode.OK)
                {
                    Console.WriteLine(
                        $"获取机器人状态失败/Get robot state failed: {robotStat
usCode.GetDescription()}"
                    );
                    return robotStatusCode;
                }

                (servoStatus, servoStatusCode) =
                    controller.GetServoState();
                if (servoStatusCode != StatusCode.OK)
                {
                    Console.WriteLine(
                        $"获取伺服状态失败/Get servo state failed: {servoStatu
sCode.GetDescription()}"
                    );
                    return servoStatusCode;
                }

                Console.WriteLine(
                    $"机器人状态/Robot state: {robotStatus}, 伺服状态/Servo sta
te: {servoStatus}"
                );

                if (
```

```
                    robotStatus == RobotState.ROBOT_IDLE
                    && servoStatus == ServoState.SERVO_IDLE
                )
                {
                    Console.WriteLine(
                        "机器人和伺服器已空闲/Robot and servo are idle"
                    );
                    break;
                }

                Thread.Sleep(2000); // 等待2秒

                if (
                    System.DateTime.Now - startTime
                    > System.TimeSpan.FromSeconds(60)
                )
                {
                    Console.WriteLine(
                        "等待机器人和伺服器空闲超时/Waiting for robot and servo
idle timeout"
                    );
                    break;
                }
            } while (true);

            // [ZH] 执行离线轨迹
            // [EN] Execute offline trajectory
            code =
                controller.Trajectory.ExecuteOfflineTrajectory();
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "执行离线轨迹成功/Execute offline trajectory success"
                );
                Console.WriteLine(
                    "机器人开始执行轨迹程序/Robot started executing trajectory
program"
                );
            }
            else
            {
                Console.WriteLine(
```

```
                    $"执行离线轨迹失败/Execute offline trajectory failed: {cod
e.GetDescription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution/Except
ion occurred during execution: {ex.Message}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }

    /// <summary>
    /// 获取test_files文件夹中文件的路径示例方法
    /// 展示如何获取当前程序目录下的test_files文件夹中的文件路径
    /// </summary>
    private static string GetTestFilePath(string fileName)
    {
        // 获取当前程序集的目录
        string? codeFilePath =
            new System.Diagnostics.StackTrace(true)
                .GetFrame(0)
```

```
                ?.GetFileName();
            if (string.IsNullOrEmpty(codeFilePath))
            {
                throw new InvalidOperationException(
                    "无法获取当前文件路径/Cannot get current file path"
                );
            }

            string? codeDirectory = Path.GetDirectoryName(
                codeFilePath
            );
            if (string.IsNullOrEmpty(codeDirectory))
            {
                throw new InvalidOperationException(
                    "无法获取当前目录路径/Cannot get current directory path"
                );
            }

            // 构建test_files文件夹路径
            string testFilesDirectory = Path.Combine(
                codeDirectory,
                "test_files"
            );
            // 构建文件完整路径
            string filePath = Path.Combine(
                testFilesDirectory,
                fileName
            );
            return filePath;
        }
    }
```

4.7.6 开始记录轨迹

方法名	Trajectory.TrajectoryRecordBegin(string `name`)
描述	开始轨迹复现功能的轨迹记录

方法名	Trajectory.TrajectoryRecordBegin(string `name`)
请求参数	`name` : string 轨迹程序名
返回值	StatusCode: 记录开始操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.7.7 结束记录轨迹

方法名	Trajectory.TrajectoryRecordFinish(string `name`)
描述	结束轨迹复现功能的轨迹记录
请求参数	`name` : string 轨迹程序名
返回值	StatusCode: 记录结束操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.7.8 开始播放轨迹

方法名	Trajectory.TrajectoryReplayStart(string `name`)
描述	开始播放指定轨迹
请求参数	`name` : string 轨迹程序名
返回值	StatusCode: 播放开始操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.7.9 停止播放轨迹

方法名	Trajectory.TrajectoryReplayStop(string `name`)
描述	停止播放指定轨迹
请求参数	`name` : string 轨迹程序名
返回值	StatusCode: 播放停止操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.7.10 删除轨迹

方法名	Trajectory.TrajectoryRecordDelete(string `name`)
描述	删除指定轨迹
请求参数	`name` : string 轨迹程序名
返回值	StatusCode: 删除操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.7.11 获取轨迹列表

方法名	Trajectory.GetTrajectoryRecordList()
描述	获取当前已录制轨迹名称列表
请求参数	无参数
返回值	List<string>: 轨迹名称列表 StatusCode: 查询操作执行结果

方法名	Trajectory.GetTrajectoryRecordList()
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.7.12 获取轨迹起始位置

方法名	Trajectory.GetTrajectoryRecordStartPose(string `name`)
描述	获取指定录制轨迹的起始位置
请求参数	`name` : string 轨迹程序名
返回值	MotionPose: 轨迹起始位置 StatusCode: 查询操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.7.13 开始记录轨迹表 / 路径表

方法名	Trajectory.PathRecordBegin(string `name` , string `comment` , double `param` , double `angle` = 1)
描述	开始记录轨迹表（ `.traj` ）或路径表（ `.path` ）
请求参数	`name` : string 轨迹表 / 路径表文件名，必须以 `.path` 或 `.traj` 结尾 `comment` : string 描述信息 `param` : double 记录参数（ `.path` 为距离阈值，且需 >=0.5； `.traj` 为记录间隔，且需 >=2） `angle` : double 路径表旋转角度阈值（仅 `.path` 生效，需 >=1）
返回值	StatusCode: 记录开始操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.7.1.0+ 工业 (Bronze): 不支持

4.7.14 结束记录轨迹表 / 路径表

方法名	Trajectory.PathRecordFinish()
描述	结束当前轨迹表 / 路径表记录
请求参数	无参数
返回值	StatusCode: 记录结束操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.7.1.0+ 工业 (Bronze): 不支持

4.7.15 获取轨迹表 / 路径表起始姿态

方法名	Trajectory.GetPathStartPose(string `name`)
描述	获取指定轨迹表 / 路径表的起始姿态
请求参数	`name` : string 轨迹表 / 路径表名称
返回值	MotionPose: 起始姿态 StatusCode: 查询操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.7.1.0+ 工业 (Bronze): 不支持

4.7.16 获取轨迹表 / 路径表状态

方法名	Trajectory.GetPathState(List<string> `pathList`)
描述	批量查询轨迹表 / 路径表当前录制状态
请求参数	`pathList` : List<string> 轨迹表 / 路径表名称列表
返回值	Dictionary<string, MotionTableState>: 文件名与状态映射 StatusCode: 查询操作执行结果

方法名	Trajectory.GetPathState(List<string> pathList)
兼容的机器人软件版本	协作 (Copper): v7.7.1.0+ 工业 (Bronze): 不支持

4.7.17 设置路径规划参数

方法名	Trajectory.SetPathPlannerParameter(double transitionTime , double scalingFactor)
描述	设置路径规划器参数，影响轨迹过渡和平滑性
请求参数	transitionTime : double 启停过渡时长，需 >=0.2 数值越小加减速越快，最小 0.2 scalingFactor : double 缩放系数，范围 [0,1] =0，机器人将以恒定时间通过每个路径点，优先保证点与点之间时间间隔 =1，严格缩放，机器人以恒定线速度通过每个路径点，优先保证每个点匀速经过 在 0~1 之间则是折衷的行为
返回值	StatusCode: 设置操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.7.1.0+ 工业 (Bronze): 不支持

4.7.18 获取路径规划参数

方法名	Trajectory.GetPathPlannerParameter()
描述	获取当前路径规划器参数
请求参数	无参数
返回值	double: transitionTime 启停过渡时长，数值越小加减速越快，最小 0.2 double: scalingFactor 缩放系数，=0 时以恒定时间通过每个路径点，=1 时以恒定线速度通过每个路径点，0~1 之间为折衷行为 StatusCode: 查询操作执行结果

方法名	Trajectory.GetPathPlannerParameter()
兼容的机器人软件版本	协作 (Copper): v7.8.0.0+ 工业 (Bronze): 不支持

4.7.19 沿轨迹表 / 路径表运动

方法名	Trajectory.MovePath(string `name` , double `vel` = 100, double `acc` = 1)
描述	让机器人末端沿指定轨迹表 / 路径表运动
请求参数	`name` : string 轨迹表 / 路径表名称 `vel` : double 运动速度，范围 [0,5000] (mm/s) `acc` : double 加速度系数，范围 [0,1.2]
返回值	StatusCode: 运动操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.7.1.0+ 工业 (Bronze): 不支持

4.8 报警信息

概述

Alarm 模块提供机器人报警信息的读取、复位和查询功能，用于监控和处理机器人运行时可能出现的异常情况。

核心功能

- 支持报警复位
- 支持获取所有活动报警
- 支持获取最高优先级报警

使用场景

- 监控机器人运行状态，及时发现异常
- 处理机器人运行过程中的错误和报警
- 记录和分析机器人报警历史
- 实现自动化报警处理流程
- 集成到机器人监控系统中

4.8.1 获取最高优先级报警

方法名	Alarm.GetTopAlarm()
描述	获取当前最高优先级的报警
请求参数	无参数
返回值	string: 报警信息字符串 StatusCode: 函数执行结果

方法名	Alarm.GetTopAlarm()
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Alarm/GetTopAlarm.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Types;

public class GetTopAlarm
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
```

```
        {
            // [ZH] 获取最严重的一条报警
            // [EN] Get the most severe alarm
            string topError;
            (topError, code) =
                controller.Alarm.GetTopAlarm();
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "获取最严重报警成功/Get Top Alarm Success"
                );
                if (string.IsNullOrEmpty(topError))
                {
                    Console.WriteLine(
                        "当前无报警/No current alarms"
                    );
                }
                else
                {
                    Console.WriteLine(
                        $"最严重报警/Most Severe Alarm: {topError}"
                    );
                }
            }
            else
            {
                Console.WriteLine(
                    $"获取最严重报警失败/Get Top Alarm Failed: {code.GetDescrip
tion()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
```

```
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.8.2 获取所有活动报警

方法名	Alarm.GetAllActiveAlarms()
描述	获取所有当前活动的报警
请求参数	无参数
返回值	List<string>: 活动报警条目列表 StatusCode: 函数执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Alarm/GetAllActiveAlarms.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Types;
```

```
public class GetAllActiveAlarms
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            // [ZH] 获取所有的活动的报警
            // [EN] Get all active alarms
            List<string> errors;
            (errors, code) =
                controller.Alarm.GetAllActiveAlarms();
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "获取所有活动报警成功/Get All Active Alarm Success"
                );
                Console.WriteLine(
```

```
                    $"活动报警数量/Active Alarm Count: {errors.Count}"
                );

                if (errors.Count == 0)
                {
                    Console.WriteLine(
                        "当前无活动报警/No active alarms"
                    );
                }
                else
                {
                    Console.WriteLine(
                        "活动报警列表/Active Alarm List:"
                    );
                    for (int i = 0; i < errors.Count; i++)
                    {
                        Console.WriteLine(
                            $"  {i + 1}. {errors[i]}"
                        );
                    }
                }
            }
            else
            {
                Console.WriteLine(
                    $"获取所有活动报警失败/Get All Active Alarm Failed: {code.G
etDescription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
```

```
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.8.3 复位报警

方法名	Alarm.ResetAlarms()
描述	复位当前错误 / 报警
请求参数	无参数
返回值	StatusCode: 函数执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.9 文件服务类

概述

FileManager 用于在上位机和机器人控制器之间上传、下载、删除或搜索程序、轨迹及临时文件等资源，支持多种文件类型的管理和操作。

核心功能

- 支持本地文件上传到机器人控制器
- 支持机器人文件下载到本地目录
- 支持从机器人控制器删除文件
- 支持按文件名模式搜索文件
- 支持多种文件类型管理（UserProgram、BlockProgram、TrajectoryProgram、TmpFile）
- 支持上传 / 下载时的覆盖控制

使用场景

- 部署程序到机器人控制器
- 备份机器人上的程序和轨迹文件
- 同步调试产线数据
- 批量管理和查询机器人文件
- 上传临时数据文件到机器人
- 下载机器人日志或输出文件到本地

4.9.1 上传本地文件到机器人

方法名	FileManager.Upload(string `filePath` , FileType `ft` , bool `overWriting` = false)
描述	将本地文件上传到机器人控制器

方法名	FileManager.Upload(string `filePath` , FileType `ft` , bool `overWriting` = false)
请求参数	`filePath` : string 本地文件绝对路径。 `ft` : FileType 文件类型枚举值（UserProgram： `filePath` 为程序名路径，上传同目录 `.json` / `.xml` ；BlockProgram： `filePath` 为积木程序名路径，上传同目录 `.block` / `.json` / `.xml` ；TrajectoryProgram： `filePath` 为完整轨迹文件路径；TmpFile： `filePath` 为完整临时文件路径）。 `overWriting` : bool 是否覆盖同名文件（默认 `false` ）。
备注	`UserProgram` 、 `BlockProgram` 上传时请提供对应 `.xml` / `.block` 文件的完整路径，系统会同时上传同名 `.json` / `.xml` 文件。
返回值	StatusCode: 上传操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.9.2 下载机器人文件到本地

方法名	FileManager.Download(string `fileName` , FileType `ft` , string `savePath`)
描述	将机器人上的文件下载到本地目录。
请求参数	`fileName` : string 文件名。 `ft` : FileType 文件类型枚举值。 `savePath` : string 本地保存目录。
备注	`UserProgram` / `BlockProgram` / `TrajectoryProgram` 下载时仅填写程序名（不含后缀），系统会下载对应 `.json` / `.xml` 或 `.trajectory` 文件； `TmpFile` 请填写带后缀的完整文件名（如 `.csv` ）。
返回值	StatusCode: 下载操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.9.3 删除机器人上的文件

方法名	FileManager.Delete(string `fileName` , FileType `ft`)
描述	删除机器人控制器上的文件
请求参数	`fileName` : string 要删除的文件名 `ft` : FileType 要删除的文件类型
备注	`UserProgram` / `BlockProgram` / `TrajectoryProgram` 删除时仅填写程序名（不含后缀）； `TmpFile` 请填写带后缀的完整文件名。
返回值	StatusCode: 删除操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

FileManager/UserProgramOperations.cs

```cs
using System.Collections.Generic;
using System.IO;
using Agilebot.IR;
using Agilebot.IR.FileManager;
using Agilebot.IR.Types;

public class UserProgramOperations
{
    /// <summary>
    /// 测试用户程序文件的完整操作流程：上传、下载、搜索和删除
    /// </summary>
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
```

```
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            Console.WriteLine(
                "开始用户程序文件操作测试/Starting User Program File Operations
Test"
            );

            // [ZH] 获取测试文件路径
            // [EN] Get test file path
            string file_user_program = GetTestFilePath(
                "test_prog.xml"
            );
            string fileName = "test_prog";
            string save_path = GetTestFilePath("download");

            // [ZH] 上传用户程序文件
            // [EN] Upload user program file
            code = controller.FileManager.Upload(
                file_user_program,
                FileType.UserProgram,
                true
            );
            if (code == StatusCode.OK)
            {
```

```
                Console.WriteLine(
                    $"用户程序文件上传成功/User Program File Upload Success: {f
ileName}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"用户程序文件上传失败/User Program File Upload Failed: {co
de.GetDescription()}"
                );
                return code;
            }

            // [ZH] 等待下载
            // [EN] Wait before download
            Thread.Sleep(1000);

            // [ZH] 下载用户程序文件
            // [EN] Download user program file
            code = controller.FileManager.Download(
                fileName,
                FileType.UserProgram,
                save_path
            );
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    $"用户程序文件下载成功/User Program File Download Success:
{fileName}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"用户程序文件下载失败/User Program File Download Failed:
{code.GetDescription()}"
                );
                return code;
            }

            // [ZH] 搜索用户程序文件
```

```
            // [EN] Search user program file
            List<string> results = new List<string>();
            (results, code) = controller.FileManager.Search(
                fileName
            );
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    $"用户程序文件搜索成功/User Program File Search Success"
                );
                Console.WriteLine(
                    $"搜索结果数量/Search Results Count: {results.Count}"
                );
                foreach (var result in results)
                {
                    Console.WriteLine(
                        $"  找到文件/Found File: {result}"
                    );
                }
            }
            else
            {
                Console.WriteLine(
                    $"用户程序文件搜索失败/User Program File Search Failed: {co
de.GetDescription()}"
                );
                return code;
            }

            // [ZH] 等待删除
            // [EN] Wait before delete
            Thread.Sleep(1000);

            // [ZH] 删除用户程序文件
            // [EN] Delete user program file
            code = controller.FileManager.Delete(
                fileName,
                FileType.UserProgram
            );
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
```

```
                    $"用户程序文件删除成功/User Program File Delete Success: {f
ileName}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"用户程序文件删除失败/User Program File Delete Failed: {co
de.GetDescription()}"
                );
                return code;
            }

            Console.WriteLine(
                "用户程序文件操作测试完成/User Program File Operations Test Comp
leted"
            );
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }
```

```
        return code;
    }

    /// <summary>
    /// 获取test_files文件夹中文件的路径示例方法
    /// 展示如何获取当前程序目录下的test_files文件夹中的文件路径
    /// </summary>
    private static string GetTestFilePath(string fileName)
    {
        // [ZH] 获取当前程序集的目录
        // [EN] Get current assembly directory
        string? codeFilePath =
            new System.Diagnostics.StackTrace(true)
                .GetFrame(0)
                ?.GetFileName();
        if (string.IsNullOrEmpty(codeFilePath))
        {
            throw new InvalidOperationException(
                "无法获取当前文件路径/Cannot get current file path"
            );
        }

        string? codeDirectory = Path.GetDirectoryName(
            codeFilePath
        );
        if (string.IsNullOrEmpty(codeDirectory))
        {
            throw new InvalidOperationException(
                "无法获取当前目录路径/Cannot get current directory path"
            );
        }

        // [ZH] 构建test_files文件夹路径
        // [EN] Build test_files folder path
        string testFilesDirectory = Path.Combine(
            codeDirectory,
            "test_files"
        );
        // [ZH] 构建文件完整路径
        // [EN] Build complete file path
        string filePath = Path.Combine(
            testFilesDirectory,
```

```
                fileName
            );
            return filePath;
        }
    }
```

4.9.4 按文件名模式搜索文件

方法名	FileManager.Search(string `pattern` , ref List<string> `fl`)
描述	在机器人控制器上按文件名模式搜索文件。
请求参数	`pattern` : string 文件名匹配模式 `fl` : ref List<string> 返回的文件列表
返回值	StatusCode: 搜索操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.10 BasScript 脚本程序类

概述

BasScript 用于在上位机以结构化方式构建机器人中的 BAS 指令序列，涵盖运动、逻辑、程序调用、通信、视觉和 Modbus 等多种指令类型。

核心功能

- 支持构建运动指令（MoveJoint、MoveLine、MoveCircle 等）
- 支持构建逻辑指令（If、While、Switch、Goto 等）
- 支持构建赋值、等待、暂停和中断指令
- 支持构建程序调用和管理指令
- 支持构建 Socket 通信指令
- 支持构建 Modbus 寄存器读写指令
- 支持构建视觉程序指令
- 支持设置额外参数（加速度、RTCP、帧偏移等）
- 支持快速运动指令（JUMP 系列）

使用场景

- 自动化生成复杂的机器人程序
- 编辑和修改机器人程序
- 复用常用程序模块和指令序列
- 实现上位机与机器人程序的无缝对接
- 构建定制化的机器人运动轨迹
- 集成视觉、通信和 Modbus 功能到机器人程序中

类构造函数

方法名	BasScript(string `name`)
描述	构建用于机器人中的 BAS 指令序列类
请求参数	`name` : string 脚本程序名称
兼容的机器人软件版本	协作 (Copper): v7.5.2.0+ 工业 (Bronze): 不支持
备注	本类下所有方法的兼容版本要求与本类一致

子类结构

BasScript 类包含以下子类，用于组织不同类型的指令：

1. **ExtraParam**：额外参数类，用于构建复杂的机器人控制命令
2. **BasMotion**：运动指令子类，包含所有机器人运动相关的方法
3. **BasLogical**：逻辑指令子类，包含所有逻辑控制相关的方法
4. **BasStructure**：结构指令子类，包含所有结构控制相关的方法
5. **BasSocket**：Socket 通信子类，包含所有 Socket 通信相关的方法
6. **BasModbus**：Modbus 通信子类，包含所有 Modbus 通信相关的方法
7. **BasVision**：视觉指令子类，包含所有视觉相关的方法

4.10.1 ExtraParam 额外参数类

方法名	ExtraParam()
描述	额外参数类，用于构建复杂的机器人控制命令
请求参数	无
返回值	ExtraParam 对象

4.10.1.1 设置加速度

方法名	ExtraParam.Acceleration(double `value`)
描述	设置加速度，范围为 1~120%
请求参数	`value` : double 加速度值，单位：%（浮点数）
返回值	StatusCode: 参数设置执行结果

4.10.1.2 设置 RTCP 参数

方法名	ExtraParam.RTCP()
描述	设置 RTCP 参数，仅支持 MoveL 和 MoveC 指令
请求参数	无
返回值	StatusCode: 参数设置执行结果

4.10.1.3 设置帧偏移

方法名	ExtraParam.Offset(int `index`)
描述	设置帧偏移
请求参数	`index` : int 偏移索引（整数）
返回值	StatusCode: 参数设置执行结果

4.10.1.4 设置时间基准运行或赋值

方法名	ExtraParam.TB(double second , string type , string name)
描述	设置时间基准运行或赋值，范围为 0.01-30s，若输入小于 0.01 的数不会保存成功
请求参数	second : double 秒数，单位：sec（浮点数） type : string 执行类型（字符串） name : string 名称（用于运行，字符串）
返回值	StatusCode: 参数设置执行结果

方法名	ExtraParam.TB(double second , string type , int index , int status)
描述	设置时间基准运行或赋值，范围为 0.01-30s，若输入小于 0.01 的数不会保存成功
请求参数	second : double 秒数，单位：sec（浮点数） type : string IO 类型（字符串） index : int 索引（用于赋值，整数） status : int 状态（用于赋值，整数）
返回值	StatusCode: 参数设置执行结果

4.10.1.5 设置跳转

方法名	ExtraParam.SKIP(int index)
描述	设置跳转，当满足 SKIP CONDITION 指令设置的跳转条件时，从含有 SKIP 指令的当前行直接跳转到目标指令行或目标程序
请求参数	index : int 目标标签的索引（整数）
返回值	StatusCode: 参数设置执行结果

4.10.1.6 设置出发距离和接近距离

方法名	ExtraParam.Approach(double departureDist , double approachingDist)
描述	设置门型运动的出发距离和接近距离，仅用于 JUMP 指令
请求参数	departureDist : double 出发距离，单位：mm（浮点数） approachingDist : double 接近距离，单位：mm（浮点数）
返回值	StatusCode: 参数设置执行结果

4.10.2 BasMotion 运动指令子类

BasMotion 子类包含所有机器人运动相关的方法，用于构建运动指令序列。

4.10.2.1 MoveJoint 运动到点指令

方法名	BasScript.BasMotion.MoveJoint(MovePoseType poseType , int poseIndex , SpeedType speedType , double speedValue , SmoothType smoothType , double smoothDistance = 0, ExtraParam extraParam = null)
描述	关节运动指令，以指定的移动速度和移动方法使机器人向作业空间内的指定位置移动，对应机器人程序编写中的 MoveJoint 指令
请求参数	poseType : MovePoseType 位姿类型 poseIndex : int 位姿索引 speedType : SpeedType 速度类型 speedValue : double 速度值，单位：% smoothType : SmoothType 平滑类型 smoothDistance : double 平滑距离，单位：mm，取值范围：0~1000 extraParam : ExtraParam 附加参数
返回值	StatusCode: 运动指令执行结果

4.10.2.2 MoveLine 直线运动到点指令

方法名	**BasScript.BasMotion.MoveLine(MovePoseType** `poseType` **, int** `poseIndex` **, SpeedType** `speedType` **, double** `speedValue` **, SmoothType** `smoothType` **, double** `smoothDistance` **= 0, ExtraParam** `extraParam` **= null)**
描述	直线运动指令，以指定的移动速度和移动方法使机器人向作业空间内的指定位置直线移动，对应机器人程序编写中的 MoveLine 指令
请求参数	`poseType` : MovePoseType 位姿类型 `poseIndex` : int 位姿索引 `speedType` : SpeedType 速度类型 `speedValue` : double 速度值，单位：mm/s `smoothType` : SmoothType 平滑类型 `smoothDistance` : double 平滑距离，单位：mm，取值范围：0~1000 `extraParam` : ExtraParam 附加参数
返回值	StatusCode: 运动指令执行结果

4.10.2.3 MoveCircle 弧线运动到点指令

方法名	**BasScript.BasMotion.MoveCircle(MovePoseType** `poseType1` **, int** `poseIndex1` **, MovePoseType** `poseType2` **, int** `poseIndex2` **, SpeedType** `speedType` **, double** `speedValue` **, SmoothType** `smoothType` **, double** `smoothDistance` **= 0, ExtraParam** `extraParam` **= null)**
描述	圆弧运动指令，以指定的移动速度和移动方法使机器人向作业空间内的指定位置圆弧移动，对应机器人程序编写中的 MoveCircle 指令
请求参数	`poseType1` : MovePoseType 第一个位姿类型 `poseIndex1` : int 第一个位姿索引 `poseType2` : MovePoseType 第二个位姿类型 `poseIndex2` : int 第二个位姿索引 `speedType` : SpeedType 速度类型 `speedValue` : double 速度值，单位：mm/s `smoothType` : SmoothType 平滑类型

方法名	**BasScript.BasMotion.MoveCircle(MovePoseType** `poseType1` **, int** `poseIndex1` **, MovePoseType** `poseType2` **, int** `poseIndex2` **, SpeedType** `speedType` **, double** `speedValue` **, SmoothType** `smoothType` **, double** `smoothDistance` **= 0, ExtraParam** `extraParam` **= null)**
	`smoothDistance` : double 平滑距离，单位：mm，取值范围：0~1000 `extraParam` : ExtraParam 附加参数
返回值	StatusCode: 运动指令执行结果

4.10.2.4 Jump 点对点移动指令

方法名	**BasScript.BasMotion.Jump(MovePoseType** `poseType` **, int** `poseIndex` **, double** `speedValue` **, double** `speedRatio` **, SpeedType** `limZType` **, double** `limZValue` **, SmoothType** `smoothType` **, double** `smoothDistance` **= 0, ExtraParam** `extraParam` **= null)**
描述	门型运动指令，指定机器人做门型运动（首先垂直上升、然后水平移动，最后垂直下降），对应机器人程序编写中的 JUMP 指令
请求参数	`poseType` : MovePoseType 目标位姿存储类型 `poseIndex` : int 目标位置的索引 `speedValue` : double 移动速度的值，单位：mm/s `speedRatio` : double 移动速度的比率，单位：% `limZType` : SpeedType Z 轴限制的类型 `limZValue` : double Z 轴限制的值 `smoothType` : SmoothType 平滑类型 `smoothDistance` : double 平滑距离，单位：mm，取值范围：0~1000 `extraParam` : ExtraParam 额外参数
返回值	StatusCode: 运动指令执行结果

4.10.2.5 Jump3 三点跳跃指令

方法名	**BasScript.BasMotion.Jump3(MovePoseType** `poseType` **, int** `poseIndex` **, double** `speedValue` **, double** `speedRatio` **, SmoothType** `smoothType` **, double** `smoothDistance` **= 0, ExtraParam** `extraParam` **= null)**
描述	门型运动指令，指定机器人做门型运动，包含出发、接近和目标三个位置，对应机器人程序编写中的 JUMP3 指令
请求参数	`poseType` : MovePoseType 目标位姿存储类型 `poseIndex` : int 3 个目标位置的索引 `speedValue` : double 移动速度的值，单位：mm/s `speedRatio` : double 移动速度的比率，单位：% `smoothType` : SmoothType 平滑类型 `smoothDistance` : double 平滑距离，单位：mm，取值范围：0~1000 `extraParam` : ExtraParam 额外参数
返回值	StatusCode: 运动指令执行结果

4.10.2.6 Jump3CP 三点跳跃 CP 指令

方法名	**BasScript.BasMotion.Jump3CP(MovePoseType** `poseType` **, int** `poseIndex` **, double** `speedValue` **, SmoothType** `smoothType` **, double** `smoothDistance` **= 0, ExtraParam** `extraParam` **= null)**
描述	门型运动指令，指定机器人做门型运动，包含出发、接近和目标三个位置，对应机器人程序编写中的 JUMP3CP 指令
请求参数	`poseType` : MovePoseType 目标位姿存储类型 `poseIndex` : int 3 个目标位置的索引 `speedValue` : double 移动速度的值，单位：mm/s `smoothType` : SmoothType 平滑类型 `smoothDistance` : double 平滑距离，单位：mm，取值范围：0~1000 `extraParam` : ExtraParam 额外参数
返回值	StatusCode: 运动指令执行结果

4.10.3 BasLogical 逻辑指令子类

BasLogical 子类包含所有逻辑控制相关的方法，用于构建逻辑控制指令序列。

4.10.3.1 IF 条件指令

方法名	BasScript.BasLogical.IF(param1, index, param2, value, operatorType)
描述	添加一个逻辑 IF 语句到脚本中
请求参数	param1 : 第一个参数，类型为 RegisterType 或 IOType index : 索引（整数） param2 : 第二个参数，类型为 RegisterType、IOType 或 OtherType value : 值，类型为索引、数值、字符串或 IOStatus operatorType : 布尔操作符，默认为等于
返回值	StatusCode: 函数执行结果

4.10.3.2 ELSE_IF 条件分支指令

方法名	BasScript.BasLogical.ELSE_IF(param1, index, param2, value, operatorType)
描述	添加一个逻辑 ELSE IF 语句到脚本中
请求参数	param1 : 第一个参数，类型为 RegisterType 或 IOType index : 索引（整数） param2 : 第二个参数，类型为 RegisterType、IOType 或 OtherType value : 值，类型为索引、数值、字符串或 IOStatus operatorType : 布尔操作符，默认为等于
返回值	StatusCode: 函数执行结果

4.10.3.3 ELSE 否则指令

方法名	BasScript.BasLogical.ELSE()
描述	添加一个逻辑 ELSE 语句到脚本中
请求参数	无
返回值	StatusCode: 函数执行结果

4.10.3.4 END_IF 结束条件指令

方法名	BasScript.BasLogical.END_IF()
描述	结束逻辑 IF 语句
请求参数	无
返回值	StatusCode: 函数执行结果

示例代码

```cs
using Agilebot.IR;
using Agilebot.IR.Types;
using Agilebot.IR.BasScript;

public class Test
{
    public static async Task Main()
    {
        string controllerIP = "10.27.1.254";

        // 初始化捷勃特机器人
        Arm controller = new Arm(controllerIP);
        // 连接捷勃特机器人
        StatusCode code = await controller.Connect();
        Console.WriteLine(code != StatusCode.OK ? code.GetDescription() : "S
uccessfully connected.");

        // 生成脚本程序
        BasScript script = new BasScript("test");
```

```
            code = script.BasLogical.IF(RegisterType.R, 1, OtherType.VALUE, 1);
            code = script.BasMotion.MoveJoint(MovePoseType.PR, 1, SpeedType.VALU
E, 25, SmoothType.FINE);
            code = script.BasLogical.ELSE_IF(RegisterType.R, 1, OtherType.VALUE,
2);
            code = script.BasMotion.MoveJoint(MovePoseType.PR, 2, SpeedType.VALU
E, 50, SmoothType.FINE);
            code = script.BasLogical.ELSE();
            code = script.BasMotion.MoveJoint(MovePoseType.PR, 3, SpeedType.VALU
E, 50, SmoothType.FINE);
            code = script.BasLogical.END_IF();

            // 执行脚本程序
            code = controller.Execution.ExecuteBasScript(script);

            // 关闭连接
            code = controller.Disconnect();
            Console.WriteLine(code != StatusCode.OK ? code.GetDescription() : "S
uccessfully disconnected.");
        }
}
```

4.10.3.5 WHILE 循环指令

方法名	BasScript.BasLogical.WHILE(param1, index, param2, value, operatorType)
描述	添加一个逻辑 WHILE 语句到脚本中
请求参数	`param1`：第一个参数，类型为 RegisterType 或 IOType `index`：索引（整数） `param2`：第二个参数，类型为 RegisterType、IOType 或 OtherType `value`：值，类型为索引、数值、字符串或 IOStatus `operatorType`：布尔操作符，默认为等于
返回值	StatusCode: 函数执行结果

4.10.3.6 END_WHILE 结束循环指令

方法名	BasScript.BasLogical.END_WHILE()
描述	结束逻辑 While 语句
请求参数	无
返回值	StatusCode: 函数执行结果

示例代码

```cs
using Agilebot.IR;
using Agilebot.IR.Types;
using Agilebot.IR.BasScript;

public class Test
{
    public static async Task Main()
    {
        string controllerIP = "10.27.1.254";

        // 初始化捷勃特机器人
        Arm controller = new Arm(controllerIP);
        // 连接捷勃特机器人
        StatusCode code = await controller.Connect();
        Console.WriteLine(code != StatusCode.OK ? code.GetDescription() : "Successfully connected.");

        // 生成脚本程序
        BasScript script = new BasScript("test");
        code = script.BasLogical.WHILE(IOType.DO, 1, OtherType.IO_STATUS, IOStatus.ON);
        code = script.BasMotion.MoveJoint(MovePoseType.PR, 1, SpeedType.VALUE, 25, SmoothType.FINE);
        code = script.BasLogical.END_WHILE();

        // 执行脚本程序
        code = controller.Execution.ExecuteBasScript(script);
```

```
        // 关闭连接
        code = controller.Disconnect();
        Console.WriteLine(code != StatusCode.OK ? code.GetDescription() : "S
uccessfully disconnected.");
    }
}
```

4.10.3.7 SWITCH 多分支选择指令

方法名	BasScript.BasLogical.SWITCH(param, index)
描述	添加一个逻辑 SWITCH 语句到脚本中
请求参数	`param` : 参数，类型为 RegisterType 或 IOType `index` : 参数的索引
返回值	StatusCode: 函数执行结果

4.10.3.8 CASE 分支指令

方法名	BasScript.BasLogical.CASE(param, value)
描述	添加一个逻辑 CASE 语句到脚本中
请求参数	`param` : 参数，类型为 RegisterType、IOType 或 OtherType `value` : 值，类型为索引、数值、字符串
返回值	StatusCode: 函数执行结果

4.10.3.9 DEFAULT 默认分支指令

方法名	BasScript.BasLogical.DEFAULT()
描述	添加一个逻辑 DEFAULT 语句到脚本中

方法名	BasScript.BasLogical.DEFAULT()
请求参数	无
返回值	StatusCode: 函数执行结果

4.10.3.10 END_SWITCH 结束多分支选择指令

方法名	BasScript.BasLogical.END_SWITCH()
描述	结束逻辑 SWITCH 语句
请求参数	无
返回值	StatusCode: 函数执行结果

4.10.3.11 SKIP_CONDITION 跳过条件指令

方法名	BasScript.BasLogical.SKIP_CONDITION(param1, index, param2, value, operatorType)
描述	添加一个逻辑 SKIP CONDITION 语句到脚本中
请求参数	param1 : 第一个参数，类型为 RegisterType 或 IOType index : 参数一的索引 param2 : 第二个参数，类型为 RegisterType、IOType 或 OtherType value : 值，类型为索引、数值、字符串或 IOStatus operatorType : 布尔操作符，默认为等于
返回值	StatusCode: 函数执行结果

4.10.3.12 GOTO 跳转指令

方法名	BasScript.BasLogical.GOTO(index)
描述	GOTO 跳转语句
请求参数	`index` : 目标标签的索引
返回值	StatusCode: 函数执行结果

4.10.3.13 LABEL 标签指令

方法名	BasScript.BasLogical.LABEL(index)
描述	LABEL 语句
请求参数	`index` : 标签的索引
返回值	StatusCode: 函数执行结果

4.10.3.14 BREAK 跳出循环指令

方法名	BasScript.BasLogical.BREAK()
描述	BREAK 语句
请求参数	无
返回值	StatusCode: 函数执行结果

4.10.3.15 CONTINUE 跳过循环指令

方法名	BasScript.BasLogical.CONTINUE()
描述	CONTINUE 语句
请求参数	无

方法名	BasScript.BasLogical.CONTINUE()
返回值	StatusCode: 函数执行结果

4.10.4 BasStructure 结构指令子类

BasStructure 子类包含所有结构控制相关的方法，用于构建结构控制指令序列。

4.10.4.1 WAIT 等待条件指令

方法名	BasScript.BasStructure.WAIT(RegisterType `param1` , int `index` , ValueType `param2` , object `value` , BooleanOperator `operatorType` = BooleanOperator.EQ)
描述	等待条件指令，执行 WAIT 指令只有当条件满足时，程序才能继续向下执行，否则一直等待直到条件满足为止，对应机器人程序编写中的 WAIT COND 等待条件语句
请求参数	`param1` : RegisterType 第一个参数（寄存器或 IO 信号） `index` : int 参数 1 的索引 `param2` : ValueType 第二个参数（寄存器、IO 信号或其他类型） `value` : object 参数 2 的索引或值 `operatorType` : BooleanOperator 逻辑运算符，默认为等于
返回值	StatusCode: 函数执行结果

4.10.4.2 WAIT_TIME 等待时间指令

方法名	BasScript.BasStructure.WAIT_TIME(ValueType `param` , double `value`)
描述	等待时间指令，执行 WAIT TIME 指令，机器人等待指定的时间后继续执行后续指令，对应机器人程序编写中的 WAIT TIME 等待时间语句

方法名	BasScript.BasStructure.WAIT_TIME(ValueType `param` , double `value`)
请求参数	`param` : ValueType 参数类型（R 寄存器或数值） `value` : double 时间值，单位：sec
返回值	StatusCode: 函数执行结果

4.10.4.3 PAUSE 暂停指令

方法名	BasScript.BasStructure.PAUSE()
描述	PAUSE 语句
请求参数	无
返回值	StatusCode: 函数执行结果

4.10.4.4 ABORT 中断指令

方法名	BasScript.BasStructure.ABORT()
描述	ABORT 语句
请求参数	无
返回值	StatusCode: 函数执行结果

4.10.4.5 CALL 同步调用程序指令

方法名	BasScript.BasStructure.CALL(name)
描述	CALL 同步调用程序

方法名	BasScript.BasStructure.CALL(name)
请求参数	`name` : 程序名
返回值	StatusCode: 函数执行结果

4.10.4.6 RUN 异步调用程序指令

方法名	BasScript.BasStructure.RUN(name)
描述	RUN 异步调用程序
请求参数	`name` : 程序名
返回值	StatusCode: 函数执行结果

4.10.4.7 LOAD 加载程序指令

方法名	BasScript.BasStructure.LOAD(param, value)
描述	LOAD 载入程序
请求参数	`param` : 参数，R 寄存器、SR 寄存器、数值或字符串 `value` : 参数的值，数值或字符串
返回值	StatusCode: 函数执行结果

4.10.4.8 UNLOAD 卸载程序指令

方法名	BasScript.BasStructure.UNLOAD(param, value)
描述	UNLOAD 卸载程序

方法名	BasScript.BasStructure.UNLOAD(param, value)
请求参数	`param` : 参数，R 寄存器、SR 寄存器、数值或字符串 `value` : 参数的值，数值或字符串
返回值	StatusCode: 函数执行结果

4.10.4.9 EXEC 执行程序指令

方法名	BasScript.BasStructure.EXEC(param, value)
描述	EXEC 执行程序
请求参数	`param` : 参数，R 寄存器、SR 寄存器、数值或字符串 `value` : 参数的值，数值或字符串
返回值	StatusCode: 函数执行结果

4.10.5 BasSocket Socket 通信子类

BasSocket 子类包含所有 Socket 通信相关的方法，用于构建 Socket 通信指令序列。

4.10.5.1 OPEN 打开 socket 连接指令

方法名	BasScript.BasSocket.OPEN(int `index`)
描述	使用 Socket Open 指令创建一个 Server 并等待 Client 连接，对应机器人程序编写中的 SOCKET OPEN 打开 socket 连接
请求参数	`index` : int SK 寄存器序号
返回值	StatusCode: 函数执行结果

4.10.5.2 CLOSE 关闭 socket 连接指令

方法名	BasScript.BasSocket.CLOSE(int `index`)
描述	关闭指定的 socket 连接，对应机器人程序编写中的 SOCKET CLOSE 关闭 socket 连接
请求参数	`index` : int SK 寄存器序号
返回值	StatusCode: 函数执行结果

4.10.5.3 CONNECT 连接 socket 指令

方法名	BasScript.BasSocket.CONNECT(int `index`)
描述	使用 Socket Connect 指令连接到指定的 socket 服务器，对应机器人程序编写中的 SOCKET CONNECT 连接 socket
请求参数	`index` : int SK 寄存器序号
返回值	StatusCode: 函数执行结果

4.10.5.4 SEND 发送 socket 数据指令

方法名	BasScript.BasSocket.SEND(int `index` , StrType `msgType` , object `value`)
描述	使用 Socket Send 指令向指定的 socket 连接发送数据，对应机器人程序编写中的 SOCKET SEND 发送数据
请求参数	`index` : int SK 寄存器序号 `msgType` : StrType 消息类型 `value` : object 消息内容或序号
返回值	StatusCode: 函数执行结果

4.10.5.5 RECV 接收 socket 数据指令

方法名	**BasScript.BasSocket.RECV(int** `index` **, int** `msgLength` **, StrType** `msgType` **, object** `value` **)**
描述	使用 Socket Recv 指令从指定的 socket 连接读取字符，可以指定最大长度，对应机器人程序编写中的 SOCKET RECV 接收数据
请求参数	`index` : int SK 寄存器序号 `msgLength` : int 消息最大长度 `msgType` : StrType 消息类型 `value` : object 消息内容或序号
返回值	StatusCode: 函数执行结果

4.10.6 BasModbus Modbus 通信子类

BasModbus 子类包含所有 Modbus 通信相关的方法，用于构建 Modbus 寄存器读写指令序列。

4.10.6.1 READ_MH 读取 Modbus 保持寄存器指令

方法名	**BasScript.BasModbus.READ_MH(int** `index` **, int** `id` **, int** `address` **, int** `length` **, int** `rIndex` **)**
描述	读取 Modbus 保持寄存器指令，对应机器人程序编写中的 READ_MH 读取 Modbus 保持寄存器
请求参数	`index` : int 通道序号 `id` : int Modbus ID `address` : int 寄存器起始地址 `length` : int 寄存器长度，即要读取的寄存器数量 `rIndex` : int 写入结果的 R 寄存器起始序号
返回值	StatusCode: 函数执行结果

4.10.6.2 READ_MI 读取 Modbus 输入寄存器指令

方法名	**BasScript.BasModbus.READ_MI(int** `index` **, int** `id` **, int** `address` **, int** `length` **, int** `rIndex` **)**
描述	读取 Modbus 输入寄存器指令，对应机器人程序编写中的 READ_MI 读取 Modbus 输入寄存器
请求参数	`index` : int 通道序号 `id` : int Modbus ID `address` : int 寄存器起始地址 `length` : int 寄存器长度，即要读取的寄存器数量 `rIndex` : int 写入结果的 R 寄存器起始序号
返回值	StatusCode: 函数执行结果

4.10.6.3 WRITE_MH 写入 Modbus 保持寄存器指令

方法名	**BasScript.BasModbus.WRITE_MH(int** `index` **, int** `id` **, int** `address` **, int** `length` **, ValueType** `valueType` **, int** `value` **)**
描述	写入 Modbus 保持寄存器指令，对应机器人程序编写中的 WRITE_MH 写入 Modbus 保持寄存器
请求参数	`index` : int 通道序号 `id` : int Modbus ID `address` : int 寄存器起始地址 `length` : int 寄存器长度，即要写入的寄存器数量 `valueType` : ValueType 值类型 `value` : int 值或 R 寄存器起始索引
返回值	StatusCode: 函数执行结果

4.10.7 BasVision 视觉指令子类

BasVision 子类包含所有视觉相关的方法，用于构建视觉程序指令序列。

4.10.7.1 FIND 寻找视觉程序指令

方法名	BasScript.BasVision.FIND(string `name`)
描述	执行视觉寻找程序，对应机器人程序编写中的 VISION FIND 寻找视觉程序
请求参数	`name` : string 视觉程序名称
返回值	StatusCode: 函数执行结果

4.10.7.2 GET_OFFSET 获取视觉程序偏移量指令

方法名	BasScript.BasVision.GET_OFFSET(string `name` , int `index` , int `labelIndex`)
描述	获取视觉程序执行后的偏移量，对应机器人程序编写中的 VISION GET OFFSET 获取视觉程序偏移量
请求参数	`name` : string 视觉程序名称 `index` : int 视觉寄存器索引 `labelIndex` : int 标签索引
返回值	StatusCode: 函数执行结果

4.10.7.3 GET_QUANTITY 获取视觉程序结果指令

方法名	BasScript.BasVision.GET_QUANTITY(string `name` , int `index`)
描述	获取视觉程序执行后的结果数量，对应机器人程序编写中的 VISION GET QUANTITY 获取视觉程序结果
请求参数	`name` : string 视觉程序名称 `index` : int 结果存储的 R 寄存器索引

方法名	BasScript.BasVision.GET_QUANTITY(string `name` , int `index`)
返回值	StatusCode: 函数执行结果

4.10.8 AssignValue 赋值指令（完整参数）

方法名	BasScript.AssignValue(AssignType `param1` , int `index` , AssignType `param2` , object `value` , int `optIndex` = 0, int `optValue` = 0)
描述	赋值指令，用于给寄存器、IO 信号等赋值，对应机器人程序编写中的 ASSIGN 赋值语句
请求参数	`param1` : AssignType 第一个参数（寄存器或 IO 信号） `index` : int 参数 1 的索引 `param2` : AssignType 第二个参数（寄存器、IO 信号或其他类型） `value` : object 参数 2 的索引或值 `optIndex` : int 参数 1 为 PR_ELEMENT 时的额外索引，默认 0 `optValue` : int 参数 2 为 PR_ELEMENT 时的额外索引或 value 为 IOStatus.PULSE 时的脉冲值，默认 0
返回值	StatusCode: 赋值指令执行结果

4.10.9 AssignValue 赋值指令（简化参数）

方法名	BasScript.AssignValue(AssignType `param` , int `index` , object `value`)
描述	赋值指令（简化参数），用于给寄存器、IO 信号等赋值，对应机器人程序编写中的 ASSIGN 赋值语句
请求参数	`param` : AssignType 参数类型（寄存器或 IO 信号） `index` : int 参数索引 `value` : object 值（IOStatus、double 或 string）
返回值	StatusCode: 赋值操作执行结果

示例代码

```cs
using Agilebot.IR;
using Agilebot.IR.Types;
using Agilebot.IR.BasScript;

public class Test
{
    public static async Task Main()
    {
        string controllerIP = "10.27.1.254";

        // 初始化捷勃特机器人
        Arm controller = new Arm(controllerIP);
        // 连接捷勃特机器人
        StatusCode code = await controller.Connect();
        Console.WriteLine(code != StatusCode.OK ? code.GetDescription() : "S
uccessfully connected.");

        // 生成脚本程序
        BasScript script = new BasScript("test");
        code = script.BasMotion.MoveJoint(MovePoseType.PR, 1, SpeedType.VALU
E, 25, SmoothType.FINE);
        BasScript.ExtraParam param = new();
        param.Acceleration(80);
        code = script.BasMotion.MoveJoint(MovePoseType.PR, 2, SpeedType.VALU
E, 50, SmoothType.FINE, extraParam: param);

        // 执行脚本程序
        code = controller.Execution.ExecuteBasScript(script);

        // 关闭连接
        code = controller.Disconnect();
        Console.WriteLine(code != StatusCode.OK ? code.GetDescription() : "S
uccessfully disconnected.");
    }
}
```

4.10.10 SetParam 设置参数指令

方法名	**BasScript.SetParam(ParamType** `type` **, ValueType** `valueType` **, object** `value` **)**
描述	设置参数指令，用于设置机器人的各种参数，对应机器人程序编写中的 SET PARAM 设置参数
请求参数	`type` : ParamType 参数类型 `valueType` : ValueType 值类型 `value` : object 值
返回值	StatusCode: 函数执行结果

4.11 坐标系类

概述

CoordinateSystem 类用于管理机器人用户坐标系（UF）与工具坐标系（TF），提供新增、删除、更新、查询和计算坐标系的统一接口。

核心功能

- 支持获取用户 / 工具坐标系摘要信息列表
- 支持添加、删除、更新和查询用户 / 工具坐标系
- 支持根据输入的多组位姿计算用户 / 工具坐标系
- 提供统一的坐标系管理接口，便于维护控制器中的坐标系列表

使用场景

- 工具切换时管理不同工具坐标系
- 多工位调试时保持一致的空间参考
- 批量管理和查询机器人坐标系
- 根据示教点快速计算新的用户 / 工具坐标系

4.11.1 获取用户 / 工具坐标系列表

方法名	**CoordinateSystem.GetCoordinateList(CoordinateType `type`)**
描述	根据指定的坐标系类型，获取所有坐标系摘要信息列表
请求参数	`type` : CoordinateType 坐标系类型（UF 或 TF）
返回值	List<CoordSummary>: 坐标系摘要信息列表 StatusCode: 获取操作执行结果

方法名	CoordinateSystem.GetCoordinateList(CoordinateType `type`)
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.11.2 添加用户 / 工具坐标系

方法名	CoordinateSystem.Add(CoordinateType `type` , Coordinate `coordinate`)
描述	根据指定的坐标系类型，添加一个新的坐标系
请求参数	`type` : CoordinateType 坐标系类型（UF 或 TF） `coordinate` : Coordinate 要添加的坐标系信息
返回值	StatusCode: 添加操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.11.3 删除用户 / 工具坐标系

方法名	CoordinateSystem.Delete(CoordinateType `type` , int `index`)
描述	根据指定的坐标系类型和索引，删除对应坐标系
请求参数	`type` : CoordinateType 坐标系类型（UF 或 TF） `index` : int 坐标系索引
返回值	StatusCode: 删除操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.11.4 更新用户 / 工具坐标系

方法名	CoordinateSystem.Update(CoordinateType `type` , Coordinate `coordinate`)
描述	根据指定的坐标系类型和坐标系信息，更新对应坐标系
请求参数	`type` : CoordinateType 坐标系类型（UF 或 TF） `coordinate` : Coordinate 要更新的坐标系信息
返回值	StatusCode: 更新操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.11.5 获取用户 / 工具坐标系信息

方法名	CoordinateSystem.Get(CoordinateType `type` , int `index`)
描述	根据指定的坐标系类型和索引，获取对应坐标系信息
请求参数	`type` : CoordinateType 坐标系类型（UF 或 TF） `index` : int 坐标系索引
返回值	Coordinate: 坐标系信息数据 StatusCode: 获取操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

4.11.6 计算用户 / 工具坐标系

方法名	CoordinateSystem.Calculate(CoordinateType `type` , List<Position> `pose`)
描述	根据输入的位姿点集，计算用户 / 工具坐标系
请求参数	`type` : CoordinateType 坐标系类型（UF 或 TF） `pose` : List<Position> 输入位姿列表，支持 4 点法或 7 点法。4 点法为 4 组工具指向同一点的位姿；7 点法在此基础上增加坐标原点、X 方向点和 Y 方向点。角度单位为度（°）
返回值	Position: 计算后的坐标系位姿 StatusCode: 计算操作执行结果
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

CoordinateSystem/TFCoordinateTest.cs

```cs
using Agilebot.IR;
using Agilebot.IR.CoordinateSystem;
using Agilebot.IR.Types;

public class TFCoordinateTest
{
    /// <summary>
    /// 测试 TF 坐标系的计算、添加、获取列表、获取单个坐标系、更新和删除操作
    /// </summary>
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );
```

```
// [ZH] 连接捷勃特机器人
// [EN] Connect to the Agilebot robot
StatusCode code = controller.ConnectSync();
Console.WriteLine(
    code != StatusCode.OK
        ? code.GetDescription()
        : "连接成功/Successfully connected."
);

if (code != StatusCode.OK)
{
    return code;
}

try
{
    // [ZH] 准备测试数据
    // [EN] Prepare test data
    var poseData = new List<Position>
    {
        new Position(
            847.0999429718556,
            166.7999999999656,
            276.8195498896624,
            90,
            0,
            -70
        ),
        new Position(
            809.0227439212846,
            166.79999999994843,
            459.80354972094295,
            90,
            0,
            -45
        ),
        new Position(
            717.1223240422377,
            166.79999999993265,
            654.0891675073312,
            90,
            0,
```

```
                    -30
                ),
                new Position(
                    572.917828754028,
                    166.79999999992168,
                    825.1862002007621,
                    90,
                    0,
                    -40
                ),
            };

            Console.WriteLine(
                "开始TF坐标系测试/Starting TF Coordinate Test"
            );

            // [ZH] 计算坐标系
            // [EN] Calculate coordinate system
            Coordinate calculatedCoord = new Coordinate();
            (Position coord, StatusCode calculateCode) =
                controller.CoordinateSystem.Calculate(
                    CoordinateType.ToolCoordinate,
                    poseData
                );

            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "计算TF坐标系成功/Calculate TF Coordinate Success"
                );
            }
            else
            {
                Console.WriteLine(
                    $"计算TF坐标系失败/Calculate TF Coordinate Failed: {code.G
etDescription()}"
                );
                return code;
            }
            calculatedCoord.Id = 5;
            calculatedCoord.Data = coord;
```

```
            // [ZH] 删除可能存在的坐标系
            // [EN] Delete existing coordinate if exists
            StatusCode deleteCode =
                controller.CoordinateSystem.Delete(
                    CoordinateType.ToolCoordinate,
                    calculatedCoord.Id
                );
            Console.WriteLine(
                $"删除现有坐标系/Delete Existing Coordinate: {deleteCode.GetDe
scription()}"
            );

            // [ZH] 添加坐标系
            // [EN] Add coordinate system
            StatusCode addCode =
                controller.CoordinateSystem.Add(
                    CoordinateType.ToolCoordinate,
                    calculatedCoord
                );
            if (addCode == StatusCode.OK)
            {
                Console.WriteLine(
                    "添加TF坐标系成功/Add TF Coordinate Success"
                );
            }
            else
            {
                Console.WriteLine(
                    $"添加TF坐标系失败/Add TF Coordinate Failed: {addCode.GetD
escription()}"
                );
                return addCode;
            }

            // [ZH] 获取坐标系列表
            // [EN] Get coordinate list
            List<CoordSummary> listRes;
            (listRes, code) =
                controller.CoordinateSystem.GetCoordinateList(
                    CoordinateType.ToolCoordinate
                );
            if (code == StatusCode.OK)
```

```
            {
                Console.WriteLine(
                    "获取TF坐标系列表成功/Get TF Coordinate List Success"
                );
                Console.WriteLine(
                    $"坐标系列表数量/Coordinate List Count: {listRes.Count}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"获取TF坐标系列表失败/Get TF Coordinate List Failed: {cod
e.GetDescription()}"
                );
                return code;
            }

            // [ZH] 获取单个坐标系
            // [EN] Get single coordinate
            Coordinate getCoord;
            (getCoord, code) =
                controller.CoordinateSystem.Get(
                    CoordinateType.ToolCoordinate,
                    calculatedCoord.Id
                );
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "获取TF坐标系成功/Get TF Coordinate Success"
                );
                Console.WriteLine(
                    $"坐标系名称/Coordinate Name: {getCoord.Name}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"获取TF坐标系失败/Get TF Coordinate Failed: {code.GetDesc
ription()}"
                );
                return code;
            }
```

```
            // [ZH] 更新坐标系
            // [EN] Update coordinate system
            getCoord.Name = "test";
            StatusCode updateCode =
                controller.CoordinateSystem.Update(
                    CoordinateType.ToolCoordinate,
                    getCoord
                );
            if (updateCode == StatusCode.OK)
            {
                Console.WriteLine(
                    "更新TF坐标系成功/Update TF Coordinate Success"
                );
            }
            else
            {
                Console.WriteLine(
                    $"更新TF坐标系失败/Update TF Coordinate Failed: {updateCod
e.GetDescription()}"
                );
                return updateCode;
            }

            // [ZH] 删除坐标系
            // [EN] Delete coordinate system
            deleteCode = controller.CoordinateSystem.Delete(
                CoordinateType.ToolCoordinate,
                calculatedCoord.Id
            );
            if (deleteCode == StatusCode.OK)
            {
                Console.WriteLine(
                    "删除TF坐标系成功/Delete TF Coordinate Success"
                );
            }
            else
            {
                Console.WriteLine(
                    $"删除TF坐标系失败/Delete TF Coordinate Failed: {deleteCod
e.GetDescription()}"
                );
```

```
                return deleteCode;
            }

            Console.WriteLine(
                "TF坐标系测试完成/TF Coordinate Test Completed"
            );
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.12 机器人示教运动

概述

Jogging 类提供机器人在示教模式下的点动控制接口，支持单轴步进、单轴连续点动、多轴联动及停止操作。

核心功能

- 支持机器人单轴步进示教运动
- 支持机器人单轴连续点动示教运动
- 支持机器人多轴连续示教运动
- 支持停止机器人连续示教运动
- 支持通过方向符号控制正向 / 反向点动
- 支持设置步进长度和旋转步进角度

使用场景

- 机器人调试和示教过程
- 位置微调和姿态修正
- 上位机远程手动调姿
- 机器人安装和校准
- 复杂轨迹执行前的定位确认

4.12.1 单轴步进示教运动

方法名	**Jogging.Move(int `ajNum` , MoveMode `moveMode` , double `stepLength` = 0, double `stepAngle` = 0)**
描述	控制机器人按设定步进量进行单轴示教运动（`moveMode` 取 `MoveMode.Stepping` ）

方法名	**Jogging.Move(int** `ajNum` **, MoveMode** `moveMode` **, double** `stepLength` **= 0, double** `stepAngle` **= 0)**
请求参数	`ajNum` : int 轴编号（1~6 对应当前坐标系各轴；笛卡尔坐标系下 x/y/z/rx/ry/rz 对应 1~6；数值正负表示运动方向） `moveMode` : MoveMode 运动模式，单轴步进场景固定为 `MoveMode.Stepping` `stepLength` : double 单次直线步进长度，单位 mm（步进模式下生效） `stepAngle` : double 单次旋转步进角度，单位 °（步进模式下生效）
返回值	StatusCode: 返回点动是否成功的状态码
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Jogging/StepJogging.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Types;

public class StepJogging
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
```

```
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            // [ZH] 获取机器人模式
            // [EN] Get robot mode
            (UserOpMode opMode, StatusCode opCode) =
                controller.GetOpMode();
            if (opCode == StatusCode.OK)
            {
                Console.WriteLine(
                    $"当前机器人模式/Current robot mode: {opMode}"
                );
                if (
                    opMode != UserOpMode.UNLIMITED_MANUAL
                    && opMode != UserOpMode.LIMIT_MANUAL
                )
                {
                    Console.WriteLine(
                        $"示教运动必须在机器人手动模式下/Jogging must be in manua
l mode"
                    );
                    return StatusCode.OtherReason;
                }
            }
            else
            {
                Console.WriteLine(
                    $"获取机器人模式失败/Failed to get robot mode: {opCode.GetD
escription()}"
                );
            }

            // [ZH] 设置单步示教运动参数
            // [EN] Set step jogging parameters
            int ajNum = 1; // 轴序号，正数表示正方向运动
```

```
            MoveMode moveMode = MoveMode.Stepping; // 单步运动模式
            double stepLength = 5.0; // 步长，单位为mm或角度
            double stepAngle = 5.0; // 轴旋转角度，单位为角度

            Console.WriteLine(
                "开始单步示教运动/Starting Step Jogging"
            );
            Console.WriteLine(
                $"轴序号/Axis Number: {ajNum}"
            );
            Console.WriteLine(
                $"运动模式/Move Mode: {moveMode}"
            );
            Console.WriteLine(
                $"步长/Step Length: {stepLength}"
            );

            // [ZH] 执行单步示教运动
            // [EN] Execute step jogging movement
            code = controller.Jogging.Move(
                ajNum,
                moveMode,
                stepLength,
                stepAngle
            );
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "单步示教运动执行成功/Step Jogging Executed Successfully"
                );
                Console.WriteLine(
                    $"轴{ajNum}向正方向移动{stepLength}单位/Axis {ajNum} moved
{stepLength} units in positive direction"
                );
            }
            else
            {
                Console.WriteLine(
                    $"单步示教运动执行失败/Step Jogging Execution Failed: {cod
e.GetDescription()}"
                );
            }
```

```
            // [ZH] 等待一秒后执行反向运动
            // [EN] Wait one second then execute reverse movement
            Thread.Sleep(1000);

            // [ZH] 执行反向单步运动
            // [EN] Execute reverse step movement
            int reverseAjNum = -ajNum; // 负数表示负方向运动
            code = controller.Jogging.Move(
                reverseAjNum,
                moveMode,
                stepLength,
                stepAngle
            );
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "反向单步示教运动执行成功/Reverse Step Jogging Executed Succ
essfully"
                );
                Console.WriteLine(
                    $"轴{Math.Abs(reverseAjNum)}向负方向移动{stepLength}单位/Ax
is {Math.Abs(reverseAjNum)} moved {stepLength} units in negative direction"
                );
            }
            else
            {
                Console.WriteLine(
                    $"反向单步示教运动执行失败/Reverse Step Jogging Execution Fa
iled: {code.GetDescription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
```

```
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.12.2 单轴连续点动示教运动

方法名	**Jogging.Move(int** `ajNum` **, MoveMode** `moveMode` **, double** `stepLength` **= 0, double** `stepAngle` **= 0)**
描述	控制机器人进行单轴连续点动示教运动（ `moveMode` 取 `MoveMode.Continuous` ）
请求参数	`ajNum` : int 轴编号（1~6 对应当前坐标系各轴；笛卡尔坐标系下 x/y/z/rx/ry/rz 对应 1~6；数值正负表示运动方向） `moveMode` : MoveMode 运动模式，单轴连续点动场景固定为 `MoveMode.Continuous` `stepLength` : double 步长参数（连续模式下通常不使用） `stepAngle` : double 步角参数（连续模式下通常不使用）
返回值	StatusCode: 返回点动是否成功的状态码
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Jogging/ContinuousJogging.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Types;

public class ContinuousJogging
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            // [ZH] 获取机器人模式
            // [EN] Get robot mode
            (UserOpMode opMode, StatusCode opCode) =
                controller.GetOpMode();
            if (opCode == StatusCode.OK)
            {
```

```
                Console.WriteLine(
                    $"当前机器人模式/Current robot mode: {opMode}"
                );
                if (
                    opMode != UserOpMode.UNLIMITED_MANUAL
                    && opMode != UserOpMode.LIMIT_MANUAL
                )
                {
                    Console.WriteLine(
                        $"示教运动必须在机器人手动模式下/Jogging must be in manua
l mode"
                    );
                    return StatusCode.OtherReason;
                }
            }
            else
            {
                Console.WriteLine(
                    $"获取机器人模式失败/Failed to get robot mode: {opCode.GetD
escription()}"
                );
            }

            // [ZH] 设置示教运动参数
            // [EN] Set jogging parameters
            int ajNum = 3; // 轴序号，正数表示正方向运动
            MoveMode moveMode = MoveMode.Continuous; // 连续运动模式

            Console.WriteLine(
                "开始连续示教运动/Starting Continuous Jogging"
            );
            Console.WriteLine(
                $"轴序号/Axis Number: {ajNum}"
            );
            Console.WriteLine(
                $"运动模式/Move Mode: {moveMode}"
            );

            // [ZH] 启动连续示教运动
            // [EN] Start continuous jogging movement
            code = controller.Jogging.Move(ajNum, moveMode);
            if (code == StatusCode.OK)
```

```
            {
                Console.WriteLine(
                    "连续示教运动启动成功/Continuous Jogging Started Successful
ly"
                );
                Console.WriteLine(
                    "运动3秒后自动停止/Moving for 3 seconds then auto stop"
                );

                // [ZH] 运动3秒
                // [EN] Move for 3 seconds
                Thread.Sleep(3000);

                // [ZH] 停止示教运动
                // [EN] Stop jogging movement
                controller.Jogging.Stop();
                Console.WriteLine(
                    "示教运动已停止/Jogging Movement Stopped"
                );
            }
            else
            {
                Console.WriteLine(
                    $"连续示教运动启动失败/Continuous Jogging Start Failed: {co
de.GetDescription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
```

```
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.12.3 多轴连续示教运动

方法名	Jogging.MultiMove(int[] ajNums)
描述	控制机器人多轴同时进行连续示教运动
请求参数	ajNums : int [] 轴编号列表（1~6 对应当前坐标系各轴；笛卡尔坐标系下 x/y/z/rx/ry/rz 对应 1~6；数值正负表示各轴运动方向）
返回值	StatusCode: 返回点动是否成功的状态码
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Jogging/MultiJogging.cs

```cs
using Agilebot.IR;
using Agilebot.IR.Jogging;
using Agilebot.IR.Types;

public class MultiJogging
{
```

```
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            // [ZH] 获取机器人模式
            // [EN] Get robot mode
            (UserOpMode opMode, StatusCode opCode) =
                controller.GetOpMode();
            if (opCode == StatusCode.OK)
            {
                Console.WriteLine(
                    $"当前机器人模式/Current robot mode: {opMode}"
                );
                if (
                    opMode != UserOpMode.UNLIMITED_MANUAL
                    && opMode != UserOpMode.LIMIT_MANUAL
                )
                {
```

```
                    Console.WriteLine(
                        $"示教运动必须在机器人手动模式下/Jogging must be in manua
l mode"
                    );
                    return StatusCode.OtherReason;
                }
            }
            else
            {
                Console.WriteLine(
                    $"获取机器人模式失败/Failed to get robot mode: {opCode.GetD
escription()}"
                );
            }

            Console.WriteLine(
                "开始多轴示教运动/Starting Multi-axis Jogging"
            );
            Console.WriteLine(
                "演示多轴运动/Demo multi-axis step movements"
            );

            // [ZH] 多轴运动
            // [EN] Multi-axis step movement
            Console.WriteLine(
                "\n=== 多轴运动/Multi-axis Step Movement ==="
            );
            int[] axes = { 1, 2, 3 }; // 正方向运动

            code = controller.Jogging.MultiMove(axes);
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "连续示教运动启动成功/Continuous Jogging Started Successful
ly"
                );
                Console.WriteLine(
                    "运动3秒后自动停止/Moving for 3 seconds then auto stop"
                );

                // [ZH] 运动3秒
                // [EN] Move for 3 seconds
```

```
                Thread.Sleep(3000);

                // [ZH] 停止示教运动
                // [EN] Stop jogging movement
                controller.Jogging.Stop();
                Console.WriteLine(
                    "示教运动已停止/Jogging Movement Stopped"
                );
            }
            else
            {
                Console.WriteLine(
                    $"连续示教运动启动失败/Continuous Jogging Start Failed: {co
de.GetDescription()}"
                );
            }

            Console.WriteLine(
                "\n多轴示教运动完成/Multi-axis Jogging Completed"
            );
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
```

```
            }
        }

        return code;
    }
}
```

4.12.4 停止机器人连续运动

方法名	Jogging.Stop()
描述	终止机器人当前示教点动（JOG）连续运动
请求参数	无
返回值	void
备注	仅在连续点动模式下需要调用此方法停止运动
兼容的机器人软件版本	协作 (Copper): v7.5.0.0+ 工业 (Bronze): v7.5.0.0+

示例代码

Jogging/ContinuousJogging.cs

cs

```cs
using Agilebot.IR;
using Agilebot.IR.Types;

public class ContinuousJogging
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
```

```
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
        );

        if (code != StatusCode.OK)
        {
            return code;
        }

        try
        {
            // [ZH] 获取机器人模式
            // [EN] Get robot mode
            (UserOpMode opMode, StatusCode opCode) =
                controller.GetOpMode();
            if (opCode == StatusCode.OK)
            {
                Console.WriteLine(
                    $"当前机器人模式/Current robot mode: {opMode}"
                );
                if (
                    opMode != UserOpMode.UNLIMITED_MANUAL
                    && opMode != UserOpMode.LIMIT_MANUAL
                )
                {
                    Console.WriteLine(
                        $"示教运动必须在机器人手动模式下/Jogging must be in manua
l mode"
                    );
                    return StatusCode.OtherReason;
                }
            }
            else
```

```
            {
                Console.WriteLine(
                    $"获取机器人模式失败/Failed to get robot mode: {opCode.GetD
escription()}"
                );
            }

            // [ZH] 设置示教运动参数
            // [EN] Set jogging parameters
            int ajNum = 3; // 轴序号，正数表示正方向运动
            MoveMode moveMode = MoveMode.Continuous; // 连续运动模式

            Console.WriteLine(
                "开始连续示教运动/Starting Continuous Jogging"
            );
            Console.WriteLine(
                $"轴序号/Axis Number: {ajNum}"
            );
            Console.WriteLine(
                $"运动模式/Move Mode: {moveMode}"
            );

            // [ZH] 启动连续示教运动
            // [EN] Start continuous jogging movement
            code = controller.Jogging.Move(ajNum, moveMode);
            if (code == StatusCode.OK)
            {
                Console.WriteLine(
                    "连续示教运动启动成功/Continuous Jogging Started Successful
ly"
                );
                Console.WriteLine(
                    "运动3秒后自动停止/Moving for 3 seconds then auto stop"
                );

                // [ZH] 运动3秒
                // [EN] Move for 3 seconds
                Thread.Sleep(3000);

                // [ZH] 停止示教运动
                // [EN] Stop jogging movement
                controller.Jogging.Stop();
```

```
                Console.WriteLine(
                    "示教运动已停止/Jogging Movement Stopped"
                );
            }
            else
            {
                Console.WriteLine(
                    $"连续示教运动启动失败/Continuous Jogging Start Failed: {co
de.GetDescription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }

        return code;
    }
}
```

4.13 机器人订阅发布接口

概述

SubPub 类提供机器人控制器 WebSocket 订阅 / 发布通道的管理能力，负责建立连接、订阅机器人状态 / 寄存器 / IO 等主题，并通过回调或阻塞接口接收实时数据。

核心功能

- 支持连接和断开 WebSocket 服务器
- 支持订阅机器人状态数据
- 支持订阅寄存器数据
- 支持订阅 IO 信号数据
- 支持通过回调函数持续接收消息
- 支持阻塞式接收下一条消息
- 支持设置订阅频率
- 支持在接收流程中处理异常并进行排查

使用场景

- 上位机实时监听机器人运行状态
- 实现机器人数据可视化
- 与外部系统实现数据联动
- 实时监控寄存器和 IO 状态
- 构建机器人监控和控制系统
- 实现机器人状态的实时报警和通知

4.13.1 连接到 WebSocket 服务器

方法名	SubPub.Connect()
描述	连接到机器人控制器 WebSocket 服务器
请求参数	无
返回值	Task：异步连接操作结果
兼容的机器人软件版本	协作 (Copper): v7.7.0.0+ 工业 (Bronze): v7.7.0.0+

4.13.2 断开 WebSocket 连接

方法名	SubPub.Disconnect()
描述	断开与机器人控制器 WebSocket 服务器的连接
请求参数	无
返回值	Task：异步断开操作结果
兼容的机器人软件版本	协作 (Copper): v7.7.0.0+ 工业 (Bronze): v7.7.0.0+

4.13.3 添加机器人状态订阅

方法名	SubPub.SubscribeStatus(RobotTopicType[] `topicTypes` , int `frequency` = 200)
描述	添加机器人状态数据订阅
请求参数	`topicTypes` : RobotTopicType [] 机器人主题类型列表 `frequency` : int 订阅频率（单位 Hz，默认 200）
返回值	Task：异步订阅操作结果
兼容的机器人软件版本	协作 (Copper): v7.7.0.0+ 工业 (Bronze): v7.7.0.0+

4.13.4 添加寄存器订阅

方法名	**SubPub.SubscribeRegister(RegTopicType `regType` , int[] `regIds` , int `frequency` = 200)**
描述	添加寄存器数据订阅
请求参数	`regType` : RegTopicType 寄存器类型 `regIds` : int [] 寄存器 ID 列表 `frequency` : int 订阅频率（单位 Hz，默认 200）
返回值	Task：异步订阅操作结果
兼容的机器人软件版本	协作 (Copper): v7.7.0.0+ 工业 (Bronze): v7.7.0.0+

4.13.5 添加 IO 订阅

方法名	**SubPub.SubscribeIO((IOTopicType, int)[] `ioList` , int `frequency` = 200)**
描述	订阅 IO 信号数据，包括数字输入 / 输出等
请求参数	`ioList` : (IOTopicType, int)[] IO 列表（每个元素为 `(IO 类型, IO ID)` ） `frequency` : int 订阅频率（单位 Hz，默认 200）
返回值	Task：异步订阅操作结果
兼容的机器人软件版本	协作 (Copper): v7.7.0.0+ 工业 (Bronze): v7.7.0.0+

4.13.6 开始接收消息

方法名	**SubPub.StartReceiving(Func<Dictionary<string, object>, Task> `onMessageReceived`)**
描述	开始接收订阅消息，并通过回调函数处理接收到的数据

方法名	**SubPub.StartReceiving(Func<Dictionary<string, object>, Task> onMessageReceived)**
请求参数	onMessageReceived : Func<Dictionary<string, object>, Task> 消息接收回调函数
返回值	Task：异步接收任务
备注	若回调函数抛出异常，接收循环会终止。建议在回调内部自行捕获异常并记录日志
兼容的机器人软件版本	协作 (Copper): v7.7.0.0+ 工业 (Bronze): v7.7.0.0+

示例代码

SubPub/CallbackReceiving.cs

```cs
using Agilebot.IR;
using Agilebot.IR.SubPub;
using Agilebot.IR.Types;

public class CallbackReceiving
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
    {
        // [ZH] 初始化捷勃特机器人
        // [EN] Initialize the Agilebot robot
        Arm controller = new Arm(
            controllerIP,
            useLocalProxy
        );

        // [ZH] 连接捷勃特机器人
        // [EN] Connect to the Agilebot robot
        StatusCode code = controller.ConnectSync();
        Console.WriteLine(
            code != StatusCode.OK
                ? code.GetDescription()
                : "连接成功/Successfully connected."
```

```
);

// [ZH] 初始化捷勃特机器人SubPub
// [EN] Initialize the Agilebot robot SubPub
var subPub = controller.SubPub;

try
{
    Console.WriteLine(
        "开始回调方式接收消息测试/Starting Callback Receiving Test"
    );

    // [ZH] 连接到WebSocket服务器
    // [EN] Connect to WebSocket server
    subPub.Connect().Wait();
    Console.WriteLine(
        "WebSocket连接成功/WebSocket Connected Successfully"
    );

    // [ZH] 订阅机器人状态
    // [EN] Subscribe to robot status
    var topicTypes = new RobotTopicType[]
    {
        RobotTopicType.TopicCurrentJoint,
        RobotTopicType.TopicRobotStatus,
    };
    subPub
        .SubscribeStatus(topicTypes, frequency: 100)
        .Wait();
    Console.WriteLine(
        "机器人状态订阅成功/Robot Status Subscription Successful"
    );

    // [ZH] 订阅寄存器
    // [EN] Subscribe to registers
    var regIds = new int[] { 1, 2, 3 };
    subPub
        .SubscribeRegister(
            RegTopicType.R,
            regIds,
            frequency: 100
        )
```

```
            .Wait();
        Console.WriteLine(
            "寄存器订阅成功/Register Subscription Successful"
        );

        // [ZH] 订阅IO
        // [EN] Subscribe to IO
        var ioList = new (IOTopicType, int)[]
        {
            (IOTopicType.DI, 0),
            (IOTopicType.DO, 1),
        };
        subPub
            .SubscribeIO(ioList, frequency: 100)
            .Wait();
        Console.WriteLine(
            "IO订阅成功/IO Subscription Successful"
        );

        int messageCount = 0;
        int maxMessages = 10; // 接收10条消息后停止

        Console.WriteLine(
            "开始接收消息/Starting to receive messages..."
        );

        // [ZH] 开始接收消息（回调方式）
        // [EN] Start receiving messages (callback method)
        subPub
            .StartReceiving(async message =>
            {
                messageCount++;
                Console.WriteLine(
                    $"\n=== 收到第{messageCount}条消息/Received Message #
{messageCount} ==="
                );
                foreach (var kv in message)
                {
                    Console.WriteLine(
                        $"{kv.Key}: {kv.Value}"
                    );
                }
```

```
                    // [ZH] 接收指定数量消息后主动断开
                    // [EN] Disconnect after receiving specified number of m
essages
                    if (messageCount >= maxMessages)
                    {
                        Console.WriteLine(
                            $"已接收{maxMessages}条消息，准备断开连接/Received
{maxMessages} messages, preparing to disconnect"
                        );
                        subPub.Disconnect().Wait();
                        Console.WriteLine(
                            "WebSocket断开成功/WebSocket Disconnected Success
fully"
                        );
                    }

                    await Task.CompletedTask;
                })
                .Wait();

            Console.WriteLine(
                "回调方式接收消息测试完成/Callback Receiving Test Completed"
            );
            return StatusCode.OK;
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            return StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
```

```
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }
    }
}
```

4.13.7 接收下一条消息

方法名	SubPub.Receive()
描述	阻塞接收下一条消息并返回
请求参数	无
返回值	Task<Dictionary<string, object>>：接收到的消息字典
备注	连接未建立、连接被关闭或消息格式异常时会抛出异常
兼容的机器人软件版本	协作 (Copper): v7.7.0.0+ 工业 (Bronze): v7.7.0.0+

示例代码

SubPub/PollingReceiving.cs

cs

```cs
using Agilebot.IR;
using Agilebot.IR.SubPub;
using Agilebot.IR.Types;

public class PollingReceiving
{
    public static StatusCode Run(
        string controllerIP,
        bool useLocalProxy = true
    )
```

```
{
    // [ZH] 初始化捷勃特机器人
    // [EN] Initialize the Agilebot robot
    Arm controller = new Arm(
        controllerIP,
        useLocalProxy
    );

    // [ZH] 连接捷勃特机器人
    // [EN] Connect to the Agilebot robot
    StatusCode code = controller.ConnectSync();
    Console.WriteLine(
        code != StatusCode.OK
            ? code.GetDescription()
            : "连接成功/Successfully connected."
    );

    // [ZH] 初始化捷勃特机器人SubPub
    // [EN] Initialize the Agilebot robot SubPub
    var subPub = controller.SubPub;

    try
    {
        Console.WriteLine(
            "开始轮询方式接收消息测试/Starting Polling Receiving Test"
        );

        // [ZH] 连接到WebSocket服务器
        // [EN] Connect to WebSocket server
        subPub.Connect().Wait();
        Console.WriteLine(
            "WebSocket连接成功/WebSocket Connected Successfully"
        );

        // [ZH] 订阅机器人状态
        // [EN] Subscribe to robot status
        var topicTypes = new RobotTopicType[]
        {
            RobotTopicType.TopicCurrentJoint,
            RobotTopicType.TopicRobotStatus,
        };
        subPub
```

```
    .SubscribeStatus(topicTypes, frequency: 100)
    .Wait();
Console.WriteLine(
    "机器人状态订阅成功/Robot Status Subscription Successful"
);

// [ZH] 订阅寄存器
// [EN] Subscribe to registers
var regIds = new int[] { 1, 2, 3 };
subPub
    .SubscribeRegister(
        RegTopicType.R,
        regIds,
        frequency: 100
    )
    .Wait();
Console.WriteLine(
    "寄存器订阅成功/Register Subscription Successful"
);

// [ZH] 订阅IO
// [EN] Subscribe to IO
var ioList = new (IOTopicType, int)[]
{
    (IOTopicType.DI, 0),
    (IOTopicType.DO, 1),
};
subPub
    .SubscribeIO(ioList, frequency: 100)
    .Wait();
Console.WriteLine(
    "IO订阅成功/IO Subscription Successful"
);

int messageCount = 0;
int maxMessages = 10; // 接收10条消息后停止

Console.WriteLine(
    "开始轮询接收消息/Starting to poll messages..."
);

// [ZH] 循环接收消息直到达到期望数量
```

```
            // [EN] Loop to receive messages until reaching desired count
            do
            {
                messageCount++;
                try
                {
                    // [ZH] 接收单条消息
                    // [EN] Receive single message
                    var message = subPub.Receive().Result;
                    Console.WriteLine(
                        $"\n=== 收到第{messageCount}条消息/Received Message #
{messageCount} ==="
                    );
                    foreach (var kv in message)
                    {
                        Console.WriteLine(
                            $"{kv.Key}: {kv.Value}"
                        );
                    }
                }
                catch (Exception ex)
                {
                    Console.WriteLine(
                        $"接收消息时发生异常/Exception while receiving message:
{ex.Message}"
                    );
                    break;
                }
            } while (messageCount < maxMessages);

            // [ZH] 断开连接
            // [EN] Disconnect
            subPub.Disconnect().Wait();
            Console.WriteLine(
                "WebSocket断开成功/WebSocket Disconnected Successfully"
            );

            Console.WriteLine(
                "轮询方式接收消息测试完成/Polling Receiving Test Completed"
            );
            return StatusCode.OK;
        }
```

```
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            return StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            if (disconnectCode != StatusCode.OK)
            {
                Console.WriteLine(
                    disconnectCode.GetDescription()
                );
                if (code == StatusCode.OK)
                    code = disconnectCode;
            }
        }
    }
}
```

捷勃特机器人 C# SDK 更新说明

2.1.0.* 更新（2026/4/9）

1. 添加 UDP 反馈配置相关接口
2. `Motion.SetPositionTrajectoryParams` 指定 UDP 位置控制的相关参数接口添加 `filterLayer` 参数
3. `Motion.Payload.GetPayloadIdentifyState` 获取负载测定状态接口更新消息解析
4. 添加 `TerminatePayloadIdentify` 终止负载测定接口
5. 添加轨迹表和路径表相关接口
6. 添加轨迹整形器相关接口
7. 添加路径规划参数相关接口

2.0.3.* 更新（2025/12/17）

1. Arm 构造函数增加 teachPanelIP 参数
2. 去除 System.Text.Json 依赖
3. 增加同步连接接口 ConnectSync

2.0.2.* 更新（2025/12/12）

1. 修复 PR 寄存器结构体中数据读写顺序错误

2.0.1.* 更新（2025/10/21）

1. 修复 jogging 持续运动时卡顿的问题
2. 修复构建项目时可能会报 proxy 执行文件无法复制的问题

2.0.0.* 更新（2025/9/10）

1. 修改底层请求模式，添加本地控制器代理服务
2. 添加订阅功能
3. 调整 BasScript 类结构
4. 全面支持 .NET Framework 和 .NET

1.0.1.0 更新（2025/7/7）

1. 增加旧的寄存器接口类 RegistersOld 兼容 7.6.0.0 以前的机器人版本
2. 增加 Estop 紧急制动接口
3. 修复文档中的示例程序

1.0.0.0 更新（2025/5/30）

1. 改用 RPC 方式实现
2. 所有接口定义同步 Python 版本

帮助

Agent Skills

Agent Skills 旨在为各类 AI 智能体（如 Codex、Claude Code 等）提供能力扩展，使其在特定工业或开发场景中具备标准化的执行流程与工具链。

兼容的 Agent 类型

目前 Agent Skills 已实现对以下主流 AI 开发工具和 CLI 的自动识别与支持：

- **IDE / 编辑器**：Cursor、Windsurf、Trae、Trae CN、VS Code（通过 Copilot / Continue）、Neovate、Pochi
- **命令行工具（CLI）**：Claude Code、Kimi Code CLI、Gemini CLI、iFlow CLI、Kiro CLI、Mistral Vibe
- **开发框架 / 平台**：OpenHands、Replit、Cline、Roo Code、Codex、Amp、Antigravity、Augment、OpenClaw
- **其他**：CodeBuddy、Command Code、Droid、Junie、Kilo Code、Kode、MCPJam、Mux、OpenCode、Pi、Qoder、Qwen Code、Zencoder、AdaL

快速开始

1. 环境准备

安装前请确保本地已配置 Node.js 环境。教程参考：Node.js 下载与安装指南

2. 执行安装命令

在终端运行以下命令，进入交互式安装界面：

```bash
npx skills add https://dev.svfactory.com -g
```

TIP

安装说明：执行后，工具会自动扫描本地已安装的 Agent。您可以根据终端提示选择要注入技能的特定 Agent（如 Cursor、Claude Code 等）。

已提供的 Skills 列表

agilebot-nlu-control（捷勃特机器人自然语言控制）

该技能专为捷勃特机器人（Agilebot）设计。通过 NLU（自然语言理解）技术，让 Agent 能够直接解析并执行控制指令。

- 环境要求：必须确保系统 Python 可用且已安装 Conda。Skill 会自动检查必需依赖，若缺失会提示并引导安装。
- 触发场景：在对话中提到 “捷勃特机器人” 或 “Agilebot” 并给出具体控制意图时。
- 核心功能：自然语言指令解析与执行、机器人状态查询、基础动作控制、参数化运动指令（如距离 / 角度 / 速度）。

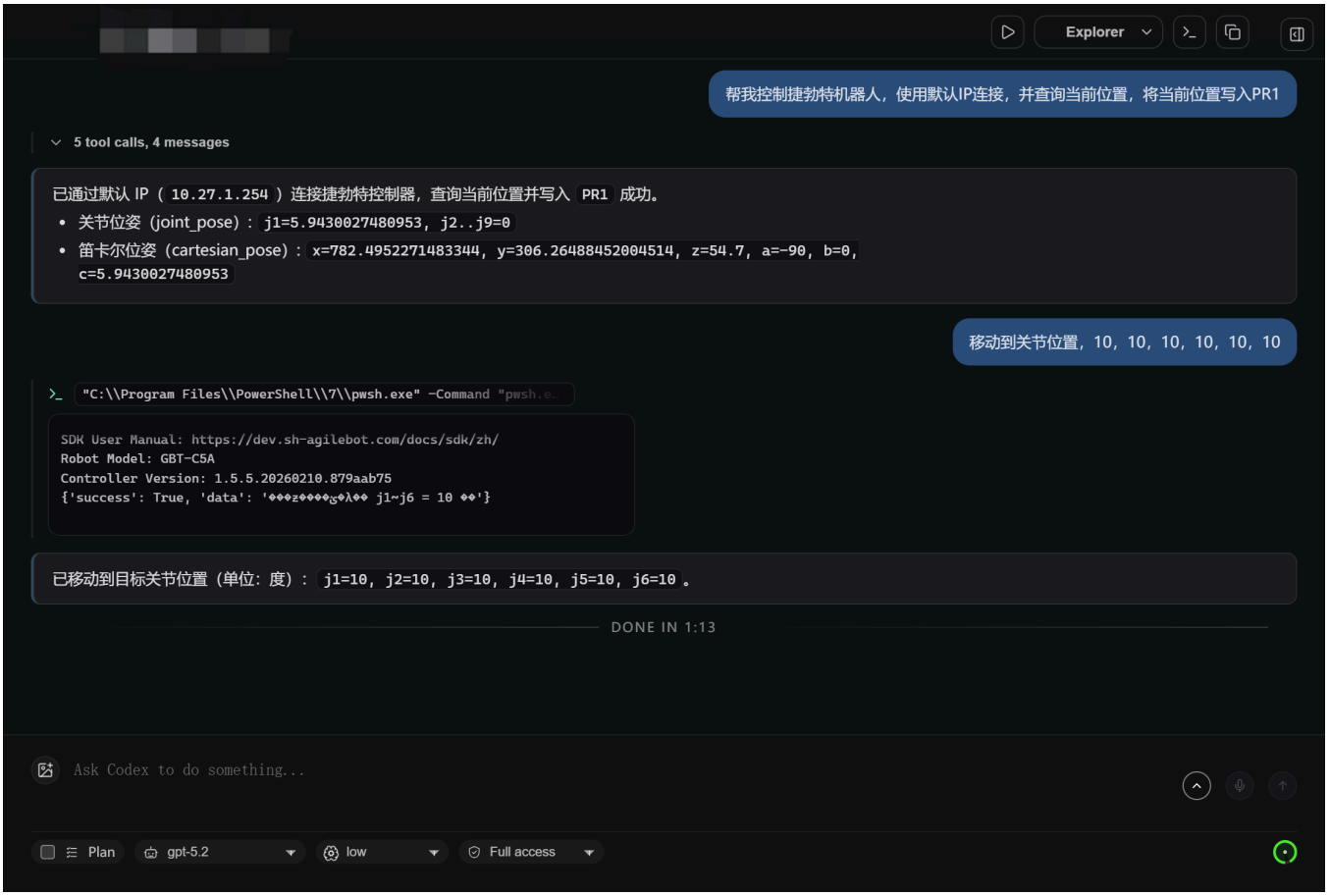
警告

安装后的技能将以 Agent 的完整权限运行，请在正式使用前确保已在受信任的环境中完成配置。

机器人运动存在风险，可能造成人身或设备伤害，请确保在安全的环境下使用，我们不负责由此带来的任何风险或损失。

使用示例

以 Codex 为例，安装成功后，直接在对话框输入指令即可：



AI 编程支持

本文档介绍如何使用 AI 辅助工具（如 CodeBuddy、Codex、Cursor 等）快速开发机器人控制程序。

准备工作

使用 AI 编程前，需要准备参考文档：

- **SDK 文档**：https://dev.svfactory.com/docs/sdk/knowledge/docs.txt

> **提示**：如果您使用的 AI Agent 无法很好地读取 URL，建议将以上 txt 文档下载到本地项目目录中，然后在 prompt 中引用本地文件路径。

示例 Prompt

以下是一个完整的示例，用于创建一个读取机器人状态的 Python 程序：

```
阅读以下文档，写一个Python程序，用于读取机器人的当前位置、坐标系编号、伺服状态等信息。

参考资料：
SDK文档：https://dev.svfactory.com/docs/sdk/knowledge/docs.txt
```

如果使用本地文档，可以修改为：

```
阅读以下文档，写一个Python程序，用于读取机器人的当前位置、坐标系编号、伺服状态等信息。

参考资料：
SDK文档：./docs/sdk_docs.txt
```

使用技巧

1. **明确需求**：清晰描述要实现的功能
2. **提供上下文**：引用相关文档和示例
3. **分步实现**：复杂功能可以分步骤让 AI 生成

注意事项

1. AI 生成的代码需要经过验证和测试
2. 确保代码符合项目编码规范
3. 涉及机器人控制的代码必须进行安全审查