Agilebot Robot SDK
Agilebot 로봇 SDK 매뉴얼

Python SDK

Python 기반의 고성능 로봇 제어 개발 키트로, 간결한 API 설계를 통해 지능형 로봇 애플리케이션을 빠르게 구축할 수 있습니다.

자세히 알아보기 →

C# SDK

.NET 생태계를 위한 엔터프라이즈급 로봇 제어 솔루션으로, 타입 안전한 강타입 API를 제공해 산업 자동화 시스템에 쉽게 통합할 수 있습니다.

자세히 알아보기 →

C# SDK

프롤로그

버전 기록

문서 버전	SDK 버전 번호	버전 날짜
V3.3	2.1.0.*	2026.04.13

업데이트 노트

로봇 버전 호환성

SDK는 Agilebot Scara, Puma 및 협동로봇 시리즈를 지원합니다. 로봇 소프트웨어가 설치되어 있고 로봇 소프트웨어 버전과 호환되는 장치와 함께 사용해야 합니다. 일부 함수는 버전 차이로 인해 다른 결과를 반환할 수 있습니다.

SDK가 로봇 팔에 연결되면 로봇 팔 모션 제어 소프트웨어의 버전을 확인합니다. 버전이 최소 요구 사항보다 낮으면 연결이 실패합니다. 권장 버전보다 낮은 경우 버전이 너무 낮다는 메시지가 나타납니다. 적시에 로봇 소프트웨어 버전을 업데이트하십시오.

SDK의 일부 인터페이스는 해당 버전의 컨트롤러만 지원합니다. 특정 인터페이스의 호환성을 확인하십시오.

SDK 버전	호환되는 로봇 소프트웨어 버전	지원현황
0.1.1.X	Copper v7.5.X.X, Bronze v7.4.X.X	단종
0.1.2.X	Copper v7.5.X.X, Bronze v7.4.X.X	단종
0.2.0.X	Copper v7.5.X.X, Bronze v7.4.X.X	단종
1.0.0.X	Copper v7.6.X.X, Bronze v7.5.X.X	지원됨
2.0.X.X	Copper v7.7.X.X, Bronze v7.7.X.X	지원됨
2.1.X.X	Copper v7.7.I.X	지원됨

1 소개 및 배포

1.1 환경 요구사항

체계:

- Windows 10 이상
  - x86_64 아키텍처
- .NET 버전
  - 6.0 이상
- .NET 프레임워크 버전
  - 4.7 이상

1.2 설치

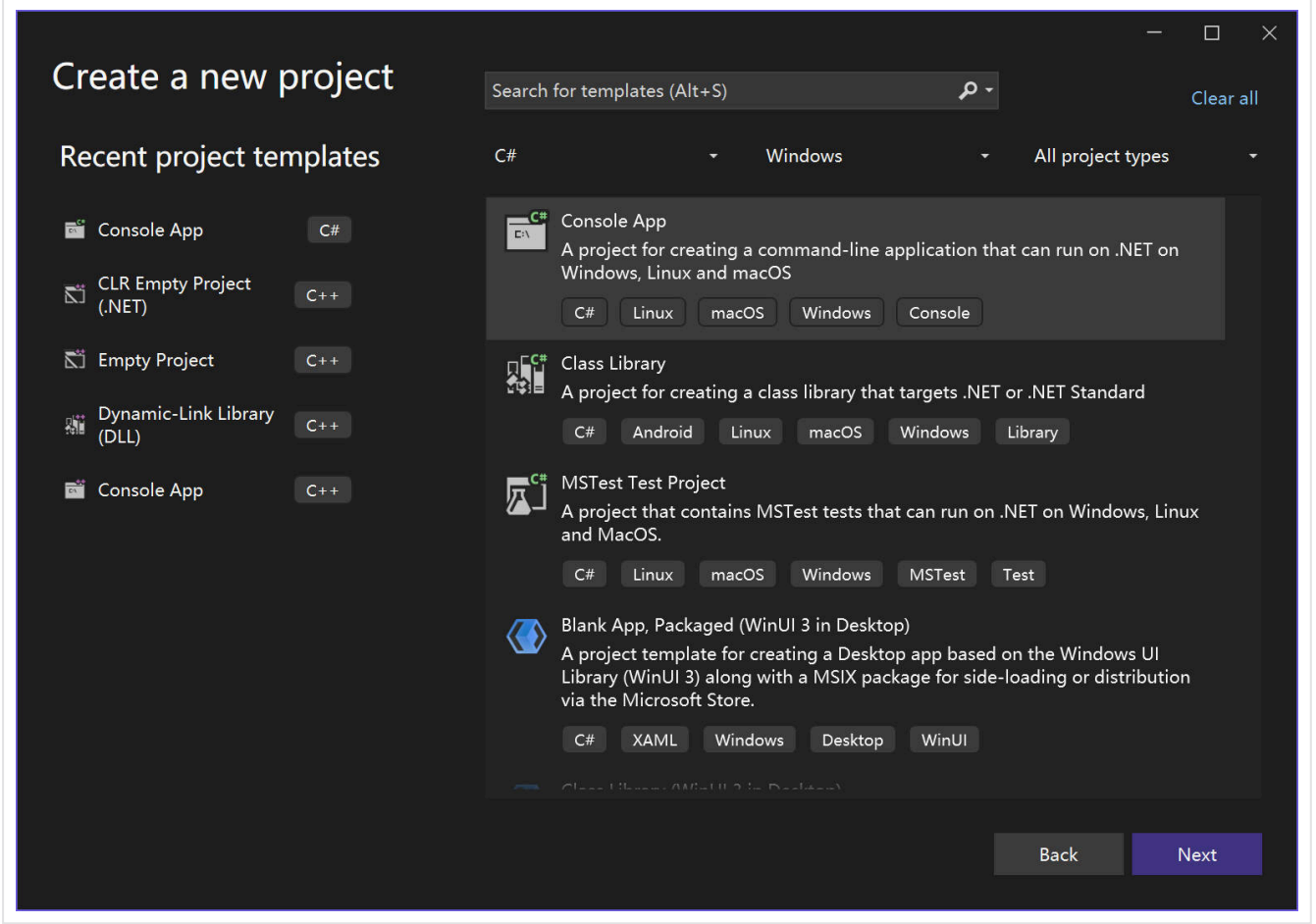
이 섹션에서는 IDE 준비, SDK 설치 및 가장 일반적인 런타임 주의 사항을 안내하므로 즉시 Agilebot SDK 실험을 시작할 수 있습니다.

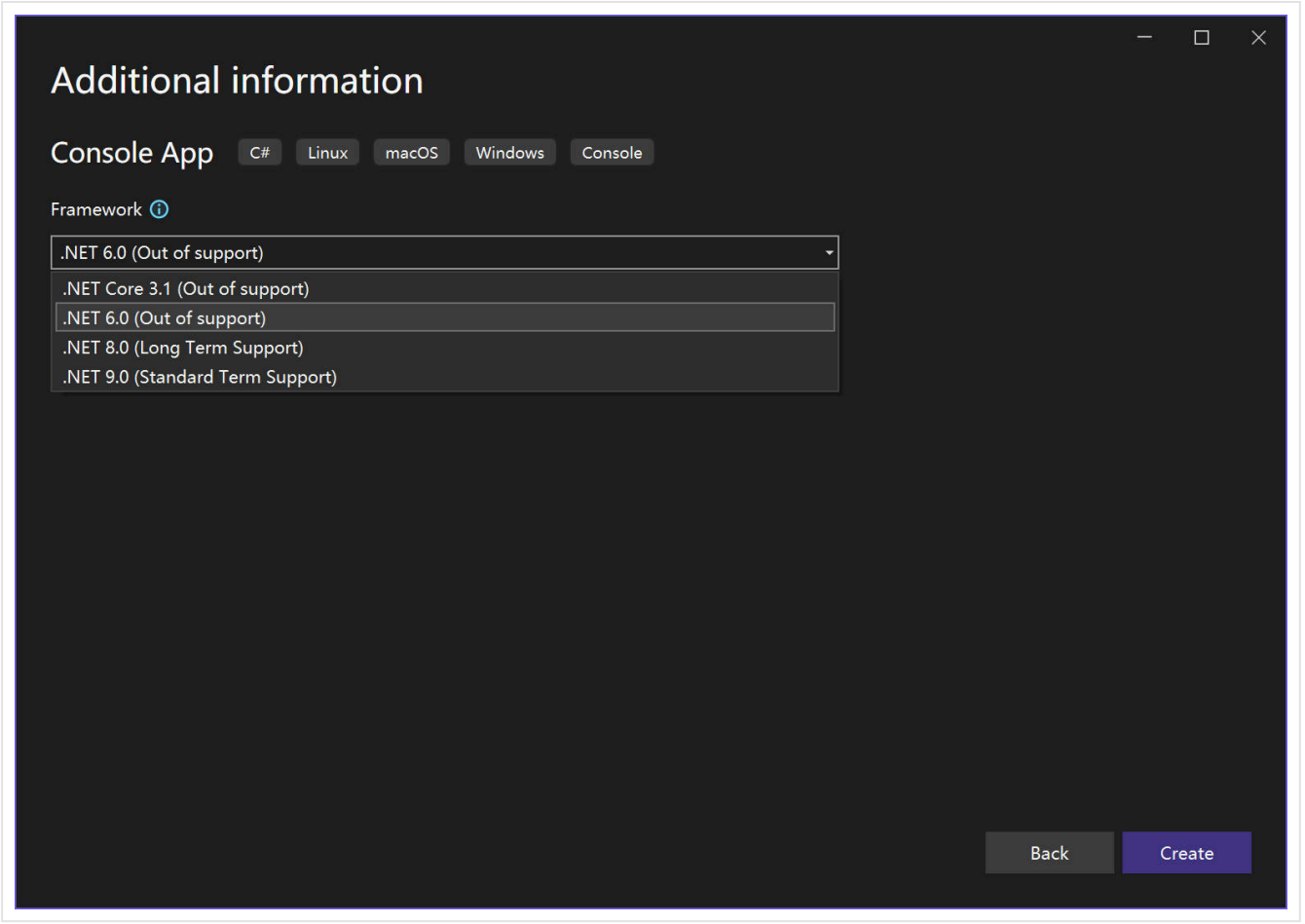
IDE 설정

1. Visual Studio는 C# 개발에 권장되는 IDE입니다. Download Visual Studio Tools - Free Install for Windows, Mac, Linux에서 다운로드하세요.
2. 설치 후 Visual Studio를 시작하고 초기 설정(로그인, 필요한 워크로드 설치 등)을 완료합니다.

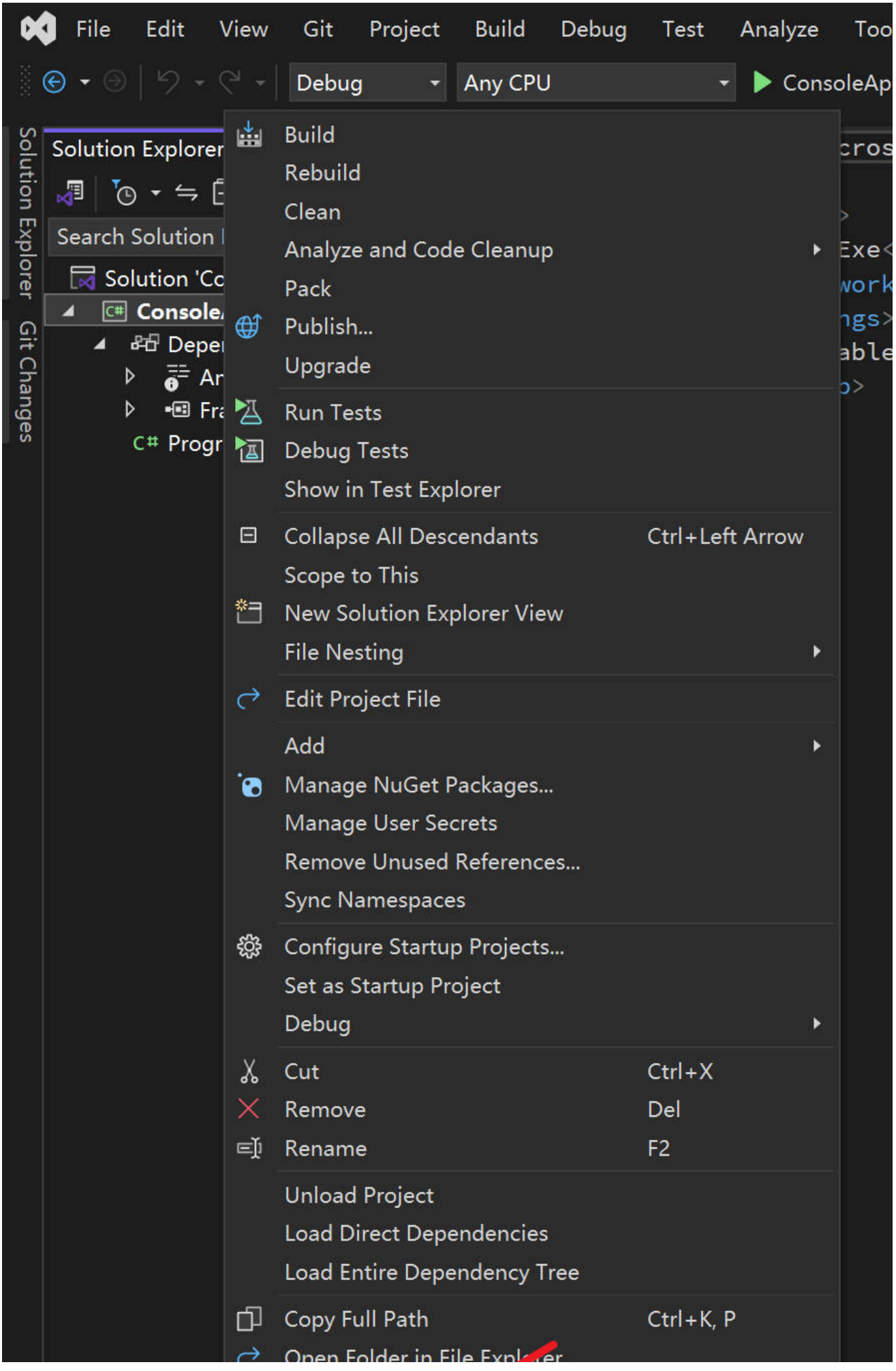
SDK 가져오기 및 프로젝트 생성

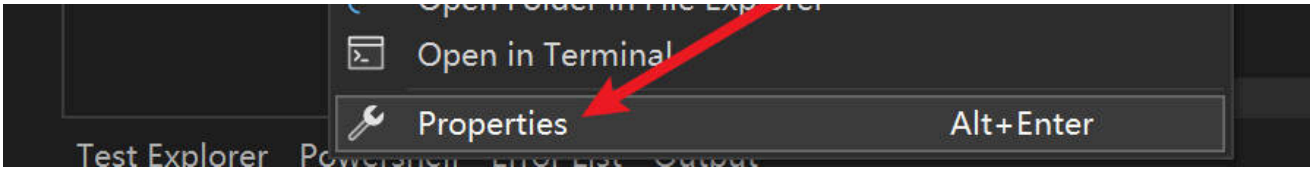
1. 새 **C# 콘솔 앱**을 만들고 `.NET 6.0` 이상을 대상 프레임워크로 선택합니다.

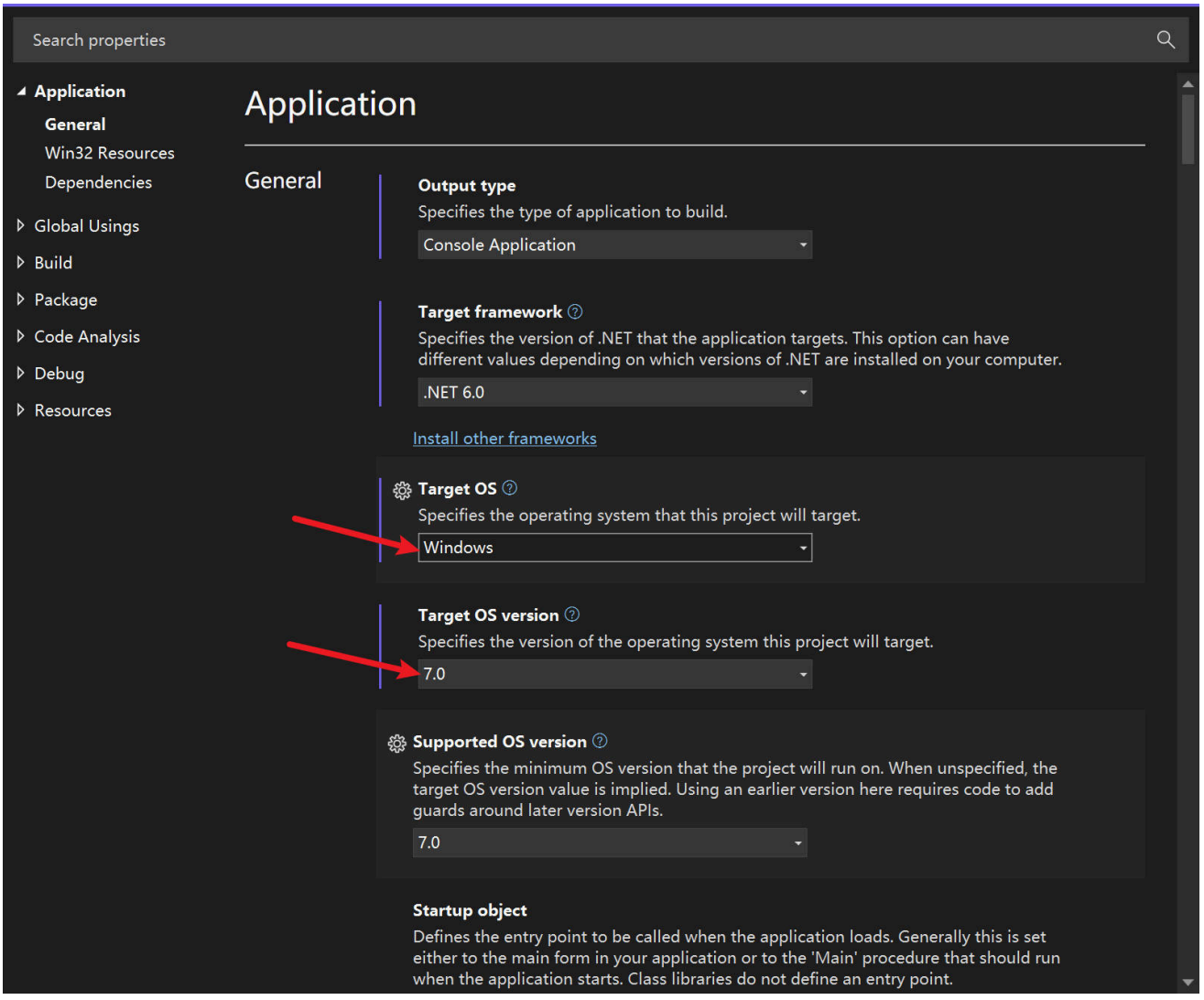




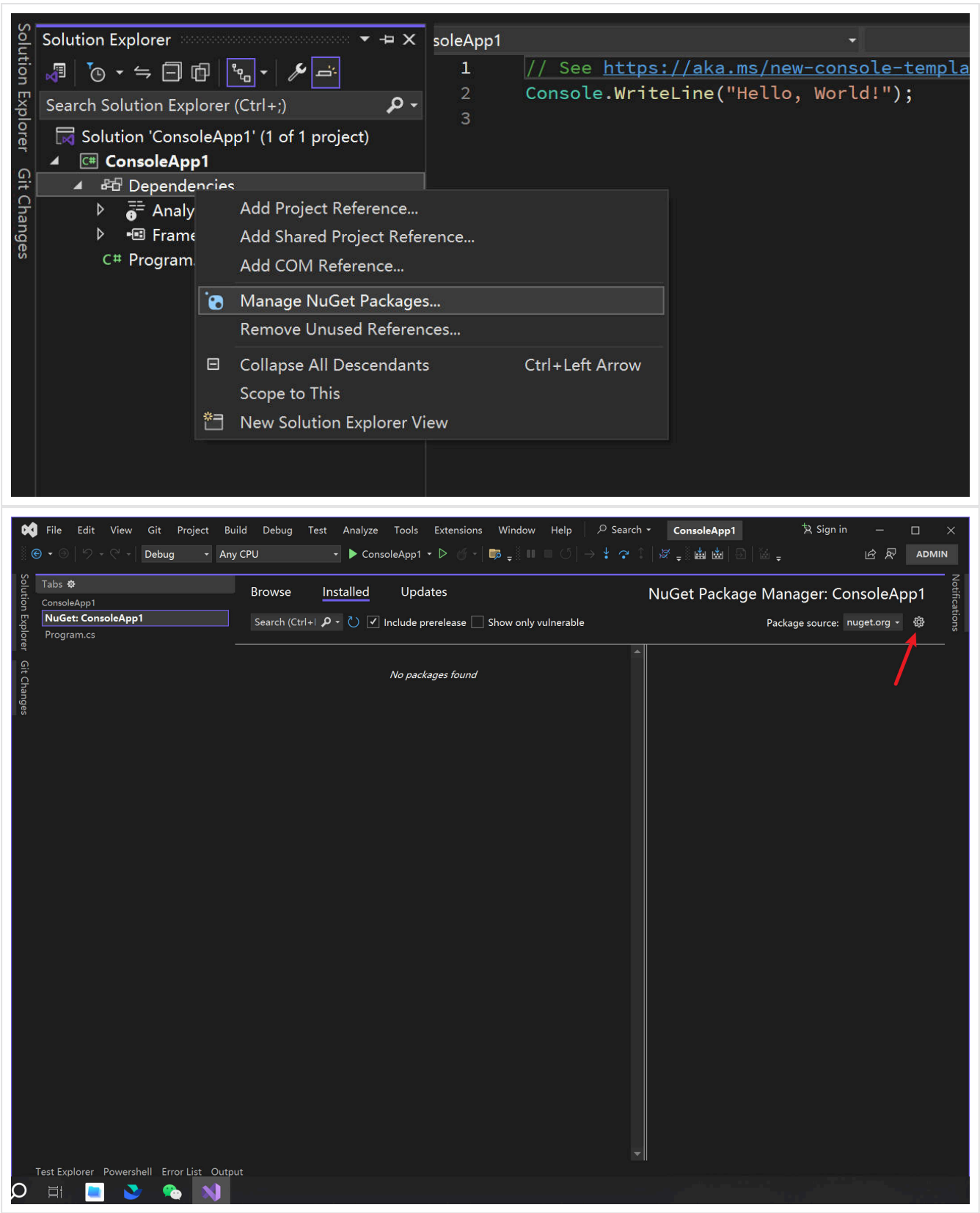
2. 프로젝트 속성을 열고 대상 OS를 **Windows**로 설정한 다음 **버전 7.0 이상**을 선택하여 최신 WinApp SDK 기능을 활용하세요.

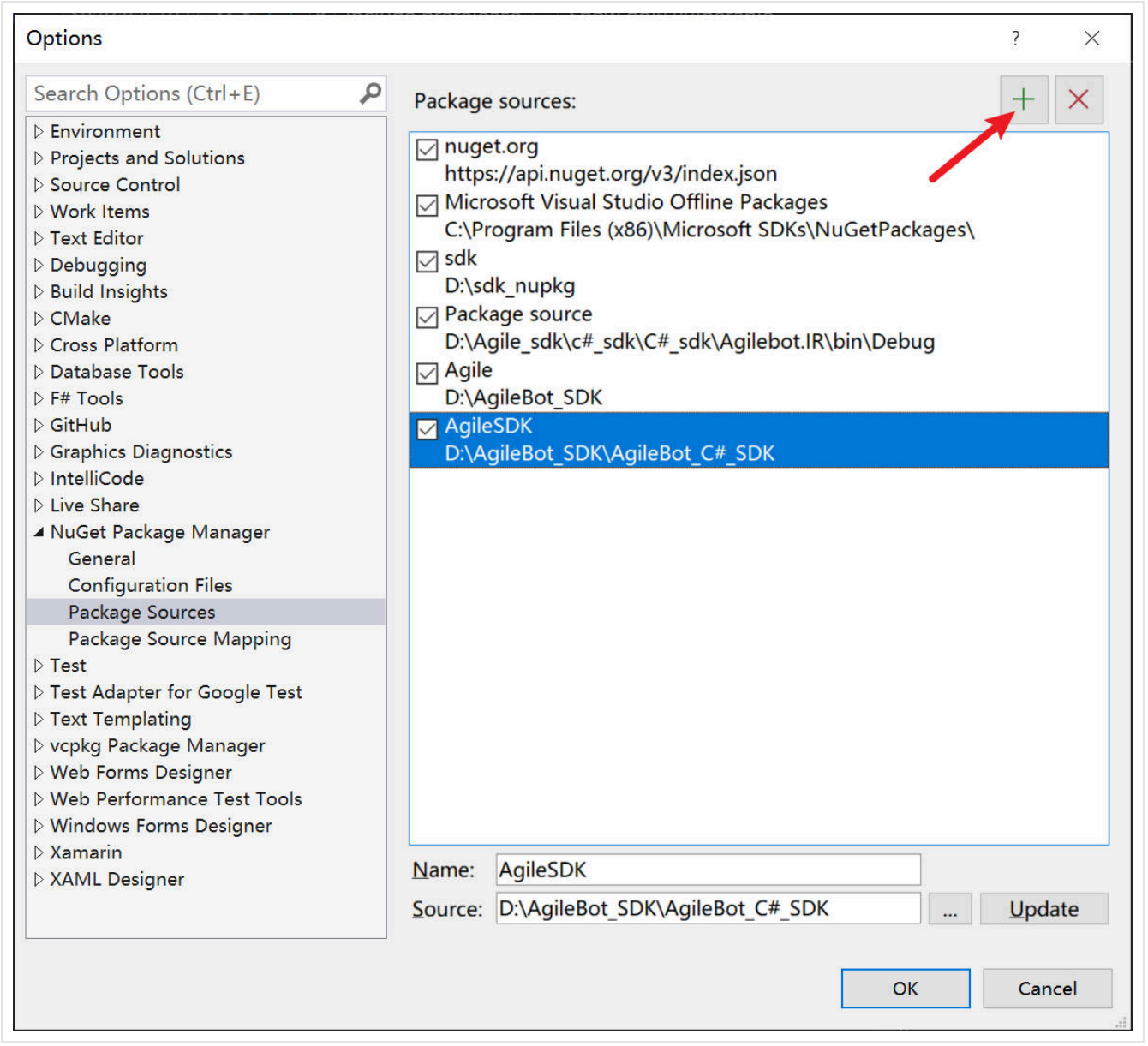


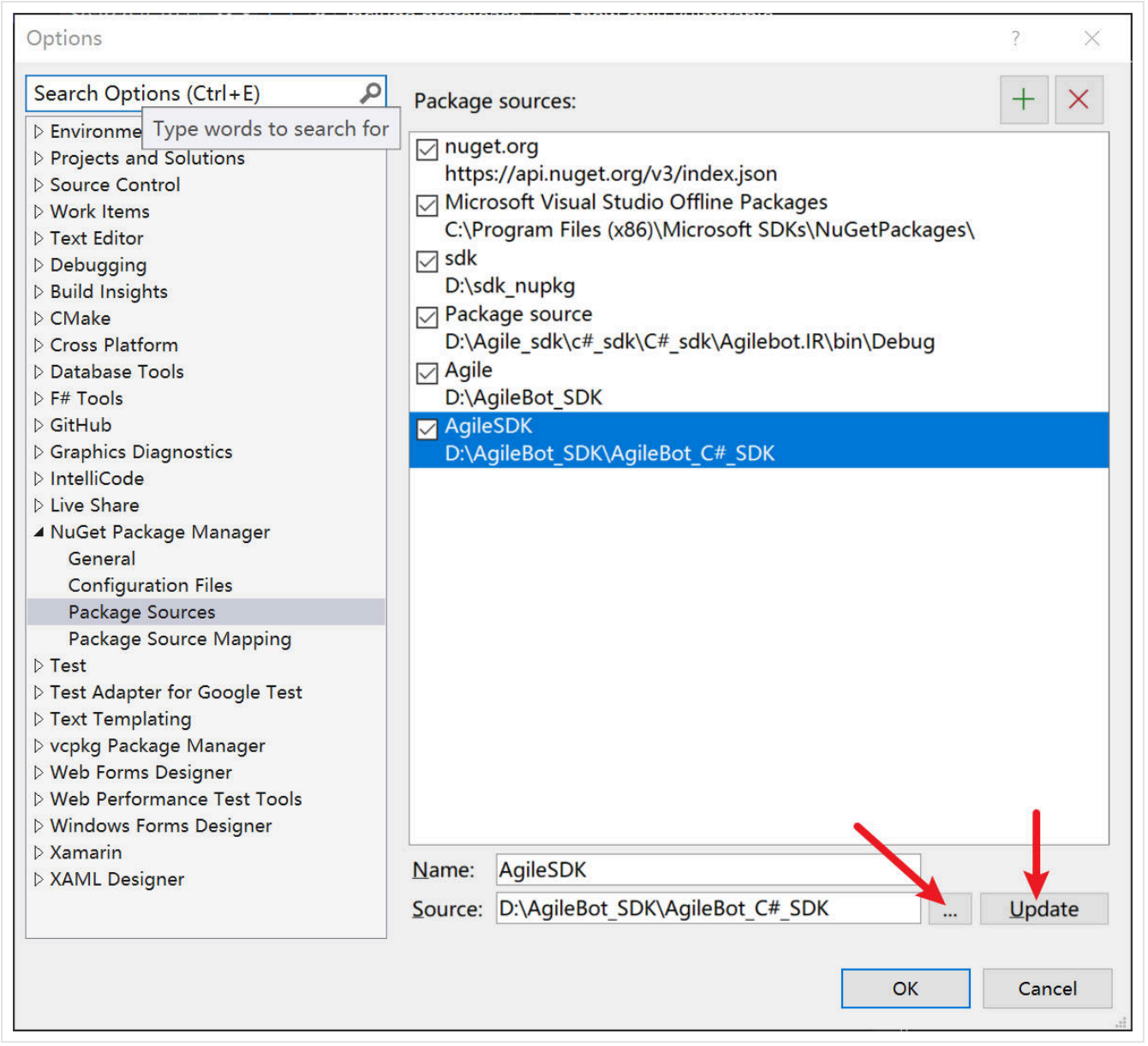




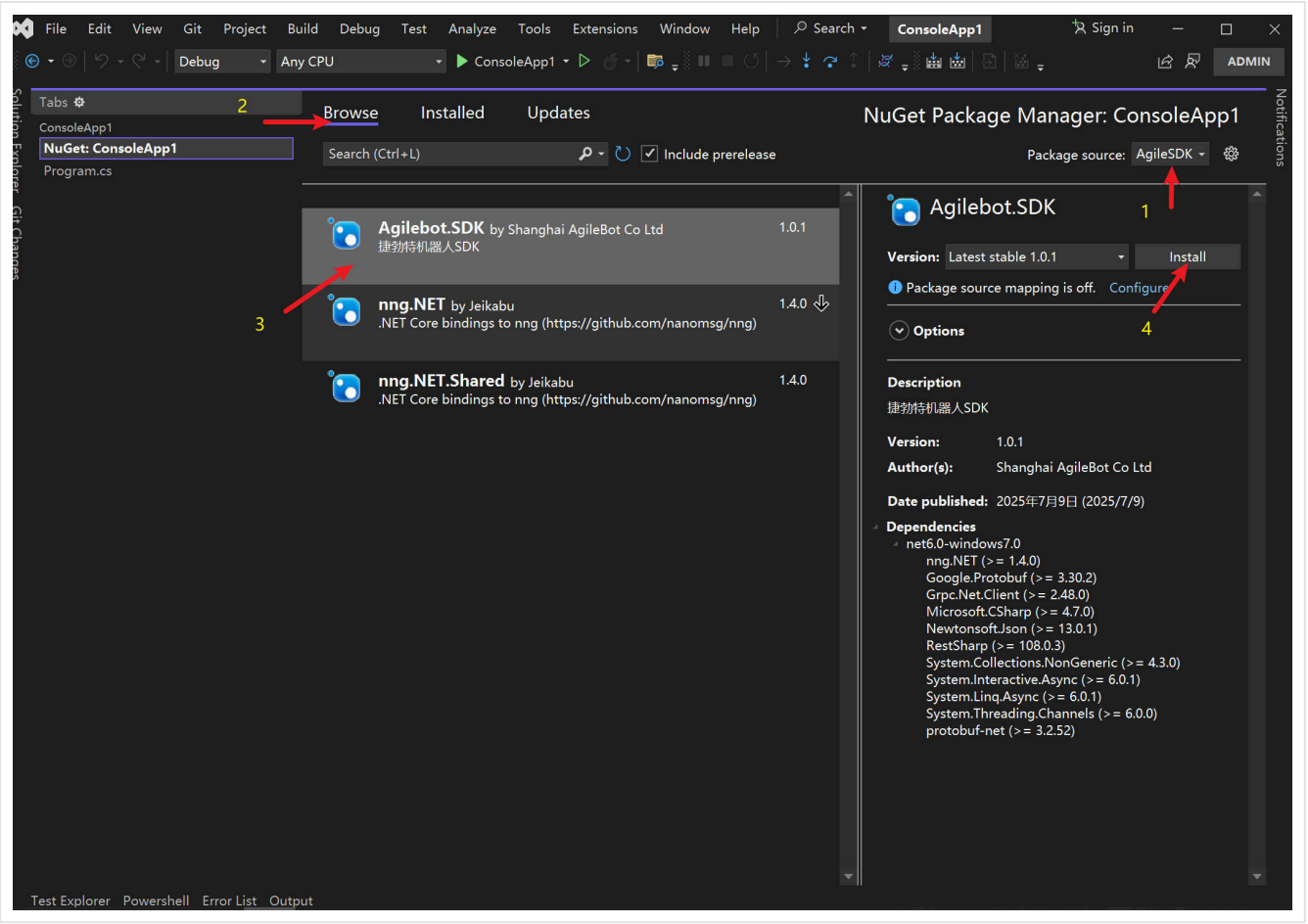
3. Tools > NuGet Package Manager > Package Manager Settings 로 이동한 다음 SDK 패키지가 포함된 디렉터리를 새 패키지 소스로 추가합니다.







4. NuGet 패키지 소스를 새로 추가된 항목으로 전환하고 `Agilebot.SDK` 패키지를 설치합니다.

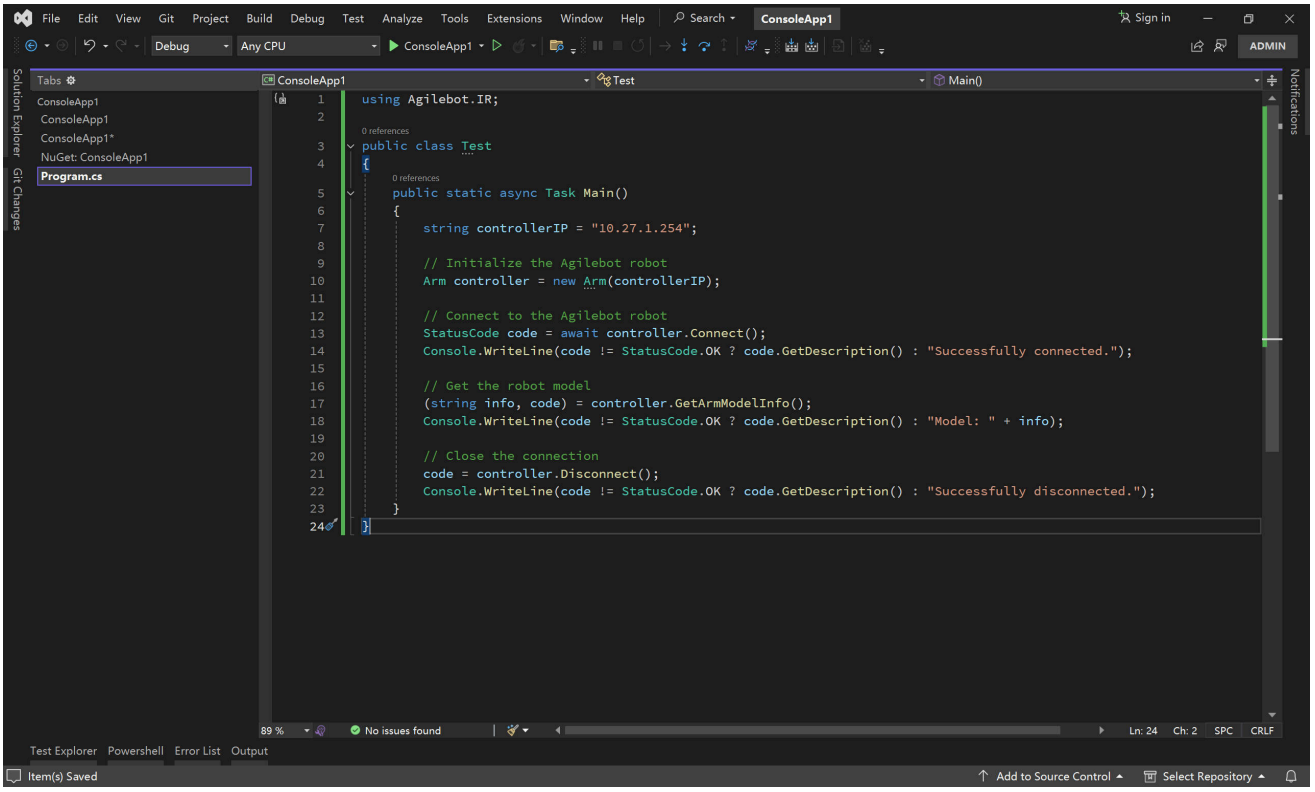


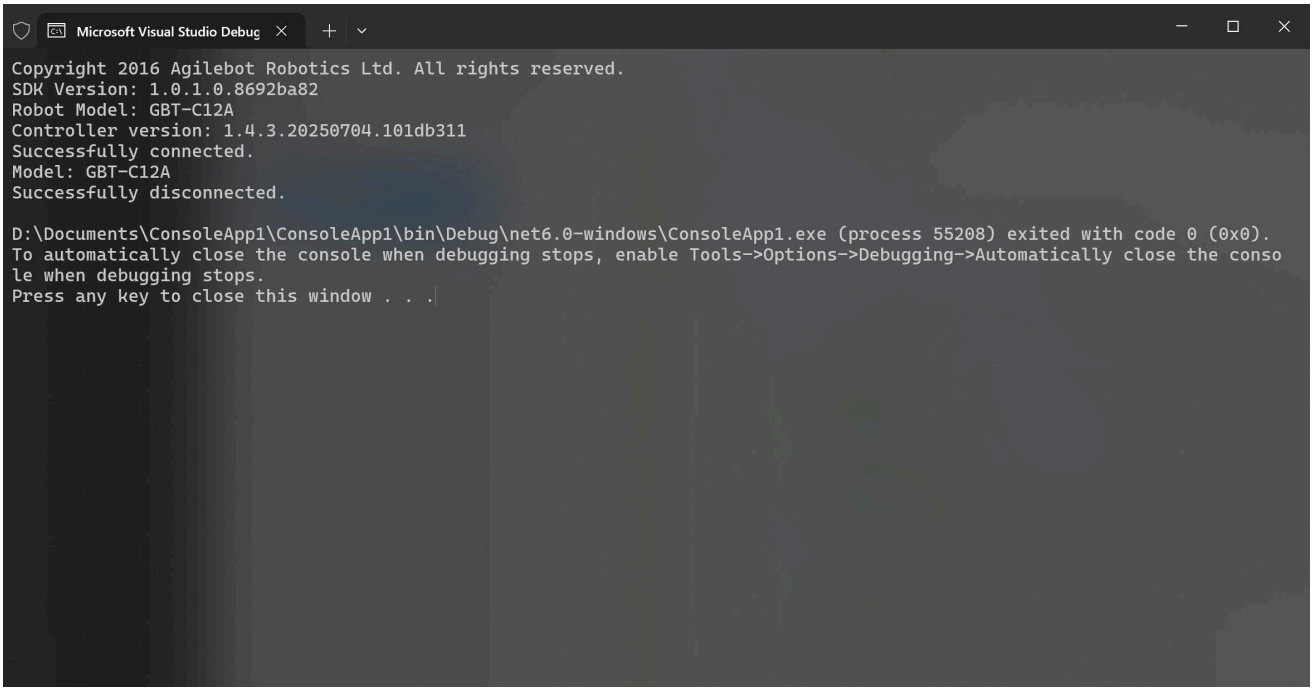
프록시 파일 및 문제 해결

- SDK를 설치한 후 프로젝트는 로컬 컨트롤러 프록시를 사용할 때 필요한 `controller_proxy_service_windows_amd64.exe` 가 포함된 `Tools` 폴더를 자동으로 얻습니다. 실행 파일이 누락된 경우 프로젝트 폴더와 빌드 출력 디렉터리 모두에 수동으로 복사하세요.
- 프로그램이 예기치 않게 종료되어 프록시 서비스가 계속 활성 상태인 경우 Windows 작업 관리자를 열고 **controller_proxy_service_windows_amd64**를 찾아 프로세스를 종료합니다.
- 프록시 서비스가 실행되는 동안에는 프록시 서비스가 있는 디렉터리를 다른 위치로 이동하지 **마세요**.

네트워킹 및 디버깅 요구 사항

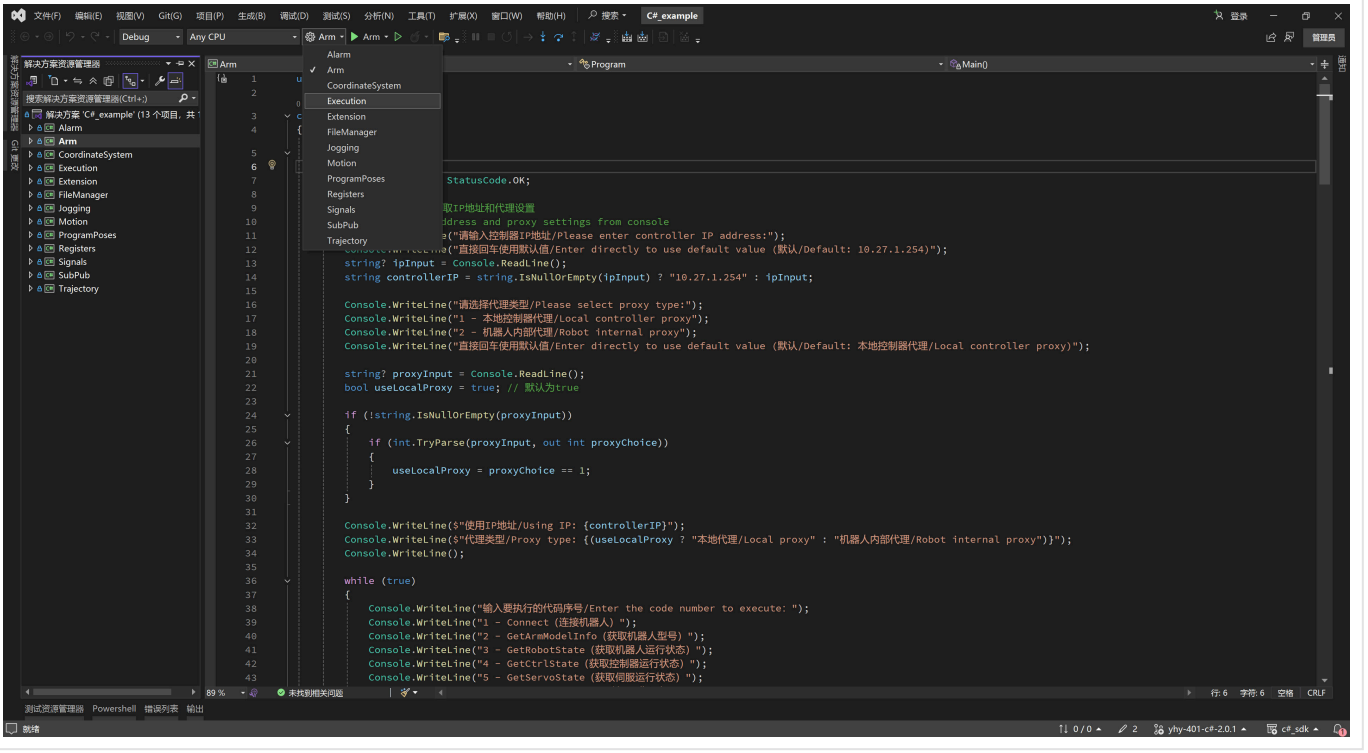
1. 코드를 실행하기 전에 호스트 PC가 로봇 네트워크에 연결되어 있는지 또는 로봇과 동일한 LAN을 공유하는지 확인하세요.
2. 프록시 서비스가 예기치 않게 삭제되는 것을 방지하려면 디버깅 중에 네트워크를 안정적으로 유지하세요.





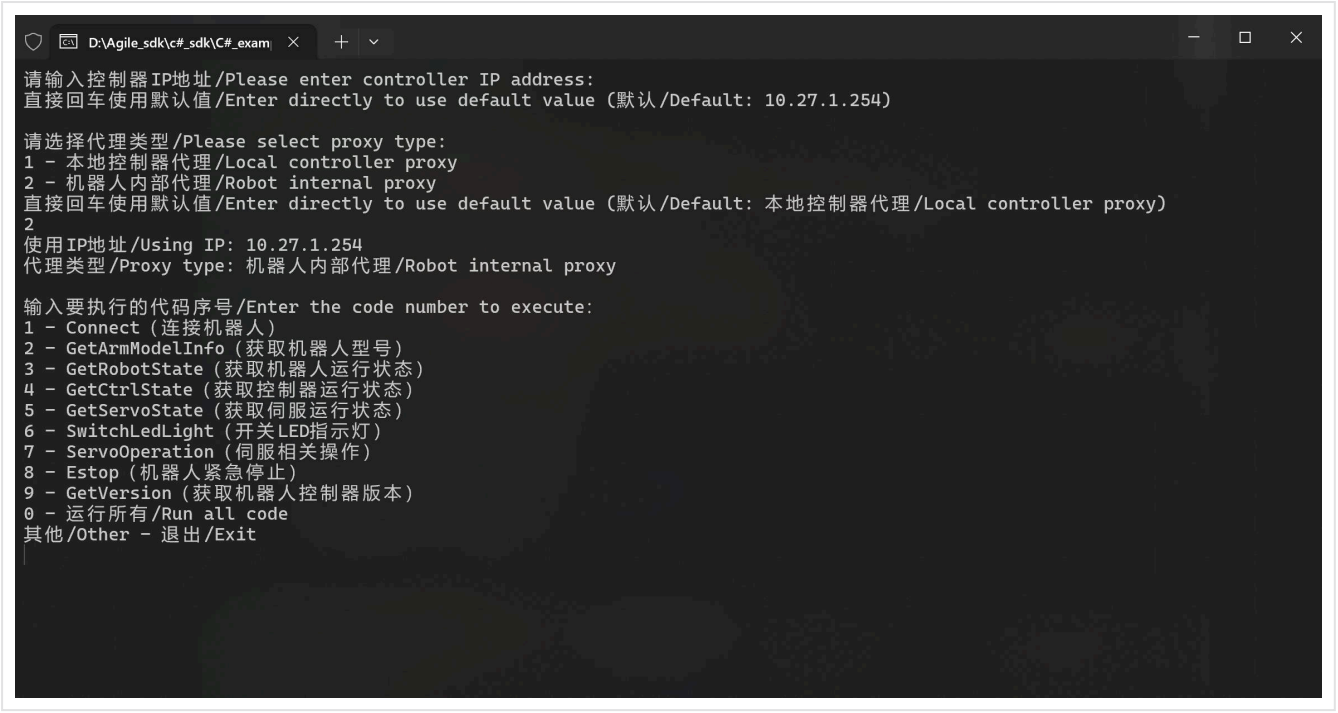
1.3 예제 프로그램 사용

이 장에서는 SDK와 함께 제공되는 `C#_example` 프로젝트를 안내합니다. 시작 항목을 전환하면 각 주요 SDK 클래스를 빠르게 시험해 볼 수 있습니다.



예제 실행

1. `C#_example` 를 열고 실행하십시오. 콘솔 창이 자동으로 나타납니다.
2. 메시지가 나타나면 로봇 IP 주소를 입력합니다.
3. 프록시 서비스를 호스팅할 위치(로봇 컨트롤러 또는 로컬 PC)를 선택합니다.
4. **시작**을 클릭하여 선택한 예제를 로드합니다.



프록시 유형

- **로봇 내부 프록시**: 로봇 컨트롤러에 내장된 프록시를 사용합니다. 컨트롤러 펌웨어 **v7.7.0.0 이상**에 권장됩니다.
- **로컬 컨트롤러 프록시**: SDK와 함께 제공되는 경량 프록시를 사용하고 호스트 PC에서 실행됩니다. 이는 컨트롤러 펌웨어가 **v7.7.0.0** 미만일 때 유일한 옵션입니다.
- **Airbot** 로봇의 경우 로봇 내부 프록시만 지원됩니다. 로컬 프록시는 Airbot 컨트롤러에 연결할 수 없습니다.

2 어휘

용어	설명
펜던트를 가르치다	로봇에 부착되어 로봇을 가르치고 제어하는 데 사용되는 펜던트입니다.
SDK	로봇 프로그래밍 및 제어에 사용되는 소프트웨어 개발 키트
로봇 네트워크	로봇과 외부 컴퓨터 간의 네트워크 연결
제어 장치	동작 명령 실행, 센서 데이터 처리, 로봇 상태 관리를 담당하는 로봇의 제어 장치입니다.
로봇 팔	다수의 관절과 링크로 구성된 로봇의 주요 움직이는 부분
서보 시스템	로봇 관절의 움직임을 제어하는 모터 구동 시스템으로 정밀한 위치 및 속도 제어가 가능합니다.
가르침	로봇이나 티치펜던트의 수동 조작을 통해 로봇의 운동 궤적과 동작을 기록하는 과정
관절	로봇 팔의 다양한 링크를 연결하는 가동 부품으로, 각 관절은 1자유도에 해당합니다.
데카르트 좌표	공간에서 로봇의 위치와 방향을 설명하는 데 사용되는 서로 수직인 3개의 X, Y, Z 축을 기반으로 하는 3차원 좌표계
포즈	위치 좌표 및 회전 각도를 포함하여 공간에서 로봇의 위치와 방향의 조합
궤도	공간에서 이동하는 로봇 엔드 이펙터의 경로는 일반적으로 일련의 포즈 포인트로 구성됩니다.
유효 탑재량	로봇의 엔드 이펙터가 운반하는 무게와 물체는 로봇의 모션 성능과 정확성에 영향을 미칩니다.
좌표계	기본 좌표계, 도구 좌표계, 사용자 좌표계 등을 포함하여 로봇 위치 및 방향을 설명하는 데 사용되는 참조 시스템입니다.
OVC	전체 속도 제어(Overall Velocity Control)는 로봇의 전체 동작 속도 배율을 설정하는 데 사용됩니다.
OAC	로봇의 전체 가속 배율을 설정하는 데 사용되는 전체 가속도 제어
TF	로봇의 엔드툴을 원점으로 하는 좌표계인 Tool Frame
UF	편리한 프로그래밍 및 위치 지정을 위한 사용자 정의 좌표계인 사용자 프레임

용어	설명
TCS	티칭 좌표계(Teach Coordinate System)는 티칭 시 사용되는 좌표계입니다.
DH 매개변수	Denavit-Hartenberg 매개변수, 로봇 링크의 기하학적 관계를 설명하는 데 사용되는 표준 매개변수
PR 레지스터	포즈 레지스터(Pose Register)는 로봇 포즈 정보를 저장하는 데 사용되는 레지스터입니다.
MR 레지스터	모션 레지스터(Motion Register)는 모션 관련 매개변수를 저장하는 데 사용되는 레지스터입니다.
SR 레지스터	문자열 레지스터, 문자열 정보를 저장하는 데 사용되는 레지스터
R 레지스터	숫자 정보를 저장하는 데 사용되는 레지스터인 Real Register
MH 레지스터	Modbus Holding Register, Modbus 통신을 위한 Holding Register
MI 레지스터	Modbus 입력 레지스터, Modbus 통신을 위한 입력 레지스터
BAS	로봇 제어 프로그램을 작성하는 데 사용되는 고급 프로그래밍 언어인 기본 스크립트
스카라	선택적 컴플라이언스 조립 로봇 암은 4축 산업용 로봇의 일종입니다.
협동로봇	인간과 안전하게 협업할 수 있는 로봇으로, 일반적으로 힘 감지 및 충돌 감지 기능을 갖추고 있습니다.
산업용 로봇	산업 자동화 생산에 사용되는 로봇으로 일반적으로 고정밀, 고속, 고부하 능력을 갖추고 있습니다.
Copper	Agilebot의 협동로봇 제품 라인의 코드명
Bronze	Agilebot 산업용 로봇 제품 라인의 코드명

3 데이터 구조

3.1 상태 코드

설명

인터페이스에서 반환된 상태 코드입니다.

가져오기

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

필드

Name	Enum Value	Description
OK	0	실행 성공
INCOMPATIBLE_VERSION	-1	호환되지 않는 버전
TIMEOUT	-3	연결 시간 초과
INTERFACE_NOT_IMPLEMENTED	-4	인터페이스가 구현되지 않았습니다.
INDEX_OUT_OF_RANGE	-5	범위를 벗어난 인덱스
UNSUPPORTED_FILETYPE	-6	지원되지 않는 파일 형식
UNSUPPORTED_PARAMETER	-7	지원되지 않는 로봇 매개변수
UNSUPPORTED_SIGNALTYPE	-8	지원되지 않는 IO 신호 유형
PROGRAM_NOT_FOUND	-9	프로그램을 찾을 수 없습니다
PROGRAM_POSE_NOT_FOUND	-10	프로그램 포즈 정보를 찾을 수 없습니다

Name	Enum Value	Description
WRITE_PROGRAM_FAILED	-11	프로그램 포즈 정보를 업데이트하지 못했습니다.
GET_ALARM_CODE_FAILED	-12	알람 코드를 가져오기 위해 알람 서비스에 액세스하지 못했습니다.
WRONG_POSITION_INFO	-13	컨트롤러가 잘못된 위치 정보를 반환함
UNSUPPORTED_TRA_TYPE	-14	지원되지 않는 모션 유형
FILE_NOT_FOUND	-15	파일 또는 폴더를 찾을 수 없습니다
FILE_ALREADY_EXIST	-16	파일이 이미 존재합니다.
INVALID_PARAMETER	-27	잘못된 매개변수
GET_ALARM_DESC_FAILED	-17	알람 코드를 기준으로 알람 정보를 가져오지 못했습니다.
RESET_ALARM_ERRORS_FAILED	-18	알람 정보를 재설정하지 못했습니다.
GET_ALL_ALARMS_FAILED	-19	모든 알람 정보를 가져오지 못했습니다.
WRONG_DATA_FORMAT	-20	잘못된 데이터 형식이 수신되었습니다.
CONNECT_FAILED	-21	초기화 연결에 실패했습니다. IP 주소 또는 제어 캐비닛 서비스를 확인하십시오.
POSE_INDEX_DUPLICATED	-23	포즈 인덱스가 중복되었습니다.
CONTROLLER_ERROR	-254	컨트롤러 오류입니다. 개발자에게 문의하세요.
OTHER_REASON	-255	기타 이유

3.2 로봇상태

설명

로봇 작동 상태.

가져오기

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

필드

Name	Enum Value	Description
WRONG_DATA	-1	알 수 없는 상태
ROBOT_IDLE	0	로봇 유휴
ROBOT_RUNNING	1	로봇이 달리고 있다
ROBOT_TEACHING	2	로봇티칭
ROBOT_IDLE_TO_RUNNING	101	로봇 중간 상태, 유휴 상태에서 실행 중
ROBOT_IDLE_TO_TEACHING	102	로봇 중간 상태, 교육 중 유휴 상태
ROBOT_RUNNING_TO_IDLE	103	로봇 중간 상태, 유휴 상태로 실행 중
ROBOT_TEACHING_TO_IDLE	104	로봇 중간 상태, 유휴 교육

3.3 Ctrl상태

설명

컨트롤러 작동 상태.

가져오기

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

필드

Name	Enum Value	Description
WRONG_DATA	-1	알 수 없는 컨트롤러 상태
CTRL_INIT	0	컨트롤러 초기화 중
CTRL_ENGAGED	1	컨트롤러 활성화됨
CTRL_ESTOP	2	컨트롤러 비상 정지
CTRL_TERMINATED	3	컨트롤러가 종료되었습니다.
CTRL_ANY_TO_ESTOP	101	컨트롤러 중간 상태, 모두 비상 정지
CTRL_ESTOP_TO_ENGAGED	102	컨트롤러 중간 상태, 비상 정지 활성화
CTRL_ESTOP_TO_TERMINATED	103	컨트롤러 중간 상태, 비상 정지 ~ 종료

3.4 서보상태

설명

서보 컨트롤러 상태.

가져오기

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

필드

Name	Enum Value	Description
WRONG_DATA	-1	알 수 없는 서보 컨트롤러 상태
SERVO_IDLE	1	서보 컨트롤러 유휴
SERVO_RUNNING	2	서보 컨트롤러 실행 중

Name	Enum Value	Description
SERVO_DISABLE	3	서보 컨트롤러 비활성화됨
SERVO_WAIT_READY	4	준비를 기다리는 서보 컨트롤러
SERVO_WAIT_DOWN	5	종료를 기다리는 서보 컨트롤러
SERVO_INIT	10	서보 컨트롤러 초기화 중

3.5 TransformStatusEnum

설명

오프라인 궤적 파일 변환 상태에 대한 열거형입니다.

가져오기

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

필드

Name	Enum Value	Description
TRANSFORM_START	0	변환 작업이 시작되었습니다.
TRANSFORM_RUNNING	1	변환 작업 진행 중
TRANSFORM_SUCCESS	2	변환 작업이 성공적으로 완료되었습니다.
TRANSFORM_FAILED	3	변환 작업 실패
TRANSFORM_NOT_FOUND	4	변환 작업을 찾을 수 없습니다
TRANSFORM_UNKNOWN	-1	알 수 없는 변환 작업 상태

3.6 PayloadInfo

설명

`PayloadInfo` 클래스는 페이로드 ID, 무게, 질량 중심 및 관성 모멘트를 포함한 로봇의 페이로드 정보를 저장하는 데 사용됩니다. 이 정보는 부하 조건에서 로봇의 운동학적 및 동적 분석, 특히 경로 계획 및 토크 계산에 중요합니다.

가져오기

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

속성

Property	Type	Description
`Id`	`uint`	다양한 페이로드 구성을 고유하게 식별하는 데 사용되는 페이로드 ID
`Comment`	`string`	페이로드에 대한 추가 정보를 설명하는 데 사용되는 주석
`Weight`	`double`	페이로드 중량(단위: 킬로그램)
`MassCenter`	`MassCenter`	페이로드 질량 중심(X, Y, Z 좌표)
`InertiaMoment`	`InertiaMoment`	페이로드 관성 모멘트(LX, LY, LZ)

예

```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 매스센터

설명

`MassCenter` 클래스는 X, Y 및 Z 좌표를 포함하는 페이로드의 질량 중심을 나타내는 데 사용됩니다. 질량 중심은 공간에서 페이로드의 기하학적 중심이며 로봇 동작 제어 및 토크 계산에 중요합니다.

가져오기

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

속성

Property	Type	Description
`X`	`double`	질량 중심의 X 좌표(단위: 밀리미터)
`Y`	`double`	질량 중심의 Y 좌표(단위: 밀리미터)
`Z`	`double`	질량 중심의 Z 좌표(단위: 밀리미터)

3.6.2 관성모멘트

설명

`InertiaMoment` 클래스는 LX, LY 및 LZ 구성 요소를 포함하는 페이로드의 관성 모멘트를 나타내는 데 사용됩니다. 관성 모멘트는 회전 변화에 대한 페이로드의 저항을 나타내며 로봇 역학 분석 및 제어에 중요합니다.

가져오기

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

속성

Property	Type	Description
`LX`	`double`	관성 모멘트의 X 성분(단위: 킬로그램·밀리미터^2)

Property	Type	Description
LY	double	관성모멘트의 Y성분(단위: 킬로그램·밀리미터^2)
LZ	double	관성모멘트의 Z 성분 (단위: 킬로그램·밀리미터^2)

3.7 변환상태

설명

오프라인 궤적 파일 변환 상태에 대한 열거형입니다.

가져오기

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

필드

Enum Value	Value	Description
TRANSFORM_START	0	변환 작업이 시작되었습니다.
TRANSFORM_RUNNING	1	변환 작업 진행 중
TRANSFORM_SUCCESS	2	변환 작업이 성공적으로 완료되었습니다.
TRANSFORM_FAILED	3	변환 작업 실패
TRANSFORM_NOT_FOUND	4	변환 작업을 찾을 수 없습니다
TRANSFORM_UNKNOWN	-1	데이터 오류, 알 수 없는 상태

3.8 TCSType

설명

TCS 좌표계 유형.

가져오기

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

필드

Name	Enum Value	Description
WRONG_TYPE	-1	잘못된 유형
JOINT	0	Joint 공간
BASE	1	기본 좌표계
WORLD	2	세계 좌표계
USER	3	사용자 좌표계
TOOL	4	도구 좌표계
RTCP_USER	5	RTCP 사용자 좌표계
RTCP_TOOL	6	RTCP 도구 좌표계

3.9 MotionPose

설명

로봇의 위치 구조를 설명합니다. 좌표 데이터에서 XYZ 방향의 거리는 밀리미터(mm) 단위로 측정되고, 각도 데이터는 각도(°) 단위로 측정됩니다. 일부 버전에서는 각도 정보가 라디안 단위입니다. 자세한 내용은 함수 목록 반환 결과 설명을 참조하세요.

가져오기

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

속성

Property	Type	Description
CartData	BaseCartData	데카르트 데이터
Joint	Joint	Joint 데이터
Pt	PoseType	Position 유형, 기본값은 알 수 없음

예

```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

설명

직교 좌표계에서의 로봇의 위치 및 자세 정보를 설명합니다. 공간 좌표는 밀리미터(mm) 단위로 측정되며, 자세 정보에는 손목, 팔 자세와 각 축의 회전 횟수가 포함됩니다.

가져오기

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

속성

Property	Type	Description
Position	Position	로봇의 공간좌표(X, Y, Z, A, B, C)
Posture	Posture	로봇의 자세정보(손목, 팔자세, 축회전수)

예

```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

설명

데카르트 좌표계에서 로봇의 위치 및 회전 각도 좌표를 설명합니다. X, Y, Z 방향의 거리는 밀리미터(mm) 단위로 측정되고, A, B, C 방향의 각도는 도(°) 단위로 측정됩니다.

가져오기

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

속성

Property	Type	Description
X	double	직교 좌표계의 X 방향 거리(단위: 밀리미터)
Y	double	직교 좌표계의 Y 방향 거리(단위: 밀리미터)
Z	double	직교 좌표계의 Z 방향 거리(단위: 밀리미터)
A	double	직교 좌표계의 A 방향 각도(단위: 도)
B	double	직교 좌표계의 B 방향 각도(단위: 도)
C	double	직교 좌표계의 C 방향 각도(단위: 도)

예

```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

설명

손목, 팔 자세, 각 축의 회전수 등 로봇의 자세 정보를 설명합니다. Posture 정보는 공간에서 로봇의 특정 자세를 정의하는 데 사용됩니다.

가져오기

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

속성

Property	Type	Description
WristFlip	int	손목 뒤집기 자세. 범위: -1, 0, 1. 6축 로봇 J5 관절 구성의 경우: 1 = 손목이 아래로 뒤집힌 상태, -1 = 손목이 위로 뒤집힌 상태.
ArmUpDown	int	팔 올리기/down 자세. 범위: -1, 0, 1. 6축 로봇 J3 관절 구성의 경우: 1 = 팔 위(전방 조건: 관절 4에서 관절 2까지의 선 위의 관절 3 및 관절 3 각도 < 0), -1 = 팔 아래(관절 3 각도 > 0)
ArmBackFront	int	Arm front/back posture. 범위: -1, 0, 1. 6축 로봇 J1 관절 구성의 경우: 1 = 팔 앞(협동 로봇은 앞을 향하고, 조인트-2는 관절-1의 왼쪽에 있음), -1 = 팔 뒤(관절-1의 오른쪽에 있는 관절-2).
ArmLeftRight	int	팔 왼쪽/right 자세. 범위: -1, 0, 1. 4축 SCARA 로봇 J2 조인트 구성의 경우: 1 = 오른쪽의 SCARA 팔, -1 = 왼쪽의 SCARA 팔.
TurnCircle	List<int>	각 축의 다중 회전 횟수입니다. 범위: -1, 0, 1. 축이 0°에 있으면 회전 횟수 = 0. 선형 또는 원형 이동 중에 컨트롤러는 시작 포즈에 가장 가까운 회전 횟수를 자동 선택하므로 최종 값은 학습된 자세와 다를 수 있습니다. 축 1, 4, 5, 6의 경우: ≥ 180° → 값 ≥ 1; -179.99° ~ 179.99° → 0; ≤ -180° → 값 ≤ -1.

예

```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());
```

3.11 Joint

설명

Describes the angle data of each robot joint. 각 관절 각도 값은 관절 공간에서 로봇의 특정 위치를 정의하는 데 사용됩니다. 각도 단위는 일반적으로 도(°)이지만, 구체적인 단위는 실제 로봇 시스템을 기준으로 확인해야 합니다.

가져오기

c#

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

속성

Property	Type	Description
J1	double	로봇의 첫 번째 관절 각도
J2	double	로봇의 두 번째 관절 각도
J3	double	로봇의 세 번째 관절 각도
J4	double	로봇의 네 번째 관절 각도
J5	double	로봇의 다섯 번째 관절 각도
J6	double	로봇의 여섯 번째 관절 각도
J7	double	로봇의 일곱 번째 관절 각도
J8	double	로봇의 8번째 관절 각도

Property	Type	Description
J9	double	로봇의 9번째 관절 각도

예

```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());
```

메모

- 관절 각도의 단위는 일반적으로 도(°)이지만 일부 로봇 시스템에서는 라디안(rad)을 사용하는 경우도 있습니다. 실제 로봇 시스템 문서를 기반으로 장치를 확인하십시오.
- 관절 각도의 범위는 일반적으로 로봇 하드웨어에 의해 제한됩니다. 범위를 초과하면 오류가 발생하거나 장비가 손상될 수 있습니다.

3.12 PoseType

설명

관절 각도 데이터인지, 데카르트 공간 좌표인지, 알 수 없는 유형인지 구분하는 데 사용되는 로봇 자세 데이터의 유형을 정의합니다. 이 열거형은 다양한 유형의 데이터가 프로그램에서 올바르게 처리될 수 있도록 로봇 포즈 데이터의 형식을 식별하는 데 사용됩니다.

가져오기

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

열거형 값

Enum Value	Description
Unknown	알 수 없는 유형, 포즈 데이터 유형이 정의되지 않았음을 나타냅니다.
Joint	데이터가 관절 각도임을 나타내는 Joint 각도 데이터 유형
Cart	데카르트 공간 좌표 데이터 유형(데이터가 데카르트 좌표임을 나타냄)

3.13 DHparam

설명

DHparam 클래스는 Denavit-Hartenberg parameters(D-H 매개변수)를 기반으로 로봇 링크 매개변수를 설명하는 데 사용됩니다. 이러한 매개변수는 로봇 관절 간의 기하학적 관계를 정의하는 데 사용되며 로봇 운동학 및 역학 분석의 기초가 됩니다.

가져오기

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

속성

Property	Type	Description
id	uint	서로 다른 링크를 구별하는 데 사용되는 링크의 고유 식별자
a	double	인접한 관절 사이의 축 거리를 나타내는 링크 길이(단위: 밀리미터)
alpha	double	인접한 관절 축 사이의 각도를 나타내는 링크 비틀림 각도(단위: 도 또는 라디안)

Property	Type	Description
d	double	Joint 거리, 현재 관절 축을 따라 다음 관절까지의 거리를 나타냄(단위: 밀리미터)
offset	double	Joint 각도 오프셋, 관절의 초기 각도 오프셋을 나타냄(단위: 도 또는 라디안)

생성자

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

메모

- **단위 일관성**: a 및 d 의 단위는 일관성이 있어야 하며(일반적으로 밀리미터), alpha 및 offset 의 단위도 일관성이 있어야 합니다(일반적으로 도 또는 라디안).
- **각도 단위**: 일부 로봇 시스템에서는 각도 단위가 도 대신 라디안일 수 있습니다. 실제 요구 사항에 따라 단위를 확인하고 통합하십시오.
- **D-H 매개변수 정의**: D-H 매개변수의 정의는 특정 로봇 모델 및 좌표계 규칙에 따라 다릅니다. DHparam 클래스를 사용할 때 매개변수 정의가 로봇의 실제 기하학적 구조와 일치하는지 확인하세요.

3.14 장바구니 상태

설명

CartStatus 클래스는 데카르트 좌표계에서 각 축의 상태를 나타내는 데 사용됩니다. 각 축의 상태는 부울 값으로 표시되며, true 는 축을 사용할 수 있음을 나타내고 false 는 축을 사용할 수 없음을 나타냅니다. 이 상태 클래스는 일반적으로 로봇 동작 제어에서 특정 축이 제대로 작동할 수 있는지 확인하는 데 사용됩니다.

가져오기

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

속성

Property	Type	Description
`X`	`bool`	X 방향 상태, 기본값은 `true` (사용 가능)
`Y`	`bool`	Y 방향 상태, 기본값은 `true` (사용 가능)
`Z`	`bool`	Z 방향 상태, 기본값은 `true` (사용 가능)
`A`	`bool`	A 방향 상태, 기본값은 `true` (사용 가능)
`B`	`bool`	B 방향 상태, 기본값은 `true` (사용 가능)
`C`	`bool`	C 방향 상태, 기본값은 `true` (사용 가능)

3.15 조인트상태

설명

`JointStatus` 클래스는 각 로봇 관절의 상태를 나타내는 데 사용됩니다. 각 관절의 상태는 부울 값으로 표시되며 `true` 는 관절이 사용 가능함을 나타내고 `false` 는 관절이 사용 가능하지 않음을 나타냅니다. 이 상태 클래스는 일반적으로 로봇 동작 제어에서 특정 관절이 제대로 작동할 수 있는지 여부를 결정하는 데 사용됩니다.

가져오기

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

속성

Property	Type	Description
`J1`	`bool`	Joint 1의 상태, 기본값은 `true` (사용 가능)
`J2`	`bool`	Joint 2의 상태, 기본값은 `true` (사용 가능)

Property	Type	Description
J3	bool	Joint 3의 상태, 기본값은 true (사용 가능)
J4	bool	Joint 4의 상태, 기본값은 true (사용 가능)
J5	bool	Joint 5의 상태, 기본값은 true (사용 가능)
J6	bool	Joint 6의 상태, 기본값은 true (사용 가능)
J7	bool	Joint 7의 상태, 기본값은 true (사용 가능)
J8	bool	Joint 8의 상태, 기본값은 true (사용 가능)
J9	bool	Joint 9의 상태, 기본값은 true (사용 가능)

3.16 끌기 상태

설명

DragStatus 클래스는 직교 좌표계 및 관절의 상태를 포함하여 로봇 팔의 드래그 상태를 나타내는 데 사용됩니다. 또한 로봇이 연속 드래그 모드에 있는지 여부를 나타내는 플래그 IsContinuousDrag 가 포함되어 있습니다. 이 상태 클래스는 현재 드래그 모드와 각 축의 상태를 결정하기 위해 로봇 드래그 제어에 일반적으로 사용됩니다.

가져오기

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

속성

Property	Type	Description
CartStatus	CartStatus	직교좌표계 현황
JointStatus	JointStatus	관절의 상태

Property	Type	Description
IsContinuousDrag	bool	로봇이 연속 드래그 모드에 있는지 여부, 기본값은 false

생성자

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

- CartStatus 및 JointStatus 를 초기화하고 IsContinuousDrag 를 false 로 설정합니다.

예

```c#
DragStatus dragStatus = new DragStatus();
dragStatus.CartStatus.X = false; // X-axis unavailable
dragStatus.JointStatus.J3 = false; // Joint 3 unavailable
dragStatus.IsContinuousDrag = true; // Set to continuous drag mode
Console.WriteLine($"X-axis status: {dragStatus.CartStatus.X}, Joint 3 statu
s: {dragStatus.JointStatus.J3}, Is continuous drag: {dragStatus.IsContinuous
Drag}");
```

3.17 ProgramPose

설명

ProgramPose 클래스는 프로그램에서 관절 좌표 또는 데카르트 좌표일 수 있는 포즈(위치 및 방향)를 나타내는 데 사용됩니다. 이 클래스에는 포즈, 데이터(관절 또는 직교 좌표 정보), 이름 및 설명에 대한 고유 식별자가 포함됩니다. 이 클래스는 로봇 프로그램에서 포즈 정보의 관리 및 조작을 용이하게 합니다.

가져오기

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

속성

Property	Type	Description
Id	int	포즈의 고유 식별자
PoseData	ProgramPoseData	관절 또는 데카르트 좌표 정보를 포함한 포즈 데이터
Name	string	포즈의 이름
Comment	string	포즈에 대한 코멘트

생성자

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

- Id , PoseData , Name 및 Comment 를 초기화합니다.

예

```c#
ProgramPose programPose = new ProgramPose();
programPose.Id = 1; // Set the unique identifier for the pose
programPose.PoseData = new ProgramPoseData(); // Create pose data
programPose.Name = "Pose1"; // Set the name of the pose
programPose.Comment = "This is a sample pose"; // Set the comment for the po
se
Console.WriteLine($"Pose ID: {programPose.Id}, Name: {programPose.Name}, Com
ment: {programPose.Comment}");
```

3.17.1 ProgramPoseData

설명

ProgramPoseData 클래스는 데카르트 공간 좌표 및 자세 정보, 관절 각도 정보 및 포즈 유형을 포함하여 프로그램에서 포즈 데이터를 나타내는 데 사용됩니다. 이 클래스는 특정 포즈 데이터의 저장 및 관리를 용이하게 합니다.

가져오기

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

속성

Property	Type	Description
CartData	ProgramCartData	데카르트 데이터
Joint	Joint	Joint 데이터
Pt	PoseType	포즈 유형, 기본값은 알 수 없음

3.17.2 ProgramCartData

설명

ProgramCartData 클래스는 프로그램에서 직교 좌표계 데이터를 나타내는 데 사용됩니다. BaseCartData 클래스를 참조하여 공간 좌표 및 자세 정보를 포함하고, Uf 및 Tf 값을 통해 좌표계 유형을 결정합니다. Uf 는 사용자 프레임을 나타내고 Tf 는 도구 프레임을 나타냅니다. Uf 및 Tf 의 값이 -1 인 경우 시스템의 기본 좌표계를 사용함을 나타냅니다. 이 클래스는 로봇 프로그래밍에서 데카르트 공간의 포즈 정보를 정의하고 관리하는 데 사용됩니다.

가져오기

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

속성

Property	Type	Description
BaseCart	BaseCartData	로봇의 직교위치 및 자세정보
Uf	int	사용자 프레임, -1 는 시스템의 좌표계 사용을 나타냅니다.
Tf	int	도구 프레임, -1 는 시스템 좌표계의 사용을 나타냅니다.

3.18 FileType

설명

`FileType` 열거형은 업로드가 허용되는 파일 유형을 정의하는 데 사용됩니다. 소스와 형식에 따라 다양한 유형의 로봇 프로그램 파일을 구별합니다. 이 열거형은 파일 관리, 업로드 및 프로그램 구문 분석을 위해 로봇 프로그래밍 환경에서 사용되며 시스템이 다양한 유형의 프로그램 파일을 올바르게 식별하고 처리하는 데 도움이 됩니다.

가져오기

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

열거형 값

Enum Value	Description
`UserProgram`	사용자가 포인트 선택을 통해 생성한 프로그램 파일로, 각 프로그램에는 `.xml` 및 `.json` 파일이 포함되어 있습니다.
`BlockProgram`	블록 프로그래밍을 통해 사용자가 생성한 프로그램 파일로, 각 프로그램에는 `.block` , `.xml` , `.json` 파일이 포함됩니다.
`TrajectoryProgram`	일반적으로 오프라인 프로그래밍의 경로 계획에 사용되는 오프라인 궤적 프로그램 파일입니다.

3.19 SignalType

설명

`SignalType` 열거형은 로봇 시스템에서 지원되는 신호 유형을 정의하는 데 사용됩니다. 목적과 소스에 따라 다양한 디지털 신호와 아날로그 신호를 구별합니다. 이 열거형은 신호 구성, 신호 처리 및 논리 판단을 위해 로봇 제어 시스템에서 사용되며 시스템이 다양한 유형의 신호를 정확하게 식별하고 관리하는 데 도움이 됩니다.

가져오기

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

열거형 값

Enum Value	Description
DI	디지털 입력, 외부 디지털 신호를 수신하는 데 사용됩니다.
DO	외부 장치나 액추에이터를 제어하는 데 사용되는 디지털 출력입니다.
RI	로봇의 손목에서 디지털 신호를 수신하는 데 사용되는 로봇 입력입니다.
RO	로봇 손목의 액추에이터를 제어하는 데 사용되는 로봇 출력.
UI	사용자 입력, 사용자 정의 디지털 신호를 수신하는 데 사용됩니다.
UO	사용자 출력, 사용자 정의 디지털 신호를 출력하는 데 사용됩니다.
TDI	도구 디지털 입력, 도구 끝에서 디지털 신호를 수신하는 데 사용됩니다.
TDO	도구 끝에 있는 액추에이터를 제어하는 데 사용되는 도구 디지털 출력입니다.
GI	그룹 입력, 디지털 신호 조합을 수신하는 데 사용됩니다.
GO	그룹 출력, 디지털 신호의 조합을 출력하는 데 사용됩니다.
AI	아날로그 입력, 연속적인 아날로그 신호를 수신하는 데 사용됩니다.
AO	아날로그 출력, 연속적인 아날로그 신호를 출력하는 데 사용됩니다.
TAI	도구 아날로그 입력, 도구 끝에서 아날로그 신호를 수신하는 데 사용됩니다.

3.20 PoseRegister

설명

`PoseRegister` 클래스는 관절 좌표 또는 데카르트 좌표일 수 있는 PR 레지스터에서 포즈(위치 및 방향)를 나타내는 데 사용됩니다. 이 클래스에는 포즈, 데이터(관절 또는 직교 좌표 정보), 이름 및 설명에 대한 고유 식별자가 포함됩니다. 이 클래스는 로봇 프로그램에서 포즈 정보의 관리 및 조작을 용이하게 합니다.

가져오기

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

속성

Property	Type	Description
`Id`	`int`	포즈의 고유 식별자
`PoseData`	`PoseRegisterData`	관절 또는 데카르트 좌표 정보를 포함한 포즈 데이터
`Name`	`string`	포즈의 이름
`Comment`	`string`	포즈에 대한 코멘트

생성자

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

- `Id` , `PoseData` , `Name` 및 `Comment` 를 초기화합니다.

예

```c#
PoseRegister pose = new PoseRegister();
pose.Id = 1; // Set the unique identifier for the pose
pose.PoseData = new PoseRegisterData(); // Create pose data
pose.Name = "Pose1"; // Set the name of the pose
pose.Comment = "This is a sample pose"; // Set the comment for the pose
Console.WriteLine($"Pose ID: {pose.Id}, Name: {pose.Name}, Comment: {pose.Co
mment}");
```

3.20.1 PoseRegisterData

설명

PoseRegisterData 클래스는 데카르트 공간 좌표 및 자세 정보, 관절 각도 정보 및 포즈 유형을 포함하여 PR 레지스터의 포즈 데이터를 나타내는 데 사용됩니다. 이 클래스는 특정 포즈 데이터의 저장 및 관리를 용이하게 합니다.

가져오기

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

속성

Property	Type	Description
CartData	BaseCartData	데카르트 데이터
Joint	Joint	Joint 데이터
Pt	PoseType	포즈 유형, 기본값은 알 수 없음

3.22 동등 어구

설명

Coordinate 클래스는 로봇 시스템의 좌표계를 나타내는 데 사용됩니다. 고유 식별자(ID), 이름, 설명, 동작 그룹 번호, 특정 포즈 데이터 등 좌표계에 대한 기본 정보가 포함됩니다. 이 클래스는 로봇 프로그래밍 및 제어 시스템에서 좌표계의 위치와 방향을 정의 및 관리하고 프로그램의 동작 계획 및 경로 제어를 용이하게 하는 데 사용됩니다.

가져오기

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

속성

Property	Type	Description
Id	int	좌표계의 고유 식별자
Name	string	좌표계를 식별하고 설명하는 데 사용되는 좌표계의 이름
Comment	string	좌표계의 목적이나 특성을 더 자세히 설명하는 데 사용되는 좌표계에 대한 설명입니다.
GroupId	int	좌표계가 속한 모션 그룹 번호로, 좌표계 분류 및 관리에 사용됩니다.
Data	Position	위치 및 방향 정보를 포함한 좌표계의 특정 포즈 데이터

예

```c#
// Create a Coordinate instance
Coordinate coordinate = new Coordinate
{
    Id = 1, // Set the unique identifier
    Name = "UserCoordinate1", // Set the name
    Comment = "This is a user-defined coordinate system", // Set the comment
    GroupId = 1, // Set the motion group number
    Data = new Position { X = 100, Y = 200, Z = 300, A = 45, B = 30, C = 60
} // Set the pose data
};
```

3.22.1 CoordinateType

설명

CoordinateType 열거형은 좌표계 유형을 정의하는 데 사용됩니다. 이는 사용자 좌표계와 도구 좌표계를 구별합니다. 이 열거형은 로봇 프로그래밍 및 제어 시스템에서 좌표계의 목적을 명확하게 지정하여 시스템이 좌표계와 관련된 작업을 올바르게 처리하도록 돕는 데 사용됩니다.

가져오기

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

열거형 값

Enum Value	Description
UserCoordinate	사용자 정의 좌표계를 정의하는 데 사용되는 사용자 좌표계입니다.
ToolCoordinate	도구 좌표계(예: 엔드 이펙터)의 좌표계를 정의하는 데 사용됩니다.

3.22.2 CoordSummary

설명

CoordSummary 클래스는 좌표계의 요약 정보를 나타내는 데 사용됩니다. 여기에는 좌표계의 유형, 고유 식별자, 이름, 설명 및 그룹 ID가 포함됩니다. 이 클래스는 로봇 프로그래밍 환경에서 좌표계의 메타데이터를 관리 및 저장하여 프로그램에서 좌표계에 대한 빠른 액세스 및 조작을 용이하게 하는 데 사용됩니다.

가져오기

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

속성

Property	Type	Description
Type	CoordinateType	사용자 좌표계 또는 공구 좌표계일 수 있는 좌표계 유형
Id	int	좌표계의 고유 식별자
Name	string	좌표계 이름
Comment	string	목적이나 특성을 설명하는 데 사용되는 좌표계에 대한 설명입니다.
GroupId	int	좌표계가 속한 그룹 ID로, 좌표계의 분류 및 관리에 사용됩니다.

예

```c#
// Create a CoordSummary instance
CoordSummary coordSummary = new CoordSummary
{
    Type = CoordinateType.UserCoordinate, // Set to user coordinate system
    Id = 1, // Set the unique identifier
    Name = "UserCoord1", // Set the name
    Comment = "This is a user-defined coordinate system", // Set the comment
    GroupId = 0 // Set the group ID
};
```

3.23 RobotTopicType

설명

`RobotTopicType` 열거형은 `SubPub.SubscribeStatus` 에서 로봇 상태 구독 항목을 지정하는 데 사용됩니다.

가져오기

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

필드

이름	설명
`TopicCurrentJoint`	로봇 관절 상태 피드백 게시
`TopicCurrentCartesian`	현재 TCP 데카르트 좌표를 게시합니다.
`TopicUF`	현재 사용자 프레임 정보를 게시합니다.
`TopicTF`	현재 도구 프레임 정보를 게시합니다.
`TopicVelocity`	전역 속도 비율 게시

이름	설명
TopicRunningStatus	컨트롤러 실행 상태 게시
TopicInterpreterStatus	통역사 현황 공개
TopicRobotStatus	로봇 상태 게시
TopicCtrlStatus	컨트롤러 상태 게시
TopicServoStatus	서보 컨트롤러 상태 게시
TopicTrajectoryRecordsStatus	궤적 재생 녹화 상태
UserOpMode	사용자 조작 모드

3.24 RegTopicType

설명

RegTopicType 열거형은 SubPub.SubscribeRegister 에서 등록 구독 유형을 지정하는 데 사용됩니다.

가져오기

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

필드

이름	설명
R	R 등록 주제
MR	MR 등록 주제
SR	SR 등록 주제
PR	PR 등록 주제

3.25 IOTopicType

설명

`IOTopicType` 열거형은 `SubPub.SubscribeIO` 에서 I/O 구독 유형을 지정하는 데 사용됩니다.

가져오기

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

필드

이름	설명
DI	디지털 입력 주제
DO	디지털 출력 주제
GI	그룹 디지털 입력 주제
GO	그룹 디지털 출력 주제
RI	로봇 입력 주제
RO	로봇 출력 주제
UI	사용자 입력 주제
UO	사용자 출력 주제
TDI	도구 디지털 입력 항목
TDO	도구 디지털 출력 주제
TAI	도구 아날로그 입력 항목
AI	아날로그 입력 주제
AO	아날로그 출력 주제

4 방법 및 예

4.1 로봇의 기본 동작

개요

`Arm` 클래스는 연결 관리, 상태 쿼리 및 제어 명령 발송을 포함하여 Agilebot 로봇을 위한 대부분의 고주파수 인터페이스를 캡슐화합니다. 일반적인 작업 흐름:

1. `Arm(controllerIP, teachPanelIP, localProxy)` 를 인스턴스화합니다.
2. 컨트롤러/teach 펜던트와의 통신을 설정하려면 `ConnectSync()` 를 호출하세요.
3. 시나리오에 따른 통화 모션, 상태, I/O, 및 기타 인터페이스.
4. 리소스를 해제하려면 `DisconnectSync()` 를 호출하세요.

인스턴스화 후 클래스는 내부 설정을 자동으로 처리합니다.

- SDK 버전 확인
- 컨트롤러 유형 식별
- 자동 프록시 서비스 선택
- 로그에 기록되는 통신 경로

메모

- 로봇 소프트웨어 버전이 7.7.0 미만인 경우 `localProxy=true` 를 유지하고 PC에서 로컬 통신 기능을 보장하십시오.
- PC 모드의 산업용 로봇의 경우 연결 후 티치 펜던트 제어 권한을 획득하고 작동 후 해제되는지 확인하십시오.

클래스 생성자

메서드 이름	**Arm(string `controllerIP` , string `teachPanelIP` = null, bool `localProxy` = true)**
설명	사용 가능한 모든 로봇 제어 인터페이스를 포함하는 Agilebot 로봇 클래스 생성자. 다른 기능을 사용하려면 먼저 로봇을 초기화하고 연결해야 합니다.
요청 매개변수	`controllerIP` : 문자열 로봇 컨트롤러 IP 주소 `teachPanelIP` : 문자열 옵션 티치 펜던트 IP; 생략하면 `controllerIP` 로 대체됩니다. `localProxy` : bool 로컬 컨트롤러 프록시 서비스를 사용할지 여부, 기본값은 true입니다. true인 경우 로컬에서 컨트롤러 프록시 서비스를 시작합니다. false인 경우 로봇 컨트롤러에 프록시 서비스가 이미 설치되어 있어야 합니다(로봇 소프트웨어 버전 7.7 이상이 필요함).
반환 값	StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.1.1 로봇에 연결하기

메서드 이름	**ConnectSync()**
설명	Agilebot 로봇과 네트워크 연결을 설정합니다. 로봇 인스턴스를 초기화하려면 Arm 생성자를 먼저 호출해야 합니다. 이 메서드는 생성자에 지정된 프록시 모드를 기반으로 해당 프록시 서비스를 시작합니다.
요청 매개변수	없음
반환 값	StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.1.2 로봇팔과의 연결 확인하기

메서드 이름	IsConnected()
설명	로봇과의 네트워크 연결이 유효한지 확인합니다. 연결이 유효하고 통신이 가능한 경우 `true` 를 반환합니다. 연결이 끊어지거나 설정되지 않은 경우 `false` 를 반환합니다.
요청 매개변수	없음
반환 값	bool: 연결 상태, true는 연결이 유효함을 나타내고, false는 연결이 유효하지 않거나 연결되지 않았음을 나타냅니다.
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.1.3 로봇과의 연결 끊기

메서드 이름	DisconnectSync()
설명	Disconnects from the Agilebot robot and releases related resources. After disconnection, `ConnectSync()` must be called again to communicate.
요청 매개변수	없음
반환 값	StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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 현재 로봇 모델 가져오기

메서드 이름	GetArmModelInfo()
설명	현재 연결된 Agilebot 로봇의 모델 정보를 가져옵니다. 로봇 모델 문자열(예: "GBT-C5A")과 작업 실행 상태를 반환합니다.
요청 매개변수	없음
반환 값	string: 로봇 모델 문자열, 예: "GBT-C5A" StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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 로봇의 작동 상태 가져오기

메서드 이름	GetRobotState()
설명	Agilebot 로봇의 현재 작동 상태를 가져옵니다. 로봇 동작 상태 열거값과 동작 실행 상태를 반환한다.
요청 매개변수	없음
반환 값	RobotState: 로봇 동작 상태 열거형 값 StatusCode: 함수 실행 결과

메서드 이름	GetRobotState()
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

Arm/GetRobotState.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 현재 컨트롤러 작동 상태 가져오기

메서드 이름	GetCtrlState()
설명	Agilebot 로봇 컨트롤러의 현재 작동 상태를 가져옵니다. 컨트롤러 작동 상태 열거 값과 작업 실행 상태를 반환합니다.
요청 매개변수	없음
반환 값	CtrlState: 컨트롤러 동작 상태 열거형 값 StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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 현재 서보 상태 가져오기

메서드 이름	GetServoState()
설명	Agilebot 로봇 서보 시스템의 현재 상태를 가져옵니다. 서보 시스템 상태 열거값과 작업 실행 상태를 반환합니다.
요청 매개변수	없음
반환 값	ServoState: 서보 시스템 상태 열거형 값 StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

Arm/GetServoState.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 로봇 컨트롤러 버전 얻기

메서드 이름	GetVersion()
설명	Agilebot 로봇 컨트롤러의 소프트웨어 버전 정보를 가져옵니다. 컨트롤러 소프트웨어 버전 문자열과 작업 실행 상태를 반환합니다.
요청 매개변수	없음

메서드 이름	GetVersion()
반환 값	string: 컨트롤러 소프트웨어 버전 문자열 StatusCode: 기능 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

Arm/GetVersion.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 로봇의 LED 표시등 설정

메서드 이름	SwitchLedLight(bool `mode`)
설명	Agilebot 로봇 LED 표시등의 on/off 상태를 제어합니다. `true` 는 표시등을 켜고, `false` 는 표시등을 끕니다.
요청 매개변수	`mode` : bool LED 표시 등 제어 모드, true는 켜짐을 나타내고 false는 꺼짐을 나타냅니다.
반환 값	StatusCode: 작업 실행 결과
호환성	협동 로봇만 지원하며 컨트롤러 버전 1.3.6 이상이 필요하며 산업용 로봇은 지원되지 않습니다.
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.1.3+ 산업용(Bronze): 지원되지 않음

예제 코드

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 로봇 서보 온

메서드 이름	ServoOn()
설명	Agilebot 로봇의 서보 시스템을 시작하여 로봇을 제어 가능한 상태로 만듭니다. 서보 시작 후 로봇은 모션 명령을 받아들일 수 있습니다.
요청 매개변수	없음
반환 값	StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.1.11 로봇 서보 끄기

메서드 이름	ServoOff()
설명	Agilebot 로봇의 서보 시스템을 꺼서 로봇을 안전 정지 상태로 만듭니다. 서보 종료 후 로봇은 움직일 수 없습니다.
요청 매개변수	없음
반환 값	StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.1.12 로봇 서보 재설정

메서드 이름	ServoReset()
설명	Agilebot 로봇의 서보 시스템을 재설정하여 오류 상태를 지우고 재시작을 준비합니다. 이 메서드는 일반적으로 시스템을 복원하기 위해 서보 오류가 발생한 후에 호출됩니다.
요청 매개변수	없음
반환 값	StatusCode: 함수 실행 결과

메서드 이름	ServoReset()
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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 비상 정지

메서드 이름	Estop()
설명	Agilebot 로봇의 비상 정지를 실행하여 즉시 모든 동작을 멈추고 안전 상태로 진입합니다. 비상 정지 후 동작을 재개하려면 서보 시스템을 다시 시작해야 합니다.
요청 매개변수	없음
반환 값	StatusCode : 비상 정지 동작 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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 로봇 모션 제어 및 상태

개요

`Motion` 클래스는 로봇 모션 제어의 핵심 객체입니다. 이는 다음을 캡슐화합니다:

- Velocity/acceleration 매개변수 관리
- 좌표계 관리
- 포즈 변환
- 궤적 모션 제어
- 드래그 티칭
- 실시간 제어
- 페이로드 관리

일반적인 작업흐름

`Arm` 가 연결된 후 `arm.Motion` 를 통해 모션 인스턴스를 얻습니다. 별도의 초기화가 필요하지 않습니다.

4.2.1 로봇 매개변수 가져오기

4.2.1.1 OVC 전체 속도 계수 얻기

메서드 이름	Motion.GetOVC()
설명	현재 로봇의 OVC(Overall Velocity Control) 전역 속도 비율을 0~1 범위로 가져옵니다.
요청 매개변수	없음
반환 값	double: 전역 속도 비율 값 StatusCode: 함수 실행 결과

메서드 이름	Motion.GetOVC()
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.2.1.2 OAC 전체 가속 계수 얻기

메서드 이름	Motion.GetOAC()
설명	현재 로봇의 OAC(전체 가속 제어) 전역 가속 비율을 0~1.2 범위로 가져옵니다.
요청 매개변수	없음
반환 값	double: 전역 가속도 비율 값 StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.2.1.3 현재 TF 가져오기

메서드 이름	Motion.GetTF()
설명	로봇이 사용하는 현재 TF(Tool Frame) 도구 좌표계 인덱스를 0~10 범위로 가져옵니다.
요청 매개변수	없음
반환 값	int: TF 공구 좌표계 인덱스 StatusCode: 기능 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.2.1.4 현재 UF 가져오기

메서드 이름	Motion.GetUF()
설명	로봇이 사용하는 현재 UF(사용자 프레임) 사용자 좌표계 인덱스를 0~10 범위로 가져옵니다.

메서드 이름	Motion.GetUF()
요청 매개변수	없음
반환 값	int: UF 사용자 좌표계 인덱스 StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.2.1.5 현재 TCS 교육 좌표계 가져오기

메서드 이름	Motion.GetTCS()
설명	로봇이 사용하는 현재 TCS(교시 좌표계) 좌표계 유형을 가져옵니다. 자세한 내용은 TCSType를 참조하세요.
요청 매개변수	없음
반환 값	TCSType: TCS 티칭 좌표계 유형 열거형 값 StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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 로봇 매개변수 설정

4.2.2.1 OVC 전체 속도 계수 설정

메서드 이름	**Motion.SetOVC(double** `value` **)**
설명	현재 로봇의 OVC(Overall Velocity Control) 전역 속도 비율을 설정합니다.
요청 매개변수	`value` : 이중 속도 비율, 범위 0~1
반환 값	StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.2.2.2 OAC 전체 가속 계수 설정

메서드 이름	**Motion.SetOAC(double** `value` **)**
설명	현재 로봇의 OAC(전체 가속 제어) 전역 가속 비율을 설정합니다.
요청 매개변수	`value` : 이중 가속도 비율, 범위 0.01~1.2
반환 값	StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.2.2.3 현재 TF 공구 좌표계 지수 설정

메서드 이름	**Motion.SetTF(int** `value` **)**
설명	현재 TF(도구 프레임) 도구 좌표계 색인을 설정합니다.
요청 매개변수	`value` : int TF 인덱스, 범위 0~10
반환 값	StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.2.2.4 현재 UF 사용자 좌표계 지수 설정

메서드 이름	**Motion.SetUF(int** `value` **)**
설명	현재 UF(사용자 프레임) 사용자 좌표계 인덱스를 설정합니다.
요청 매개변수	`value` : int UF 인덱스, 범위 0~10
반환 값	StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.2.2.5 현재 TCS 티칭 좌표계 설정

메서드 이름	Motion.SetTCS(TCSType `value`)
설명	현재 TCS(교시 좌표계) 교육 좌표계를 설정합니다. 자세한 내용은 TCSType를 참조하세요.
요청 매개변수	`value` : TCSType TCS 티칭 좌표계 유형
반환 값	StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

Motion/SetMotionParameters.cs

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 데카르트 Position를 Joint 값으로 변환

메서드 이름	**Motion.ConvertCartToJoint(MotionPose `pose` , int `ufIndex` = 0, int `tfIndex` = 0)**
설명	포즈 데이터를 데카르트 좌표에서 관절 좌표로 변환합니다.
요청 매개변수	`pose` : MotionPose 로봇의 직교 자세(PoseType.CART; SDK는 자세가 지정되지 않은 경우 자동으로 가능한 자세를 해결합니다.) `ufIndex` : int 사용자 좌표계 인덱스, 기본값은 0 `tfIndex` : int 도구 좌표계 인덱스, 기본값은 0
반환 값	MotionPose: 변환된 로봇 포즈 데이터 StatusCode: 변환 연산 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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.Joint.J2}, J3={convertPose.Joint.J3}, J4={convertPose.Joint.J4}, J5={convertPose.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.Message}"
            );
            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 Joint 값을 데카르트 Position로 변환

메서드 이름	**Motion.ConvertJointToCart(MotionPose `pose` , int `ufIndex` = 0, int `tfIndex` = 0)**
설명	포즈 데이터를 관절 좌표에서 데카르트 좌표로 변환합니다.
요청 매개변수	`pose` : MotionPose 로봇의 관절 포즈 `ufIndex` : int 사용자 좌표계 인덱스, 기본값은 0 `tfIndex` : int 도구 좌표계 인덱스, 기본값은 0
반환 값	MotionPose: 변환된 로봇 포즈 데이터 StatusCode: 변환 연산 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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 Joint 모션

메서드 이름	**Motion.MoveJoint(MotionPose `pose` , double `vel` = 1, double `acc` = 1)**
설명	관절 공간에서 가장 빠른 경로를 따라 지정된 위치로 이동하도록 로봇 엔드 이펙터를 제어합니다.
요청 매개변수	`pose` : MotionPose 직교 공간 또는 관절 좌표계의 목표 위치 좌표 `vel` : double 동작 속도, 범위 0~1, 최대 속도의 배수를 나타냄 `acc` : double Acceleration, 범위 0~1.2, 최대 가속도의 배수를 나타냄
반환 값	StatusCode: 모션 명령 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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 선형 운동

메서드 이름	**Motion.MoveLine(MotionPose `pose` , double `vel` = 100, double `acc` = 1)**
설명	로봇 엔드 이펙터가 직선을 따라 지정된 위치로 이동하도록 제어합니다. 모션 궤적은 두 지점 사이의 직선입니다.
요청 매개변수	`pose` : MotionPose 데카르트 공간 또는 관절 좌표계의 대상 위치 좌표 `vel` : 이중 모션 속도, 범위 0~5000mm/s,는 로봇 엔드 이펙터 이동 속도를 나타냅니다. `acc` : 이중 가속도, 범위 0~1.2, 최대 가속도의 배수를 나타냅니다.
반환 값	StatusCode: 모션 명령 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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 원형 운동

메서드 이름	**Motion.MoveCircle(MotionPose `pose1` , MotionPose `pose2` , double `vel` = 100, double `acc` = 1)**
설명	중간점과 끝점에 의해 결정되는 호를 사용하여 원형 궤적을 따라 지정된 위치로 이동하도록 로봇 엔드 이펙터를 제어합니다.
요청 매개변수	`pose1` : MotionPose 로봇 모션 중간 포즈 `pose2` : MotionPose 로봇 모션 최종 포즈 `vel` : double 모션 속도, 범위 0~5000mm/s,는 로봇 엔드 이펙터 이동 속도를 나타냅니다. `acc` : double 가속도, 범위 0~1.2, 최대 가속도의 배수를 나타냄
반환 값	StatusCode: 모션 명령 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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 현재 포즈 얻기

메서드 이름	**Motion.GetCurrentPose(PoseType `pt` , int `ufIndex` = 0, int `tfIndex` = 0)**
설명	로봇의 현재 자세를 가져옵니다. 이는 직교 공간 또는 관절 좌표계의 자세 정보일 수 있습니다.
요청 매개변수	`pt` : PoseType 포즈 유형 `ufIndex` : int 사용자 좌표계 인덱스(PoseType.CART에만 유효; 기본값 0) `tfIndex` : int 공구 좌표계 인덱스(PoseType.CART에만 유효; 기본값 0)
반환 값	MotionPose: 로봇 포즈 데이터 StatusCode: 작업 실행 결과 가져오기
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

Motion/GetCurrentPose.cs

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 DH 매개변수 가져오기

메서드 이름	Motion.GetDHParam()
설명	로봇의 DH(Denavit-Hartenberg) 매개변수를 가져옵니다.
요청 매개변수	없음
반환 값	List<DHparam>: DH 매개변수 목록 StatusCode: 연산 실행 결과 가져오기
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): 지원되지 않음

예제 코드

Motion/GetDHParam.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 DH 매개변수 설정

메서드 이름	Motion.SetDHParam(List<DHparam> dHparams)
설명	로봇의 DH(Denavit-Hartenberg) 매개변수를 설정합니다.
요청 매개변수	dHparams : 목록<DHparam> DH 매개변수 목록
반환 값	StatusCode: 작업 실행 결과 설정
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): 지원되지 않음

예제 코드

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 축 잠금 상태 가져오기

메서드 이름	Motion.GetDragSet()
설명	동작 티칭에만 적용되는 현재 로봇 축 잠금 상태를 가져옵니다.
요청 매개변수	없음
반환 값	DragStatus: 축 잠금 상태, True는 축이 이동 가능함을 나타내고, False는 잠겨 있음을 나타냅니다. StatusCode: 기능 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): 지원되지 않음

4.2.12 축 잠금 상태 설정

메서드 이름	**Motion.SetDragSet(DragStatus** `dragStatus`)
설명	티칭 동작에만 적용되는 현재 로봇 축 잠금 상태를 설정합니다.
요청 매개변수	`dragStatus` : DragStatus 축 잠금 상태, 기본값은 모두 True: 잠금 해제 상태
반환 값	StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): 지원되지 않음

4.2.13 드래그 티칭 활성화

메서드 이름	**Motion.EnableDrag(bool** `dragState`)
설명	로봇에 대한 드래그 티칭을 활성화하거나 비활성화합니다.
요청 매개변수	`dragState` : bool 로봇의 드래그 상태, true는 드래그 모드로 들어가는 것을 나타내고, false는 드래그 모드를 종료하는 것을 나타냅니다.
반환 값	StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): 지원되지 않음

예제 코드

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 실시간 Position 제어 모드 진입

메서드 이름	Motion.EnterPositionControl()
설명	실시간 위치 제어 모드로 진입하여 로봇의 정밀한 위치 제어가 가능합니다.
요청 매개변수	없음
반환 값	StatusCode: 함수 실행 결과
메모	실시간 제어 모드로 진입한 후에는 UDP를 통해 제어 명령을 전송해야 합니다.
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.2.0+ 산업용(Bronze): 지원되지 않음

4.2.15 실시간 Position 제어 모드 종료

메서드 이름	Motion.ExitPositionControl()
설명	실시간 위치 제어 모드를 종료하고 기본 로봇 제어 상태로 복귀합니다.
요청 매개변수	없음
반환 값	StatusCode: 함수 실행 결과
메모	종료 후 로봇은 더 이상 실시간 제어 명령을 수락하지 않습니다.

메서드 이름	Motion.ExitPositionControl()
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.2.0+ 산업용(Bronze): 지원되지 않음

4.2.16 UDP 피드백 구성 가져오기

메서드 이름	Motion.GetUDPFeedbackConfig(int `id`)
설명	UDP 로봇 상태 구성 파일 rtf_info.json을 읽습니다.
요청 매개변수	`id` : int 구성 ID, 1부터 시작
반환 값	UDPFeedbackConfig: 지정된 ID의 UDP 로봇 상태 구성 StatusCode: 함수 실행 결과
메모	구성 제약: 1. 여러 구성이 존재하는 경우 단 하나의 구성만 sub_flag를 true로 가질 수 있으며 나머지는 false여야 합니다. 2. use_multicast가 true이면 client_ip는 239...* 형식의 멀티캐스트 주소 하나만 지원합니다. 3. use_multicast가 false이면 client_ip는 여러 개의 LAN 유니캐스트 주소를 지원합니다.
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.2.17 UDP 피드백 구성 설정

메서드 이름	Motion.SetUDPFeedbackConfig(UDPFeedbackConfig `config`)
설명	UDP 로봇 상태 구성 파일 rtf_info.json에 기록합니다.
요청 매개변수	`config` : UDPFeedbackConfig UDP 로봇 상태 구성, 구성 내 ID는 1부터 시작
반환 값	StatusCode: 함수 실행 결과

메서드 이름	Motion.SetUDPFeedbackConfig(UDPFeedbackConfig `config`)
메모	구성 제약: 1. 여러 구성이 존재하는 경우 단 하나의 구성만 sub_flag를 true로 가질 수 있으며 나머지는 false여야 합니다. 2. use_multicast가 true이면 client_ip는 239...* 형식의 멀티캐스트 주소 하나만 지원합니다. 3. use_multicast가 false이면 client_ip는 여러 개의 LAN 유니캐스트 주소를 지원합니다. 본 인터페이스는 구성 파일에 기록만 하며 즉시 적용되지는 않습니다.
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.2.18 UDP 피드백 활성화

메서드 이름	Motion.EnableUDPFeedback(int `id`)
설명	지정된 ID의 UDP 로봇 상태 구성을 활성화하고 rtf_info.json을 동기화하여 즉시 적용합니다.
요청 매개변수	`id` : int 구성 ID, 1부터 시작
반환 값	StatusCode: 활성화 작업 실행 결과
메모	실행 후 지정된 ID의 sub_flag만 true가 되고 나머지 구성의 sub_flag는 모두 false가 됩니다.
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.2.19 UDP 피드백 비활성화

메서드 이름	Motion.DisableUDPFeedback(int `id`)
설명	지정된 ID의 UDP 로봇 상태 구성을 비활성화하고 rtf_info.json을 동기화하여 즉시 적용합니다.

메서드 이름	**Motion.DisableUDPFeedback(int `id`)**
요청 매개변수	`id` : int 구성 ID, 1부터 시작
반환 값	StatusCode: 비활성화 작업 실행 결과
메모	실행 후 모든 구성의 sub_flag가 false가 됩니다.
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.2.20 구독 매개변수 설정

메서드 이름	**Motion.SetUDPFeedbackParams(bool `flag` , string `ip` , int `interval` , int `feedbackType` , List<int> `DOList` = null)**
설명	로봇이 지정된 IP 주소로 데이터를 푸시하도록 UDP 피드백 매개변수를 구성합니다.
요청 매개변수	`flag` : bool UDP 데이터 푸시 활성화 여부; `ip` : 문자열 수신자의 IP 주소; `interval` : int 데이터 전송 간격 (단위: 밀리초); `feedbackType` : int 피드백 데이터 형식(0: XML 형식); `DOList` : List<int> 획득할 DO 신호 목록(최대 10개, 선택 사항)
반환 값	StatusCode: 함수 실행 결과
메모	매개변수 설정은 UDP 데이터 푸시 기능이 활성화된 경우에만 유효합니다.
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.2.0+ 산업용(Bronze): 지원되지 않음

예제 코드

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;
    }
}
```

데이터 푸시 설명

이름	필드	설명
RIst: Cartesian Position	엑스	공구 좌표계의 X 방향 값, 단위는 밀리미터
	와이	공구 좌표계의 Y 방향 값, 단위는 밀리미터
	지	공구 좌표계의 Z 방향 값, 단위는 밀리미터

이름	필드	설명
	에이	공구 좌표계에서 X축을 중심으로 한 회전, 단위는 도입니다.
	비	공구 좌표계에서 Y축을 중심으로 한 회전, 단위는 도입니다.
	기음	공구 좌표계에서 Z축을 중심으로 한 회전, 단위는 도입니다.
AIPos: Joint Position	A1-A6	6개 관절의 값, 단위는 도
EIPos: Additional Axis Data	EIPos	추가 축 데이터
WristBtnState: Wrist Button State	버튼 상태	1 = 버튼이 눌림, 0 = 버튼이 해제됨
	드래그모델	드래그 버튼 상태
	레코드조인트	녹음 버튼 상태 가르치기
	일시중지이력서	일시정지/resume 버튼 상태
Digout: DO Output	디그아웃	디지털 출력 상태(DO)
ProgramStatus: Program Status	ProgId	프로그램 ID
	상태	인터프리터 실행 상태: 0 = INTERPRETER_IDLE 1 = INTERPRETER_EXECUTE 2 = INTERPRETER_PAUSED
	Xpath	프로그램 세그먼트 반환 값, 형식은 `program name: line number`
IPOC: Timestamp	IPOC	타임스탬프

4.2.21 로봇의 소프트 한계 가져오기

메서드 이름	Motion.GetUserSoftLimit()
설명	로봇의 현재 소프트 제한을 가져옵니다.
요청 매개변수	없음
반환 값	List<List<double>>: 로봇의 소프트 리미트, 리스트의 첫 번째 레이어는 각 축을 나타내고, 두 번째 레이어는 각 축의 하한값과 상한값을 나타냅니다. StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.2.22 UDP Position 제어 매개변수 설정

메서드 이름	Motion.SetPositionTrajectoryParams(int `maxTimeoutCount` , int `timeout` , int `filterLayer` , double `wristElbowThreshold` , double `shoulderThreshold`)
설명	UDP 위치 제어에 관련된 매개변수를 설정합니다.
요청 매개변수	`maxTimeoutCount` : int 최대 시간 초과 수, 범위 [1,100] `timeout` : int 시간 초과 기간(즉, 전송 간격, 기본값 20ms), 범위 [1,100] `filterLayer` : int 필터링 레벨, 범위 [1,100] `wristElbowThreshold` : double 손목/팔꿈치 특이점 접근 임계값, 범위 [10,100] `shoulderThreshold` : double 어깨 특이점 접근 임계값, 범위 [100,300]
반환 값	StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.2.23 궤적 정형기 매개변수 조회

메서드 이름	Motion.GetShapingParam()
설명	현재 궤적 정형기(trajectory shaper) 매개변수 값을 조회합니다.

메서드 이름	Motion.GetShapingParam()
요청 매개변수	없음
반환 값	int: 궤적 정형기 매개변수 값(범위 5~15, 0은 비활성화) StatusCode: 조회 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.7.1.0+ 산업용(Bronze): 지원되지 않음

4.2.24 궤적 정형기 매개변수 설정

메서드 이름	Motion.SetShapingParam(int `value`)
설명	궤적 정형기(trajectory shaper) 매개변수 값을 설정합니다.
요청 매개변수	`value` : int 궤적 정형기 매개변수(범위 5~15, 0은 비활성화)
반환 값	StatusCode: 설정 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.7.1.0+ 산업용(Bronze): 지원되지 않음

4.2.25 페이로드 관련 인터페이스

4.2.25.1 현재 활성 페이로드 가져오기

메서드 이름	Motion.Payload.GetCurrentPayload()
설명	현재 활성 페이로드 인덱스를 가져옵니다.
요청 매개변수	없음
반환 값	int: 현재 활성화된 페이로드의 인덱스 StatusCode: 함수 실행 결과

메서드 이름	Motion.Payload.GetCurrentPayload()
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.2.25.2 인덱스로 페이로드 얻기

메서드 이름	Motion.Payload.GetPayloadById(int `index`)
설명	인덱스별로 페이로드 정보를 가져옵니다.
요청 매개변수	`index` : 페이로드 인덱스
반환 값	StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.2.25.3 지정된 페이로드 활성화

메서드 이름	Motion.Payload.SetCurrentPayload(int `index`)
설명	인덱스별로 지정된 페이로드를 활성화합니다.
요청 매개변수	`index` : 페이로드 인덱스
반환 값	StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+
메모	페이로드 ID는 현재 장치에 존재해야 합니다.

4.2.25.4 모든 페이로드 정보 얻기

메서드 이름	Motion.Payload.GetAllPayloadInfo()
설명	모든 페이로드에 대한 자세한 정보를 가져옵니다.
요청 매개변수	없음

메서드 이름	Motion.Payload.GetAllPayloadInfo()
반환 값	Dictionary<단위, 문자열>: 페이로드 정보 사전을 반환합니다. StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.2.25.5 페이로드 추가

메서드 이름	Motion.Payload.AddPayload(PayloadInfo `payload`)
설명	새로운 페이로드를 추가합니다.
요청 매개변수	`payload` : PayloadInfo 페이로드 객체
반환 값	StatusCode: 함수 실행 결과
메모	새 페이로드 ID는 현재 장치에 없어야 하며 1에서 10 사이여야 합니다.
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.2.25.6 지정된 페이로드 삭제

메서드 이름	Motion.Payload.DeletePayload(int `index`)
설명	지정된 인덱스가 있는 페이로드를 삭제합니다.
요청 매개변수	`index` : int 페이로드 인덱스
반환 값	StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+
메모	참고: 현재 활성 페이로드는 삭제할 수 없습니다. 활성 페이로드를 삭제하려면 먼저 다른 페이로드를 활성화한 후 현재 페이로드를 삭제하세요.

4.2.25.7 지정된 페이로드 업데이트

메서드 이름	**Motion.Payload.UpdatePayload(PayloadInfo `payload`)**
설명	지정된 페이로드의 정보를 업데이트합니다.
요청 매개변수	`payload` : PayloadInfo 페이로드 객체
반환 값	StatusCode: 함수 실행 결과
메모	페이로드 ID는 현재 장치에 존재해야 합니다.
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

Motion/PayloadControl.cs

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 축 3이 수평인지 확인하기

메서드 이름	Motion.Payload.CheckAxisThreeHorizontal()
설명	축 3이 수평인지 확인합니다.
요청 매개변수	없음
반환 값	double : 3축의 수평각 StatusCode : 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.2.0+ 산업용(Bronze): 지원되지 않음
메모	페이로드 식별을 수행하려면 수평 각도가 -1과 1 사이여야 합니다.

4.2.25.9 페이로드 식별 상태 가져오기

메서드 이름	Motion.Payload.GetPayloadIdentifyState()
설명	페이로드 식별 상태를 가져옵니다.
요청 매개변수	없음
반환 값	PayloadIdentifyState: 페이로드 식별 상태 StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.2.0+ 산업용(Bronze): 지원되지 않음

4.2.25.10 페이로드 식별 시작

메서드 이름	Motion.Payload.StartPayloadIdentify(double `weight` , double `angle`)
설명	페이로드 식별을 시작합니다.

메서드 이름	**Motion.Payload.StartPayloadIdentify(double** `weight` **, double** `angle` **)**
요청 매개변수	`weight` : double 페이로드 중량(무게를 알 수 없는 경우 -1 사용) `angle` : double 축 6의 허용되는 회전 각도(30-90도)
반환 값	StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.2.0+ 산업용(Bronze): 지원되지 않음
메모	페이로드 식별을 시작하기 전에 페이로드 식별 상태로 진입해야 합니다.

4.2.25.11 페이로드 식별 결과 가져오기

메서드 이름	**Motion.Payload.PayloadIdentifyResult()**
설명	페이로드 식별 결과를 가져옵니다.
요청 매개변수	없음
반환 값	PayloadInfo: 페이로드 식별 결과 StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.2.0+ 산업용(Bronze): 지원되지 않음

4.2.25.12 페이로드 식별을 위한 간섭 확인 시작

메서드 이름	**Motion.Payload.InterferenceCheckForPayloadIdentify(double** `weight` **, double** `angle` **)**
설명	잠재적인 충돌을 확인하기 위해 페이로드 식별에 대한 간섭 확인을 시작합니다.
요청 매개변수	`weight` : double 페이로드 중량(무게를 알 수 없는 경우 -1 사용) `angle` : double 축 6의 허용되는 회전 각도(30-90도)
반환 값	StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.2.0+ 산업용(Bronze): 지원되지 않음

4.2.25.13 페이로드 식별 상태 입력

메서드 이름	Motion.Payload.PayloadIdentifyStart()
설명	페이로드 식별 상태로 들어갑니다.
요청 매개변수	없음
반환 값	StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.2.0+ 산업용(Bronze): 지원되지 않음

4.2.25.14 페이로드 식별 상태 종료

메서드 이름	Motion.Payload.PayloadIdentifyDone()
설명	페이로드 식별 상태를 종료합니다.
요청 매개변수	없음
반환 값	StatusCode: 함수 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.2.0+ 산업용(Bronze): 지원되지 않음

4.2.25.15 전체 페이로드 식별 프로세스

메서드 이름	Motion.Payload.PayloadIdentify(double `weight` = -1, double `angle` = 90)
설명	위에 언급된 모든 인터페이스를 포함하여 페이로드 식별 프로세스를 완료합니다. 일반적인 페이로드 식별에는 이 인터페이스로 충분합니다.
요청 매개변수	`weight` : double 페이로드 중량(무게를 알 수 없는 경우 -1 사용) `angle` : double 축 6의 허용되는 회전 각도(30-90도)
반환 값	PayloadInfo: 페이로드 식별 결과 StatusCode: 함수 실행 결과
메모	반환된 페이로드는 로봇에 추가되거나 로봇의 기존 페이로드에 저장될 수 있습니다. 전체 프로세스 단계는 다음과 같습니다. 1. 페이로드 식별 상태 입력

메서드 이름	Motion.Payload.PayloadIdentify(double `weight` = -1, double `angle` = 90)
	2. 페이로드 식별 시작 3. 페이로드 식별 결과 가져오기 4. 페이로드 식별 상태 종료
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.2.0+ 산업용(Bronze): 지원되지 않음

4.2.25.16 페이로드 식별 종료

메서드 이름	Motion.Payload.TerminatePayloadIdentify()
설명	페이로드 식별 상태를 강제 종료합니다.
요청 매개변수	없음
반환 값	StatusCode: 상태 전환 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.7.1.0+ 산업용(Bronze): 지원되지 않음

예제 코드

Motion/PayloadIdentify.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 로봇 프로그램 실행 수업

개요

`Execution` 클래스는 로봇 프로그램 및 모션 작업을 위한 통합 예약 인터페이스를 제공합니다. 다음을 담당합니다.

- /stopping/pausing/resuming 교육 프로그램 시작
- 동시 실행 작업 목록 관리
- 사용자 정의 BAS 스크립트 흐름 실행

`Arm` 및 `Motion` 와 결합된 `Execution` 는 호스트 측 트리거링과 컨트롤러 측 프로그램 흐름 제어를 처리합니다.

4.3.1 지정된 프로그램 실행

메서드 이름	**Execution.Start(string `programName`)**
설명	지정된 프로그램을 실행합니다.
요청 매개변수	`programName` : 문자열 실행할 프로그램 이름
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.3.2 현재 실행 중인 프로그램 중지

메서드 이름	**Execution.Stop(string `programName` = null)**
설명	현재 실행 중인 프로그램을 중지하거나 로봇의 현재 동작 명령을 중지합니다.

메서드 이름	**Execution.Stop(string `programName` = null)**
요청 매개변수	`programName` : 문자열 중지할 프로그램 이름, 기본값은 null입니다. 이는 현재 실행 중인 프로그램 또는 모션 명령을 중지함을 의미합니다.
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.3.3 실행 중인 모든 프로그램의 세부 정보 반환

메서드 이름	**Execution.AllRunningPrograms()**
설명	프로그램 ID 및 프로그램 이름을 포함하여 실행 중인 모든 프로그램에 대한 자세한 정보를 반환합니다.
요청 매개변수	없음
반환 값	Dictionary<string, int>: 프로그램 ID에서 프로그램 이름으로 매핑 StatusCode: 연산 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.3.4 프로그램 실행 일시 중지

메서드 이름	**Execution.Pause(string `programName` = null)**
설명	현재 실행 중인 프로그램을 일시 정지하거나 로봇의 현재 동작을 일시 정지합니다.
요청 매개변수	`programName` : 문자열 일시 중지할 프로그램의 이름입니다. 통과하지 못한 경우 기본값은 현재 실행 중인 프로그램 또는 실행 중인 작업을 제어하는 것입니다.
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.3.5 프로그램 실행 재개

메서드 이름	**Execution.Resume(string `programName` = null)**
설명	일시정지된 상태에서 프로그램을 계속 실행합니다.
요청 매개변수	`programName` : 문자열 재개할 프로그램 이름, 기본값은 null입니다. 즉, 현재 일시 중지된 프로그램 또는 모션 명령을 재개한다는 의미입니다.
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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 BAS 스크립트 프로그램 실행

메서드 이름	Execution.ExecuteBasScript(BasScript `script`)
설명	사용자 정의 BAS 스크립트 프로그램을 실행합니다.
요청 매개변수	`script` : BasScript 사용자 정의 BAS 스크립트 프로그램
반환 값	StatusCode: 작업 실행 결과
메모	BAS 스크립트 프로그램 일시 중지, 재개 및 중지 방법은 일반 프로그램과 동일합니다.
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.2.0+ 산업용(Bronze): 지원되지 않음 산업용 로봇: v7.6.0.0+

예제 코드

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 프로그램 포즈 읽기/Write

개요

`ProgramPoses` 클래스는 로봇 티칭 프로그램에서 포즈 포인트를 읽고, 쓰고, 변환하는 데 사용됩니다. 이 수업을 통해 다음을 수행할 수 있습니다.

- 지정된 프로그램 및 포즈 인덱스를 찾습니다.
- 추가, 삭제, 수정 및 쿼리 작업 수행
- 데카르트 표현과 결합 표현 간 변환

호스트 측 시나리오에서 프로그램 포인트의 일괄 유지 관리 또는 오프라인 편집에 편리합니다.

4.4.1 프로그램에서 지정된 포즈의 값 가져오기

메서드 이름	**ProgramPoses.Read(string `programName` , int `index` , FileType `ft` = FileType.UserProgram)**
설명	지정된 프로그램의 지정된 인덱스에서 포즈 값을 가져옵니다.
요청 매개변수	`programName` : 문자열 프로그램 이름 `index` : int 포즈 인덱스 `ft` : FileType 파일 형식 (기본값: FileType.UserProgram)
반환 값	ProgramPose: 포즈 정보 StatusCode: 연산 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

ProgramPoses/ReadProgramPose.cs

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 프로그램에서 지정된 포즈의 값 수정

메서드 이름	**ProgramPoses.Write(string** `programName` **, int** `index` **, ProgramPose** `value` **, FileType** `ft` **= FileType.UserProgram)**
설명	지정된 프로그램의 지정된 인덱스에서 포즈 값을 수정합니다.
요청 매개변수	`programName` : 문자열 프로그램 이름 `index` : int 포즈 인덱스 `value` : ProgramPose 업데이트할 포즈 값 `ft` : FileType 파일 형식 (기본값: FileType.UserProgram)
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

ProgramPoses/WriteProgramPose.cs

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.GetDe
scription()}"
                );
            }
        }
        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;
    }
}
```

4.4.3 지정된 프로그램에 포즈 추가

메서드 이름	**ProgramPoses.Add(string `programName` , int `index` , ProgramPose `value` , FileType `ft` = FileType.UserProgram)**
설명	지정된 프로그램의 지정된 인덱스 위치에 새 포즈를 추가합니다.
요청 매개변수	`programName` : string 프로그램 이름 `index` : int 포즈 인덱스 `value` : ProgramPose 추가할 포즈 값 `ft` : FileType 파일 형식 (기본값: FileType.UserProgram)
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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 프로그램에서 지정된 포즈 삭제하기

메서드 이름	**ProgramPoses.Delete(string** `programName` **, int** `index` **, FileType** `ft` **= FileType.UserProgram)**
설명	지정된 프로그램의 지정된 인덱스에 있는 포즈를 삭제합니다.

메서드 이름	**ProgramPoses.Delete(string** `programName` **, int** `index` **, FileType** `ft` **= FileType.UserProgram)**
요청 매개변수	`programName` : 문자열 프로그램 이름 `index` : int 포즈 인덱스 `ft` : FileType 파일 형식 (기본값: FileType.UserProgram)
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

ProgramPoses/DeleteProgramPose.cs

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 지정된 프로그램에서 모든 포즈 가져오기

메서드 이름	**ProgramPoses.ReadAllPoses(string `programName` , FileType `ft` = FileType.UserProgram)**
설명	지정된 프로그램에서 모든 포즈 정보를 가져옵니다.
요청 매개변수	`programName` : 문자열 프로그램 이름 `ft` : FileType 파일 형식(기본값: FileType.UserProgram)
반환 값	List<ProgramPose>: 포즈 정보 목록 StatusCode: 연산 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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 로봇 프로그램에서 포즈 유형 변환

메서드 이름	**ProgramPoses.ConvertPose(ProgramPose `pose` , PoseType `toType`)**
설명	관절 좌표와 데카르트 공간 좌표 사이에서 로봇 프로그램 포즈를 변환합니다.
요청 매개변수	`pose` : ProgramPose 변환할 포즈 값 `toType` : PoseType 변환 후 대상 유형
반환 값	ProgramPose: 변환된 포즈 정보 StatusCode: 연산 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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 신호

개요

`Signals` 클래스는 다음을 포함하여 컨트롤러 I/O,에 대한 통합 read/write 인터페이스를 제공합니다.

- Digital/analog 입력 및 출력 작업
- 일괄 I/O 작업

`Signals` 를 사용하면 다음을 수행할 수 있습니다.

- 현재 신호 상태 읽기
- 일괄 쓰기 DO/RO/GO 포트
- 그리퍼, 센서 및 생산 라인 장치 통합

4.5.1 지정된 유형 및 포트 IO의 값 읽기

메서드 이름	**Signals.Read(SignalType `type` , int `index`)**
설명	지정된 유형과 포트의 IO 값을 읽습니다. (DI/DO/UI/UO/RI/RO/GI/GO/TAI/TDI/TDO/AI/AO 지원)
요청 매개변수	`type` : 읽을 SignalType IO 유형 `index` : int IO 포트 인덱스, 1부터 시작
반환 값	int: IO 값, 1은 상위 레벨, 0은 하위 레벨 StatusCode: 연산 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+
메모	UI/UO는 읽기만 가능하고 쓰기는 불가능합니다.

예제 코드

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 지정된 유형 및 포트 IO의 값 쓰기

메서드 이름	**Signals.Write(SignalType** `type` **, int** `index` **, double** `value` **)**
설명	지정된 유형 및 포트의 IO 값을 씁니다. 현재 DO/RO/GO/TDO/AO.만 지원합니다.
요청 매개변수	`type` : 쓸 SignalType IO 유형 `index` : 1부터 시작하는 int IO 포트 인덱스 `value` : 이중 IO 값(DO/RO/TDO는 0 또는 1만 허용, GO는 정수, AO는 부동 소수점 아날로그)
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+
메모	UI/UO는 읽기만 가능하고 쓰기는 불가능합니다.

예제 코드

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 일괄 쓰기 DO(디지털 출력) 신호

메서드 이름	Signals.MultiWrite(SignalType `type` , List<int> `ioData`)
설명	DO(디지털 출력) 신호를 일괄 쓰기합니다.
요청 매개변수	`type` : 쓸 SignalType IO 유형(DO 전용) `ioData` : List<int> 포트 번호 및 포트 값 목록(예: [port1, state1, port2, state2], 길이는 짝수이고 0보다 커야 함)
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+
메모	DO만 일괄 쓰기를 지원합니다. UI/UO는 쓰기를 지원하지 않으며, DI/RI는 단일 포인트 읽기만 지원합니다.

예제 코드

```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.");

        // Batch write 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 일괄 읽기 DO(디지털 출력) 포트 값

메서드 이름	**Signals.MultiRead(SignalType `type` , List<int> `indexes`)**
설명	DO(디지털 출력) 포트 값을 일괄적으로 읽습니다.
요청 매개변수	`type` : 읽을 SignalType IO 유형(DO 전용) `indexes` : List<int> 포트 번호 목록(비워둘 수 없음)
반환 값	List<int>: 포트 값 목록(입력과 일치하는 순서) StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+
메모	DO만 일괄 읽기를 지원합니다. UI/UO는 쓰기를 지원하지 않으며, DI/RI는 단일 포인트 읽기만 지원합니다.

예제 코드

```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.");

        // Batch read 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 등록정보

개요

`Registers` 클래스는 컨트롤러의 호스트 측 레지스터 액세스를 위한 통합 항목을 제공하여 여러 레지스터 유형에 대한 작업을 지원합니다.

핵심 기능

- /write 숫자 레지스터 읽기(R)
- /write 모션 레지스터 읽기(MR)
- /write 문자열 레지스터 읽기(SR)
- /write 포즈 레지스터 읽기(PR)
- /write Modbus 레지스터 읽기(MH 보유, MI 입력)

사용 사례

- 런타임 매개변수 전달
- 로봇 상태 동기화
- 외부 시스템과 구성 공유
- 호스트-장치 상호작용 제어

4.6.1 R 숫자 레지스터 작업

4.6.1.1 R 레지스터 값 읽기

메서드 이름	**Registers.Read_R(int `index`)**
설명	R 숫자 레지스터의 값을 읽습니다.
요청 매개변수	`index` : 읽을 int R 레지스터 번호

메서드 이름	Registers.Read_R(int index)
반환 값	double: 레지스터 값 StatusCode: 연산 실행 결과
호환 로봇 소프트웨어 버전	협동로봇: 7.6.0.1 산업용 로봇: 7.6.0.0

4.6.1.2 R 레지스터 값 읽기(메타데이터 포함)

메서드 이름	Registers.Read_R(int index , bool withMeta)
설명	선택적 메타데이터(name/comment)를 사용하여 R 숫자 레지스터의 값을 읽습니다.
요청 매개변수	index : int 읽을 레지스터 번호 withMeta : bool 메타데이터 반환 여부(true인 경우 레지스터 개체 반환)
반환 값	레지스터: 레지스터 객체(id/name/comment/value) StatusCode: 연산 실행 결과
호환 로봇 소프트웨어 버전	협동로봇: 7.7.0.0+ 산업용 로봇: 7.7.0.0+

4.6.1.3 R 레지스터의 값 쓰기

메서드 이름	Registers.Write_R(int index , double value)
설명	R 숫자 레지스터의 값을 씁니다.
요청 매개변수	index : int 쓸 레지스터 번호 value : double 쓸 레지스터 숫자값
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협동로봇: 7.6.0.1 산업용 로봇: 7.6.0.0

4.6.1.4 메타데이터로 R 레지스터 작성

메서드 이름	Registers.Write_R(Register `register`)
설명	이름과 설명을 포함하여 Register 개체를 사용하여 R 레지스터를 작성합니다.
요청 매개변수	`register` : 등록 객체(id/name/comment/value)
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협동로봇: 7.7.0.0+ 산업용 로봇: 7.7.0.0+

4.6.1.5 R 레지스터 삭제

메서드 이름	Registers.Delete_R(int `index`)
설명	지정된 R 숫자 레지스터를 삭제합니다.
요청 매개변수	`index` : 삭제할 int R 레지스터 번호
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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 모션 레지스터 작업

4.6.2.1 MR 레지스터 값 읽기

메서드 이름	Registers.Read_MR(int `index`)
설명	MR 모션 레지스터의 값을 읽습니다.
요청 매개변수	`index` : int MR 읽을 레지스터 번호

메서드 이름	Registers.Read_MR(int `index`)
반환 값	int: 레지스터 값 StatusCode: 연산 실행 결과
호환 로봇 소프트웨어 버전	협동로봇: 7.6.0.1 산업용 로봇: 7.6.0.0

4.6.2.2 MR 레지스터 값 읽기(메타데이터 포함)

메서드 이름	Registers.Read_MR(int `index` , bool `withMeta`)
설명	선택적 메타데이터(name/comment)와 함께 MR 모션 레지스터의 값을 읽습니다.
요청 매개변수	`index` : int 읽을 레지스터 번호 `withMeta` : bool 메타데이터 반환 여부(true인 경우 레지스터 개체 반환)
반환 값	MotionRegister: 레지스터 객체(id/name/comment/value) StatusCode: 연산 실행 결과
호환 로봇 소프트웨어 버전	협동로봇: 7.7.0.0+ 산업용 로봇: 7.7.0.0+

4.6.2.3 MR 레지스터의 값 쓰기

메서드 이름	Registers.Write_MR(int `index` , int `value`)
설명	MR 모션 레지스터의 값을 씁니다.
요청 매개변수	`index` : int 쓸 레지스터 번호 `value` : int 쓸 레지스터 숫자값
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협동로봇: 7.6.0.1 산업용 로봇: 7.6.0.0

4.6.2.4 메타데이터로 MR 레지스터 작성

메서드 이름	Registers.Write_MR(MotionRegister `register`)
설명	이름과 설명을 포함하여 MotionRegister 개체를 사용하여 MR 레지스터를 작성합니다.
요청 매개변수	`register` : MotionRegister 객체(id/name/comment/value)
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협동로봇: 7.7.0.0+ 산업용 로봇: 7.7.0.0+

4.6.2.5 MR 레지스터 삭제

메서드 이름	Registers.Delete_MR(int `index`)
설명	지정된 MR 모션 레지스터를 삭제합니다.
요청 매개변수	`index` : int MR 삭제할 레지스터 번호
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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 문자열 레지스터 작업

4.6.3.1 SR 레지스터 값 읽기

메서드 이름	Registers.Read_SR(int `index`)
설명	SR 문자열 레지스터의 값을 읽습니다.
요청 매개변수	`index` : int SR 읽을 레지스터 번호

메서드 이름	Registers.Read_SR(int `index`)
반환 값	string: 레지스터 문자열 값 StatusCode: 연산 실행 결과
호환 로봇 소프트웨어 버전	협동로봇: 7.6.0.1 산업용 로봇: 7.6.0.0

4.6.3.2 SR 레지스터 값 읽기(메타데이터 포함)

메서드 이름	Registers.Read_SR(int `index` , bool `withMeta`)
설명	선택적 메타데이터(name/comment)와 함께 SR 문자열 레지스터의 값을 읽습니다.
요청 매개변수	`index` : int 읽을 레지스터 번호 `withMeta` : bool 메타데이터 반환 여부(true인 경우 레지스터 개체 반환)
반환 값	StringRegister: 레지스터 객체(id/name/comment/value) StatusCode: 연산 실행 결과
호환 로봇 소프트웨어 버전	협동로봇: 7.7.0.0+ 산업용 로봇: 7.7.0.0+

4.6.3.3 SR 레지스터의 값 쓰기

메서드 이름	Registers.Write_SR(int `index` , string `value`)
설명	SR 문자열 레지스터의 값을 씁니다.
요청 매개변수	`index` : int 쓸 레지스터 번호 `value` : string 쓸 레지스터 스트링 값
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협동로봇: 7.6.0.1 산업용 로봇: 7.6.0.0

4.6.3.4 메타데이터로 SR 레지스터 작성

메서드 이름	**Registers.Write_SR(StringRegister** `register` **)**
설명	이름과 설명을 포함하여 StringRegister 개체를 사용하여 SR 레지스터를 작성합니다.
요청 매개변수	`register` : StringRegister 객체(id/name/comment/value)
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협동로봇: 7.7.0.0+ 산업용 로봇: 7.7.0.0+

4.6.3.5 SR 레지스터 삭제

메서드 이름	**Registers.Delete_SR(int** `index` **)**
설명	지정된 SR 문자열 레지스터를 삭제합니다.
요청 매개변수	`index` : int SR 삭제할 레지스터 번호
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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 포즈 레지스터 작업

4.6.4.1 PR 레지스터 값 읽기

메서드 이름	Registers.Read_PR(int `index`)
설명	PR 포즈 레지스터의 값을 읽습니다.
요청 매개변수	`index` : int PR 읽을 레지스터 번호

메서드 이름	Registers.Read_PR(int `index`)
반환 값	PoseRegister: 포즈 데이터 StatusCode: 연산 실행 결과
호환 로봇 소프트웨어 버전	협동로봇: 7.6.0.1 산업용 로봇: 7.6.0.0

4.6.4.2 PR 레지스터 값 읽기(메타데이터 포함)

메서드 이름	Registers.Read_PR(int `index` , bool `withMeta`)
설명	선택적 메타데이터(name/comment)와 함께 PR 포즈 레지스터의 값을 읽습니다.
요청 매개변수	`index` : int 읽을 레지스터 번호 `withMeta` : bool 메타데이터 반환 여부(true인 경우 name/comment 포함)
반환 값	PoseRegister: 포즈 데이터(withMeta=true인 경우 name/comment 포함) StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협동로봇: 7.7.0.0+ 산업용 로봇: 7.7.0.0+

4.6.4.3 PR 레지스터 값 쓰기

메서드 이름	Registers.Write_PR(PoseRegister `pose`)
설명	PR 포즈 레지스터의 값을 씁니다.
요청 매개변수	`pose` : PoseRegister 쓸 포즈 데이터
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협동로봇: 7.6.0.1 산업용 로봇: 7.6.0.0

4.6.4.4 메타데이터로 PR 레지스터 작성

메서드 이름	Registers.Write_PR(PoseRegister `pose` , bool `withMeta`)
설명	메타데이터(name/comment) 쓰기 여부를 제어하여 PR 포즈 레지스터의 값을 씁니다.
요청 매개변수	`pose` : PoseRegister 포즈 데이터를 작성 `withMeta` : bool 메타데이터 작성 여부(true인 경우 name/comment 작성)
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협동로봇: 7.7.0.0+ 산업용 로봇: 7.7.0.0+

4.6.4.5 PR 레지스터 삭제

메서드 이름	Registers.Delete_PR(int `index`)
설명	지정된 PR 포즈 레지스터를 삭제합니다.
요청 매개변수	`index` : int PR 삭제할 레지스터 번호
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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 레지스터(MH 홀딩 레지스터, MI 입력 레지스터)

4.6.5.1 MH 레지스터 값 읽기

메서드 이름	Registers.Read_MH(int `index`)
설명	MH 보유 레지스터의 값을 읽습니다.
요청 매개변수	`index` : int 읽을 레지스터 번호
반환 값	int: 레지스터 값 StatusCode: 연산 실행 결과
호환 로봇 소프트웨어 버전	협동로봇: 7.6.0.0 산업용 로봇: 7.6.0.0

4.6.5.2 MI 레지스터 값 읽기

메서드 이름	Registers.Read_MI(int `index`)
설명	MI 입력 레지스터의 값을 읽습니다.
요청 매개변수	`index` : int 읽을 레지스터 번호
반환 값	int: 레지스터 값 StatusCode: 연산 실행 결과
호환 로봇 소프트웨어 버전	협동로봇: 7.6.0.0 산업용 로봇: 7.6.0.0

4.6.5.3 MH 레지스터의 값 쓰기

메서드 이름	Registers.Write_MH(int `index` , int `value`)
설명	MH 보유 레지스터의 값을 씁니다.
요청 매개변수	`index` : int 쓸 레지스터 번호 `value` : int 쓸 레지스터 숫자값
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협동로봇: 7.6.0.0 산업용 로봇: 7.6.0.0

4.6.5.4 MI 레지스터의 값 쓰기

메서드 이름	Registers.Write_MI(int `index` , int `value`)
설명	MI 입력 레지스터의 값을 씁니다.
요청 매개변수	`index` : int 쓸 레지스터 번호 `value` : int 쓸 레지스터 숫자값
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협동로봇: 7.6.0.0 산업용 로봇: 7.6.0.0

예제 코드

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 궤도 제어

개요

`Trajectory` 모듈은 오프라인 궤도 실행 및 궤도 파일 변환을 위한 인터페이스를 제공하여 복잡한 궤도 재생을 지원합니다.

핵심 기능

- 오프라인 궤적 파일 설정 및 실행
- 안전한 속도로 로봇을 궤적 시작점으로 이동
- CSV 궤적 파일을 `.trajectory` 형식으로 변환
- 궤적 변환 상태 쿼리

사용 사례

- 복잡한 궤적을 정확하게 재현
- 외부 궤적 데이터를 실행 가능한 로봇 형식으로 변환

4.7.1 실행할 오프라인 궤적 파일 설정

메서드 이름	**Trajectory.SetOffLineTrajectoryFile(string `path`)**
설명	실행할 오프라인 궤적 파일을 설정합니다.
요청 매개변수	`path` : 문자열 오프라인 궤적 파일 프로그램 이름(형식 설명은 아래 설명 참조)
반환 값	StatusCode: 작업 실행 결과
비고	A.trajectory trajectory file format is a **text file**: - Line 1: 6 represents 6 axes, 0.001 represents 1ms interval between two points, 8093 represents a total of 8093 trajectory points

메서드 이름	**Trajectory.SetOffLineTrajectoryFile(string** `path` **)**
	- Line 2: Represents the initial positions of the 6 axes - Lines 3-8095: Represent trajectory points, including positions, velocities, accelerations, torque feedforward, do ports, and values of do ports for the 6 axes - do_port represents the used do port (range 1-24) - do_port is -1, indicating no IO signal will be triggered at this position - do_port is 1, do_state is 1, indicating do1 port will trigger ON signal at this position - do_port is 1, do_state is 0, indicating do1 port will trigger OFF signal at this position Users upload the offline file to the robot controller root directory using FileManager.upload, then execute the trajectory using instructions 4.7.2 and 4.7.3
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.7.2 안전한 속도로 로봇을 오프라인 궤적의 시작점으로 이동

메서드 이름	**Trajectory.PrepareOfflineTrajectory()**
설명	안전한 속도로 로봇을 오프라인 궤적의 시작점으로 이동시킵니다.
요청 매개변수	없음
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.7.3 오프라인 궤적 파일 실행 시작

메서드 이름	**Trajectory.ExecuteOfflineTrajectory()**
설명	오프라인 궤적 프로그램의 실행을 시작합니다.

메서드 이름	Trajectory.ExecuteOfflineTrajectory()
요청 매개변수	없음
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.7.4 궤적 파일 변환 기능

메서드 이름	Trajectory.TransformCsvToTrajectory(string `fileName`)
설명	궤적 CSV 파일을 궤적 형식으로 변환하여 컨트롤러의 궤적 파일 디렉터리에 저장합니다.
요청 매개변수	`fileName` : 문자열 CSV 궤적 파일 이름(.csv 접미사가 필요 없음)
오버로드 메서드	**Trajectory.TransformCsvToTrajectory**(string `fileName` , string `separator` , string `ioFlag`) `separator` : 문자열 구분자, 공백 또는 쉼표 지원 `ioFlag` : 문자열 IO 소스 식별자, "1"은 기본 IO 값 사용, "2"는 사용자 지정 IO 사용
반환 값	string : 변환된 궤적 파일 경로 StatusCode: 연산 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.7.5 궤적 파일 변환 상태 조회

메서드 이름	Trajectory.CheckTransformStatus(string `fileName`)
설명	궤적 파일 변환 현황을 조회합니다.
요청 매개변수	`fileName` : 문자열 궤적 파일 경로(TransformCsvToTrajectory 인터페이스에서 반환됨)

메서드 이름	Trajectory.CheckTransformStatus(string fileName)
반환 값	TransformState: 변환 상태 StatusCode: 연산 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

Trajectory/OfflineTrajectory.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 e
xecution 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] 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.GetDescriptio
n()}"
                );
                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 궤적 기록 시작

메서드 이름	Trajectory.TrajectoryRecordBegin(string `name`)
설명	궤적 재현 기능의 궤적 기록을 시작합니다.
요청 매개변수	`name` : 문자열 궤적 프로그램 이름
반환 값	StatusCode: 기록 시작 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.7.7 궤적 기록 종료

메서드 이름	Trajectory.TrajectoryRecordFinish(string `name`)
설명	궤적 재현 기능의 궤적 기록을 종료합니다.
요청 매개변수	`name` : 문자열 궤적 프로그램 이름
반환 값	StatusCode: 기록 종료 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.7.8 궤적 재생 시작

메서드 이름	Trajectory.TrajectoryReplayStart(string `name`)
설명	지정된 궤적 재생을 시작합니다.

메서드 이름	**Trajectory.TrajectoryReplayStart(string** `name` **)**
요청 매개변수	`name` : 문자열 궤적 프로그램 이름
반환 값	StatusCode: 재생 시작 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.7.9 궤적 재생 중지

메서드 이름	**Trajectory.TrajectoryReplayStop(string** `name` **)**
설명	지정된 궤적 재생을 중지합니다.
요청 매개변수	`name` : 문자열 궤적 프로그램 이름
반환 값	StatusCode: 재생 중지 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.7.10 궤적 삭제

메서드 이름	**Trajectory.TrajectoryRecordDelete(string** `name` **)**
설명	지정된 궤적을 삭제합니다.
요청 매개변수	`name` : 문자열 궤적 프로그램 이름
반환 값	StatusCode: 삭제 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.7.11 궤적 목록 조회

메서드 이름	Trajectory.GetTrajectoryRecordList()
설명	현재 기록된 궤적 이름 목록을 조회합니다.
요청 매개변수	없음
반환 값	List<string>: 궤적 이름 목록 StatusCode: 조회 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.7.12 궤적 시작 위치 조회

메서드 이름	Trajectory.GetTrajectoryRecordStartPose(string `name`)
설명	지정된 기록 궤적의 시작 위치를 조회합니다.
요청 매개변수	`name` : 문자열 궤적 프로그램 이름
반환 값	MotionPose: 궤적 시작 위치 StatusCode: 조회 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.7.13 궤적 테이블/경로 테이블 기록 시작

메서드 이름	Trajectory.PathRecordBegin(string `name` , string `comment` , double `param` , double `angle` = 1)
설명	궤적 테이블(`.traj`) 또는 경로 테이블(`.path`) 기록을 시작합니다.
요청 매개변수	`name` : 문자열 궤적 테이블/경로 테이블 파일명, 반드시 `.path` 또는 `.traj` 로 끝나야 함

메서드 이름	**Trajectory.PathRecordBegin(string `name` , string `comment` , double `param` , double `angle` = 1)**
	`comment` : 문자열 설명 정보 `param` : double 기록 파라미터(`.path` 는 거리 임계값, >=0.5 필요; `.traj` 는 기록 간격, >=2 필요) `angle` : double 경로 테이블 회전 각도 임계값(`.path` 에만 적용, >=1 필요)
반환 값	StatusCode: 기록 시작 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.7.1.0+ 산업용(Bronze): 지원 안 함

4.7.14 궤적 테이블/경로 테이블 기록 종료

메서드 이름	Trajectory.PathRecordFinish()
설명	현재 궤적 테이블/경로 테이블 기록을 종료합니다.
요청 매개변수	없음
반환 값	StatusCode: 기록 종료 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.7.1.0+ 산업용(Bronze): 지원 안 함

4.7.15 궤적 테이블/경로 테이블 시작 자세 조회

메서드 이름	Trajectory.GetPathStartPose(string `name`)
설명	지정된 궤적 테이블/경로 테이블의 시작 자세를 조회합니다.
요청 매개변수	`name` : 문자열 궤적 테이블/경로 테이블 이름
반환 값	MotionPose: 시작 자세 StatusCode: 조회 작업 실행 결과

메서드 이름	Trajectory.GetPathStartPose(string `name`)
호환 로봇 소프트웨어 버전	협업(Copper): v7.7.1.0+ 산업용(Bronze): 지원 안 함

4.7.16 궤적 테이블/경로 테이블 상태 조회

메서드 이름	Trajectory.GetPathState(List<string> `pathList`)
설명	궤적 테이블/경로 테이블의 현재 기록 상태를 일괄 조회합니다.
요청 매개변수	`pathList` : List<string> 궤적 테이블/경로 테이블 이름 목록
반환 값	Dictionary<string, MotionTableState>: 파일명과 상태 매핑 StatusCode: 조회 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.7.1.0+ 산업용(Bronze): 지원 안 함

4.7.17 경로 계획 파라미터 설정

메서드 이름	Trajectory.SetPathPlannerParameter(double `transitionTime` , double `scalingFactor`)
설명	경로 계획기 파라미터를 설정합니다. 궤적 전환 및 평활성에 영향을 줍니다.
요청 매개변수	`transitionTime` : double 시작/정지 전환 시간, >=0.2 필요 값이 작을수록 가·감속이 빨라지며 최소값은 0.2 `scalingFactor` : double 스케일링 계수, 범위 [0,1] =0이면 로봇은 일정한 시간 간격으로 각 경로점을 통과하며 점 간 시간 간격을 우선 보장 =1이면 엄격한 스케일링으로 로봇은 일정한 선속도로 각 경로점을 통과하며 각 점의 등속 통과를 우선 보장 0과 1 사이의 값은 절충된 동작
반환 값	StatusCode: 설정 작업 실행 결과

메서드 이름	**Trajectory.SetPathPlannerParameter(double `transitionTime` , double `scalingFactor`)**
호환 로봇 소프트웨어 버전	협업(Copper): v7.7.1.0+ 산업용(Bronze): 지원 안 함

4.7.18 경로 계획 파라미터 조회

메서드 이름	**Trajectory.GetPathPlannerParameter()**
설명	현재 경로 계획기 파라미터를 조회합니다.
요청 매개변수	없음
반환 값	double: `transitionTime` 시작/정지 전환 시간, 값이 작을수록 가·감속이 빨라지며 최소값은 0.2 double: `scalingFactor` 스케일링 계수, =0이면 일정한 시간으로 각 경로점 통과, =1이면 일정한 선속도로 통과, 0~1 사이는 절충 동작 StatusCode: 조회 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.8.0.0+ 산업용(Bronze): 지원 안 함

4.7.19 궤적 테이블/경로 테이블을 따라 이동

메서드 이름	**Trajectory.MovePath(string `name` , double `vel` = 100, double `acc` = 1)**
설명	로봇 말단이 지정된 궤적 테이블/경로 테이블을 따라 이동하도록 합니다.
요청 매개변수	`name` : 문자열 궤적 테이블/경로 테이블 이름 `vel` : double 이동 속도, 범위 [0,5000] (mm/s) `acc` : double 가속도 계수, 범위 [0,1.2]
반환 값	StatusCode: 이동 작업 실행 결과

메서드 이름	**Trajectory.MovePath(string `name` , double `vel` = 100, double `acc` = 1)**
호환 로봇 소프트웨어 버전	협업(Copper): v7.7.I.0+ 산업용(Bronze): 지원 안 함

4.8 알람 정보

개요

`Alarm` 모듈은 로봇 작동 중에 발생할 수 있는 비정상적인 상황을 모니터링하고 처리하는 데 사용되는 로봇 알람 정보에 대한 읽기, 재설정 및 쿼리 기능을 제공합니다.

핵심 기능

- 알람 재설정
- 모든 활성 알람 가져오기
- 우선순위가 가장 높은 알람 받기

사용 사례

- 로봇 작동 상태를 모니터링하고 적시에 이상 징후를 감지
- 로봇 작동 중 오류 및 알람 처리
- 로봇 알람 이력 기록 및 분석
- 자동화된 경보 처리 워크플로 구현
- 로봇 모니터링 시스템에 통합

4.8.1 최우선 순위 알람 받기

메서드 이름	Alarm.GetTopAlarm()
설명	현재 가장 높은 우선순위 경보를 가져옵니다.
요청 매개변수	없음
반환 값	string: 알람 정보 문자열 StatusCode: 연산 실행 결과

메서드 이름	Alarm.GetTopAlarm()
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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 모든 활성 경보 가져오기

메서드 이름	Alarm.GetAllActiveAlarms()
설명	현재 활성화된 모든 알람을 가져옵니다.
요청 매개변수	없음
반환 값	List<string>: 활성 알람 항목 목록 StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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 알람 재설정

메서드 이름	Alarm.ResetAlarms()
설명	현재 오류를 재설정합니다./alarms.
요청 매개변수	없음
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.9 파일 서비스 클래스

개요

`FileManager` 는 호스트와 로봇 컨트롤러 간의 프로그램 /trajectory/temporary 파일을 업로드, 다운로드, 삭제 및 검색하는 데 사용됩니다.

핵심 기능

- 컨트롤러에 로컬 파일 업로드
- 컨트롤러 파일을 로컬 경로로 다운로드
- 컨트롤러에서 파일 삭제
- 파일 이름 패턴으로 파일 검색
- 여러 파일 형식 관리(`UserProgram` , `BlockProgram` , `TrajectoryProgram` , `TmpFile`)
- 업로드 중 덮어쓰기 동작 제어/download

사용 사례

- 컨트롤러에 프로그램 배포
- 컨트롤러 측 프로그램 및 궤적 백업
- 생산 라인 디버깅 데이터 동기화
- 컨트롤러의 배치 파일 관리

4.9.1 로봇에 로컬 파일 업로드

메서드 이름	**FileManager.Upload(string `filePath` , FileType `ft` , bool `overWriting` = false)**
설명	로봇 컨트롤러에 로컬 파일을 업로드합니다.
요청 매개변수	`filePath` : string 업로드할 로컬 파일의 절대 경로 `ft` : FileType 업로드할 파일 유형

메서드 이름	**FileManager.Upload(string `filePath` , FileType `ft` , bool `overWriting` = false)**
	`overWriting` : bool 로봇 컨트롤러의 기존 파일을 덮어쓸지 여부, 기본값은 false(덮어쓰기 없음)
메모	USER_PROGRAM 및 BLOCK_PROGRAM의 경우 `.xml` / `.block` 파일의 전체 경로를 제공하십시오. 시스템은 또한 동일한 이름의 `.json` (및 BlockProgram의 경우 `.xml`)를 업로드합니다.
반환 값	StatusCode: 업로드 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.9.2 로컬 머신에 로봇 파일 다운로드

메서드 이름	**FileManager.Download(string `fileName` , FileType `ft` , string `savePath`)**
설명	로봇 컨트롤러에서 로컬로 파일을 다운로드합니다.
요청 매개 변수	`fileName` : string 다운로드할 파일 이름 `ft` : FileType 다운로드할 파일 유형 `savePath` : string 다운로드한 파일의 로컬 저장 경로
메모	`UserProgram` / `BlockProgram` / `TrajectoryProgram` ,의 경우 프로그램 이름(확장자가 없는 파일 이름)만 지정하십시오. `TmpFile` 의 경우 확장명을 포함한 전체 파일 이름을 제공하십시오.
반환 값	StatusCode: 작업 실행 결과 다운로드
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.9.3 로봇에서 파일 삭제

메서드 이름	FileManager.Delete(string `fileName` , FileType `ft`)
설명	로봇 컨트롤러에서 파일을 삭제합니다.
요청 매개변수	`fileName` : 문자열 삭제할 파일 이름 `ft` : FileType 삭제할 파일 유형
메모	`UserProgram` / `BlockProgram` / `TrajectoryProgram` ,의 경우 프로그램 이름(확장자가 없는 파일 이름)만 지정하십시오. `TmpFile` 의 경우 확장명을 포함한 전체 파일 이름을 제공하십시오.
반환 값	StatusCode: 작업 실행 결과 삭제
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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 파일 이름 패턴으로 파일 검색

메서드 이름	FileManager.Search(string `pattern` , ref List<string> `fl`)
설명	파일 이름 패턴과 일치하는 로봇 컨트롤러의 파일을 검색합니다.
요청 매개변수	`pattern` : 문자열 파일 이름 일치 패턴 `fl` : ref List<string> 반환된 파일 목록
반환 값	StatusCode: 검색 연산 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.10 BasScript 스크립트 프로그램 클래스

개요

`BasScript` 는 모션, 로직, 프로그램 제어, 통신, 비전 및 Modbus 명령어를 포함하여 구조화된 방식으로 호스트 측에서 BAS 명령어 시퀀스를 구성하는 데 사용됩니다.

핵심 기능

- 모션 명령어 빌드(`MoveJoint` , `MoveLine` , `MoveCircle` 등)
- 로직 명령어 빌드(`IF` , `WHILE` , `SWITCH` , `GOTO` 등)
- 할당 작성, 대기, 일시 중지 및 중단 지침
- 프로그램 호출 및 관리 지침 작성
- 소켓 통신 지침 작성
- Modbus 레지스터 read/write 명령어 빌드
- 비전 지침 빌드
- 추가 매개변수 구성(가속, RTCP, 프레임 오프셋 등)
- 높은 수준의 점프 지침 지원(`JUMP` 시리즈)

사용 사례

- 복잡한 로봇 프로그램을 자동으로 생성
- 로봇 프로그램 편집 및 수정
- 공통 프로그램 모듈 및 명령어 시퀀스 재사용
- 호스트와 로봇 프로그램 간의 원활한 통합 구현
- 맞춤형 로봇 모션 궤적 구축
- 비전, 통신, Modbus 기능을 로봇 프로그램에 통합

클래스 생성자

메서드 이름	BasScript(string `name`)
설명	로봇에서 사용할 BAS 명령 시퀀스 클래스를 구성합니다.
요청 매개변수	`name` : 문자열 스크립트 프로그램 이름
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.2.0+ 산업용(Bronze): 지원되지 않음
메모	이 클래스의 모든 메서드에는 이 클래스와 동일한 호환 버전 요구사항이 있습니다.

서브클래스 구조

`BasScript` 클래스에는 다양한 명령 범주를 구성하기 위한 다음 하위 클래스가 포함되어 있습니다.

1. **ExtraParam**: 복잡한 로봇 제어 명령을 구축하기 위한 추가 매개변수 클래스
2. **BasMotion**: 모션 명령 하위 클래스, 모든 로봇 모션 관련 메서드 포함
3. **BasLogical**: 논리 명령 하위 클래스, 모든 논리 제어 관련 메서드 포함
4. **BasStructure**: 구조 명령 하위 클래스, 모든 구조 제어 관련 메서드 포함
5. **BasSocket**: 소켓 통신 하위 클래스, 모든 소켓 통신 관련 메서드 포함
6. **BasModbus**: Modbus 통신 하위 클래스, 모든 Modbus 통신 관련 방법 포함
7. **BasVision**: 비전 지침 하위 클래스, 모든 비전 관련 메서드 포함

4.10.1 ExtraParam 클래스

메서드 이름	ExtraParam()
설명	복잡한 로봇 제어 명령을 구축하기 위한 추가 매개변수 클래스
요청 매개변수	없음
반환 값	ExtraParam 객체

4.10.1.1 가속도 설정

메서드 이름	ExtraParam.Acceleration(double `value`)
설명	가속도를 설정하며, 범위는 1~120%입니다.
요청 매개변수	`value` : double 가속도 값, 단위: %(부동소수점)
반환 값	StatusCode: 파라미터 설정 실행 결과

4.10.1.2 RTCP 매개변수 설정

메서드 이름	ExtraParam.RTCP()
설명	RTCP 매개변수 설정, MoveL 및 MoveC 명령어만 지원
요청 매개변수	없음
반환 값	StatusCode: 파라미터 설정 실행 결과

4.10.1.3 프레임 오프셋 설정

메서드 이름	ExtraParam.Offset(int `index`)
설명	프레임 오프셋을 설정합니다.
요청 매개변수	`index` : int 오프셋 인덱스(정수)
반환 값	StatusCode: 파라미터 설정 실행 결과

4.10.1.4 시간 기반 실행 설정/Assign

메서드 이름	**ExtraParam.TB(double** `second` **, string** `type` **, string** `name` **)**
설명	시간 기반 실행 또는 할당을 설정합니다. 범위는 0.01-30초이며, 0.01 미만의 값은 성공적으로 저장되지 않습니다.
요청 매개변수	`second` : double 초, 단위: sec(부동소수점) `type` : 문자열 실행 유형(문자열) `name` : 문자열 이름(실행용, 문자열)
반환 값	StatusCode: 파라미터 설정 실행 결과

메서드 이름	**ExtraParam.TB(double** `second` **, string** `type` **, int** `index` **, int** `status` **)**
설명	시간 기반 실행 또는 할당을 설정합니다. 범위는 0.01-30초이며, 0.01 미만의 값은 성공적으로 저장되지 않습니다.
요청 매개변수	`second` : double 초, 단위: sec(부동 소수점) `type` : 문자열 IO 유형(문자열) `index` : int 인덱스(할당용, 정수) `status` : int 상태(할당용, 정수)
반환 값	StatusCode: 파라미터 설정 실행 결과

4.10.1.5 건너뛰기 설정

메서드 이름	**ExtraParam.SKIP(int** `index` **)**
설명	점프를 설정합니다. SKIP CONDITION 명령에 의해 설정된 점프 조건이 충족되면 SKIP 명령이 포함된 현재 라인에서 대상 명령 라인 또는 대상 프로그램으로 직접 점프합니다.
요청 매개변수	`index` : int 대상 라벨의 인덱스(정수)
반환 값	StatusCode: 파라미터 설정 실행 결과

4.10.1.6 출발 및 접근 거리 설정

메서드 이름	**ExtraParam.Approach(double** departureDist **, double** approachingDist **)**
설명	Gate형 모션의 출발 및 접근 거리를 설정하며 JUMP 명령에만 사용됩니다.
요청 매개변수	departureDist : double 출발 거리, 단위: mm(부동 소수점) approachingDist : double 접근 거리, 단위: mm(부동 소수점)
반환 값	StatusCode: 파라미터 설정 실행 결과

4.10.2 BasMotion 모션 명령어 하위 클래스

BasMotion 하위 클래스에는 모션 명령 시퀀스를 구축하는 데 사용되는 모든 로봇 모션 관련 메서드가 포함되어 있습니다.

4.10.2.1 MoveJoint 모션-포인트 명령

메서드 이름	**BasScript.BasMotion.MoveJoint(MovePoseType** poseType **, int** poseIndex **, SpeedType** speedType **, double** speedValue **, SmoothType** smoothType **, double** smoothDistance **= 0, ExtraParam** extraParam **= null)**
설명	Joint 모션 명령은 지정된 이동 속도와 방법으로 작업 공간의 지정된 위치로 로봇을 이동하며 로봇 프로그램 작성의 MoveJoint 명령에 해당합니다.
요청 매개변수	poseType : MovePoseType 포즈 유형 poseIndex : int 포즈 인덱스 speedType : SpeedType 속도 유형 speedValue : double 속도 값, 단위: % smoothType : SmoothType 부드러운 유형 smoothDistance : double Smooth 거리, 단위: mm, 값 범위: 0~1000 extraParam : ExtraParam 추가 매개변수
반환 값	StatusCode: 모션 명령 실행 결과

4.10.2.2 MoveLine 선형 모션-포인트 명령어

메서드 이름	**BasScript.BasMotion.MoveLine(MovePoseType** `poseType` **, int** `poseIndex` **, SpeedType** `speedType` **, double** `speedValue` **, SmoothType** `smoothType` **, double** `smoothDistance` **= 0, ExtraParam** `extraParam` **= null)**
설명	선형 동작 명령은 지정된 이동 속도와 방법으로 작업 공간의 지정된 위치로 로봇을 선형으로 이동시키는 것으로, 로봇 프로그램 작성의 MoveLine 명령에 해당합니다.
요청 매개변수	`poseType` : MovePoseType 포즈 유형 `poseIndex` : int 포즈 인덱스 `speedType` : SpeedType 속도 유형 `speedValue` : double 속도 값, 단위: mm/s `smoothType` : SmoothType Smooth 유형 `smoothDistance` : double Smooth 거리, 단위: mm, 값 범위: 0~1000 `extraParam` : ExtraParam 추가 매개변수
반환 값	StatusCode: 모션 명령 실행 결과

4.10.2.3 MoveCircle 호 모션 투 포인트 명령어

메서드 이름	**BasScript.BasMotion.MoveCircle(MovePoseType** `poseType1` **, int** `poseIndex1` **, MovePoseType** `poseType2` **, int** `poseIndex2` **, SpeedType** `speedType` **, double** `speedValue` **, SmoothType** `smoothType` **, double** `smoothDistance` **= 0, ExtraParam** `extraParam` **= null)**
설명	원호 모션 명령은 로봇을 지정된 이동 속도와 방법으로 작업 공간의 지정된 위치로 호 형태로 이동시키는 로봇 프로그램 작성의 MoveCircle 명령에 해당합니다.
요청 매개변수	`poseType1` : MovePoseType 첫 번째 포즈 유형 `poseIndex1` : int 첫 번째 포즈 인덱스 `poseType2` : MovePoseType 두 번째 포즈 유형 `poseIndex2` : int 두 번째 포즈 인덱스 `speedType` : SpeedType 속도 유형 `speedValue` : double 속도 값, 단위: mm/s `smoothType` : SmoothType Smooth 유형

메서드 이름	**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 Smooth 거리, 단위: mm, 값 범위: 0~1000 `extraParam` : ExtraParam 추가 매개변수
반환 값	StatusCode: 모션 명령 실행 결과

4.10.2.4 점대점 동작 명령어 점프

메서드 이름	**BasScript.BasMotion.Jump(MovePoseType** `poseType` **, int** `poseIndex` **, double** `speedValue` **, double** `speedRatio` **, SpeedType** `limZType` **, double** `limZValue` **, SmoothType** `smoothType` **, double** `smoothDistance` **= 0, ExtraParam** `extraParam` **= null)**
설명	게이트형 모션 명령어는 게이트형 모션을 수행하도록 로봇을 지정합니다(먼저 수직으로 위로 이동한 다음 수평으로 이동하고 마지막으로 수직으로 아래로 이동). 로봇 프로그램 작성의 JUMP 명령어에 해당합니다.
요청 매개변수	`poseType` : MovePoseType 대상 포즈 저장 유형 `poseIndex` : int 대상 위치 인덱스 `speedValue` : double 동작 속도 값, 단위: mm/s `speedRatio` : double 동작 속도 비율, 단위: % `limZType` : SpeedType Z축 제한 유형 `limZValue` : double Z축 제한 값 `smoothType` : SmoothType Smooth 유형 `smoothDistance` : double Smooth 거리, 단위: mm, 값 범위: 0~1000 `extraParam` : ExtraParam 추가 매개변수
반환 값	StatusCode: 모션 명령 실행 결과

4.10.2.5 Jump3 3점 점프 명령어

메서드 이름	**BasScript.BasMotion.Jump3(MovePoseType** `poseType` **, int** `poseIndex` **, double** `speedValue` **, double** `speedRatio` **, SmoothType** `smoothType` **, double** `smoothDistance` **= 0, ExtraParam** `extraParam` **= null)**
설명	게이트형 동작 명령은 로봇이 출발, 접근, 목표 위치 등 게이트형 동작을 수행하도록 지정하며, 로봇 프로그램 작성 시 JUMP3 명령에 해당합니다.
요청 매개변수	`poseType` : MovePoseType 대상 포즈 저장 유형 `poseIndex` : int 3 대상 위치 인덱스 `speedValue` : double 동작 속도 값, 단위: mm/s `speedRatio` : double 동작 속도 비율, 단위: % `smoothType` : SmoothType 부드러운 유형 `smoothDistance` : double Smooth 거리, 단위: mm, 값 범위: 0~1000 `extraParam` : ExtraParam 추가 매개변수
반환 값	StatusCode: 모션 명령 실행 결과

4.10.2.6 Jump3CP 3점 점프 CP 명령어

메서드 이름	**BasScript.BasMotion.Jump3CP(MovePoseType** `poseType` **, int** `poseIndex` **, double** `speedValue` **, SmoothType** `smoothType` **, double** `smoothDistance` **= 0, ExtraParam** `extraParam` **= null)**
설명	게이트형 모션 명령은 로봇이 출발, 접근 및 목표 위치를 포함하여 게이트형 모션을 수행하도록 지정하며 로봇 프로그램 작성 시 JUMP3CP 명령에 해당합니다.
요청 매개변수	`poseType` : MovePoseType 대상 포즈 저장 유형 `poseIndex` : int 3 대상 위치 인덱스 `speedValue` : double 동작 속도 값, 단위: mm/s `smoothType` : SmoothType Smooth 유형 `smoothDistance` : double Smooth distance, 단위: mm, 값 범위: 0~1000 `extraParam` : ExtraParam 추가 매개변수

메서드 이름	**BasScript.BasMotion.Jump3CP(MovePoseType `poseType` , int `poseIndex` , double `speedValue` , SmoothType `smoothType` , double `smoothDistance` = 0, ExtraParam `extraParam` = null)**
반환 값	StatusCode: 모션 명령 실행 결과

4.10.3 BasLogical 논리 명령어 하위 클래스

BasLogical 하위 클래스에는 논리 제어 명령 시퀀스를 구축하는 데 사용되는 모든 논리 제어 관련 메서드가 포함되어 있습니다.

4.10.3.1 IF 조건부 명령어

메서드 이름	**BasScript.BasLogical.IF(param1, index, param2, value, operatorType)**
설명	스크립트에 논리적 IF 문을 추가합니다.
요청 매개변수	`param1` : 첫 번째 매개변수, 유형 RegisterType 또는 IOType `index` : 인덱스(정수) `param2` : 두 번째 매개변수, 유형 RegisterType, IOType 또는 OtherType `value` : 값, 유형 인덱스, 숫자, 문자열 또는 IOStatus `operatorType` : 부울 연산자, 기본값은 같음
반환 값	StatusCode: 작업 실행 결과

4.10.3.2 ELSE_IF 조건부 분기 명령어

메서드 이름	**BasScript.BasLogical.ELSE_IF(param1, index, param2, value, operatorType)**
설명	스크립트에 논리적 ELSE IF 문을 추가합니다.
요청 매개변수	`param1` : 첫 번째 매개변수, 유형 RegisterType 또는 IOType `index` : 인덱스(정수)

메서드 이름	BasScript.BasLogical.ELSE_IF(param1, index, param2, value, operatorType)
	param2 : 두 번째 매개변수, 유형 RegisterType, IOType 또는 OtherType value : 값, 유형 인덱스, 숫자, 문자열 또는 IOStatus operatorType : 부울 연산자, 기본값은 같음
반환 값	StatusCode: 작업 실행 결과

4.10.3.3 ELSE 명령

메서드 이름	BasScript.BasLogical.ELSE()
설명	스크립트에 논리적 ELSE 문을 추가합니다.
요청 매개변수	없음
반환 값	StatusCode: 작업 실행 결과

4.10.3.4 END_IF 조건부 명령어 종료

메서드 이름	BasScript.BasLogical.END_IF()
설명	논리적 IF 문을 종료합니다.
요청 매개변수	없음
반환 값	StatusCode: 작업 실행 결과

4.10.3.5 WHILE 루프 명령어

메서드 이름	BasScript.BasLogical.WHILE(param1, index, param2, value, operatorType)
설명	스크립트에 논리적 WHILE 문을 추가합니다.

메서드 이름	BasScript.BasLogical.WHILE(param1, index, param2, value, operatorType)
요청 매개변수	param1 : 첫 번째 매개변수, 유형 RegisterType 또는 IOType index : 인덱스(정수) param2 : 두 번째 매개변수, 유형 RegisterType, IOType 또는 OtherType value : 값, 유형 인덱스, 숫자, 문자열 또는 IOStatus operatorType : 부울 연산자, 기본값은 같음
반환 값	StatusCode: 작업 실행 결과

4.10.3.6 END_WHILE 루프 종료 명령

메서드 이름	BasScript.BasLogical.END_WHILE()
설명	논리적인 While 문을 종료합니다.
요청 매개변수	없음
반환 값	StatusCode: 작업 실행 결과

4.10.3.7 SWITCH 다중 지점 선택 지침

메서드 이름	BasScript.BasLogical.SWITCH(param, index)
설명	스크립트에 논리적 SWITCH 문을 추가합니다.
요청 매개변수	param : 매개변수, 유형 RegisterType 또는 IOType index : 매개변수의 인덱스
반환 값	StatusCode: 작업 실행 결과

4.10.3.8 CASE 분기 지시사항

메서드 이름	BasScript.BasLogical.CASE(param, value)
설명	스크립트에 논리적 CASE 문을 추가합니다.
요청 매개변수	`param` : 매개변수, 유형 RegisterType, IOType 또는 OtherType `value` : 값, 유형 인덱스, 숫자, 문자열
반환 값	StatusCode: 작업 실행 결과

4.10.3.9 DEFAULT 분기 지침

메서드 이름	BasScript.BasLogical.DEFAULT()
설명	스크립트에 논리적 DEFAULT 문을 추가합니다.
요청 매개변수	없음
반환 값	StatusCode: 작업 실행 결과

4.10.3.10 END_SWITCH 다중 지점 선택 종료 명령

메서드 이름	BasScript.BasLogical.END_SWITCH()
설명	논리적 SWITCH 문을 종료합니다.
요청 매개변수	없음
반환 값	StatusCode: 작업 실행 결과

4.10.3.11 SKIP_CONDITION 조건 건너뛰기 명령

메서드 이름	BasScript.BasLogical.SKIP_CONDITION(param1, index, param2, value, operatorType)
설명	스크립트에 논리적 SKIP CONDITION 문을 추가합니다.

메서드 이름	BasScript.BasLogical.SKIP_CONDITION(param1, index, param2, value, operatorType)
요청 매개변수	`param1` : 첫 번째 매개변수, 유형 RegisterType 또는 IOType `index` : 매개변수 1의 인덱스 `param2` : 두 번째 매개변수, 유형 RegisterType, IOType 또는 OtherType `value` : 값, 유형 인덱스, 숫자, 문자열 또는 IOStatus `operatorType` : 부울 연산자, 기본값은 같음
반환 값	StatusCode: 작업 실행 결과

4.10.3.12 GOTO 점프 지시

메서드 이름	BasScript.BasLogical.GOTO(index)
설명	GOTO 점프문
요청 매개변수	`index` : 대상 라벨의 인덱스
반환 값	StatusCode: 작업 실행 결과

4.10.3.13 라벨 지시사항

메서드 이름	BasScript.BasLogical.LABEL(index)
설명	LABEL 문
요청 매개변수	`index` : 라벨 인덱스
반환 값	StatusCode: 작업 실행 결과

4.10.3.14 BREAK 루프 중단 명령

메서드 이름	BasScript.BasLogical.BREAK()
설명	BREAK 문
요청 매개변수	없음
반환 값	StatusCode: 작업 실행 결과

4.10.3.15 CONTINUE 루프 건너뛰기 명령어

메서드 이름	BasScript.BasLogical.CONTINUE()
설명	CONTINUE 문
요청 매개변수	없음
반환 값	StatusCode: 작업 실행 결과

4.10.4 BasStructure 구조 명령어 서브클래스

BasStructure 하위 클래스에는 구조 제어 명령 시퀀스를 구축하는 데 사용되는 모든 구조 제어 관련 메서드가 포함되어 있습니다.

4.10.4.1 WAIT 대기 조건 명령어

메서드 이름	**BasScript.BasStructure.WAIT(RegisterType `param1` , int `index` , ValueType `param2` , object `value` , BooleanOperator `operatorType` = BooleanOperator.EQ)**
설명	대기 조건 명령, 조건이 만족될 때만 WAIT 명령을 실행하고, 프로그램은 계속 하향 실행될 수 있습니다. 그렇지 않으면 조건이 만족될 때까지 대기합니다. 이는 로봇 프로그램 작성 중 WAIT COND 문에 해당합니다.
요청 매개	`param1` : RegisterType 첫 번째 매개변수(레지스터 또는 IO 신호) `index` : int 매개변수 1의 인덱스

메서드 이름	**BasScript.BasStructure.WAIT(RegisterType** `param1` **, int** `index` **, ValueType** `param2` **, object** `value` **, BooleanOperator** `operatorType` **= BooleanOperator.EQ)**
변수	`param2` : ValueType 두 번째 매개변수(레지스터, IO 신호 또는 기타 유형) `value` : 개체 매개변수 2의 인덱스 또는 값 `operatorType` : BooleanOperator 논리 연산자, 기본값은 같음
반환 값	StatusCode: 작업 실행 결과

4.10.4.2 WAIT_TIME 대기 시간 지시

메서드 이름	**BasScript.BasStructure.WAIT_TIME(ValueType** `param` **, double** `value` **)**
설명	대기 시간 명령, WAIT TIME 명령 실행, 로봇은 지정된 시간 동안 기다린 후 후속 명령을 계속 실행합니다. 로봇 프로그램 작성 중 WAIT TIME 문에 해당합니다.
요청 매개변수	`param` : ValueType 매개변수 유형(R 레지스터 또는 값) `value` : double 시간 값, 단위: 초
반환 값	StatusCode: 작업 실행 결과

4.10.4.3 PAUSE 지시사항

메서드 이름	BasScript.BasStructure.PAUSE()
설명	PAUSE 문
요청 매개변수	없음
반환 값	StatusCode: 작업 실행 결과

4.10.4.4 ABORT 명령

메서드 이름	BasScript.BasStructure.ABORT()
설명	ABORT 문
요청 매개변수	없음
반환 값	StatusCode: 작업 실행 결과

4.10.4.5 CALL 동기 프로그램 호출 명령어

메서드 이름	BasScript.BasStructure.CALL(name)
설명	CALL 동기 프로그램 호출
요청 매개변수	`name` : 프로그램 이름
반환 값	StatusCode: 작업 실행 결과

4.10.4.6 RUN 비동기 프로그램 호출 명령어

메서드 이름	BasScript.BasStructure.RUN(name)
설명	RUN 비동기 프로그램 호출
요청 매개변수	`name` : 프로그램 이름
반환 값	StatusCode: 작업 실행 결과

4.10.4.7 LOAD 로드 프로그램 명령어

메서드 이름	BasScript.BasStructure.LOAD(param, value)
설명	LOAD 로드 프로그램

메서드 이름	BasScript.BasStructure.LOAD(param, value)
요청 매개변수	`param` : 매개변수, R 레지스터, SR 레지스터, 숫자 또는 문자열 `value` : 매개변수, 숫자 또는 문자열의 값
반환 값	StatusCode: 작업 실행 결과

4.10.4.8 UNLOAD 언로드 프로그램 명령어

메서드 이름	BasScript.BasStructure.UNLOAD(param, value)
설명	UNLOAD 언로드 프로그램
요청 매개변수	`param` : 매개변수, R 레지스터, SR 레지스터, 숫자 또는 문자열 `value` : 매개변수, 숫자 또는 문자열의 값
반환 값	StatusCode: 작업 실행 결과

4.10.4.9 EXEC 프로그램 실행 명령

메서드 이름	BasScript.BasStructure.EXEC(param, value)
설명	EXEC 실행 프로그램
요청 매개변수	`param` : 매개변수, R 레지스터, SR 레지스터, 숫자 또는 문자열 `value` : 매개변수, 숫자 또는 문자열의 값
반환 값	StatusCode: 작업 실행 결과

4.10.5 BasSocket 소켓 통신 하위 클래스

BasSocket 하위 클래스에는 소켓 통신 명령 시퀀스를 구축하는 데 사용되는 모든 소켓 통신 관련 메서드가 포함되어 있습니다.

4.10.5.1 OPEN 오픈 소켓 연결 지침

메서드 이름	**BasScript.BasSocket.OPEN(int** `index` **)**
설명	Socket Open 명령을 사용하여 Server를 생성하고 Client 연결을 기다립니다. 로봇 프로그램 작성 시 SOCKET OPEN에 해당합니다.
요청 매개변수	`index` : int SK 레지스터 인덱스
반환 값	StatusCode: 작업 실행 결과

4.10.5.2 CLOSE 소켓 연결 종료 명령

메서드 이름	**BasScript.BasSocket.CLOSE(int** `index` **)**
설명	지정된 소켓 연결을 닫습니다. 로봇 프로그램 작성 시 SOCKET CLOSE에 해당합니다.
요청 매개변수	`index` : int SK 레지스터 인덱스
반환 값	StatusCode: 작업 실행 결과

4.10.5.3 CONNECT 소켓 연결 지침

메서드 이름	**BasScript.BasSocket.CONNECT(int** `index` **)**
설명	Socket Connect 명령을 사용하여 지정된 소켓 서버에 연결하며, 로봇 프로그램 작성 시 SOCKET CONNECT에 해당합니다.
요청 매개변수	`index` : int SK 레지스터 인덱스
반환 값	StatusCode: 작업 실행 결과

4.10.5.4 SEND 소켓 데이터 전송 명령

메서드 이름	**BasScript.BasSocket.SEND(int** `index` **, StrType** `msgType` **, object** `value` **)**
설명	소켓 보내기 명령을 사용하여 지정된 소켓 연결로 데이터를 보냅니다. 로봇 프로그램 작성의 SOCKET SEND에 해당합니다.
요청 매개변수	`index` : int SK 레지스터 인덱스 `msgType` : StrType 메시지 유형 `value` : 객체 메시지 내용 또는 인덱스
반환 값	StatusCode: 작업 실행 결과

4.10.5.5 RECV 수신 소켓 데이터 명령어

메서드 이름	**BasScript.BasSocket.RECV(int** `index` **, int** `msgLength` **, StrType** `msgType` **, object** `value` **)**
설명	소켓 Recv 명령을 사용하여 지정된 소켓 연결에서 문자를 읽고, 최대 길이를 지정할 수 있으며, 로봇 프로그램 작성의 SOCKET RECV에 해당합니다.
요청 매개변수	`index` : int SK 레지스터 인덱스 `msgLength` : int 메시지 최대 길이 `msgType` : StrType 메시지 유형 `value` : 객체 메시지 내용 또는 인덱스
반환 값	StatusCode: 작업 실행 결과

4.10.6 BasModbus Modbus 통신 하위 클래스

BasModbus 하위 클래스에는 Modbus 레지스터 read/write 명령어 시퀀스를 구축하는 데 사용되는 모든 Modbus 통신 관련 메서드가 포함되어 있습니다.

4.10.6.1 READ_MH Modbus 보유 레지스터 읽기 명령

메서드 이름	**BasScript.BasModbus.READ_MH(int** `index` **, int** `id` **, int** `address` **, int** `length` **, int** `rIndex` **)**
설명	Modbus 보유 레지스터 명령 읽기, 로봇 프로그램 작성의 READ_MH에 해당
요청 매개변수	`index` : int 채널 인덱스 `id` : int Modbus ID `address` : int 레지스터 시작 주소 `length` : int 레지스터 길이, 즉 읽을 레지스터 수 `rIndex` : int 결과를 쓸 R 레지스터 시작 인덱스
반환 값	StatusCode: 작업 실행 결과

4.10.6.2 READ_MI Modbus 입력 레지스터 읽기 명령

메서드 이름	**BasScript.BasModbus.READ_MI(int** `index` **, int** `id` **, int** `address` **, int** `length` **, int** `rIndex` **)**
설명	Modbus 입력 레지스터 읽기 명령은 로봇 프로그램 작성의 READ_MI에 해당합니다.
요청 매개변수	`index` : int 채널 인덱스 `id` : int Modbus ID `address` : int 레지스터 시작 주소 `length` : int 레지스터 길이, 즉 읽을 레지스터 수 `rIndex` : int 결과를 쓸 R 레지스터 시작 인덱스
반환 값	StatusCode: 작업 실행 결과

4.10.6.3 WRITE_MH Modbus 홀딩 레지스터 쓰기 명령어

메서드 이름	**BasScript.BasModbus.WRITE_MH(int** `index` **, int** `id` **, int** `address` **, int** `length` **, ValueType** `valueType` **, int** `value` **)**
설명	Modbus 보유 레지스터 쓰기 명령은 로봇 프로그램 작성의 WRITE_MH에 해당합니다.
요청 매개변수	`index` : int 채널 인덱스 `id` : int Modbus ID `address` : int 레지스터 시작 주소 `length` : int 레지스터 길이, 즉 쓸 레지스터 수 `valueType` : ValueType 값 유형 `value` : int 값 또는 R 레지스터 시작 인덱스
반환 값	StatusCode: 작업 실행 결과

4.10.7 BasVision Vision 명령어 하위 클래스

BasVision 하위 클래스에는 비전 프로그램 명령 시퀀스를 구축하는 데 사용되는 모든 비전 관련 메서드가 포함되어 있습니다.

4.10.7.1 FIND 비전 프로그램 찾기 프로그램 안내

메서드 이름	**BasScript.BasVision.FIND(string** `name` **)**
설명	비전파인드 프로그램을 실행하며, 로봇 프로그램 작성시 VISION FIND에 해당합니다.
요청 매개변수	`name` : 문자열 비전 프로그램 이름
반환 값	StatusCode: 작업 실행 결과

4.10.7.2 GET_OFFSET 비전 프로그램 오프셋 명령 가져오기

메서드 이름	**BasScript.BasVision.GET_OFFSET(string** `name` **, int** `index` **, int** `labelIndex` **)**
설명	비전 프로그램 실행 후 오프셋을 가져옵니다. 로봇 프로그램 작성 시 VISION GET OFFSET에 해당합니다.
요청 매개변수	`name` : 문자열 비전 프로그램 이름 `index` : int 비전 레지스터 인덱스 `labelIndex` : int 라벨 인덱스
반환 값	StatusCode: 작업 실행 결과

4.10.7.3 GET_QUANTITY 비전 프로그램 결과 지시사항 가져오기

메서드 이름	**BasScript.BasVision.GET_QUANTITY(string** `name` **, int** `index` **)**
설명	비전 프로그램 실행 후 결과 수량을 가져옵니다. 로봇 프로그램 작성의 VISION GET QUANTITY에 해당합니다.
요청 매개변수	`name` : 문자열 비전 프로그램 이름 `index` : int R 결과를 저장할 인덱스를 등록합니다.
반환 값	StatusCode: 작업 실행 결과

4.10.8 AssignValue 할당 지침(전체 매개변수)

메서드 이름	**BasScript.AssignValue(AssignType** `param1` **, int** `index` **, AssignType** `param2` **, object** `value` **, int** `optIndex` **= 0, int** `optValue` **= 0)**
설명	레지스터, IO 신호 등에 값을 할당하는 데 사용되는 할당 명령은 로봇 프로그램 작성의 ASSIGN 문에 해당합니다.
요청 매개변수	`param1` : AssignType 첫 번째 매개변수(레지스터 또는 IO 신호) `index` : int 매개변수 1의 인덱스 `param2` : AssignType 두 번째 매개변수(레지스터, IO 신호 또는 기타 유형) `value` : 개체 매개변수 2의 인덱스 또는 값 `optIndex` : int param1이 PR_ELEMENT일 때 매개변수 1에 대한 추가 인덱스, 기본값 0

메서드 이름	**BasScript.AssignValue(AssignType** `param1` **, int** `index` **, AssignType** `param2` **, object** `value` **, int** `optIndex` **= 0, int** `optValue` **= 0)**
	`optValue` : int param2가 PR_ELEMENT일 때 매개변수 2에 대한 추가 인덱스 또는 값이 IOStatus.PULSE일 때 펄스 값, 기본값 0
반환 값	StatusCode: 작업 실행 결과

4.10.9 AssignValue 할당 지침(단순화된 매개변수)

메서드 이름	**BasScript.AssignValue(AssignType** `param` **, int** `index` **, object** `value` **)**
설명	레지스터, IO 신호 등에 값을 할당하는 데 사용되는 할당 명령(간단화된 매개변수)은 로봇 프로그램 작성의 ASSIGN 문에 해당합니다.
요청 매개변수	`param` : AssignType 매개변수 유형(레지스터 또는 IO 신호) `index` : int 매개변수 인덱스 `value` : 개체 값(IOStatus, double 또는 문자열)
반환 값	StatusCode: 작업 실행 결과

4.10.10 SetParam 매개변수 설정 명령

메서드 이름	**BasScript.SetParam(ParamType** `type` **, ValueType** `valueType` **, object** `value` **)**
설명	로봇의 각종 파라메타를 설정하는데 사용되는 파라메타 설정 명령은 로봇 프로그램 작성시 SET PARAM 에 해당합니다.
요청 매개변수	`type` : ParamType 매개변수 유형 `valueType` : ValueType 값 유형 `value` : 객체 값
반환 값	StatusCode: 작업 실행 결과

4.11 좌표계 클래스

개요

`CoordinateSystem` 클래스는 로봇 사용자 프레임(UF) 및 툴 프레임(TF)을 관리하는 데 사용되며 좌표계 추가, 삭제, 업데이트, 쿼리 및 계산을 위한 통합 인터페이스를 제공합니다.

핵심 기능

- user/tool 좌표계 요약 정보 목록 가져오기 지원
- user/tool 좌표계 추가, 삭제, 업데이트 및 쿼리 지원
- 여러 입력 포즈를 기반으로 user/tool 좌표계 계산 지원
- 컨트롤러에서 좌표계 목록을 쉽게 유지 관리할 수 있도록 통합 좌표계 관리 인터페이스를 제공합니다.

사용 사례

- 도구 전환 중 다양한 도구 좌표계 관리
- 다중 스테이션 디버깅 중에 일관된 공간 참조 유지
- 로봇 좌표계 일괄 관리 및 조회
- 학습된 점에서 새로운 사용자 /tool 좌표계를 빠르게 계산합니다.

4.11.1 User/Tool 좌표계 목록 가져오기

메서드 이름	**CoordinateSystem.GetCoordinateList(CoordinateType `type`)**
설명	지정된 좌표 유형에 대한 모든 좌표계 요약 정보 목록을 가져옵니다.
요청 매개변수	`type` : CoordinateType 좌표 유형(UF 또는 TF)
반환 값	List<CoordSummary>: 좌표 요약 목록 StatusCode: 연산 실행 결과

메서드 이름	CoordinateSystem.GetCoordinateList(CoordinateType `type`)
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.11.2 User/Tool 좌표계 추가

메서드 이름	CoordinateSystem.Add(CoordinateType `type` , Coordinate `coordinate`)
설명	지정된 좌표 유형에 대한 새 좌표계를 추가합니다.
요청 매개변수	`type` : CoordinateType 좌표 유형 (UF 또는 TF) `coordinate` : Coordinate 추가할 좌표 데이터
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.11.3 User/Tool 좌표계 삭제

메서드 이름	CoordinateSystem.Delete(CoordinateType `type` , int `index`)
설명	좌표 유형 및 인덱스로 지정된 좌표계를 삭제합니다.
요청 매개변수	`type` : CoordinateType 좌표 유형(UF 또는 TF) `index` : int 좌표 인덱스
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.11.4 User/Tool 좌표계 업데이트

메서드 이름	**CoordinateSystem.Update(CoordinateType `type` , Coordinate `coordinate`)**
설명	지정된 좌표 유형 및 좌표 정보에 대한 좌표계를 업데이트합니다.
요청 매개변수	`type` : CoordinateType 좌표 유형 (UF 또는 TF) `coordinate` : Coordinate 업데이트할 좌표 데이터
반환 값	StatusCode: 작업 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.11.5 User/Tool 좌표계 정보 가져오기

메서드 이름	**CoordinateSystem.Get(CoordinateType `type` , int `index`)**
설명	좌표 유형 및 색인별로 좌표계 정보를 가져옵니다.
요청 매개변수	`type` : CoordinateType 좌표 유형(UF 또는 TF) `index` : int 좌표 인덱스
반환 값	Coordinate: 좌표 데이터 StatusCode: 연산 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

4.11.6 User/Tool 좌표계 계산

메서드 이름	**CoordinateSystem.Calculate(CoordinateType `type` , List<Position> `pose`)**
설명	입력 포즈 포인트에서 user/tool 좌표계를 계산합니다.
요청 매개변수	`type` : CoordinateType 좌표 유형(UF 또는 TF) `pose` : 목록<Position> 입력 포즈 목록입니다. 4점 또는 7점 방법을 지원합니다. 4점 방식에서는 도구가 동일한 목표점을 가리키는 4가지 포즈를 제공합니다. 7-point 방식에서는 원점, X방향 포인트, Y방향 포인트를 추가로 제공합니다. 각도 단위: 도(°).
반환 값	Position: 계산된 좌표 포즈 StatusCode: 연산 실행 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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 로봇 조깅 모션

개요

`Jogging` 클래스는 단일 축 스테핑, 단일 축 연속 조그, 다축 연속 조그 및 정지 작업을 포함하여 수동 교육 모드에서 조그 제어 API를 제공합니다.

핵심 기능

- 단축 스텝 조깅
- 단축 연속 조깅
- 다축 연속 조깅
- 지속적인 조깅을 중지
- 포지티브/negative 축 기호를 사용한 방향 제어
- 구성 가능한 선형 스텝 길이 및 회전 스텝 각도

사용 사례

- 로봇 디버깅 및 교육 과정
- 미세 위치 및 자세 조정
- 호스트 측 원격 수동 자세 조정
- 로봇 설치 및 교정
- 복잡한 궤적을 실행하기 전 Position 확인

4.12.1 단일축 스텝 조깅

메서드 이름	**Jogging.Move(int** `ajNum` **, MoveMode** `moveMode` **, double** `stepLength` **= 0, double** `stepAngle` **= 0)**
설명	단일축 스텝 조깅(`moveMode` = `MoveMode.Stepping`)을 실행합니다.

메서드 이름	**Jogging.Move(int** `ajNum` **, MoveMode** `moveMode` **, double** `stepLength` **= 0, double** `stepAngle` **= 0)**
요청 매개변수	`ajNum` : int 축 인덱스(1~6은 현재 좌표계 축에 매핑됩니다. 데카르트 모드에서는 x/y/z/rx/ry/rz가 1~6에 매핑됩니다. 부호는 방향을 나타냅니다.) `moveMode` : 이 시나리오에서는 이동 모드가 `MoveMode.Stepping` 로 고정되었습니다. `stepLength` : double 선형 스텝 길이(mm)(스테핑 모드에서 유효). `stepAngle` : 이중 회전 스텝 각도(°)(스테핑 모드에서 유효).
반환 값	StatusCode: 조그 동작 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

Jogging/StepJogging.cs

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 단일축 연속 조깅

메서드 이름	**Jogging.Move(int** `ajNum` **, MoveMode** `moveMode` **, double** `stepLength` **= 0, double** `stepAngle` **= 0)**
설명	단축 연속 조깅(`moveMode` = `MoveMode.Continuous`)을 실행합니다.
요청 매개변수	`ajNum` : int 축 인덱스(1~6은 현재 좌표계 축에 매핑됩니다. 데카르트 모드에서는 x/y/z/rx/ry/rz가 1~6에 매핑됩니다. 부호는 방향을 나타냅니다.) `moveMode` : 이 시나리오에서는 이동 모드가 `MoveMode.Continuous` 로 고정되었습니다. `stepLength` : 이중 단계 매개변수(일반적으로 연속 모드에서는 사용되지 않음). `stepAngle` : 이중 단계 각도 매개변수(일반적으로 연속 모드에서는 사용되지 않음).
반환 값	StatusCode: 조그 동작 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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;
    }
}
```

4.12.3 다축 연속 조깅

메서드 이름	**Jogging.MultiMove(int[] `ajNums`)**
설명	여러 축에서 동시에 연속 조깅을 실행합니다.
요청 매개변수	`ajNums` : int[] 축 목록(1~6은 현재 좌표계 축에 매핑됩니다. 데카르트 모드에서는 x/y/z/rx/ry/rz가 1~6에 매핑됩니다. 기호는 각 축 방향을 나타냅니다.)
반환 값	StatusCode: 조그 동작 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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 지속적인 조깅 중지

메서드 이름	Jogging.Stop()
설명	현재 연속적인 조그 이동을 중지합니다.
요청 매개변수	없음
반환 값	무효의
메모	이 방법은 연속 조깅 모드에서만 필요합니다.
호환 로봇 소프트웨어 버전	협업(Copper): v7.5.0.0+ 산업용(Bronze): v7.5.0.0+

예제 코드

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 로봇 구독 및 게시 인터페이스

개요

`SubPub` 클래스는 로봇 컨트롤러에 대한 WebSocket 구독/publish 채널 관리를 제공합니다. 연결을 설정하고, 로봇 status/register/IO 주제를 구독하고, 콜백 또는 차단 API를 통해 실시간 데이터를 수신하는 데 사용됩니다.

핵심 기능

- WebSocket 서버 연결 및 연결 해제 지원
- 로봇 상태 데이터 구독 지원
- 데이터 등록 구독 지원
- IO 신호 데이터 구독 지원
- 콜백 기능을 통한 지속적인 메시지 수신 지원
- 다음 메시지 수신 차단 지원
- 지원 설정 구독 빈도
- 메시지 수신 중 예외 처리 및 문제 해결 지원

사용 사례

- 호스트 측 로봇 작동 상태 실시간 모니터링
- 로봇 데이터 시각화 구현
- 외부 시스템과의 데이터 연계 실현
- 레지스터 및 IO 상태의 실시간 모니터링
- 로봇 모니터링 및 제어 시스템 구축
- 로봇 상태에 대한 실시간 알람 및 알림 구현

4.13.1 WebSocket 서버에 연결

메서드 이름	SubPub.Connect()
설명	로봇 컨트롤러 WebSocket 서버에 연결합니다.
요청 매개변수	없음
반환 값	작업: 비동기 연결 작업 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.7.0.0+ 산업용(Bronze): v7.7.0.0+

4.13.2 WebSocket 서버에서 연결 끊기

메서드 이름	SubPub.Disconnect()
설명	로봇 컨트롤러 WebSocket 서버와의 연결을 끊습니다.
요청 매개변수	없음
반환 값	작업: 비동기 연결 해제 작업 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.7.0.0+ 산업용(Bronze): v7.7.0.0+

4.13.3 로봇 상태 구독 추가

메서드 이름	SubPub.SubscribeStatus(RobotTopicType[] `topicTypes` , int `frequency` = 200)
설명	로봇 상태 데이터 구독을 추가합니다.
요청 매개변수	`topicTypes` : RobotTopicType[] 로봇 토픽 유형 목록 `frequency` : int 구독 빈도(단위:Hz, 기본값 200)
반환 값	작업: 비동기 구독 작업 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.7.0.0+ 산업용(Bronze): v7.7.0.0+

4.13.4 등록 구독 추가

메서드 이름	**SubPub.SubscribeRegister(RegTopicType `regType` , int[] `regIds` , int `frequency` = 200)**
설명	등록 데이터 구독을 추가합니다.
요청 매개변수	`regType` : RegTopicType 레지스터 유형 `regIds` : int[] 레지스터 ID 목록 `frequency` : int 구독 빈도(단위:Hz, 기본값 200)
반환 값	작업: 비동기 구독 작업 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.7.0.0+ 산업용(Bronze): v7.7.0.0+

4.13.5 IO 구독 추가

메서드 이름	**SubPub.SubscribeIO((IOTopicType, int)[] `ioList` , int `frequency` = 200)**
설명	디지털 입력/output, 등을 포함한 IO 신호 데이터를 구독합니다.
요청 매개변수	`ioList` : (IOTopicType, int)[] IO 목록 (각 요소는 `(IO Type, IO ID)` 입니다) `frequency` : int 구독 주파수 (단위:Hz, 기본값 200)
반환 값	작업: 비동기 구독 작업 결과
호환 로봇 소프트웨어 버전	협업(Copper): v7.7.0.0+ 산업용(Bronze): v7.7.0.0+

4.13.6 메시지 수신 시작

메서드 이름	**SubPub.StartReceiving(Func<Dictionary<string, object>, Task> `onMessageReceived`)**
설명	구독 메시지 수신을 시작하고 콜백 함수를 통해 수신된 데이터를 처리합니다.
요청 매개변수	`onMessageReceived` : Func<Dictionary<string, object>, Task> 메시지 수신 콜백 함수
반환 값	작업: 비동기 수신 작업
메모	콜백에서 예외가 발생하면 수신 루프가 중지될 수 있습니다. 콜백 내에서 예외를 포착하고 기록하는 것이 좋습니다.
호환 로봇 소프트웨어 버전	협업(Copper): v7.7.0.0+ 산업용(Bronze): v7.7.0.0+

예제 코드

SubPub/CallbackReceiving.cs

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 다음 메시지 수신

메서드 이름	SubPub.Receive()
설명	다음 메시지 수신을 차단하고 이를 반환합니다.
요청 매개변수	없음
반환 값	Task<Dictionary<string, object>>: 수신된 메시지 사전
메모	연결이 설정되지 않거나, 연결이 닫히거나, 메시지 형식이 비정상적인 경우 예외가 발생할 수 있습니다.
호환 로봇 소프트웨어 버전	협업(Copper): v7.7.0.0+ 산업용(Bronze): v7.7.0.0+

예제 코드

SubPub/PollingReceiving.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;
            }
        }
    }
}
```

Agilebot C# SDK 업데이트 노트

2.1.0.* 업데이트(2026/4/9)

1. UDP 피드백 구성 관련 인터페이스를 추가했습니다.
2. `Motion.SetPositionTrajectoryParams` UDP 위치 제어 관련 매개변수 설정 인터페이스에 `filterLayer` 매개변수를 추가했습니다.
3. `Motion.Payload.GetPayloadIdentifyState` 부하 측정 상태 조회 인터페이스의 메시지 파싱을 업데이트 했습니다.
4. `TerminatePayloadIdentify` 부하 측정 종료 인터페이스를 추가했습니다.
5. 궤적 테이블(trajectory table) 및 경로 테이블(path table) 관련 인터페이스를 추가했습니다.
6. 궤적 정형기(trajectory shaper) 관련 인터페이스를 추가했습니다.
7. 경로 계획 파라미터 관련 인터페이스를 추가했습니다.

2.0.3.* 업데이트(2025/12/17)

1. Arm 생성자에는 이제 TeachingPanelIP 매개변수가 포함됩니다.
2. System.Text.Json 종속성을 제거했습니다.
3. 동기 연결 인터페이스 ConnectSync가 추가되었습니다.

2.0.2.* 업데이트(2025/12/12)

1. PR 레지스터 구조체에서 잘못된 /write 데이터 읽기 순서를 수정했습니다.

2.0.1.* 업데이트(2025/10/21)

1. 연속 조깅 중 끊김 현상이 수정되었습니다.
2. 프록시 실행 파일이 복사되지 않을 수 있는 빌드 시간 오류를 수정했습니다.

2.0.0.* (2025-09-10)

1. 기본 요청 패턴을 리팩터링하고 로컬 컨트롤러 프록시 서비스를 추가했습니다.
2. 구독 기능을 도입했습니다.
3. BasScript 클래스 구조를 재구성했습니다.
4. .NET Framework 및 .NET(Core/5+/6+)을 모두 완벽하게 지원합니다.

1.0.1.0 (2025년 7월 7일)

1. 7.6.0.0 이전의 로봇 버전과 호환되도록 이전 레지스터 인터페이스 클래스 `RegistersOld` 를 추가했습니다.
2. Estop 비상 제동 인터페이스 추가
3. 문서의 예제 프로그램을 수정했습니다.

1.0.0.0 (2025년 5월 30일)

1. RPC 방식을 사용하여 구현되었습니다.
2. 모든 인터페이스 정의를 Python 버전과 동기화했습니다.

돕다

에이전트 스킬

에이전트 스킬은 산업 및 개발 시나리오를 위한 표준 실행 워크플로와 도구를 AI 에이전트(예: Codex, Claude Code)에 확장해 주는 기능입니다.

지원되는 에이전트 유형

에이전트 스킬은 현재 다음 AI 도구와 CLI에 대한 자동 감지 및 설치를 지원합니다.

- **IDE / 편집기**: Cursor, Windsurf, Trae, Trae CN, VS Code (Copilot / Continue), Neovate, Pochi
- **명령줄(CLI)**: Claude Code, Kimi Code CLI, Gemini CLI, iFlow CLI, Kiro CLI, Mistral Vibe
- **프레임워크 / 플랫폼**: OpenHands, Replit, Cline, Roo Code, Codex, Amp, Antigravity, Augment, OpenClaw
- **기타**: CodeBuddy, Command Code, Droid, Junie, Kilo Code, Kode, MCPJam, Mux, OpenCode, Pi, Qoder, Qwen Code, Zencoder, AdaL

빠른 시작

1. 환경 준비

설치 전에 로컬 환경에 Node.js가 설치되어 있는지 확인하세요. 안내 문서: Node.js Download and Installation

2. 설치 명령 실행

터미널에서 다음 명령을 실행하면 대화형 설치 프로그램이 시작됩니다.

```bash
npx skills add https://dev.sh-agilebot.com -g
```

TIP

명령을 실행하면 도구가 로컬에 설치된 에이전트를 스캔합니다. 안내에 따라 스킬을 설치할 에이전트(예: Cursor 또는 Claude Code)를 선택하세요.

사용 가능한 스킬

agilebot-nlu-control (Agilebot 자연어 제어)

이 스킬은 Agilebot 로봇을 위해 설계되었습니다. NLU(Natural Language Understanding)를 사용하여 에이전트가 제어 명령을 직접 해석하고 실행할 수 있도록 합니다.

- 환경 요구 사항: 시스템 Python을 사용할 수 있어야 하며 Conda가 설치되어 있어야 합니다. 스킬은 필요한 의존성을 점검하고, 누락된 항목이 있으면 설치를 안내합니다.
- 트리거: 대화에서 "Agilebot"이 언급되고 구체적인 제어 의도가 함께 주어질 때
- 핵심 기능: 자연어 명령 해석 및 실행, 로봇 상태 조회, 기본 동작 제어, 매개변수화된 동작 명령(예: distance/angle/speed)

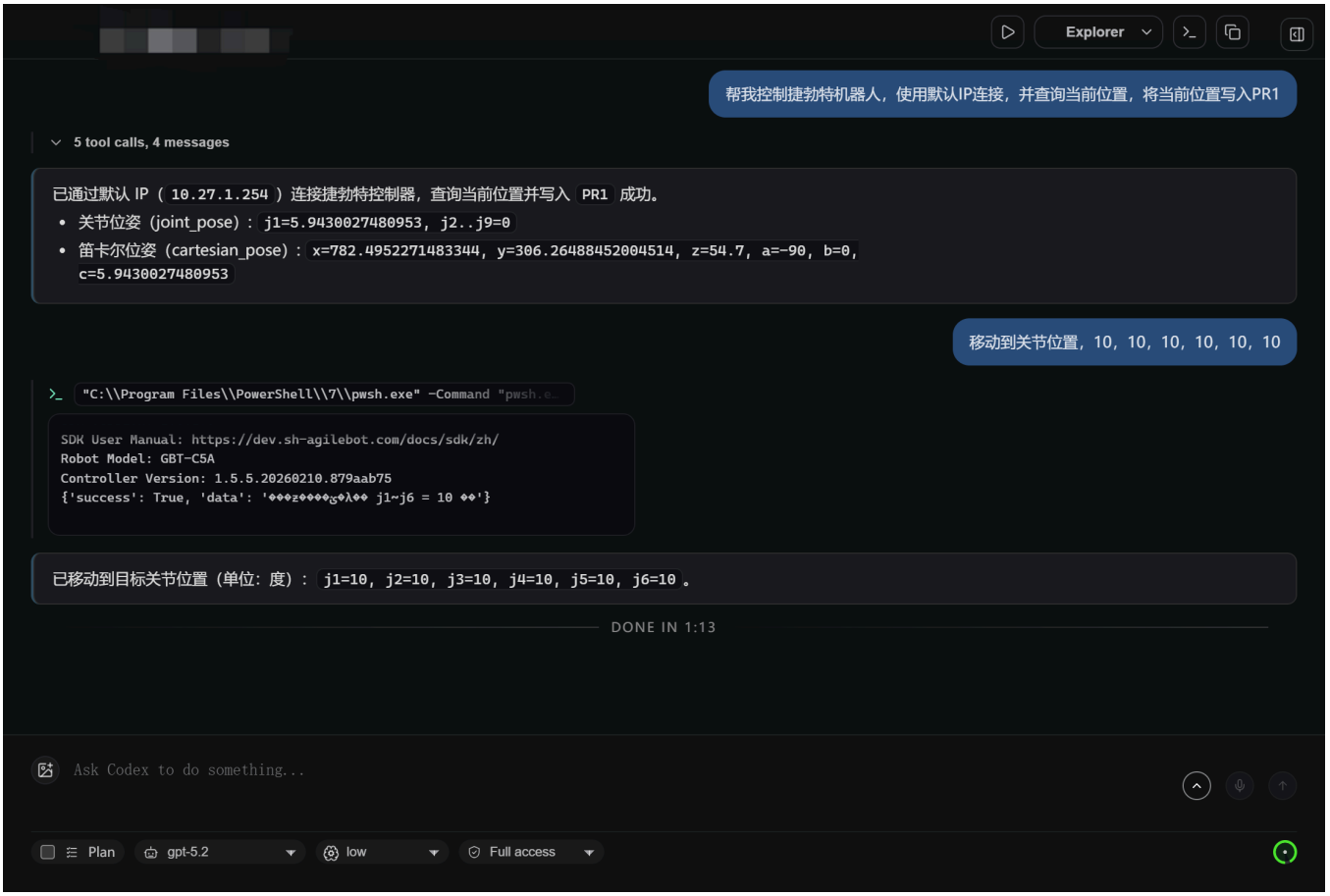
경고

설치된 스킬은 에이전트의 전체 권한으로 실행됩니다. 반드시 신뢰할 수 있는 환경에서 사용하세요.

로봇 동작은 위험할 수 있으며, 인명 피해나 장비 손상을 초래할 수 있습니다. 안전이 확보된 환경에서 사용해야 하며, 사용으로 인해 발생한 위험과 손실에 대해서는 책임지지 않습니다.

사용 예

Codex를 예로 들면, 설치 후 채팅에 직접 지시를 입력해 사용할 수 있습니다.



AI 코딩 지원

이 문서에서는 AI 지원 도구(예: CodeBuddy, Codex, Cursor 등)를 사용하여 로봇 프로그램을 빠르게 개발하는 방법을 설명합니다.

준비

AI 코딩을 사용하기 전에 참조 문서를 준비해야 합니다.

- **SDK 문서**: https://dev.sh-agilebot.com/docs/sdk/knowledge/docs.txt

> **팁**: AI 에이전트가 URL을 잘 읽을 수 없는 경우 위의 txt 문서를 로컬 프로젝트 디렉터리에 다운로드하고 프롬프트에서 로컬 파일 경로를 참조하세요.

예제 프롬프트

다음은 로봇 상태를 읽는 Python 프로그램을 생성하는 완전한 예입니다.

```
Read the following documentation and write a Python program that reads the r
obot's current position, coordinate system number, servo status, and other i
nformation.

Reference materials:
SDK Documentation: https://dev.sh-agilebot.com/docs/sdk/knowledge/docs.txt
```

로컬 문서에 의존하는 경우 다음과 같이 수정할 수 있습니다.

```
Read the following documentation and write a Python program that reads the r
obot's current position, coordinate system number, servo status, and other i
nformation.

Reference materials:
SDK Documentation: ./docs/sdk_docs.txt
```

사용 팁

1. **명확한 요구사항**: 구현하려는 기능을 명확하게 설명하세요.
2. **컨텍스트 제공**: 관련 문서 및 예시 참조
3. **단계적 구현**: AI를 사용하여 복잡한 기능을 단계별로 생성할 수 있습니다.

메모

1. AI가 생성한 코드는 검증과 테스트가 필요함
2. 코드가 프로젝트 코딩 표준을 준수하는지 확인하세요.
3. 로봇 제어와 관련된 코드는 보안 검토를 받아야 합니다.