Agilebot Robot SDK

Agilebot Robot SDK Manual

Python SDK

High-performance robot control development kit based on Python, featuring elegant and concise API design to help you rapidly build intelligent robotics applications.

Learn More →

C# SDK

Enterprise-grade robot control solution for the .NET ecosystem, providing type-safe strongly-typed APIs for seamless integration into industrial automation systems.

Learn More →

C# SDK

Prologue

Version History

Document Version	SDK Version Number	Version Date
V3.3	2.1.0.*	2026.04.13

Update Notes

Robot Version Compatibility

The SDK supports Agilebot Scara, Puma, and collaborative robot series. It must be used with devices that have the robot software installed and is compatible with the robot software versions. Some functions may return different results due to version differences.

When the SDK connects to the robotic arm, it will check the version of the robotic arm motion control software. If the version is lower than the minimum requirement, the connection will fail. If it is lower than the recommended version, a prompt indicating that the version is too low will appear. Please update the robot software version in a timely manner.

Some interfaces of the SDK only support the corresponding version of the controller. Please check the compatibility of specific interfaces.

SDK Version	Compatible Robot Software Versions	Support Status
0.1.1.X	Copper v7.5.X.X, Bronze v7.5.X.X	Discontinued
0.1.2.X	Copper v7.5.X.X, Bronze v7.5.X.X	Discontinued
0.2.0.X	Copper v7.5.X.X, Bronze v7.5.X.X	Discontinued
1.0.0.X	Copper v7.6.X.X, Bronze v7.5.X.X	Supported
2.0.X.X	Copper v7.7.X.X, Bronze v7.7.X.X	Supported
2.1.X.X	Copper v7.7.I.X	Supported

1 Introduction and Deployment

1.1 Environment Requirements

System:

- Windows 10 or later
  - x86_64 architecture
- .NET Version
  - 6.0 or higher
- .NET Framework Version
  - 4.7 or higher

1.2 Installation

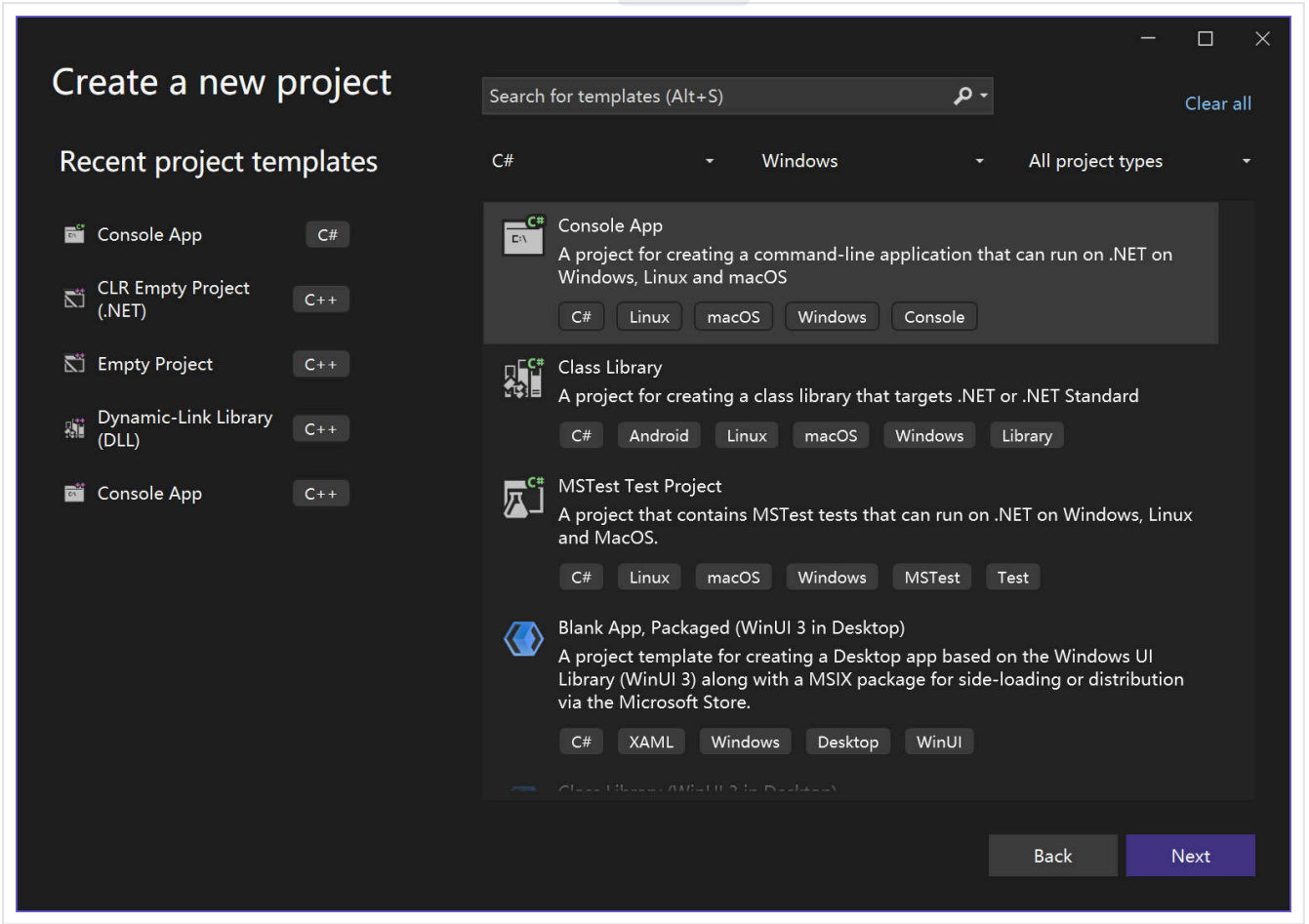
This section walks through IDE preparation, SDK installation, and the most common runtime caveats so you can start experimenting with the Agilebot SDK right away.

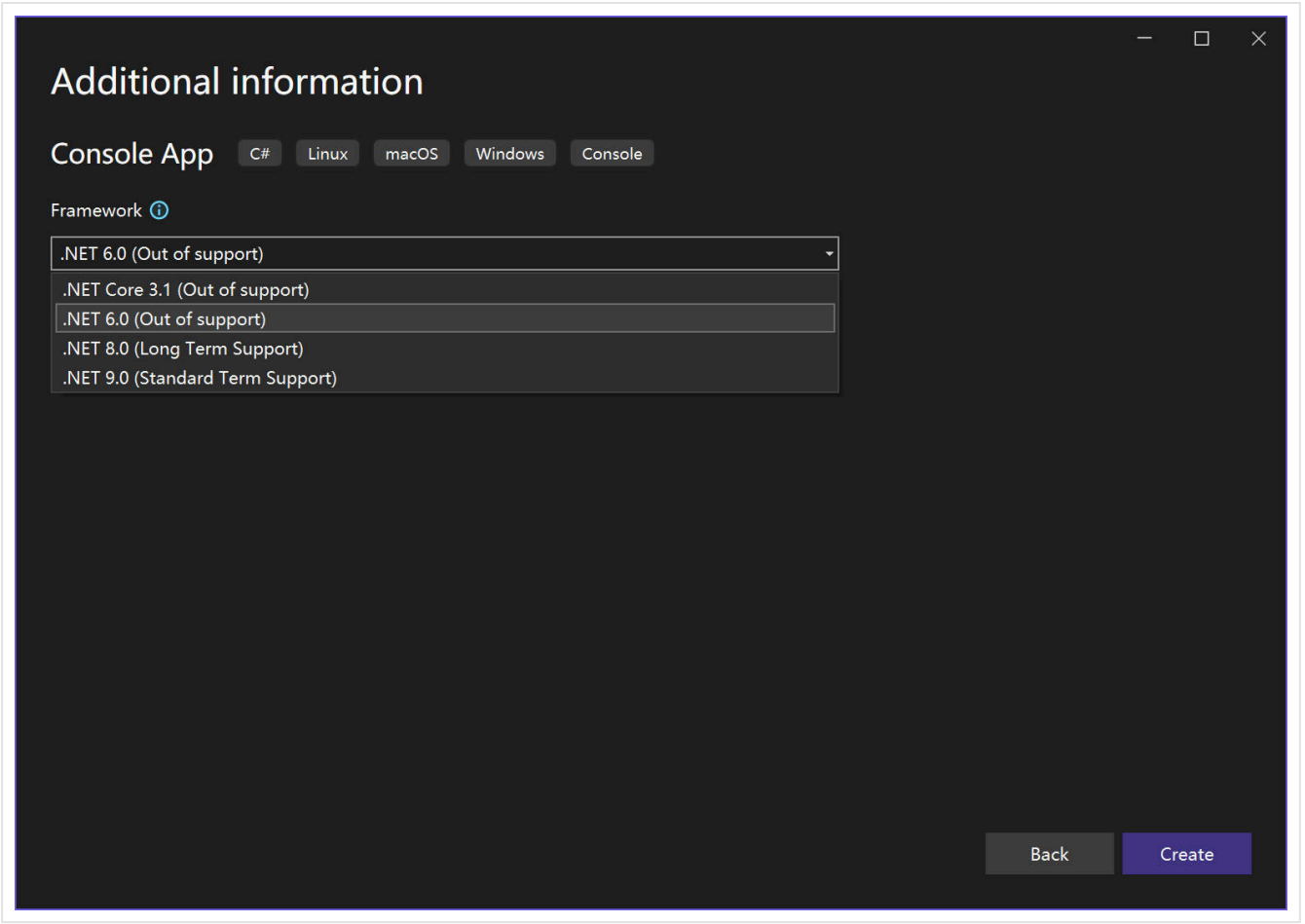
IDE Setup

1. Visual Studio is the recommended IDE for C# development. Download it from Download Visual Studio Tools - Free Install for Windows, Mac, Linux.
2. After installation, launch Visual Studio and finish the initial setup (sign in, install required workloads, etc.).

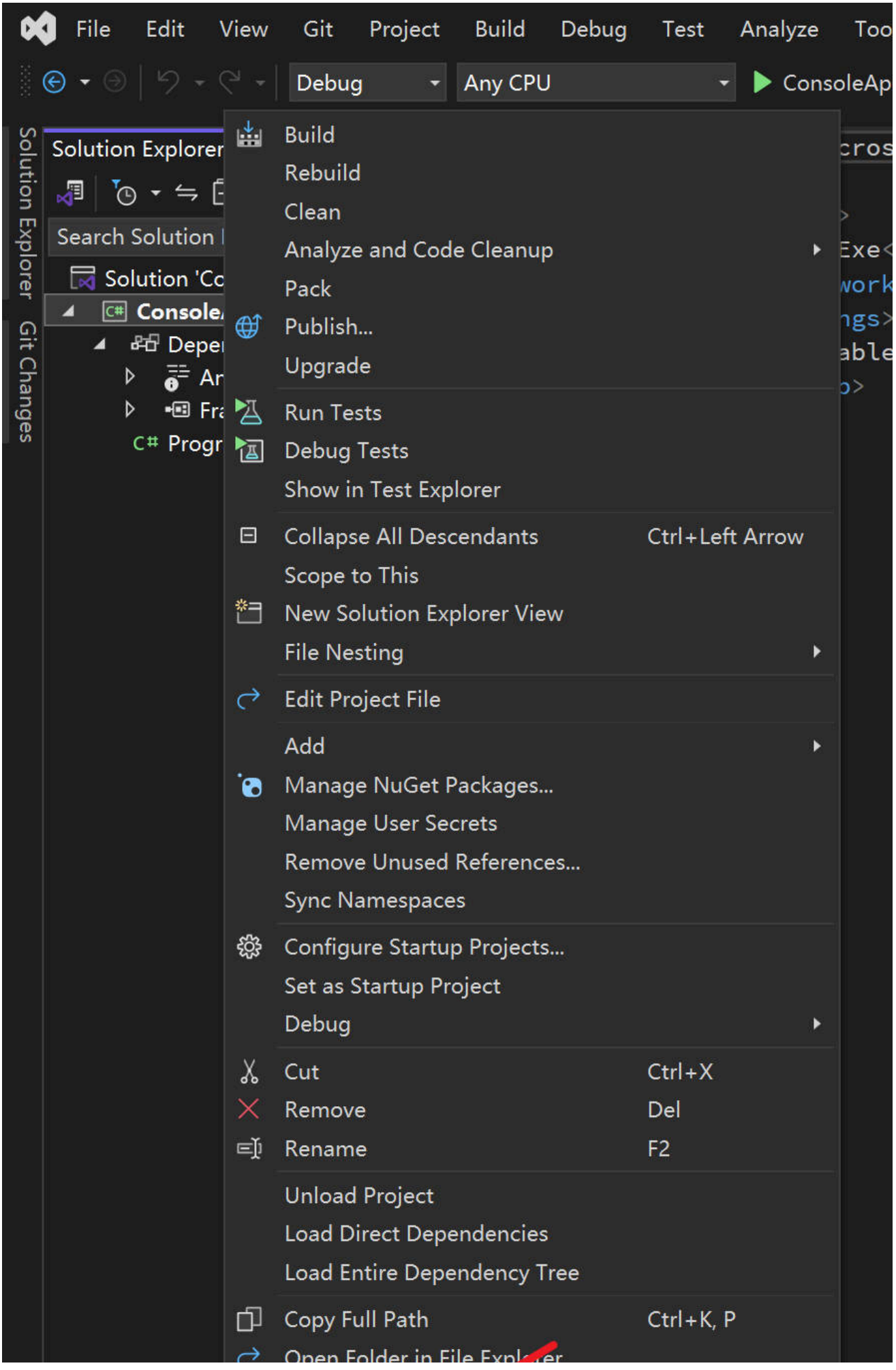
Get the SDK and Create a Project

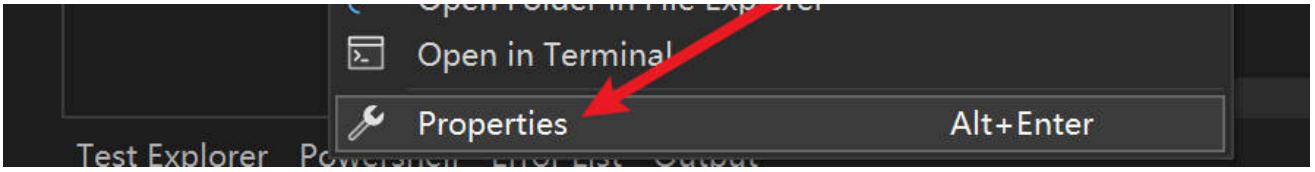
1. Create a new **C# Console App** and choose `.NET 6.0` or later as the target framework.

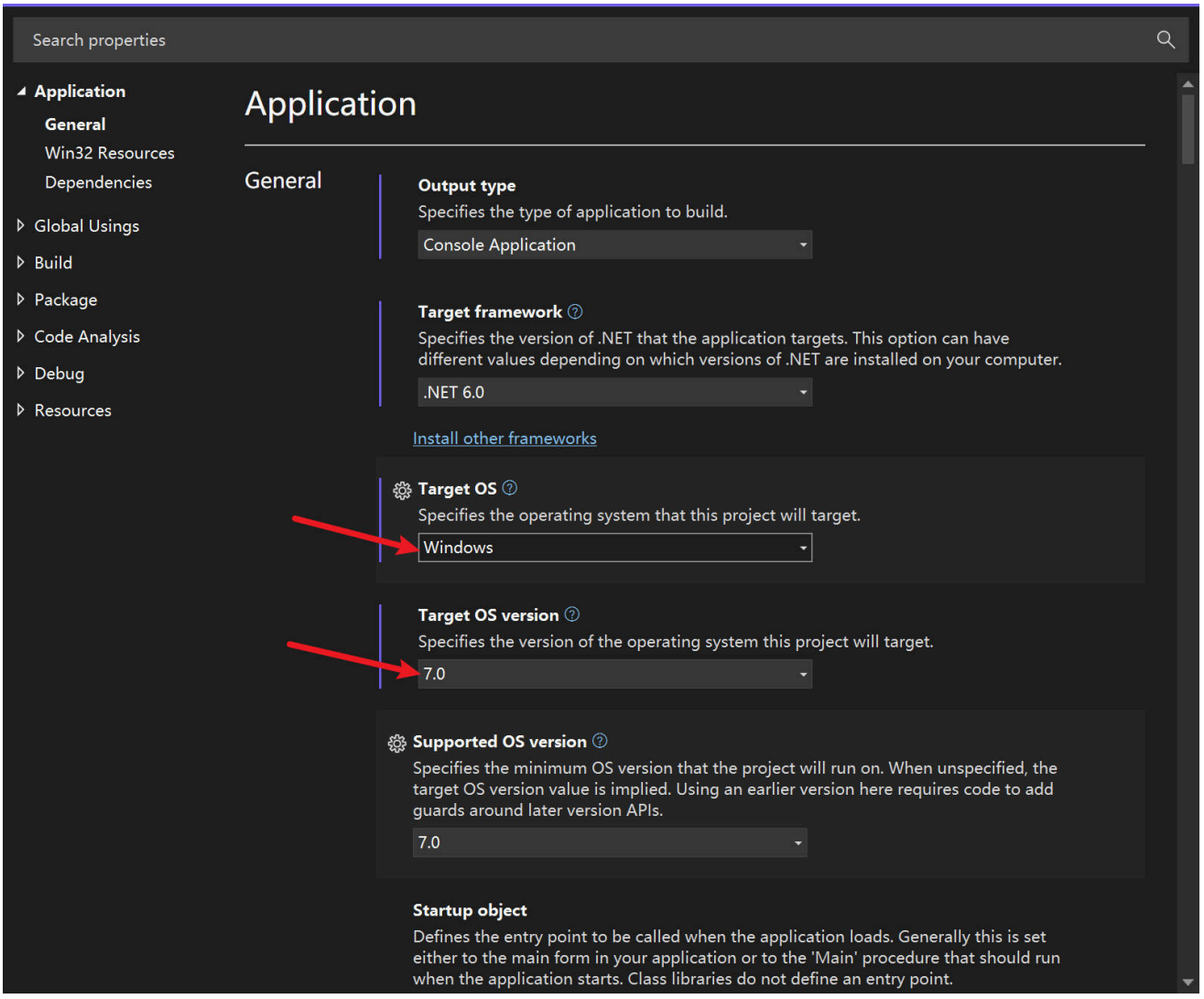




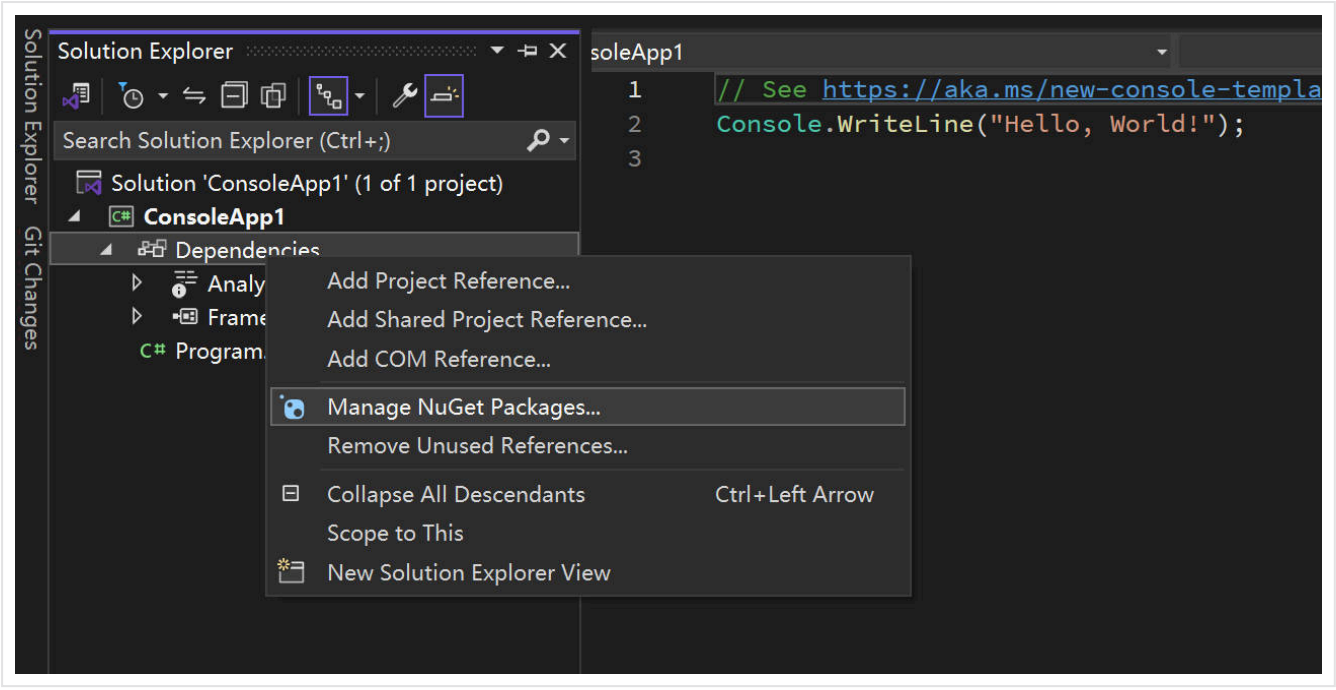
2. Open the project properties, set the target OS to **Windows**, and pick **version 7.0 or higher** to leverage the latest WinApp SDK features.

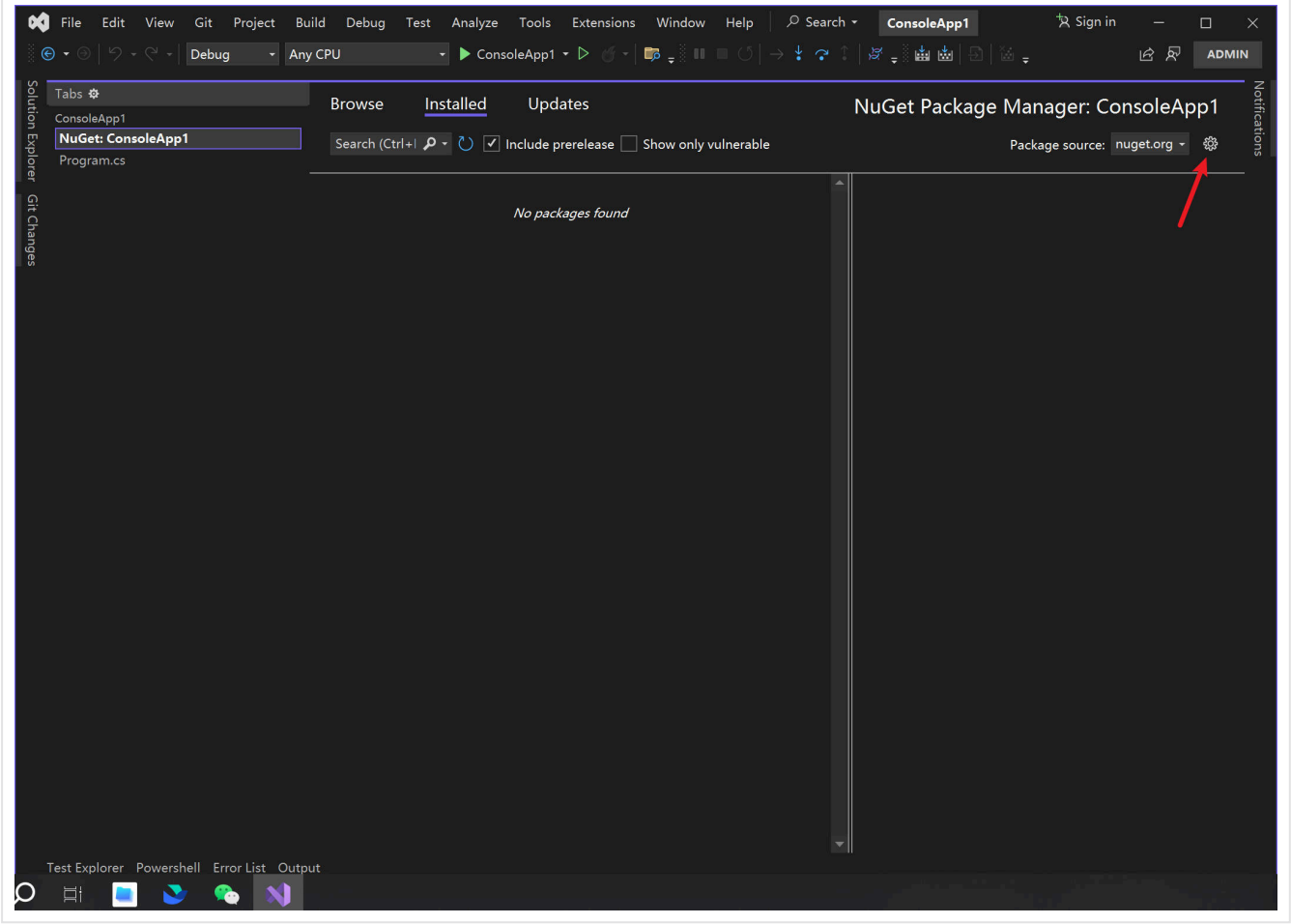


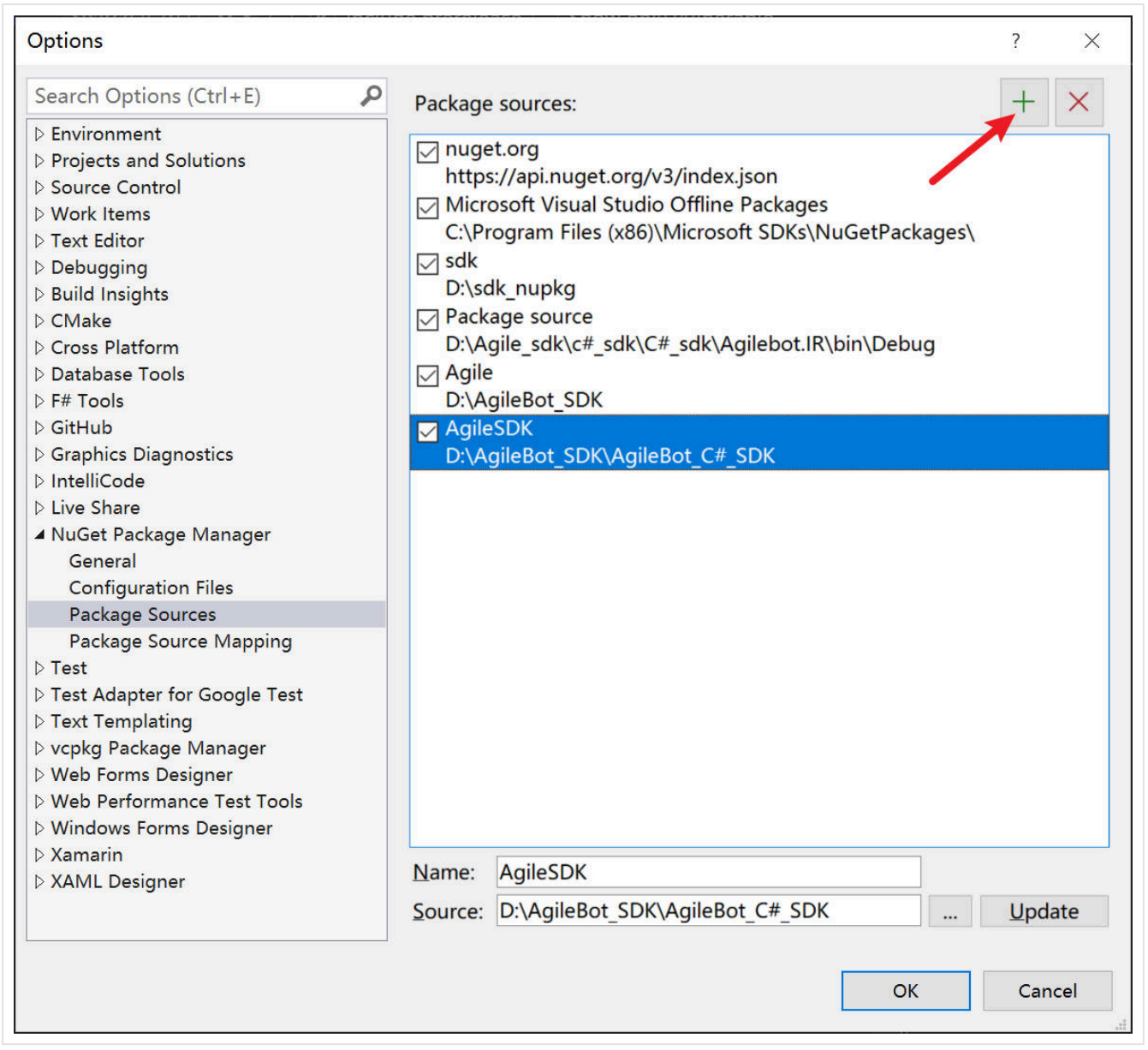


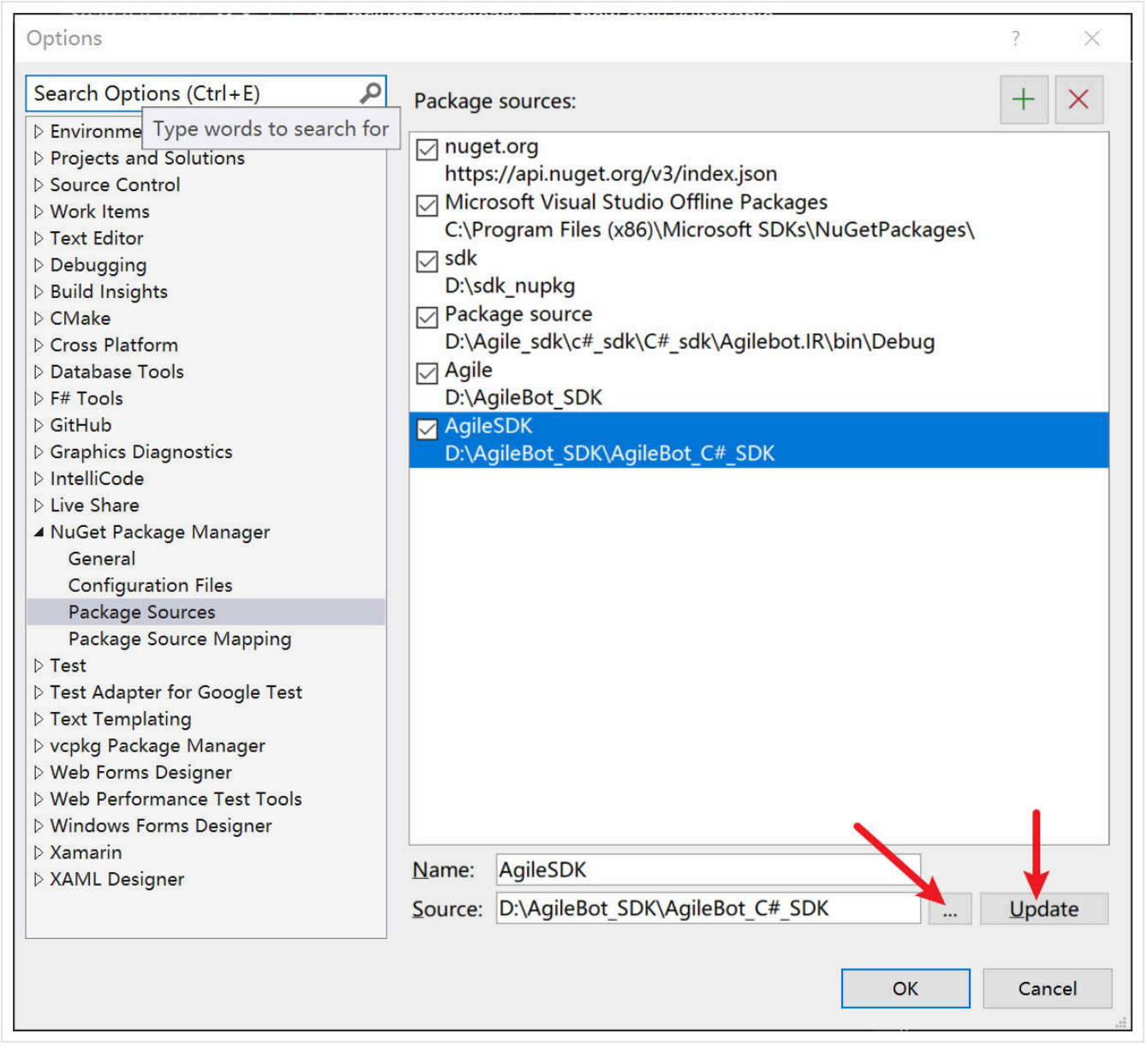


3. Navigate to `Tools > NuGet Package Manager > Package Manager Settings` , then add the directory containing the SDK package as a new package source.

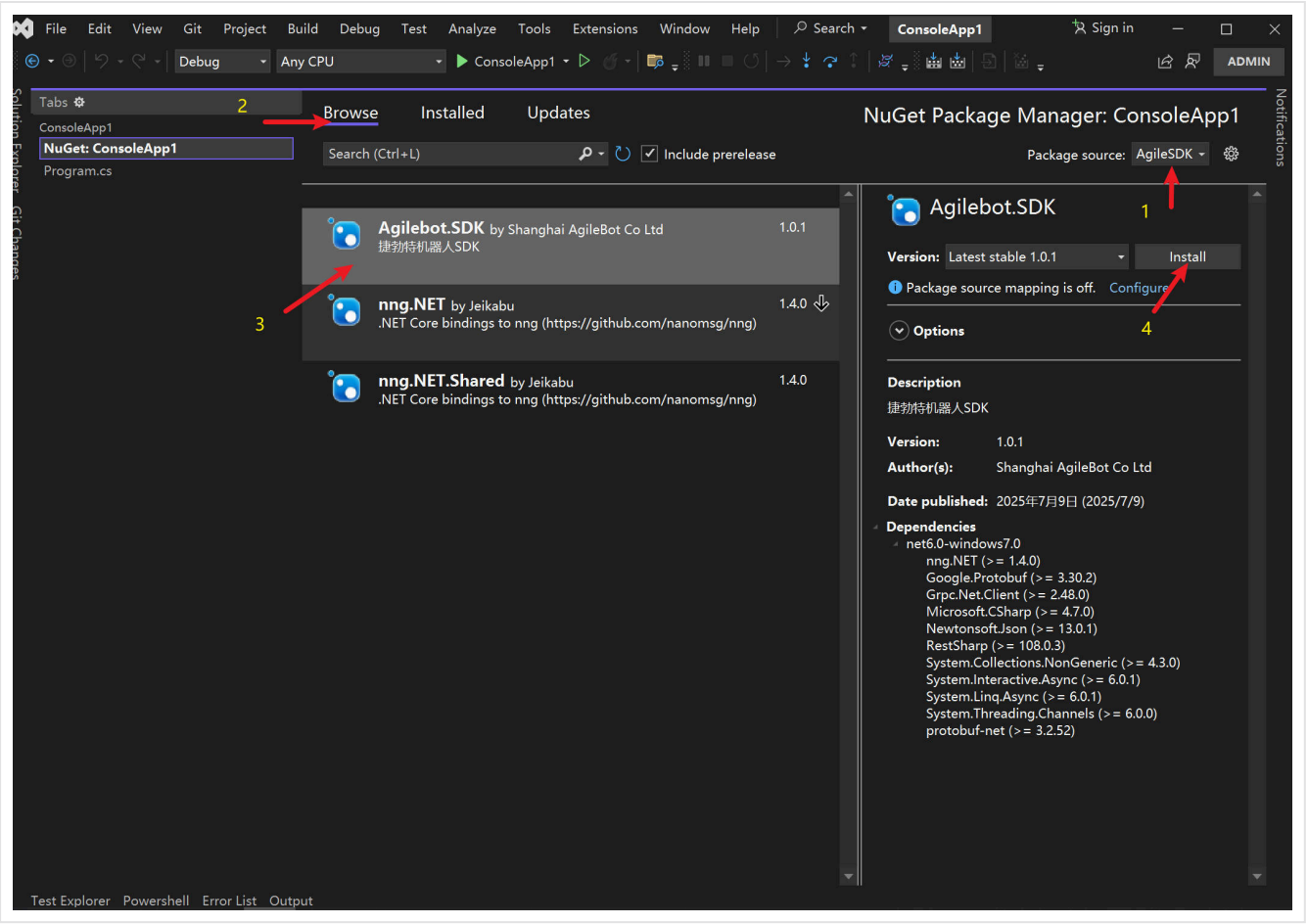








4. Switch the NuGet package source to the newly added entry and install the `Agilebot.SDK` package.



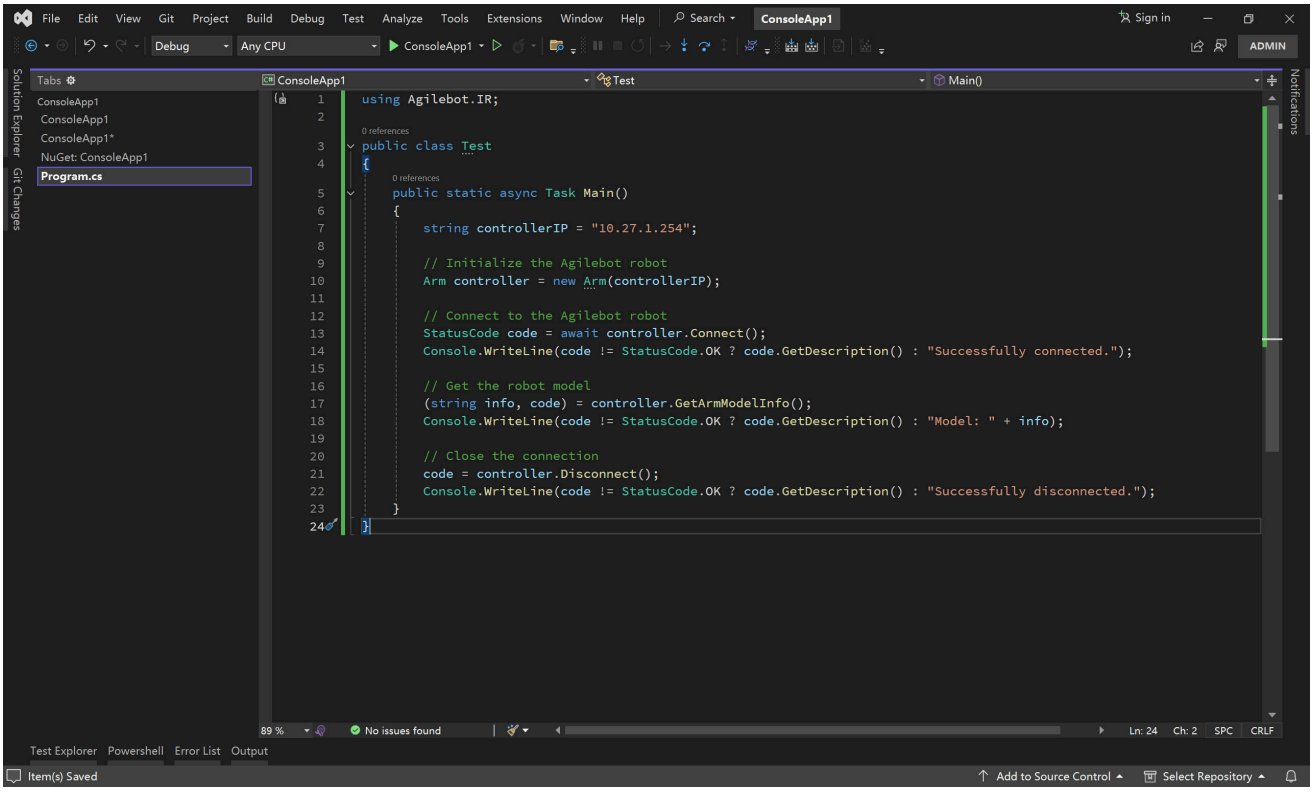
Proxy Files and Troubleshooting

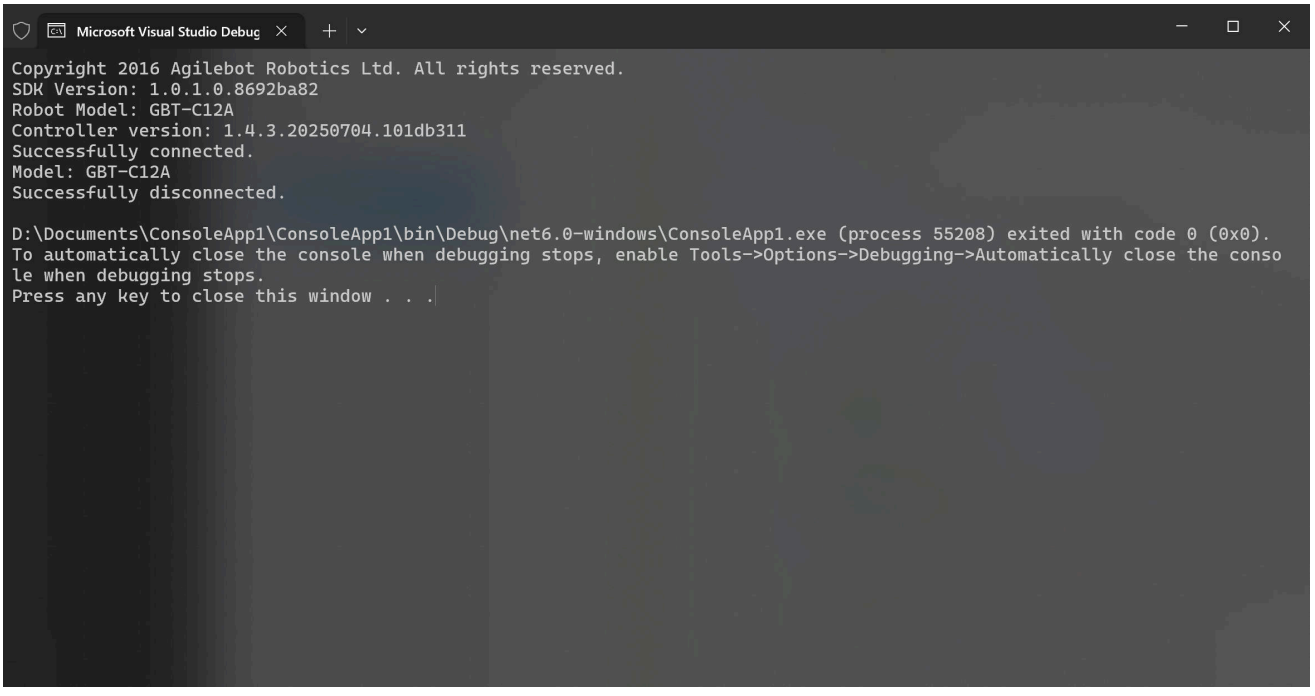
- After installing the SDK, the project automatically gains a `Tools` folder containing `controller_proxy_service_windows_amd64.exe`, which is required when using the local controller proxy. If the executable is missing, copy it manually into both the project folder and the build output directory.
- If the proxy service stays alive because the program exited unexpectedly, open Windows Task Manager, locate **controller_proxy_service_windows_amd64**, and end the process.
- While the proxy service is running, do **not** move the directory where the proxy service is located to another location.

Networking and Debugging Requirements

1. Before running your code, make sure the host PC is connected to the robot network or shares the same LAN as the robot.

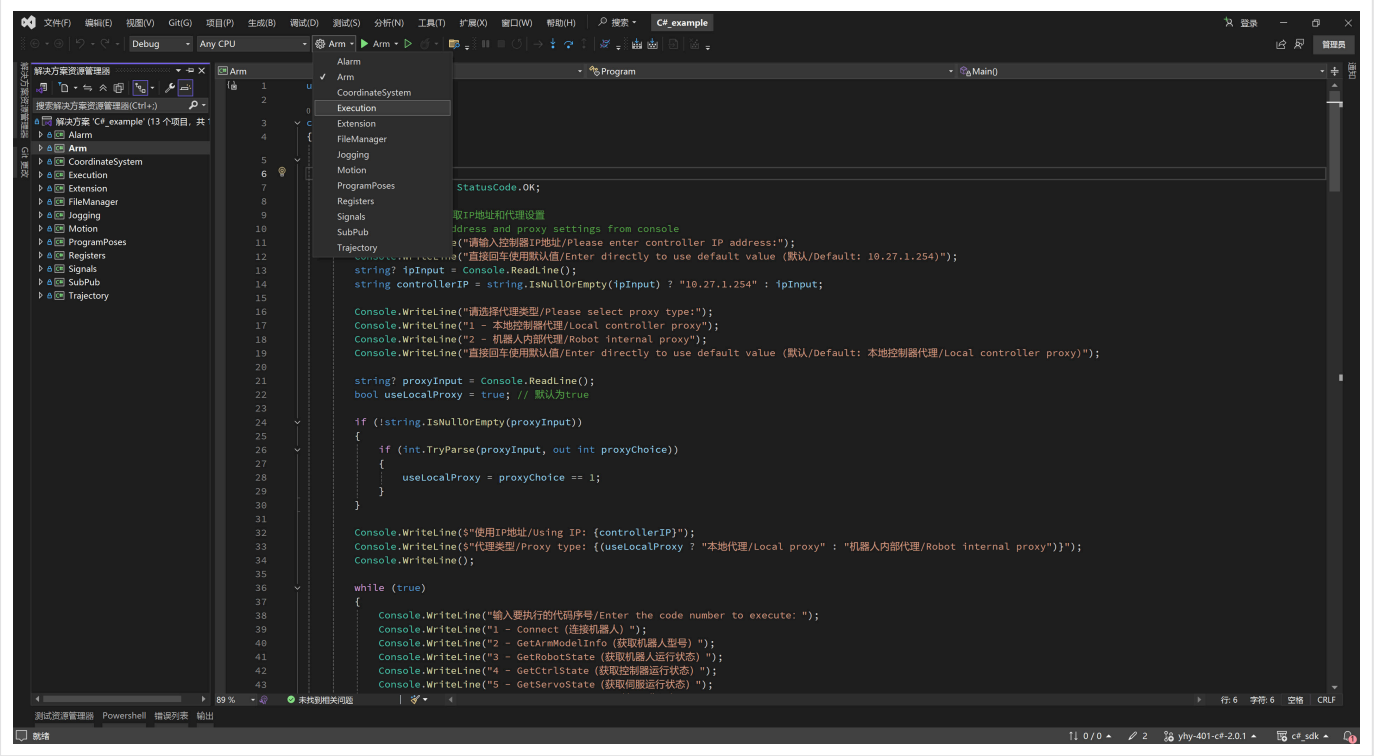
2. Keep the network stable during debugging to prevent the proxy service from dropping unexpectedly.





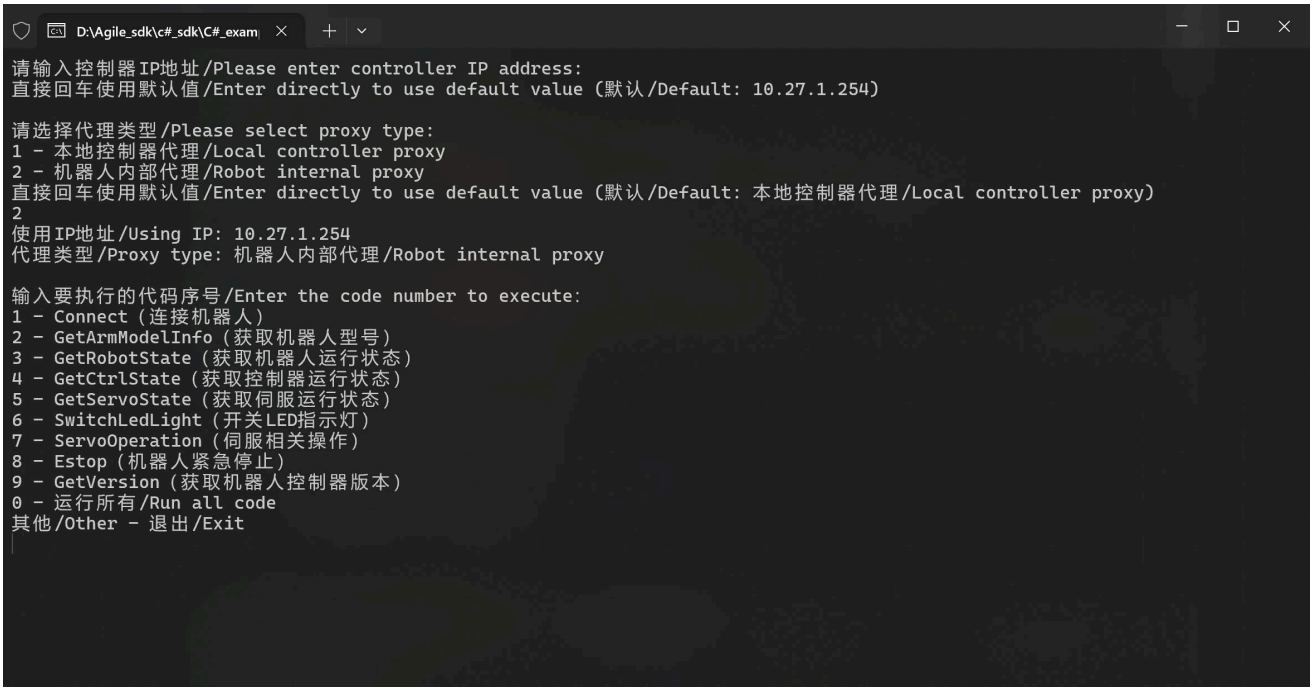
1.3 Example Program Usage

This chapter walks through the `C#_example` project that ships with the SDK. By switching the startup item you can quickly try out each major SDK class.



Run the Example

1. Open and run `C#_example` ; a console window appears automatically.
2. Enter the robot IP address when prompted.
3. Choose where to host the proxy service (robot controller or local PC).
4. Click **Start** to load the selected example.



Proxy Types

- **Robot Internal Proxy**: Uses the proxy built into the robot controller. Recommended for controller firmware **v7.7.0.0 or newer**.
- **Local Controller Proxy**: Uses the lightweight proxy shipped with the SDK and runs on the host PC. This is the only option when the controller firmware is **below v7.7.0.0**.
- For **Airbot** robots only the Robot Internal Proxy is supported; the local proxy cannot reach Airbot controllers.

2 Glossary

Term	Description
teach pendant	The pendant attached to the robot, used for teaching and controlling the robot
SDK	Software Development Kit, used for programming and controlling the robot
robot network	The network connection between the robot and the external computer
controller	The control unit of the robot, responsible for executing motion commands, processing sensor data, and managing robot status
robotic arm	The main moving part of the robot, consisting of multiple joints and links
servo system	The motor drive system that controls robot joint motion, providing precise position and speed control
teaching	The process of recording robot motion trajectories and actions through manual operation of the robot or teach pendant
joint	The movable component connecting various links in the robot arm, each joint corresponding to one degree of freedom
Cartesian coordinates	A three-dimensional coordinate system based on three mutually perpendicular X, Y, Z axes, used to describe the robot's position and orientation in space
pose	The combination of the robot's position and orientation in space, including position coordinates and rotation angles
trajectory	The path of the robot's end effector moving in space, usually composed of a series of pose points
payload	The weight and objects carried by the robot's end effector, affecting the robot's motion performance and accuracy
coordinate system	A reference system used to describe robot position and orientation, including base coordinate system, tool coordinate system, user coordinate system, etc.
OVC	Overall Velocity Control, used to set the overall motion speed multiplier of the robot

Term	Description
OAC	Overall Acceleration Control, used to set the overall acceleration multiplier of the robot
TF	Tool Frame, a coordinate system with the robot's end tool as the origin
UF	User Frame, a user-defined coordinate system for convenient programming and positioning
TCS	Teach Coordinate System, a coordinate reference system used during teaching
DH parameters	Denavit-Hartenberg parameters, standard parameters used to describe the geometric relationships of robot links
PR register	Pose Register, a register used to store robot pose information
MR register	Motion Register, a register used to store motion-related parameters
SR register	String Register, a register used to store string information
R register	Real Register, a register used to store numerical information
MH register	Modbus Holding Register, a holding register for Modbus communication
MI register	Modbus Input Register, an input register for Modbus communication
BAS	Basic Script, a high-level programming language used to write robot control programs
Scara	Selective Compliance Assembly Robot Arm, a type of four-axis industrial robot
collaborative robot	A robot capable of safe collaboration with humans, usually equipped with force sensing and collision detection capabilities
industrial robot	A robot used for industrial automation production, usually with high precision, high speed, and high load capacity
Copper	The codename for Agilebot's collaborative robot product line
Bronze	The codename for Agilebot's industrial robot product line

3 Data Structures

3.1 StatusCode

Description

Status codes returned by the interface.

Import

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

Fields

Name	Enum Value	Description
OK	0	Execution successful
INCOMPATIBLE_VERSION	-1	Incompatible version
TIMEOUT	-3	Connection timeout
INTERFACE_NOT_IMPLEMENTED	-4	Interface not implemented
INDEX_OUT_OF_RANGE	-5	Index out of range
UNSUPPORTED_FILETYPE	-6	Unsupported file type
UNSUPPORTED_PARAMETER	-7	Unsupported robot parameter
UNSUPPORTED_SIGNALTYPE	-8	Unsupported IO signal type
PROGRAM_NOT_FOUND	-9	Program not found
PROGRAM_POSE_NOT_FOUND	-10	Program pose information not found

Name	Enum Value	Description
WRITE_PROGRAM_FAILED	-11	Failed to update program pose information
GET_ALARM_CODE_FAILED	-12	Failed to access alarm service to get alarm code
WRONG_POSITION_INFO	-13	Controller returns incorrect position information
UNSUPPORTED_TRA_TYPE	-14	Unsupported motion type
FILE_NOT_FOUND	-15	File or folder not found
FILE_ALREADY_EXIST	-16	File already exists
INVALID_PARAMETER	-27	Invalid parameter
GET_ALARM_DESC_FAILED	-17	Failed to get alarm information based on alarm code
RESET_ALARM_ERRORS_FAILED	-18	Failed to reset alarm information
GET_ALL_ALARMS_FAILED	-19	Failed to get all alarm information
WRONG_DATA_FORMAT	-20	Incorrect data format received
CONNECT_FAILED	-21	Initialization connection failed, please check IP address or control cabinet service
POSE_INDEX_DUPLICATED	-23	Pose index duplicated
CONTROLLER_ERROR	-254	Controller error, please contact the developer
OTHER_REASON	-255	Other reasons

3.2 RobotState

Description

Robot operation status.

Import

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

Fields

Name	Enum Value	Description
WRONG_DATA	-1	Unknown state
ROBOT_IDLE	0	Robot idle
ROBOT_RUNNING	1	Robot running
ROBOT_TEACHING	2	Robot teaching
ROBOT_IDLE_TO_RUNNING	101	Robot intermediate state, idle to running
ROBOT_IDLE_TO_TEACHING	102	Robot intermediate state, idle to teaching
ROBOT_RUNNING_TO_IDLE	103	Robot intermediate state, running to idle
ROBOT_TEACHING_TO_IDLE	104	Robot intermediate state, teaching to idle

3.3 CtrlState

Description

Controller operation status.

Import

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

Fields

Name	Enum Value	Description
WRONG_DATA	-1	Unknown controller state
CTRL_INIT	0	Controller initializing
CTRL_ENGAGED	1	Controller enabled
CTRL_ESTOP	2	Controller emergency stop
CTRL_TERMINATED	3	Controller terminated
CTRL_ANY_TO_ESTOP	101	Controller intermediate state, any to emergency stop
CTRL_ESTOP_TO_ENGAGED	102	Controller intermediate state, emergency stop to enabled
CTRL_ESTOP_TO_TERMINATED	103	Controller intermediate state, emergency stop to terminated

3.4 ServoState

Description

Servo controller status.

Import

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

Fields

Name	Enum Value	Description
WRONG_DATA	-1	Unknown servo controller state

Name	Enum Value	Description
SERVO_IDLE	1	Servo controller idle
SERVO_RUNNING	2	Servo controller running
SERVO_DISABLE	3	Servo controller disabled
SERVO_WAIT_READY	4	Servo controller waiting for ready
SERVO_WAIT_DOWN	5	Servo controller waiting for shutdown
SERVO_INIT	10	Servo controller initializing

3.5 TransformStatusEnum

Description

Enum for offline trajectory file conversion status.

Import

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

Fields

Name	Enum Value	Description
TRANSFORM_START	0	Conversion task started
TRANSFORM_RUNNING	1	Conversion task in progress
TRANSFORM_SUCCESS	2	Conversion task completed successfully
TRANSFORM_FAILED	3	Conversion task failed
TRANSFORM_NOT_FOUND	4	Conversion task not found

Name	Enum Value	Description
TRANSFORM_UNKNOWN	-1	Unknown conversion task status

3.6 PayloadInfo

Description

The PayloadInfo class is used to store the robot's payload information, including payload ID, weight, center of mass, and moment of inertia. This information is crucial for kinematic and dynamic analysis of the robot under load conditions, especially for path planning and torque calculation.

Import

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

Properties

Property	Type	Description
Id	uint	Payload ID, used to uniquely identify different payload configurations
Comment	string	Comment, used to describe additional information about the payload
Weight	double	Payload weight (unit: kilograms)
MassCenter	MassCenter	Payload center of mass (X, Y, Z coordinates)
InertiaMoment	InertiaMoment	Payload moment of inertia (LX, LY, LZ)

Example

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

3.6.1 MassCenter

Description

The `MassCenter` class is used to represent the center of mass of the payload, containing the X, Y, and Z coordinates. The center of mass is the geometric center of the payload in space and is important for robot motion control and torque calculation.

Import

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

Properties

Property	Type	Description
`X`	`double`	X-coordinate of the center of mass (unit: millimeters)
`Y`	`double`	Y-coordinate of the center of mass (unit: millimeters)
`Z`	`double`	Z-coordinate of the center of mass (unit: millimeters)

3.6.2 InertiaMoment

Description

The `InertiaMoment` class is used to represent the moment of inertia of the payload, containing the LX, LY, and LZ components. The moment of inertia represents the payload's resistance to

rotational changes and is important for robot dynamics analysis and control.

Import

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

Properties

Property	Type	Description
LX	double	X-component of the moment of inertia (unit: kilograms·millimeters²)
LY	double	Y-component of the moment of inertia (unit: kilograms·millimeters²)
LZ	double	Z-component of the moment of inertia (unit: kilograms·millimeters²)

3.7 TransformState

Description

Enum for offline trajectory file conversion status.

Import

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

Fields

Enum Value	Value	Description
TRANSFORM_START	0	Conversion task started
TRANSFORM_RUNNING	1	Conversion task in progress
TRANSFORM_SUCCESS	2	Conversion task completed successfully

Enum Value	Value	Description
TRANSFORM_FAILED	3	Conversion task failed
TRANSFORM_NOT_FOUND	4	Conversion task not found
TRANSFORM_UNKNOWN	-1	Data error, unknown status

3.8 TCSType

Description

TCS coordinate system type.

Import

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

Fields

Name	Enum Value	Description
WRONG_TYPE	-1	Incorrect type
JOINT	0	Joint space
BASE	1	Base coordinate system
WORLD	2	World coordinate system
USER	3	User coordinate system
TOOL	4	Tool coordinate system
RTCP_USER	5	RTCP user coordinate system
RTCP_TOOL	6	RTCP tool coordinate system

3.9 MotionPose

Description

Describes the robot's position structure. In the coordinate data, the distance in the XYZ direction is measured in millimeters (mm), and the angle data is measured in degrees (°). In some versions, the angle information is in radians; see the function list return result description for details.

Import

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

Properties

Property	Type	Description
CartData	BaseCartData	Cartesian data
Joint	Joint	Joint data
Pt	PoseType	Position type, defaults to Unknown

Example

```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

Description

Describes the robot's position and posture information in the Cartesian coordinate system. The spatial coordinates are measured in millimeters (mm), and the posture information includes wrist and arm postures as well as the rotation counts of each axis.

Import

c#

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

Properties

Property	Type	Description
Position	Position	Robot's spatial coordinates (X, Y, Z, A, B, C)

Property	Type	Description
Posture	Posture	Robot's posture information (wrist, arm posture, and axis rotation counts)

Example

```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

Description

Describes the robot's position and rotation angle coordinates in the Cartesian coordinate system. The distance in the X, Y, Z directions is measured in millimeters (mm), and the angles in the A, B, C directions are measured in degrees (°).

Import

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

Properties

Property	Type	Description
X	double	Distance in the X direction of the Cartesian coordinate system (unit: millimeters)
Y	double	Distance in the Y direction of the Cartesian coordinate system (unit: millimeters)

Property	Type	Description
Z	double	Distance in the Z direction of the Cartesian coordinate system (unit: millimeters)
A	double	Angle in the A direction of the Cartesian coordinate system (unit: degrees)
B	double	Angle in the B direction of the Cartesian coordinate system (unit: degrees)
C	double	Angle in the C direction of the Cartesian coordinate system (unit: degrees)

Example

```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

Description

Describes the robot's posture information, including wrist and arm postures as well as the rotation counts of each axis. Posture information is used to define the robot's specific posture in space.

Import

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

Properties

Property	Type	Description
WristFlip	int	Wrist flip posture. Range: -1, 0, 1. For a 6-axis robot J5 joint config: 1 = wrist flipped down, -1 = wrist flipped up.
ArmUpDown	int	Arm up/down posture. Range: -1, 0, 1. For a 6-axis robot J3 joint config: 1 = arm above (forward condition: joint-3 above the line from joint-4 to joint-2 and joint-3 angle < 0), -1 = arm below (joint-3 angle > 0).
ArmBackFront	int	Arm front/back posture. Range: -1, 0, 1. For a 6-axis robot J1 joint config: 1 = arm in front (collaborative robot facing forward, joint-2 on the left side of joint-1), -1 = arm behind (joint-2 on the right side of joint-1).
ArmLeftRight	int	Arm left/right posture. Range: -1, 0, 1. For a 4-axis SCARA robot J2 joint config: 1 = SCARA arm on the right, -1 = SCARA arm on the left.
TurnCircle	List<int>	Multi-turn counts for each axis. Range: -1, 0, 1. When the axis is at 0°, turn count = 0. During linear or circular moves the controller auto-selects the turn count closest to the start pose, so the final value may differ from the taught posture. For axes 1, 4, 5, 6: ≥ 180° → value ≥ 1; -179.99° ~ 179.99° → 0; ≤ -180° → value ≤ -1.

Example

```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

Description

Describes the angle data of each robot joint. Each joint angle value is used to define the robot's specific position in joint space. The angle unit is typically degrees (°), but the specific unit should be confirmed based on the actual robot system.

Import

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

Properties

Property	Type	Description
J1	double	Angle of the robot's first joint
J2	double	Angle of the robot's second joint
J3	double	Angle of the robot's third joint
J4	double	Angle of the robot's fourth joint
J5	double	Angle of the robot's fifth joint
J6	double	Angle of the robot's sixth joint
J7	double	Angle of the robot's seventh joint
J8	double	Angle of the robot's eighth joint
J9	double	Angle of the robot's ninth joint

Example

```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());
```

Notes

- The unit of joint angles is typically degrees (°), but some robot systems may use radians (rad). Please confirm the unit based on the actual robot system documentation.
- The range of joint angles is usually limited by the robot hardware. Exceeding the range may cause errors or damage the equipment.

3.12 PoseType

Description

Defines the type of robot pose data, used to distinguish whether the data is joint angle data, Cartesian space coordinates, or unknown type. This enum is used to identify the format of robot pose data so that different types of data can be correctly processed in the program.

Import

c#

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

Enum Values

Enum Value	Description
Unknown	Unknown type, indicating the pose data type is not defined
Joint	Joint angle data type, indicating the data is joint angles

Enum Value	Description
`Cart`	Cartesian space coordinate data type, indicating the data is Cartesian coordinates

3.13 DHparam

Description

The `DHparam` class is used to describe the robot link parameters based on the Denavit-Hartenberg parameters (D-H parameters). These parameters are used to define the geometric relationships between robot joints and are the basis for robot kinematics and dynamics analysis.

Import

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

Properties

Property	Type	Description
`id`	`uint`	Unique identifier for the link, used to distinguish different links
`a`	`double`	Link length, representing the axial distance between adjacent joints (unit: millimeters)
`alpha`	`double`	Link twist angle, representing the angle between adjacent joint axes (unit: degrees or radians)
`d`	`double`	Joint distance, representing the distance along the current joint axis to the next joint (unit: millimeters)
`offset`	`double`	Joint angle offset, representing the initial angle offset of the joint (unit: degrees or radians)

Constructor

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

Notes

- **Unit consistency**: The units of `a` and `d` should be consistent (typically millimeters), and the units of `alpha` and `offset` should also be consistent (typically degrees or radians).
- **Angle unit**: In some robot systems, the angle unit may be radians instead of degrees. Please confirm and unify the units based on actual requirements.
- **D-H parameter definition**: The definition of D-H parameters depends on the specific robot model and coordinate system conventions. When using the `DHparam` class, ensure that the parameter definitions are consistent with the robot's actual geometric structure.

3.14 CartStatus

Description

The `CartStatus` class is used to represent the status of each axis in the Cartesian coordinate system. The status of each axis is represented by a boolean value, with `true` indicating the axis is available and `false` indicating it is not. This status class is commonly used in robot motion control to determine whether a particular axis can function properly.

Import

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

Properties

Property	Type	Description
`X`	`bool`	Status of the X direction, defaults to `true` (available)

Property	Type	Description
`Y`	`bool`	Status of the Y direction, defaults to `true` (available)
`Z`	`bool`	Status of the Z direction, defaults to `true` (available)
`A`	`bool`	Status of the A direction, defaults to `true` (available)
`B`	`bool`	Status of the B direction, defaults to `true` (available)
`C`	`bool`	Status of the C direction, defaults to `true` (available)

3.15 JointStatus

Description

The `JointStatus` class is used to represent the status of each robot joint. The status of each joint is represented by a boolean value, with `true` indicating the joint is available and `false` indicating it is not. This status class is commonly used in robot motion control to determine whether a particular joint can function properly.

Import

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

Properties

Property	Type	Description
`J1`	`bool`	Status of Joint 1, defaults to `true` (available)
`J2`	`bool`	Status of Joint 2, defaults to `true` (available)
`J3`	`bool`	Status of Joint 3, defaults to `true` (available)
`J4`	`bool`	Status of Joint 4, defaults to `true` (available)

Property	Type	Description
J5	bool	Status of Joint 5, defaults to true (available)
J6	bool	Status of Joint 6, defaults to true (available)
J7	bool	Status of Joint 7, defaults to true (available)
J8	bool	Status of Joint 8, defaults to true (available)
J9	bool	Status of Joint 9, defaults to true (available)

3.16 DragStatus

Description

The DragStatus class is used to represent the drag status of the robot arm, including the status of the Cartesian coordinate system and the joints. Additionally, it includes a flag IsContinuousDrag to indicate whether the robot is in continuous drag mode. This status class is commonly used in robot drag control to determine the current drag mode and the status of each axis/joint.

Import

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

Properties

Property	Type	Description
CartStatus	CartStatus	Status of the Cartesian coordinate system
JointStatus	JointStatus	Status of the joints

Property	Type	Description
`IsContinuousDrag`	`bool`	Whether the robot is in continuous drag mode, defaults to `false`

Constructor

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

- Initializes `CartStatus` and `JointStatus`, and sets `IsContinuousDrag` to `false`.

Example

```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

Description

The `ProgramPose` class is used to represent a pose (position and orientation) in a program, which can be joint coordinates or Cartesian coordinates. This class includes a unique identifier for the pose, data (joint or Cartesian coordinate information), name, and comment. This class facilitates the management and manipulation of pose information in robot programs.

Import

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

Properties

Property	Type	Description
`Id`	`int`	Unique identifier for the pose
`PoseData`	`ProgramPoseData`	Pose data, including joint or Cartesian coordinate information
`Name`	`string`	Name of the pose
`Comment`	`string`	Comment for the pose

Constructor

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

- Initializes `Id` , `PoseData` , `Name` , and `Comment` .

Example

```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

Description

The `ProgramPoseData` class is used to represent pose data in a program, including Cartesian space coordinates and posture information, joint angle information, and pose type. This class facilitates the storage and management of specific pose data.

Import

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

Properties

Property	Type	Description
`CartData`	`ProgramCartData`	Cartesian data
`Joint`	`Joint`	Joint data
`Pt`	`PoseType`	Pose type, defaults to Unknown

3.17.2 ProgramCartData

Description

The `ProgramCartData` class is used to represent the Cartesian coordinate system data in a program. It references the `BaseCartData` class to include spatial coordinates and posture information, and determines the coordinate system type through the values of `Uf` and `Tf` . `Uf` represents the User Frame, and `Tf` represents the Tool Frame. If the values of `Uf` and `Tf` are `-1` , it indicates the use of the system's default coordinate system. This class is used in robot programming to define and manage pose information in Cartesian space.

Import

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

Properties

Property	Type	Description
`BaseCart`	`BaseCartData`	Robot's Cartesian position and posture information
`Uf`	`int`	User Frame, `-1` indicates the use of the system's coordinate system
`Tf`	`int`	Tool Frame, `-1` indicates the use of the system's coordinate system

3.18 FileType

Description

The `FileType` enum is used to define the types of files allowed for upload. It distinguishes between different types of robot program files based on their source and format. This enum is used in the robot programming environment for file management, upload, and program parsing, helping the system correctly identify and process different types of program files.

Import

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

Enum Values

Enum Value	Description
`UserProgram`	Program files generated by the user through point selection, each program includes `.xml` and `.json` files.
`BlockProgram`	Program files generated by the user through block programming, each program includes `.block` , `.xml` , and `.json` files.
`TrajectoryProgram`	Offline trajectory program files, typically used for path planning in offline programming.

3.19 SignalType

Description

The `SignalType` enum is used to define the types of signals supported in the robot system. It distinguishes between various digital and analog signals based on their purpose and source. This enum is used in the robot control system for signal configuration, signal processing, and logic judgment, helping the system accurately identify and manage different types of signals.

Import

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

Enum Values

Enum Value	Description
DI	Digital Input, used to receive external digital signals.
DO	Digital Output, used to control external devices or actuators.
RI	Robot Input, used to receive digital signals from the robot's wrist.
RO	Robot Output, used to control actuators on the robot's wrist.
UI	User Input, used to receive user-defined digital signals.
UO	User Output, used to output user-defined digital signals.
TDI	Tool Digital Input, used to receive digital signals from the tool end.
TDO	Tool Digital Output, used to control actuators on the tool end.
GI	Group Input, used to receive a combination of digital signals.
GO	Group Output, used to output a combination of digital signals.
AI	Analog Input, used to receive continuous analog signals.
AO	Analog Output, used to output continuous analog signals.
TAI	Tool Analog Input, used to receive analog signals from the tool end.

3.20 PoseRegister

Description

The `PoseRegister` class is used to represent a pose (position and orientation) in a PR register, which can be joint coordinates or Cartesian coordinates. This class includes a unique identifier for the pose, data (joint or Cartesian coordinate information), name, and comment. This class facilitates the management and manipulation of pose information in robot programs.

Import

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

Properties

Property	Type	Description
`Id`	`int`	Unique identifier for the pose
`PoseData`	`PoseRegisterData`	Pose data, including joint or Cartesian coordinate information
`Name`	`string`	Name of the pose
`Comment`	`string`	Comment for the pose

Constructor

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

- Initializes `Id` , `PoseData` , `Name` , and `Comment` .

Example

```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

Description

The `PoseRegisterData` class is used to represent pose data in a PR register, including Cartesian space coordinates and posture information, joint angle information, and pose type. This class facilitates the storage and management of specific pose data.

Import

c#

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

Properties

Property	Type	Description
`CartData`	`BaseCartData`	Cartesian data
`Joint`	`Joint`	Joint data
`Pt`	`PoseType`	Pose type, defaults to Unknown

3.22 Coordinate

Description

The `Coordinate` class is used to represent a coordinate system in the robot system. It includes basic information about the coordinate system, such as a unique identifier (ID), name, comment, motion group number, and specific pose data. This class is used in robot programming and control systems to define and manage the position and orientation of coordinate systems, facilitating motion planning and path control in programs.

Import

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

Properties

Property	Type	Description
Id	int	Unique identifier for the coordinate system
Name	string	Name of the coordinate system, used to identify and describe the coordinate system
Comment	string	Comment for the coordinate system, used to further explain the purpose or characteristics of the coordinate system
GroupId	int	Motion group number to which the coordinate system belongs, used for classification and management of coordinate systems
Data	Position	Specific pose data of the coordinate system, including position and orientation information

Example

```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

Description

The `CoordinateType` enum is used to define the type of coordinate system. It distinguishes between user coordinate systems and tool coordinate systems. This enum is used in robot programming and control systems to clearly specify the purpose of the coordinate system, helping the system correctly handle operations related to coordinate systems.

Import

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

Enum Values

Enum Value	Description
`UserCoordinate`	User coordinate system, used to define user-defined coordinate systems.
`ToolCoordinate`	Tool coordinate system, used to define the coordinate system of tools (e.g., end effectors).

3.22.2 CoordSummary

Description

The `CoordSummary` class is used to represent the summary information of a coordinate system. It includes the type, unique identifier, name, comment, and group ID of the coordinate system. This class is used in the robot programming environment to manage and store metadata of coordinate systems, facilitating quick access and manipulation of coordinate systems in programs.

Import

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

Properties

Property	Type	Description
Type	CoordinateType	Type of the coordinate system, which can be a user coordinate system or a tool coordinate system
Id	int	Unique identifier for the coordinate system
Name	string	Name of the coordinate system
Comment	string	Comment for the coordinate system, used to describe its purpose or characteristics
GroupId	int	Group ID to which the coordinate system belongs, used for classification and management of coordinate systems

Example

```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

Description

The RobotTopicType enum is used by SubPub.SubscribeStatus to specify robot status subscription topics.

Import

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

Fields

Name	Description
TopicCurrentJoint	Publishes robot joint status feedback
TopicCurrentCartesian	Publishes current TCP Cartesian coordinates
TopicUF	Publishes current user frame information
TopicTF	Publishes current tool frame information
TopicVelocity	Publishes global velocity ratio
TopicRunningStatus	Publishes controller running status
TopicInterpreterStatus	Publishes interpreter status
TopicRobotStatus	Publishes robot status
TopicCtrlStatus	Publishes controller status
TopicServoStatus	Publishes servo controller status
TopicTrajectoryRecordsStatus	Trajectory playback recording status
UserOpMode	User operation mode

3.24 RegTopicType

Description

The RegTopicType enum is used by SubPub.SubscribeRegister to specify the register subscription type.

Import

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

Fields

Name	Description
R	R register topic
MR	MR register topic
SR	SR register topic
PR	PR register topic

3.25 IOTopicType

Description

The `IOTopicType` enum is used by `SubPub.SubscribeIO` to specify the I/O subscription type.

Import

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

Fields

Name	Description
DI	Digital input topic
DO	Digital output topic
GI	Group digital input topic
GO	Group digital output topic

Name	Description
RI	Robot input topic
RO	Robot output topic
UI	User input topic
UO	User output topic
TDI	Tool digital input topic
TDO	Tool digital output topic
TAI	Tool analog input topic
AI	Analog input topic
AO	Analog output topic

4 Methods and Examples

4.1 Basic Operations of the Robot

Overview

The `Arm` class encapsulates most high-frequency interfaces for Agilebot robots, including connection management, state query, and control command dispatch. Typical workflow:

1. Instantiate `Arm(controllerIP, teachPanelIP, localProxy)` .
2. Call `ConnectSync()` to establish communication with the controller/teach pendant.
3. Call motion, status, I/O, and other interfaces based on your scenario.
4. Call `DisconnectSync()` to release resources.

After instantiation, the class handles internal setup automatically:

- SDK version check
- Controller type identification
- Automatic proxy service selection
- Communication path logging in the log

Notes

- When robot software version is below 7.7.0, keep `localProxy=true` and ensure local communication capability on the PC.
- For industrial robots in PC mode, ensure teach pendant control permission is obtained after connection and released after operations.

Class Constructor

Method Name	Arm(string `controllerIP` , string `teachPanelIP` = null, bool `localProxy` = true)
Description	Agilebot robot class constructor, which includes all available robot control interfaces. The robot must be initialized and connected before other functions can be used.

Method Name	Arm(string `controllerIP` , string `teachPanelIP` = null, bool `localProxy` = true)
Request Parameters	`controllerIP` : string Robot controller IP address `teachPanelIP` : string Optional teach pendant IP; falls back to `controllerIP` when omitted `localProxy` : bool Whether to use local controller proxy service, default is true. When true, launches controller proxy service locally; when false, requires proxy service to be already installed in robot controller (requires robot software version 7.7 or later).
Return Value	StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.1.1 Connecting to the Robot

Method Name	ConnectSync()
Description	Establishes network connection with the Agilebot robot. The Arm constructor must be called first to initialize the robot instance. This method will start the corresponding proxy service based on the proxy mode specified in the constructor.
Request Parameters	None
Return Value	StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.1.2 Checking the Connection with the Robot Arm

Method Name	IsConnected()
Description	Checks whether the network connection with the robot is valid. Returns `true` if the connection is valid and communication is possible; returns `false` if the connection is disconnected or not established.
Request Parameters	None
Return Value	bool: Connection status, true indicates connection is valid, false indicates connection is invalid or not connected
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.1.3 Disconnecting from the Robot

Method Name	DisconnectSync()
Description	Disconnects from the Agilebot robot and releases related resources. After disconnection, `ConnectSync()` must be called again to communicate.
Request Parameters	None
Return Value	StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Getting the Current Robot Model

Method Name	GetArmModelInfo()
Description	Gets the model information of the currently connected Agilebot robot. Returns the robot model string (e.g., "GBT-C5A") and the operation execution status.
Request Parameters	None
Return Value	string: Robot model string, e.g., "GBT-C5A" StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Getting the Robot's Operating State

Method Name	GetRobotState()
Description	Gets the current operating state of the Agilebot robot. Returns the robot operating state enum value and the operation execution status.
Request Parameters	None

Method Name	GetRobotState()
Return Value	RobotState: Robot operating state enum value StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Getting the Current Controller Operating State

Method Name	GetCtrlState()
Description	Gets the current operating state of the Agilebot robot controller. Returns the controller operating state enum value and the operation execution status.
Request Parameters	None
Return Value	CtrlState: Controller operating state enum value StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

Arm/GetCtrlState.cs

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 Getting the Current Servo State

Method Name	GetServoState()
Description	Gets the current state of the Agilebot robot servo system. Returns the servo system state enum value and the operation execution status.
Request Parameters	None
Return Value	ServoState: Servo system state enum value StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

Arm/GetServoState.cs

cs

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

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

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

        try
        {
            // [ZH] 获取伺服运行状态
            // [EN] Get the servo operating state
            (ServoState state, code) =
                controller.GetServoState();
            if (code != StatusCode.OK)
            {
                Console.WriteLine(
                    "Get ServoState Failed: "
                        + code.GetDescription()
                );
```

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

        return code;
    }
}
```

4.1.8 Getting the Robot Controller Version

Method Name	GetVersion()
Description	Gets the software version information of the Agilebot robot controller. Returns the controller software version string and the operation execution status.
Request Parameters	None
Return Value	string: Controller software version string StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Setting the Robot's LED Indicator Light

Method Name	SwitchLedLight(bool `mode`)
Description	Controls the on/off state of the Agilebot robot LED indicator light. `true` turns on the indicator light, `false` turns it off.
Request Parameters	`mode` : bool LED indicator light control mode, true indicates turn on, false indicates turn off
Return Value	StatusCode: Operation execution result
Compatibility	Only supports collaborative robots, requires controller version 1.3.6 and above, industrial robots not supported
Compatible robot software version	Collaborative (Copper): v7.5.1.3+ Industrial (Bronze): Not supported

Example Code

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 Robot Servo On

Method Name	ServoOn()
Description	Starts the servo system of the Agilebot robot, making the robot enter a controllable state. After servo startup, the robot can accept motion commands.
Request Parameters	None
Return Value	StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.1.11 Robot Servo Off

Method Name	ServoOff()
Description	Turns off the servo system of the Agilebot robot, making the robot enter a safe stop state. After servo shutdown, the robot will not be able to move.
Request Parameters	None
Return Value	StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.1.12 Resetting the Robot Servo

Method Name	ServoReset()
Description	Resets the servo system of the Agilebot robot, clearing error states and preparing for restart. This method is typically called after a servo failure to restore the system.

Method Name	ServoReset()
Request Parameters	None
Return Value	StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Emergency Stop

Method Name	Estop()
Description	Executes emergency stop of the Agilebot robot, immediately stopping all motion and entering a safe state. After emergency stop, the servo system must be restarted to resume motion.
Request Parameters	None
Return Value	StatusCode: Emergency stop operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

Arm/Estop.cs

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 Robot Motion Control and Status

Overview

The `Motion` class is the core object for robot motion control. It encapsulates:

- Velocity/acceleration parameter management
- Coordinate system management
- Pose conversion
- Trajectory motion control
- Drag teaching
- Realtime control
- Payload management

Common Workflow

After `Arm` is connected, obtain the motion instance through `arm.Motion` . No separate initialization is required.

4.2.1 Getting Robot Parameters

4.2.1.1 Getting OVC Overall Velocity Coefficient

Method Name	Motion.GetOVC()
Description	Gets the current robot's OVC (Overall Velocity Control) global velocity ratio, with a range of 0~1.
Request Parameters	None
Return Value	double: Global velocity ratio value StatusCode: Result of function execution

Method Name	Motion.GetOVC()
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.2.1.2 Getting OAC Overall Acceleration Coefficient

Method Name	Motion.GetOAC()
Description	Gets the current robot's OAC (Overall Acceleration Control) global acceleration ratio, with a range of 0~1.2.
Request Parameters	None
Return Value	double: Global acceleration ratio value StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.2.1.3 Getting the Current TF

Method Name	Motion.GetTF()
Description	Gets the current TF (Tool Frame) tool coordinate system index used by the robot, with a range of 0~10.
Request Parameters	None
Return Value	int: TF tool coordinate system index StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.2.1.4 Getting the Current UF

Method Name	Motion.GetUF()
Description	Gets the current UF (User Frame) user coordinate system index used by the robot, with a range of 0~10.
Request Parameters	None
Return Value	int: UF user coordinate system index StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.2.1.5 Getting the Current TCS Teaching Coordinate System

Method Name	Motion.GetTCS()
Description	Gets the current TCS (Teach Coordinate System) teaching coordinate system type used by the robot, see TCSType for details.
Request Parameters	None
Return Value	TCSType: TCS teaching coordinate system type enum value StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

Motion/GetMotionParameters.cs

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 Setting Robot Parameters

4.2.2.1 Setting OVC Overall Velocity Coefficient

Method Name	Motion.SetOVC(double `value`)
Description	Sets the current robot's OVC (Overall Velocity Control) global velocity ratio.
Request Parameters	`value` : double velocity ratio, range 0~1
Return Value	StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.2.2.2 Setting OAC Overall Acceleration Coefficient

Method Name	Motion.SetOAC(double `value`)
Description	Sets the current robot's OAC (Overall Acceleration Control) global acceleration ratio.
Request Parameters	`value` : double acceleration ratio, range 0.01~1.2
Return Value	StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.2.2.3 Setting the Current TF Tool Coordinate System Index

Method Name	Motion.SetTF(int `value`)
Description	Sets the current TF (Tool Frame) tool coordinate system index.
Request Parameters	`value` : int TF index, range 0~10
Return Value	StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.2.2.4 Setting the Current UF User Coordinate System Index

Method Name	Motion.SetUF(int `value`)
Description	Sets the current UF (User Frame) user coordinate system index.
Request Parameters	`value` : int UF index, range 0~10
Return Value	StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.2.2.5 Setting the Current TCS Teaching Coordinate System

Method Name	Motion.SetTCS(TCSType `value`)
Description	Sets the current TCS (Teach Coordinate System) teaching coordinate system, see TCSType for details.
Request Parameters	`value` : TCSType TCS teaching coordinate system type
Return Value	StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Converting Cartesian Position to Joint Values

Method Name	Motion.ConvertCartToJoint(MotionPose `pose` , int `ufIndex` = 0, int `tfIndex` = 0)
Description	Converts pose data from Cartesian coordinates to joint coordinates.
Request Parameters	`pose` : MotionPose Robot's Cartesian pose (PoseType.CART; SDK automatically solves feasible posture when posture is not specified) `ufIndex` : int User coordinate system index, default is 0 `tfIndex` : int Tool coordinate system index, default is 0
Return Value	MotionPose: Converted robot pose data StatusCode: Conversion operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

Motion/ConvertCartToJoint.cs

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

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

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

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

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

```
                    0,
                    0,
                    0,
                    0,
                },
            };

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

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

        return code;
    }
}
```

4.2.4 Converting Joint Values to Cartesian Position

Method Name	Motion.ConvertJointToCart(MotionPose `pose` , int `ufIndex` = 0, int `tfIndex` = 0)
Description	Converts pose data from joint coordinates to Cartesian coordinates.
Request Parameters	`pose` : MotionPose Robot's joint pose `ufIndex` : int User coordinate system index, default is 0 `tfIndex` : int Tool coordinate system index, default is 0
Return Value	MotionPose: Converted robot pose data StatusCode: Conversion operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Motion

Method Name	Motion.MoveJoint(MotionPose `pose` , double `vel` = 1, double `acc` = 1)
Description	Controls the robot end effector to move to a specified position along the fastest path in joint space.
Request Parameters	`pose` : MotionPose Target position coordinates in Cartesian space or joint coordinate system `vel` : double Motion speed, range 0~1, representing multiple of maximum speed `acc` : double Acceleration, range 0~1.2, representing multiple of maximum acceleration
Return Value	StatusCode: Motion command execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Linear Motion

Method Name	Motion.MoveLine(MotionPose `pose` , double `vel` = 100, double `acc` = 1)
Description	Controls the robot end effector to move to a specified position along a straight line, with the motion trajectory being a straight line between two points.
Request Parameters	`pose` : MotionPose Target position coordinates in Cartesian space or joint coordinate system `vel` : double Motion speed, range 0~5000mm/s, representing robot end effector movement speed `acc` : double Acceleration, range 0~1.2, representing multiple of maximum acceleration
Return Value	StatusCode: Motion command execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Circular Motion

Method Name	Motion.MoveCircle(MotionPose `pose1` , MotionPose `pose2` , double `vel` = 100, double `acc` = 1)
Description	Controls the robot end effector to move to a specified position along a circular trajectory, with the arc determined by the intermediate point and end point.
Request Parameters	`pose1` : MotionPose Robot motion intermediate pose `pose2` : MotionPose Robot motion final pose `vel` : double Motion speed, range 0~5000mm/s, representing robot end effector movement speed `acc` : double Acceleration, range 0~1.2, representing multiple of maximum acceleration
Return Value	StatusCode: Motion command execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Getting Current Pose

Method Name	Motion.GetCurrentPose(PoseType `pt` , int `ufIndex` = 0, int `tfIndex` = 0)
Description	Gets the current pose of the robot, which can be pose information in Cartesian space or joint coordinate system.
Request Parameters	`pt` : PoseType Pose type `ufIndex` : int User coordinate system index (only effective for PoseType.CART; default 0) `tfIndex` : int Tool coordinate system index (only effective for PoseType.CART; default 0)
Return Value	MotionPose: Robot pose data StatusCode: Get operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

Motion/GetCurrentPose.cs

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

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

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

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

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

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

```
                );
                Console.WriteLine(
                    $"姿态: A={cartPose.CartData.Position.A}, B={cartPose.Ca
rtData.Position.B}, C={cartPose.CartData.Position.C}"
                );
            }
            else
            {
                Console.WriteLine(
                    $"获取笛卡尔位姿失败: {code.GetDescription()}"
                );
            }

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

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

        return code;
    }
}
```

4.2.9 Getting DH Parameters

Method Name	Motion.GetDHParam()
Description	Gets the robot's DH (Denavit-Hartenberg) parameters.
Request Parameters	None
Return Value	List<DHparam>: DH parameter list StatusCode: Get operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): Not supported

Example Code

Motion/GetDHParam.cs

cs

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

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

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

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

        try
        {
            // [ZH] 获取机器人的DH参数
            // [EN] Get robot DH parameters
            List<DHparam> dhParamsList;
            (dhParamsList, code) =
                controller.Motion.GetDHParam(1);
            if (code == StatusCode.OK)
            {
                Console.WriteLine("获取DH参数成功:");
```

```
                for (int i = 0; i < dhParamsList.Count; i++)
                {
                    var dh = dhParamsList[i];
                    Console.WriteLine(
                        $"轴{i + 1}: Alpha={dh.alpha}, A={dh.a}, D={dh.d}, O
ffset={dh.offset}"
                    );
                }
            }
            else
            {
                Console.WriteLine(
                    $"获取DH参数失败: {code.GetDescription()}"
                );
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"执行过程中发生异常/Exception occurred during execution: {ex.M
essage}"
            );
            code = StatusCode.OtherReason;
        }
        finally
        {
            // [ZH] 关闭连接
            // [EN] Close the connection
            StatusCode disconnectCode =
                controller.Disconnect();
            Console.WriteLine(
                disconnectCode != StatusCode.OK
                    ? disconnectCode.GetDescription()
                    : "Successfully disconnected."
            );
        }

        return code;
    }
}
```

4.2.10 Setting DH Parameters

Method Name	Motion.SetDHParam(List<DHparam> dHparams)
Description	Sets the robot's DH (Denavit-Hartenberg) parameters.
Request Parameters	dHparams : List<DHparam> DH parameter list
Return Value	StatusCode: Set operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): Not supported

Example Code

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 Getting Axis Lock Status

Method Name	Motion.GetDragSet()
Description	Gets the current robot axis lock status, which only applies to teaching movements.
Request Parameters	None
Return Value	DragStatus: Axis lock status, True indicates the axis is movable, False indicates it is locked StatusCode: Result of function execution

Method Name	Motion.GetDragSet()
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): Not supported

4.2.12 Setting Axis Lock Status

Method Name	Motion.SetDragSet(DragStatus `dragStatus`)
Description	Sets the current robot axis lock status, which only applies to teaching movements.
Request Parameters	`dragStatus` : DragStatus Axis lock status, default is all True: unlocked state
Return Value	StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): Not supported

4.2.13 Enabling Drag Teaching

Method Name	Motion.EnableDrag(bool `dragState`)
Description	Enables or disables drag teaching for the robot.
Request Parameters	`dragState` : bool The drag state of the robot, true indicates entering drag mode, false indicates exiting drag mode
Return Value	StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): Not supported

Example Code

Motion/DragControl.cs

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 Entering Real-Time Position Control Mode

Method Name	Motion.EnterPositionControl()
Description	Enters real-time position control mode, allowing precise position control of the robot.
Request Parameters	None
Return Value	StatusCode: Result of function execution
Note	After entering real-time control mode, control commands must be sent via UDP.
Compatible robot software version	Collaborative (Copper): v7.5.2.0+ Industrial (Bronze): Not supported

4.2.15 Exiting Real-Time Position Control Mode

Method Name	Motion.ExitPositionControl()
Description	Exits real-time position control mode, returning to the default robot control state.
Request Parameters	None
Return Value	StatusCode: Result of function execution
Note	After exiting, the robot will no longer accept real-time control commands.
Compatible robot software version	Collaborative (Copper): v7.5.2.0+ Industrial (Bronze): Not supported

4.2.16 Getting UDP Feedback Configuration

Method Name	Motion.GetUDPFeedbackConfig(int `id`)
Description	Reads the UDP robot state configuration file `rtf_info.json` .
Request Parameters	`id` : int Configuration ID, starting from 1
Return Value	UDPFeedbackConfig: UDP robot state configuration with the specified ID StatusCode: Result of function execution
Note	Configuration constraints: 1. When multiple configurations exist, only one may have `sub_flag=true` ; all others must be `false` . 2. When `use_multicast=true` , `client_ip` supports only a single multicast address in the `239.*.*.*` range. 3. When `use_multicast=false` , `client_ip` supports multiple LAN unicast addresses.
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.2.17 Setting UDP Feedback Configuration

Method Name	Motion.SetUDPFeedbackConfig(UDPFeedbackConfig `config`)
Description	Writes the UDP robot state configuration file `rtf_info.json` .
Request Parameters	`config` : UDPFeedbackConfig UDP robot state configuration; IDs in the configuration start from 1
Return Value	StatusCode: Result of function execution
Note	Configuration constraints: 1. When multiple configurations exist, only one may have `sub_flag=true` ; all others must be `false` . 2. When `use_multicast=true` , `client_ip` supports only a single multicast address in the `239.*.*.*` range. 3. When `use_multicast=false` , `client_ip` supports multiple LAN unicast addresses. This interface only writes to the configuration file; changes do not take effect immediately.
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.2.18 Enabling UDP Feedback

Method Name	Motion.EnableUDPFeedback(int `id`)
Description	Enables the UDP robot state configuration with the specified ID and synchronizes `rtf_info.json` so the change takes effect immediately.
Request Parameters	`id` : int Configuration ID, starting from 1
Return Value	StatusCode: Result of the enable operation
Note	After execution, only the specified ID has `sub_flag=true` ; all other configurations have `sub_flag=false` .

Method Name	Motion.EnableUDPFeedback(int `id`)
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.2.19 Disabling UDP Feedback

Method Name	Motion.DisableUDPFeedback(int `id`)
Description	Disables the UDP robot state configuration with the specified ID and synchronizes `rtf_info.json` so the change takes effect immediately.
Request Parameters	`id` : int Configuration ID, starting from 1
Return Value	StatusCode: Result of the disable operation
Note	After execution, all configurations have `sub_flag=false` .
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.2.20 Setting Subscription Parameters

Method Name	Motion.SetUDPFeedbackParams(bool `flag` , string `ip` , int `interval` , int `feedbackType` , List<int> `DOList` = null)
Description	Configures the UDP feedback parameters for the robot to push data to a specified IP address.
Request Parameters	`flag` : bool Whether to enable UDP data pushing; `ip` : string IP address of the recipient; `interval` : int Interval for sending data (unit: milliseconds); `feedbackType` : int Feedback data format (0: XML format); `DOList` : List<int> List of DO signals to be obtained (up to ten, optional)
Return Value	StatusCode: Result of function execution

Method Name	Motion.SetUDPFeedbackParams(bool `flag` , string `ip` , int `interval` , int `feedbackType` , List<int> `DOList` = null)
Note	The parameter settings are only effective when the UDP data pushing function is enabled.
Compatible robot software version	Collaborative (Copper): v7.5.2.0+ Industrial (Bronze): Not supported

Example Code

Motion/PositionControl.cs

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;
        }
    }
```

Data Push Description

Name	Field	Description
RIst: Cartesian Position	X	Value in the X direction in the tool coordinate system, unit is millimeters
	Y	Value in the Y direction in the tool coordinate system, unit is millimeters
	Z	Value in the Z direction in the tool coordinate system, unit is millimeters
	A	Rotation around the X axis in the tool coordinate system, unit is degrees
	B	Rotation around the Y axis in the tool coordinate system, unit is degrees
	C	Rotation around the Z axis in the tool coordinate system, unit is degrees
AIPos: Joint Position	A1-A6	Values of the six joints, unit is degrees
EIPos: Additional Axis Data	EIPos	Additional axis data
WristBtnState: Wrist Button State	Button State	1 = Button pressed, 0 = Button released
	DragModel	Drag button state
	RecordJoint	Teach record button state
	PauseResume	Pause/resume button state

Name	Field	Description
Digout: DO Output	Digout	State of digital output (DO)
ProgramStatus: Program Status	ProgId	Program ID
	Status	Interpreter execution status: 0 = INTERPRETER_IDLE 1 = INTERPRETER_EXECUTE 2 = INTERPRETER_PAUSED
	Xpath	Program segment return value, format is `program name: line number`
IPOC: Timestamp	IPOC	Timestamp

4.2.21 Getting the Robot's Soft Limits

Method Name	Motion.GetUserSoftLimit()
Description	Gets the current soft limits of the robot.
Request Parameters	None
Return Value	List<List<double>>: Robot's soft limits, the first layer of the list represents each axis, and the second layer represents the lower and upper limit values of each axis StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.2.22 Specifying UDP Position Control Parameters

Method Name	Motion.SetPositionTrajectoryParams(int `maxTimeoutCount` , int `timeout` , int `filterLayer` , double `wristElbowThreshold` , double `shoulderThreshold`)
Description	Specifies the parameters related to UDP position control.
Request Parameters	`maxTimeoutCount` : int Maximum number of timeouts, range [1,100] `timeout` : int Timeout period (i.e., send interval, default 20ms), range [1,100] `filterLayer` : int Filtering level, range [1,100] `wristElbowThreshold` : double Threshold for wrist/elbow approaching singularity, range [10,100] `shoulderThreshold` : double Threshold for approaching shoulder singularity, range [100,300]
Return Value	StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.2.23 Getting the Trajectory Shaping Parameter

Method Name	Motion.GetShapingParam()
Description	Gets the current trajectory shaping parameter value.
Request Parameters	None
Return Value	int: trajectory shaping parameter value (range 5~15, 0 means disabled). StatusCode: execution result of query operation.
Compatible robot software version	Collaborative (Copper): v7.7.I.0+ Industrial (Bronze): Not supported

4.2.24 Setting the Trajectory Shaping Parameter

Method Name	Motion.SetShapingParam(int `value`)
Description	Sets the trajectory shaping parameter value.
Request Parameters	`value` : int trajectory shaping parameter (range 5~15, 0 means disabled)
Return Value	StatusCode: execution result of set operation.
Compatible robot software version	Collaborative (Copper): v7.7.I.0+ Industrial (Bronze): Not supported

4.2.25 Payload-Related Interfaces

4.2.25.1 Getting the Current Active Payload

Method Name	Motion.Payload.GetCurrentPayload()
Description	Gets the currently active payload index.
Request Parameters	None
Return Value	int: Index of the currently active payload StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.2.25.2 Getting the Corresponding Payload

Method Name	Motion.Payload.GetPayloadById(int `index`)
Description	Gets payload information by index.
Request Parameters	`index` : Payload index
Return Value	StatusCode: Result of function execution

Method Name	Motion.Payload.GetPayloadById(int `index`)
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.2.25.3 Activating the Corresponding Payload

Method Name	Motion.Payload.SetCurrentPayload(int `index`)
Description	Activates the specified payload by index.
Request Parameters	`index` : Payload index
Return Value	StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+
Note	The payload ID must exist in the current device.

4.2.25.4 Getting All Payload Information

Method Name	Motion.Payload.GetAllPayloadInfo()
Description	Gets detailed information of all payloads.
Request Parameters	None
Return Value	Dictionary<uint, string>: Returns a dictionary of payload information StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.2.25.5 Adding a Payload

Method Name	Motion.Payload.AddPayload(PayloadInfo `payload`)
Description	Adds a new payload.

Method Name	Motion.Payload.AddPayload(PayloadInfo `payload`)
Request Parameters	`payload` : PayloadInfo Payload object
Return Value	StatusCode: Result of function execution
Note	The new payload ID must not exist in the current device and must be between 1 and 10.
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.2.25.6 Deleting a Specified Payload

Method Name	Motion.Payload.DeletePayload(int `index`)
Description	Deletes the payload with the specified index.
Request Parameters	`index` : int Payload index
Return Value	StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+
Note	Note: The currently active payload cannot be deleted. If you want to delete the active payload, please activate another payload first and then delete the current one.

4.2.25.7 Updating a Specified Payload

Method Name	Motion.Payload.UpdatePayload(PayloadInfo `payload`)
Description	Updates the information of the specified payload.
Request Parameters	`payload` : PayloadInfo Payload object
Return Value	StatusCode: Result of function execution
Note	The payload ID must exist in the current device.

Method Name	Motion.Payload.UpdatePayload(PayloadInfo payload)
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Checking if Axis 3 is Horizontal

Method Name	Motion.Payload.CheckAxisThreeHorizontal()
Description	Checks if Axis 3 is horizontal.
Request Parameters	None
Return Value	double: The horizontal angle of Axis 3 StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.2.0+ Industrial (Bronze): Not supported
Note	The horizontal angle must be between -1 and 1 to perform payload identification.

4.2.25.9 Getting the Payload Identification State

Method Name	Motion.Payload.GetPayloadIdentifyState()
Description	Gets the state of payload identification.
Request Parameters	None
Return Value	PayloadIdentifyState: Payload identification state StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.2.0+ Industrial (Bronze): Not supported

4.2.25.10 Starting Payload Identification

Method Name	Motion.Payload.StartPayloadIdentify(double `weight` , double `angle`)
Description	Starts payload identification.
Request Parameters	`weight` : double Payload weight (use -1 for unknown weight) `angle` : double Allowed rotation angle of Axis 6 (30-90 degrees)
Return Value	StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.2.0+ Industrial (Bronze): Not supported

Method Name	Motion.Payload.StartPayloadIdentify(double `weight` , double `angle`)
Note	You must enter the payload identification state before starting payload identification.

4.2.25.11 Getting the Payload Identification Result

Method Name	Motion.Payload.PayloadIdentifyResult()
Description	Gets the result of payload identification.
Request Parameters	None
Return Value	PayloadInfo: Payload identification result StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.2.0+ Industrial (Bronze): Not supported

4.2.25.12 Starting Interference Check for Payload Identification

Method Name	Motion.Payload.InterferenceCheckForPayloadIdentify(double `weight` , double `angle`)
Description	Starts interference check for payload identification to check for potential collisions.
Request Parameters	`weight` : double Payload weight (use -1 for unknown weight) `angle` : double Allowed rotation angle of Axis 6 (30-90 degrees)
Return Value	StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.2.0+ Industrial (Bronze): Not supported

4.2.25.13 Entering Payload Identification State

Method Name	Motion.Payload.PayloadIdentifyStart()
Description	Enters the payload identification state.

Method Name	Motion.Payload.PayloadIdentifyStart()
Request Parameters	None
Return Value	StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.2.0+ Industrial (Bronze): Not supported

4.2.25.14 Exiting Payload Identification State

Method Name	Motion.Payload.PayloadIdentifyDone()
Description	Exits the payload identification state.
Request Parameters	None
Return Value	StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.5.2.0+ Industrial (Bronze): Not supported

4.2.25.15 Full Payload Identification Process

Method Name	Motion.Payload.PayloadIdentify(double `weight` = -1, double `angle` = 90)
Description	Complete payload identification process, including all the interfaces mentioned above. For general payload identification, this interface is sufficient.
Request Parameters	`weight` : double Payload weight (use -1 for unknown weight) `angle` : double Allowed rotation angle of Axis 6 (30-90 degrees)
Return Value	PayloadInfo: Payload identification result StatusCode: Result of function execution
Note	The returned payload can be added to the robot or saved to an existing payload in the robot. The full process steps are: 1. Enter payload identification state 2. Start payload identification

Method Name	Motion.Payload.PayloadIdentify(double weight = -1, double angle = 90)
	3. Get payload identification result 4. Exit payload identification state
Compatible robot software version	Collaborative (Copper): v7.5.2.0+ Industrial (Bronze): Not supported

4.2.25.16 Terminating Payload Identification

Method Name	Motion.Payload.TerminatePayloadIdentify()
Description	Terminates the payload identification state.
Request Parameters	None
Return Value	StatusCode: Result of function execution
Compatible robot software version	Collaborative (Copper): v7.7.I.0+ Industrial (Bronze): Not supported

Example Code

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 Robot Program Execution Class

Overview

The `Execution` class provides a unified scheduling interface for robot programs and motion tasks. It is responsible for:

- Starting/stopping/pausing/resuming teaching programs
- Managing concurrent running task list
- Executing custom BAS-script flows

Combined with `Arm` and `Motion` , `Execution` handles host-side triggering and control of controller-side program flow.

4.3.1 Executing a Specified Program

Method Name	Execution.Start(string `programName`)
Description	Executes the specified program.
Request Parameters	`programName` : string Name of the program to be executed
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.3.2 Stopping the Currently Executing Program

Method Name	Execution.Stop(string `programName` = null)
Description	Stops the currently executing program or stops the robot's current motion command.
Request Parameters	`programName` : string Name of the program to be stopped, default is null, meaning stop the currently running program or motion command
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.3.3 Returning Details of All Running Programs

Method Name	Execution.AllRunningPrograms()
Description	Returns detailed information of all running programs, including program IDs and program names.
Request Parameters	None
Return Value	Dictionary<string, int>: Mapping from program ID to program name StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.3.4 Pausing Program Execution

Method Name	Execution.Pause(string `programName` = null)
Description	Pauses the currently executing program or pauses the robot's current motion.
Request Parameters	`programName` : string Name of the program to be paused, when not passed, defaults to controlling the currently running program or executing action

Method Name	Execution.Pause(string `programName` = null)
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.3.5 Resuming Program Execution

Method Name	Execution.Resume(string `programName` = null)
Description	Continues running a program in paused state.
Request Parameters	`programName` : string Name of the program to be resumed, default is null, meaning resume the currently paused program or motion command
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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.Valu
e}"
                    );
                }
            }
            else
            {
                Console.WriteLine(
                    $"获取运行程序列表失败/Get Running Programs List Failed: {c
ode.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 Executing a BAS Script Program

Method Name	Execution.ExecuteBasScript(BasScript `script`)
Description	Executes a user-defined BAS script program.
Request Parameters	`script` : BasScript User-defined BAS script program
Return Value	StatusCode: Operation execution result
Note	BAS script program pause, resume, and stop methods are the same as regular programs.
Compatible robot software version	Collaborative (Copper): v7.5.2.0+ Industrial (Bronze): Not supported Industrial Robot: v7.6.0.0+

Example Code

Execution/ExecuteBasScript.cs

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 Program Pose Read/Write

Overview

The `ProgramPoses` class is used to read, write, and convert pose points in robot teaching programs. Through this class, you can:

- Locate the specified program and pose index
- Perform add, delete, modify, and query operations
- Convert between Cartesian and joint representations

It is convenient for batch maintenance of program points or offline editing in host-side scenarios.

4.4.1 Getting the Value of a Specified Pose in a Program

Method Name	ProgramPoses.Read(string `programName` , int `index` , FileType `ft` = FileType.UserProgram)
Description	Gets the pose value at the specified index in the specified program.
Request Parameters	`programName` : string Program name `index` : int Pose index `ft` : FileType File type (default: FileType.UserProgram)
Return Value	ProgramPose: Pose information StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Modifying the Value of a Specified Pose in a Program

Method Name	ProgramPoses.Write(string `programName` , int `index` , ProgramPose `value` , FileType `ft` = FileType.UserProgram)
Description	Modifies the pose value at the specified index in the specified program.
Request Parameters	`programName` : string Program name `index` : int Pose index `value` : ProgramPose Pose value to be updated `ft` : FileType File type (default: FileType.UserProgram)
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Adding a Pose to a Specified Program

Method Name	ProgramPoses.Add(string `programName` , int `index` , ProgramPose `value` , FileType `ft` = FileType.UserProgram)
Description	Adds a new pose at the specified index position in the specified program.
Request Parameters	`programName` : string Program name `index` : int Pose index `value` : ProgramPose Pose value to be added `ft` : FileType File type (default: FileType.UserProgram)
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Deleting a Specified Pose from a Program

Method Name	ProgramPoses.Delete(string `programName` , int `index` , FileType `ft` = FileType.UserProgram)
Description	Deletes the pose at the specified index in the specified program.

Method Name	ProgramPoses.Delete(string `programName` , int `index` , FileType `ft` = FileType.UserProgram)
Request Parameters	`programName` : string Program name `index` : int Pose index `ft` : FileType File type (default: FileType.UserProgram)
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

ProgramPoses/DeleteProgramPose.cs

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

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

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

```
        );

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

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

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

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

        return code;
    }
}
```

4.4.5 Getting All Poses from a Specified Program

Method Name	ProgramPoses.ReadAllPoses(string `programName` , FileType `ft` = FileType.UserProgram)
Description	Gets all pose information from the specified program.
Request Parameters	`programName` : string Program name `ft` : FileType File type (default: FileType.UserProgram)
Return Value	List<ProgramPose>: Pose information list StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Converting Pose Types in Robot Programs

Method Name	ProgramPoses.ConvertPose(ProgramPose `pose` , PoseType `toType`)
Description	Converts robot program poses between joint coordinates and Cartesian space coordinates.
Request Parameters	`pose` : ProgramPose Pose value to be converted `toType` : PoseType Target type after conversion
Return Value	ProgramPose: Converted pose information StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Signals

Overview

The `Signals` class provides a unified read/write interface for controller I/O, including:

- Digital/analog input and output operations
- Batch I/O operations

With `Signals` , you can:

- Read current signal states
- Batch write DO/RO/GO ports
- Integrate grippers, sensors, and production-line devices

4.5.1 Reading the Value of a Specified Type and Port IO

Method Name	Signals.Read(SignalType `type` , int `index`)
Description	Reads the IO value of the specified type and port (supports DI/DO/UI/UO/RI/RO/GI/GO/TAI/TDI/TDO/AI/AO).
Request Parameters	`type` : SignalType IO type to read `index` : int IO port index, starting from 1
Return Value	int: IO value, 1 represents high level, 0 represents low level StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+
Note	UI/UO can only be read, not written

Example Code

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 Writing the Value of a Specified Type and Port IO

Method Name	Signals.Write(SignalType `type` , int `index` , double `value`)
Description	Writes the IO value of the specified type and port, currently only supports DO/RO/GO/TDO/AO.
Request Parameters	`type` : SignalType IO type to write `index` : int IO port index, starting from 1 `value` : double IO value (DO/RO/TDO only allow 0 or 1; GO is integer; AO is floating-point analog)
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+
Note	UI/UO can only be read, not written

Example Code

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 Batch Write DO (Digital Output) Signals

Method Name	Signals.MultiWrite(SignalType `type` , List<int> `ioData`)
Description	Batch write DO (Digital Output) signals.
Request Parameters	`type` : SignalType IO type to write (DO only) `ioData` : List<int> Port number and port value list (e.g., [port1, state1, port2, state2]; length must be even and greater than 0)
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+
Note	Only DO supports batch write; UI/UO do not support writing, DI/RI only support single-point read

Example Code

cs

```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 Batch Read DO (Digital Output) Port Values

Method Name	Signals.MultiRead(SignalType `type` , List<int> `indexes`)
Description	Batch read DO (Digital Output) port values.
Request Parameters	`type` : SignalType IO type to read (DO only) `indexes` : List<int> Port number list (cannot be empty)
Return Value	List<int>: Port value list (order consistent with input) StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+
Note	Only DO supports batch read; UI/UO do not support writing, DI/RI only support single-point read

Example Code

```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 Register Information

Overview

The `Registers` class provides a unified entry for host-side register access on the controller, supporting operations for multiple register types.

Core Features

- Read/write numeric registers (R)
- Read/write motion registers (MR)
- Read/write string registers (SR)
- Read/write pose registers (PR)
- Read/write Modbus registers (MH holding, MI input)

Use Cases

- Runtime parameter passing
- Robot state synchronization
- Configuration sharing with external systems
- Host-device interaction control

4.6.1 R Numeric Register Operations

4.6.1.1 Reading the Value of an R Register

Method Name	Registers.Read_R(int `index`)
Description	Reads the value of an R numeric register.
Request Parameters	`index` : int R register number to read

Method Name	Registers.Read_R(int `index`)
Return Value	double: Register value StatusCode: Operation execution result
Compatible robot software version	Collaborative robot: 7.6.0.1 Industrial robot: 7.6.0.0

4.6.1.2 Reading the Value of an R Register (with Metadata)

Method Name	Registers.Read_R(int `index` , bool `withMeta`)
Description	Reads the value of an R numeric register, with optional metadata (name/comment).
Request Parameters	`index` : int Register number to read `withMeta` : bool Whether to return metadata (returns register object when true)
Return Value	Register: Register object (id/name/comment/value) StatusCode: Operation execution result
Compatible robot software version	Collaborative robot: 7.7.0.0+ Industrial robot: 7.7.0.0+

4.6.1.3 Writing the Value of an R Register

Method Name	Registers.Write_R(int `index` , double `value`)
Description	Writes the value of an R numeric register.
Request Parameters	`index` : int Register number to write `value` : double Register numeric value to write
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative robot: 7.6.0.1 Industrial robot: 7.6.0.0

4.6.1.4 Writing an R Register with Metadata

Method Name	Registers.Write_R(Register `register`)
Description	Writes an R register using a Register object, including name and comment.
Request Parameters	`register` : Register object (id/name/comment/value)
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative robot: 7.7.0.0+ Industrial robot: 7.7.0.0+

4.6.1.5 Deleting an R Register

Method Name	Registers.Delete_R(int `index`)
Description	Deletes the specified R numeric register.
Request Parameters	`index` : int R register number to delete
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Motion Register Operations

4.6.2.1 Reading the Value of an MR Register

Method Name	Registers.Read_MR(int `index`)
Description	Reads the value of an MR motion register.
Request Parameters	`index` : int MR register number to read

Method Name	Registers.Read_MR(int `index`)
Return Value	int: Register value StatusCode: Operation execution result
Compatible robot software version	Collaborative robot: 7.6.0.1 Industrial robot: 7.6.0.0

4.6.2.2 Reading the Value of an MR Register (with Metadata)

Method Name	Registers.Read_MR(int `index` , bool `withMeta`)
Description	Reads the value of an MR motion register, with optional metadata (name/comment).
Request Parameters	`index` : int Register number to read `withMeta` : bool Whether to return metadata (returns register object when true)
Return Value	MotionRegister: Register object (id/name/comment/value) StatusCode: Operation execution result
Compatible robot software version	Collaborative robot: 7.7.0.0+ Industrial robot: 7.7.0.0+

4.6.2.3 Writing the Value of an MR Register

Method Name	Registers.Write_MR(int `index` , int `value`)
Description	Writes the value of an MR motion register.
Request Parameters	`index` : int Register number to write `value` : int Register numeric value to write
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative robot: 7.6.0.1 Industrial robot: 7.6.0.0

4.6.2.4 Writing an MR Register with Metadata

Method Name	Registers.Write_MR(MotionRegister `register`)
Description	Writes an MR register using a MotionRegister object, including name and comment.
Request Parameters	`register` : MotionRegister object (id/name/comment/value)
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative robot: 7.7.0.0+ Industrial robot: 7.7.0.0+

4.6.2.5 Deleting an MR Register

Method Name	Registers.Delete_MR(int `index`)
Description	Deletes the specified MR motion register.
Request Parameters	`index` : int MR register number to delete
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 String Register Operations

4.6.3.1 Reading the Value of an SR Register

Method Name	Registers.Read_SR(int `index`)
Description	Reads the value of an SR string register.
Request Parameters	`index` : int SR register number to read

Method Name	Registers.Read_SR(int `index`)
Return Value	string: Register string value StatusCode: Operation execution result
Compatible robot software version	Collaborative robot: 7.6.0.1 Industrial robot: 7.6.0.0

4.6.3.2 Reading the Value of an SR Register (with Metadata)

Method Name	Registers.Read_SR(int `index` , bool `withMeta`)
Description	Reads the value of an SR string register, with optional metadata (name/comment).
Request Parameters	`index` : int Register number to read `withMeta` : bool Whether to return metadata (returns register object when true)
Return Value	StringRegister: Register object (id/name/comment/value) StatusCode: Operation execution result
Compatible robot software version	Collaborative robot: 7.7.0.0+ Industrial robot: 7.7.0.0+

4.6.3.3 Writing the Value of an SR Register

Method Name	Registers.Write_SR(int `index` , string `value`)
Description	Writes the value of an SR string register.
Request Parameters	`index` : int Register number to write `value` : string Register string value to write
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative robot: 7.6.0.1 Industrial robot: 7.6.0.0

4.6.3.4 Writing an SR Register with Metadata

Method Name	Registers.Write_SR(StringRegister `register`)
Description	Writes an SR register using a StringRegister object, including name and comment.
Request Parameters	`register` : StringRegister object (id/name/comment/value)
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative robot: 7.7.0.0+ Industrial robot: 7.7.0.0+

4.6.3.5 Deleting an SR Register

Method Name	Registers.Delete_SR(int `index`)
Description	Deletes the specified SR string register.
Request Parameters	`index` : int SR register number to delete
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Pose Register Operations

4.6.4.1 Reading the Value of a PR Register

Method Name	Registers.Read_PR(int `index`)
Description	Reads the value of a PR pose register.
Request Parameters	`index` : int PR register number to read

Method Name	Registers.Read_PR(int `index`)
Return Value	PoseRegister: Pose data StatusCode: Operation execution result
Compatible robot software version	Collaborative robot: 7.6.0.1 Industrial robot: 7.6.0.0

4.6.4.2 Reading the Value of a PR Register (with Metadata)

Method Name	Registers.Read_PR(int `index` , bool `withMeta`)
Description	Reads the value of a PR pose register, with optional metadata (name/comment).
Request Parameters	`index` : int Register number to read `withMeta` : bool Whether to return metadata (includes name/comment when true)
Return Value	PoseRegister: Pose data (includes name/comment when withMeta=true) StatusCode: Operation execution result
Compatible robot software version	Collaborative robot: 7.7.0.0+ Industrial robot: 7.7.0.0+

4.6.4.3 Writing the Value of a PR Register

Method Name	Registers.Write_PR(PoseRegister `pose`)
Description	Writes the value of a PR pose register.
Request Parameters	`pose` : PoseRegister Pose data to write
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative robot: 7.6.0.1 Industrial robot: 7.6.0.0

4.6.4.4 Writing a PR Register with Metadata

Method Name	Registers.Write_PR(PoseRegister `pose` , bool `withMeta`)
Description	Writes the value of a PR pose register, with control over whether to write metadata (name/comment).
Request Parameters	`pose` : PoseRegister Pose data to write `withMeta` : bool Whether to write metadata (writes name/comment when true)
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative robot: 7.7.0.0+ Industrial robot: 7.7.0.0+

4.6.4.5 Deleting a PR Register

Method Name	Registers.Delete_PR(int `index`)
Description	Deletes the specified PR pose register.
Request Parameters	`index` : int PR register number to delete
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Registers (MH Holding Registers, MI Input Registers)

4.6.5.1 Reading the Value of an MH Register

Method Name	Registers.Read_MH(int `index`)
Description	Reads the value of an MH holding register.
Request Parameters	`index` : int Register number to read
Return Value	int: Register value StatusCode: Operation execution result
Compatible robot software version	Collaborative robot: 7.6.0.0 Industrial robot: 7.6.0.0

4.6.5.2 Reading the Value of an MI Register

Method Name	Registers.Read_MI(int `index`)
Description	Reads the value of an MI input register.
Request Parameters	`index` : int Register number to read
Return Value	int: Register value StatusCode: Operation execution result
Compatible robot software version	Collaborative robot: 7.6.0.0 Industrial robot: 7.6.0.0

4.6.5.3 Writing the Value of an MH Register

Method Name	Registers.Write_MH(int `index` , int `value`)
Description	Writes the value of an MH holding register.
Request Parameters	`index` : int Register number to write `value` : int Register numeric value to write
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative robot: 7.6.0.0 Industrial robot: 7.6.0.0

4.6.5.4 Writing the Value of an MI Register

Method Name	Registers.Write_MI(int `index` , int `value`)
Description	Writes the value of an MI input register.
Request Parameters	`index` : int Register number to write `value` : int Register numeric value to write
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative robot: 7.6.0.0 Industrial robot: 7.6.0.0

Example Code

Registers/ModbusRegisterOperations.cs

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 Trajectory Control

Overview

The `Trajectory` module provides interfaces for offline trajectory execution and trajectory file conversion, supporting complex trajectory reproduction.

Core Features

- Set and execute offline trajectory files
- Move robot to trajectory start point at safe speed
- Convert CSV trajectory files to `.trajectory` format
- Query trajectory conversion status

Use Cases

- Accurate reproduction of complex trajectories
- Converting external trajectory data into executable robot format

4.7.1 Setting the Offline Trajectory File to Be Executed

Method Name	Trajectory.SetOffLineTrajectoryFile(string `path`)
Description	Sets the offline trajectory file to be executed.
Request Parameters	`path` : string Offline trajectory file program name (format description see remarks below)
Return Value	StatusCode: Operation execution result
Remarks	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 - Line 2: Represents the initial positions of the 6 axes

Method Name	Trajectory.SetOffLineTrajectoryFile(string `path`)
	- 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
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.7.2 Moving the Robot to the Start Point of the Offline Trajectory at a Safe Speed

Method Name	Trajectory.PrepareOfflineTrajectory()
Description	Moves the robot to the start point of the offline trajectory at a safe speed.
Request Parameters	None
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.7.3 Starting Execution of the Offline Trajectory File

Method Name	Trajectory.ExecuteOfflineTrajectory()
Description	Starts the execution of the offline trajectory program.
Request Parameters	None
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.7.4 Trajectory File Conversion Function

Method Name	Trajectory.TransformCsvToTrajectory(string `fileName`)
Description	Converts a trajectory CSV file into the trajectory format and saves it to the controller's trajectory file directory.
Request Parameter	`fileName` : string – name of the CSV trajectory file.
Overload Method	**Trajectory.TransformCsvToTrajectory**(string `fileName` , string `separator` , string `ioFlag`) `separator` : string delimiter, supports space or comma. `ioFlag` : string IO source flag, "1" for default IO values and "2" for user-defined IO values.
Return Value	string: path of the converted trajectory file. StatusCode: result of the conversion operation.
Compatible Robot Software Versions	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.7.5 Querying Trajectory File Conversion Status

Method Name	Trajectory.CheckTransformStatus(string `fileName`)
Description	Queries the current status of trajectory file conversion.

Method Name	Trajectory.CheckTransformStatus(string `fileName`)
Request Parameters	`fileName` : string Trajectory file path (returned by the TransformCsvToTrajectory interface)
Return Value	TransformState: Conversion state StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Start Trajectory Recording

Method Name	Trajectory.TrajectoryRecordBegin(string `name`)
Description	Starts trajectory recording for the trajectory replay feature.
Request Parameters	`name` : string trajectory program name.
Return Value	StatusCode: execution result of starting recording.
Compatible Robot Software Versions	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.7.7 Finish Trajectory Recording

Method Name	Trajectory.TrajectoryRecordFinish(string `name`)
Description	Finishes trajectory recording for the trajectory replay feature.
Request Parameters	`name` : string trajectory program name.
Return Value	StatusCode: execution result of finishing recording.
Compatible Robot Software Versions	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.7.8 Start Trajectory Replay

Method Name	Trajectory.TrajectoryReplayStart(string `name`)
Description	Starts replaying the specified trajectory.
Request Parameters	`name` : string trajectory program name.
Return Value	StatusCode: execution result of starting replay.
Compatible Robot Software Versions	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.7.9 Stop Trajectory Replay

Method Name	Trajectory.TrajectoryReplayStop(string `name`)
Description	Stops replaying the specified trajectory.
Request Parameters	`name` : string trajectory program name.
Return Value	StatusCode: execution result of stopping replay.
Compatible Robot Software Versions	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.7.10 Delete Trajectory

Method Name	Trajectory.TrajectoryRecordDelete(string `name`)
Description	Deletes the specified trajectory.
Request Parameters	`name` : string trajectory program name.
Return Value	StatusCode: execution result of delete operation.
Compatible Robot Software Versions	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.7.11 Get Trajectory List

Method Name	Trajectory.GetTrajectoryRecordList()
Description	Gets the list of currently recorded trajectory names.
Request Parameters	None
Return Value	List<string>: trajectory name list. StatusCode: execution result of query operation.
Compatible Robot Software Versions	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.7.12 Get Trajectory Start Pose

Method Name	Trajectory.GetTrajectoryRecordStartPose(string `name`)
Description	Gets the start pose of the specified recorded trajectory.
Request Parameters	`name` : string trajectory program name.
Return Value	MotionPose: trajectory start pose. StatusCode: execution result of query operation.
Compatible Robot Software Versions	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.7.13 Start Recording Trajectory Table/Path Table

Method Name	Trajectory.PathRecordBegin(string `name` , string `comment` , double `param` , double `angle` = 1)
Description	Starts recording a trajectory table (`.traj`) or path table (`.path`).

Method Name	Trajectory.PathRecordBegin(string `name` , string `comment` , double `param` , double `angle` = 1)
Request Parameters	`name` : string trajectory/path table file name, must end with `.path` or `.traj` . `comment` : string description. `param` : double recording parameter (`.path` : distance threshold, must be >=0.5; `.traj` : recording interval, must be >=2). `angle` : double rotational threshold for path table (`.path` only), must be >=1.
Return Value	StatusCode: execution result of starting recording.
Compatible Robot Software Versions	Collaborative (Copper): v7.7.I.0+ Industrial (Bronze): Not supported

4.7.14 Finish Recording Trajectory Table/Path Table

Method Name	Trajectory.PathRecordFinish()
Description	Finishes current trajectory table/path table recording.
Request Parameters	None
Return Value	StatusCode: execution result of finishing recording.
Compatible Robot Software Versions	Collaborative (Copper): v7.7.I.0+ Industrial (Bronze): Not supported

4.7.15 Get Trajectory Table/Path Table Start Pose

Method Name	Trajectory.GetPathStartPose(string `name`)
Description	Gets the start pose of the specified trajectory table/path table.
Request Parameters	`name` : string trajectory/path table name.

Method Name	Trajectory.GetPathStartPose(string `name`)
Return Value	MotionPose: start pose. StatusCode: execution result of query operation.
Compatible Robot Software Versions	Collaborative (Copper): v7.7.I.0+ Industrial (Bronze): Not supported

4.7.16 Get Trajectory Table/Path Table State

Method Name	Trajectory.GetPathState(List<string> `pathList`)
Description	Batch-queries current recording state of trajectory/path tables.
Request Parameters	`pathList` : List<string> trajectory/path table names.
Return Value	Dictionary<string, MotionTableState>: file-name to state mapping. StatusCode: execution result of query operation.
Compatible Robot Software Versions	Collaborative (Copper): v7.7.I.0+ Industrial (Bronze): Not supported

4.7.17 Set Path Planner Parameters

Method Name	Trajectory.SetPathPlannerParameter(double `transitionTime` , double `scalingFactor`)
Description	Sets path-planner parameters to affect transition and smoothness.
Request Parameters	`transitionTime` : double Start/stop transition time, must be >=0.2. Smaller values mean faster acceleration/deceleration, minimum 0.2. `scalingFactor` : double Scaling factor, range [0,1]. =0, the robot will pass through each path point at constant time, prioritizing time interval between points. =1, strict scaling, the robot passes through each path point at constant linear velocity, prioritizing constant speed through each point. Between 0~1 is a compromise behavior.

Method Name	Trajectory.SetPathPlannerParameter(double `transitionTime` , double `scalingFactor`)
Return Value	StatusCode: execution result of set operation.
Compatible Robot Software Versions	Collaborative (Copper): v7.7.I.0+ Industrial (Bronze): Not supported

4.7.18 Get Path Planner Parameters

Method Name	Trajectory.GetPathPlannerParameter()
Description	Gets current path-planner parameters.
Request Parameters	None
Return Value	double: `transitionTime` Start/stop transition time. Smaller values mean faster acceleration/deceleration, minimum 0.2. double: `scalingFactor` Scaling factor. =0, constant time through each path point. =1, constant linear velocity through each path point. 0~1 is compromise behavior. StatusCode: execution result of query operation.
Compatible Robot Software Versions	Collaborative (Copper): v7.8.0.0+ Industrial (Bronze): Not supported

4.7.19 Move Along Trajectory Table/Path Table

Method Name	Trajectory.MovePath(string `name` , double `vel` = 100, double `acc` = 1)
Description	Moves robot TCP along the specified trajectory/path table.
Request Parameters	`name` : string trajectory/path table name. `vel` : double speed, range [0,5000] (mm/s).

Method Name	Trajectory.MovePath(string `name` , double `vel` = 100, double `acc` = 1)
	`acc` : double acceleration factor, range [0,1.2].
Return Value	StatusCode: execution result of move operation.
Compatible Robot Software Versions	Collaborative (Copper): v7.7.I.0+ Industrial (Bronze): Not supported

4.8 Alarm Information

Overview

The `Alarm` module provides reading, reset, and query capabilities for robot alarm information, used for monitoring and handling abnormal situations that may occur during robot operation.

Core Features

- Reset alarms
- Get all active alarms
- Get highest-priority alarm

Use Cases

- Monitor robot operating status and detect abnormalities in a timely manner
- Handle errors and alarms during robot operation
- Record and analyze robot alarm history
- Implement automated alarm processing workflows
- Integrate into robot monitoring systems

4.8.1 Getting the Highest Priority Alarm

Method Name	Alarm.GetTopAlarm()
Description	Gets the current highest priority alarm.
Request Parameters	None
Return Value	string: Alarm information string StatusCode: Operation execution result

Method Name	Alarm.GetTopAlarm()
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Getting All Active Alarms

Method Name	Alarm.GetAllActiveAlarms()
Description	Gets all currently active alarms.
Request Parameters	None
Return Value	List<string>: Active alarm entry list StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Resetting Alarms

Method Name	Alarm.ResetAlarms()
Description	Resets current errors/alarms.
Request Parameters	None
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.9 File Service Class

Overview

`FileManager` is used to upload, download, delete, and search program/trajectory/temporary files between host and robot controller.

Core Features

- Upload local files to controller
- Download controller files to local path
- Delete files on controller
- Search files by filename pattern
- Manage multiple file types (`UserProgram` , `BlockProgram` , `TrajectoryProgram` , `TmpFile`)
- Control overwrite behavior during upload/download

Use Cases

- Deploy programs to controller
- Backup controller-side programs and trajectories
- Synchronize production-line debugging data
- Batch file management on controller

4.9.1 Uploading a Local File to the Robot

Method Name	FileManager.Upload(string `filePath` , FileType `ft` , bool `overWriting` = false)
Description	Uploads a local file to the robot controller.
Request Parameters	`filePath` : string Absolute path of the local file to be uploaded `ft` : FileType Type of the file to be uploaded

Method Name	FileManager.Upload(string `filePath` , FileType `ft` , bool `overWriting` = false)
	`overWriting` : bool Whether to overwrite existing file in robot controller, default is false (no overwrite)
Note	For USER_PROGRAM and BLOCK_PROGRAM, provide the full path to the `.xml` / `.block` file; the system also uploads the same-name `.json` (and `.xml` for BlockProgram).
Return Value	StatusCode: Upload operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.9.2 Downloading a Robot File to a Local Machine

Method Name	FileManager.Download(string `fileName` , FileType `ft` , string `savePath`)
Description	Downloads a file from the robot controller to local.
Request Parameters	`fileName` : string Name of the file to be downloaded `ft` : FileType Type of the file to be downloaded `savePath` : string Local save path for the downloaded file
Note	For `UserProgram` / `BlockProgram` / `TrajectoryProgram` , specify only the program name (filename without extension). For `TmpFile` , provide the full filename with extension.
Return Value	StatusCode: Download operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.9.3 Deleting a File from the Robot

Method Name	FileManager.Delete(string `fileName` , FileType `ft`)
Description	Deletes a file from the robot controller.

Method Name	FileManager.Delete(string `fileName` , FileType `ft`)
Request Parameters	`fileName` : string Name of the file to be deleted `ft` : FileType Type of the file to be deleted
Note	For `UserProgram` / `BlockProgram` / `TrajectoryProgram` , specify only the program name (filename without extension). For `TmpFile` , provide the full filename with extension.
Return Value	StatusCode: Delete operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Search Files by Filename Pattern

Method Name	FileManager.Search(string `pattern` , ref List<string> `fl`)
Description	Searches for files on the robot controller that match the filename pattern.
Request Parameters	`pattern` : string Filename match pattern `fl` : ref List<string> Returned file list
Return Value	StatusCode: Search operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.10 BasScript Script Program Class

Overview

`BasScript` is used to construct BAS instruction sequences on the host side in a structured way, including motion, logic, program control, communication, vision, and Modbus instructions.

Core Features

- Build motion instructions (`MoveJoint` , `MoveLine` , `MoveCircle` , etc.)
- Build logic instructions (`IF` , `WHILE` , `SWITCH` , `GOTO` , etc.)
- Build assignment, wait, pause, and abort instructions
- Build program call and management instructions
- Build Socket communication instructions
- Build Modbus register read/write instructions
- Build vision instructions
- Configure extra parameters (acceleration, RTCP, frame offset, etc.)
- Support high-level jump instructions (`JUMP` series)

Use Cases

- Automatically generate complex robot programs
- Edit and modify robot programs
- Reuse common program modules and instruction sequences
- Implement seamless integration between host and robot programs
- Build customized robot motion trajectories
- Integrate vision, communication, and Modbus functionality into robot programs

Class Constructor

Method Name	BasScript(string `name`)
Description	Constructs a BAS instruction sequence class for use in robots
Request Parameters	`name` : string Script program name
Compatible robot software version	Collaborative (Copper): v7.5.2.0+ Industrial (Bronze): Not supported
Note	All methods in this class have the same compatible version requirements as this class

Subclass Structure

The `BasScript` class contains the following subclasses for organizing different instruction categories:

1. **ExtraParam**: Extra parameter class for building complex robot control commands
2. **BasMotion**: Motion instruction subclass, contains all robot motion-related methods
3. **BasLogical**: Logical instruction subclass, contains all logic control-related methods
4. **BasStructure**: Structural instruction subclass, contains all structure control-related methods
5. **BasSocket**: Socket communication subclass, contains all Socket communication-related methods
6. **BasModbus**: Modbus communication subclass, contains all Modbus communication-related methods
7. **BasVision**: Vision instruction subclass, contains all vision-related methods

4.10.1 ExtraParam Class

Method Name	ExtraParam()
Description	Extra parameter class for building complex robot control commands
Request Parameters	None
Return Value	ExtraParam object

4.10.1.1 Set Acceleration

Method Name	ExtraParam.Acceleration(double `value`)
Description	Sets acceleration, range is 1~120%
Request Parameters	`value` : double Acceleration value, unit: % (floating point)
Return Value	StatusCode: Parameter setting execution result

4.10.1.2 Set RTCP Parameter

Method Name	ExtraParam.RTCP()
Description	Sets RTCP parameter, only supports MoveL and MoveC instructions
Request Parameters	None
Return Value	StatusCode: Parameter setting execution result

4.10.1.3 Set Frame Offset

Method Name	ExtraParam.Offset(int `index`)
Description	Sets frame offset
Request Parameters	`index` : int Offset index (integer)
Return Value	StatusCode: Parameter setting execution result

4.10.1.4 Set Time-Base Run/Assign

Method Name	ExtraParam.TB(double `second` , string `type` , string `name`)
Description	Sets time-base run or assign, range is 0.01-30s, values less than 0.01 will not save successfully
Request Parameters	`second` : double Seconds, unit: sec (floating point) `type` : string Execution type (string) `name` : string Name (for running, string)
Return Value	StatusCode: Parameter setting execution result

Method Name	ExtraParam.TB(double `second` , string `type` , int `index` , int `status`)
Description	Sets time-base run or assign, range is 0.01-30s, values less than 0.01 will not save successfully
Request Parameters	`second` : double Seconds, unit: sec (floating point) `type` : string IO type (string) `index` : int Index (for assignment, integer) `status` : int Status (for assignment, integer)
Return Value	StatusCode: Parameter setting execution result

4.10.1.5 Set Skip

Method Name	ExtraParam.SKIP(int `index`)
Description	Sets jump, when the jump condition set by SKIP CONDITION instruction is met, directly jumps from the current line containing SKIP instruction to the target instruction line or target program
Request Parameters	`index` : int Index of the target label (integer)
Return Value	StatusCode: Parameter setting execution result

4.10.1.6 Set Departure and Approach Distance

Method Name	ExtraParam.Approach(double `departureDist` , double `approachingDist`)
Description	Sets departure and approach distance for gate-type motion, only used for JUMP instruction
Request Parameters	`departureDist` : double Departure distance, unit: mm (floating point) `approachingDist` : double Approach distance, unit: mm (floating point)
Return Value	StatusCode: Parameter setting execution result

4.10.2 BasMotion Motion Instruction Subclass

BasMotion subclass contains all robot motion-related methods, used for building motion instruction sequences.

4.10.2.1 MoveJoint Motion to Point Instruction

Method Name	BasScript.BasMotion.MoveJoint(MovePoseType `poseType` , int `poseIndex` , SpeedType `speedType` , double `speedValue` , SmoothType `smoothType` , double `smoothDistance` = 0, ExtraParam `extraParam` = null)
Description	Joint motion instruction, moves the robot to a specified position in the workspace with specified movement speed and method, corresponds to MoveJoint instruction in robot program writing
Request Parameters	`poseType` : MovePoseType Pose type `poseIndex` : int Pose index `speedType` : SpeedType Speed type `speedValue` : double Speed value, unit: % `smoothType` : SmoothType Smooth type `smoothDistance` : double Smooth distance, unit: mm, value range: 0~1000 `extraParam` : ExtraParam Additional parameter
Return Value	StatusCode: Motion instruction execution result

4.10.2.2 MoveLine Linear Motion to Point Instruction

Method Name	BasScript.BasMotion.MoveLine(MovePoseType `poseType` , int `poseIndex` , SpeedType `speedType` , double `speedValue` , SmoothType `smoothType` , double `smoothDistance` = 0, ExtraParam `extraParam` = null)
Description	Linear motion instruction, moves the robot linearly to a specified position in the workspace with specified movement speed and method, corresponds to MoveLine instruction in robot program writing
Request Parameters	`poseType` : MovePoseType Pose type `poseIndex` : int Pose index `speedType` : SpeedType Speed type `speedValue` : double Speed value, unit: mm/s `smoothType` : SmoothType Smooth type `smoothDistance` : double Smooth distance, unit: mm, value range: 0~1000 `extraParam` : ExtraParam Additional parameter
Return Value	StatusCode: Motion instruction execution result

4.10.2.3 MoveCircle Arc Motion to Point Instruction

Method Name	BasScript.BasMotion.MoveCircle(MovePoseType `poseType1` , int `poseIndex1` , MovePoseType `poseType2` , int `poseIndex2` , SpeedType `speedType` , double `speedValue` , SmoothType `smoothType` , double `smoothDistance` = 0, ExtraParam `extraParam` = null)
Description	Arc motion instruction, moves the robot in an arc to a specified position in the workspace with specified movement speed and method, corresponds to MoveCircle instruction in robot program writing
Request Parameters	`poseType1` : MovePoseType First pose type `poseIndex1` : int First pose index `poseType2` : MovePoseType Second pose type `poseIndex2` : int Second pose index `speedType` : SpeedType Speed type `speedValue` : double Speed value, unit: mm/s

Method Name	BasScript.BasMotion.MoveCircle(MovePoseType `poseType1` , int `poseIndex1` , MovePoseType `poseType2` , int `poseIndex2` , SpeedType `speedType` , double `speedValue` , SmoothType `smoothType` , double `smoothDistance` = 0, ExtraParam `extraParam` = null)
	`smoothType` : SmoothType Smooth type `smoothDistance` : double Smooth distance, unit: mm, value range: 0~1000 `extraParam` : ExtraParam Additional parameter
Return Value	StatusCode: Motion instruction execution result

4.10.2.4 Jump Point-to-Point Motion Instruction

Method Name	BasScript.BasMotion.Jump(MovePoseType `poseType` , int `poseIndex` , double `speedValue` , double `speedRatio` , SpeedType `limZType` , double `limZValue` , SmoothType `smoothType` , double `smoothDistance` = 0, ExtraParam `extraParam` = null)
Description	Gate-type motion instruction, specifies the robot to perform gate-type motion (first vertically up, then horizontally move, finally vertically down), corresponds to JUMP instruction in robot program writing
Request Parameters	`poseType` : MovePoseType Target pose storage type `poseIndex` : int Target position index `speedValue` : double Motion speed value, unit: mm/s `speedRatio` : double Motion speed ratio, unit: % `limZType` : SpeedType Z-axis limit type `limZValue` : double Z-axis limit value `smoothType` : SmoothType Smooth type `smoothDistance` : double Smooth distance, unit: mm, value range: 0~1000 `extraParam` : ExtraParam Extra parameter
Return Value	StatusCode: Motion instruction execution result

4.10.2.5 Jump3 Three-Point Jump Instruction

Method Name	BasScript.BasMotion.Jump3(MovePoseType `poseType` , int `poseIndex` , double `speedValue` , double `speedRatio` , SmoothType `smoothType` , double `smoothDistance` = 0, ExtraParam `extraParam` = null)
Description	Gate-type motion instruction, specifies the robot to perform gate-type motion, including departure, approach, and target positions, corresponds to JUMP3 instruction in robot program writing
Request Parameters	`poseType` : MovePoseType Target pose storage type `poseIndex` : int 3 target position indices `speedValue` : double Motion speed value, unit: mm/s `speedRatio` : double Motion speed ratio, unit: % `smoothType` : SmoothType Smooth type `smoothDistance` : double Smooth distance, unit: mm, value range: 0~1000 `extraParam` : ExtraParam Extra parameter
Return Value	StatusCode: Motion instruction execution result

4.10.2.6 Jump3CP Three-Point Jump CP Instruction

Method Name	BasScript.BasMotion.Jump3CP(MovePoseType `poseType` , int `poseIndex` , double `speedValue` , SmoothType `smoothType` , double `smoothDistance` = 0, ExtraParam `extraParam` = null)
Description	Gate-type motion instruction, specifies the robot to perform gate-type motion, including departure, approach, and target positions, corresponds to JUMP3CP instruction in robot program writing
Request Parameters	`poseType` : MovePoseType Target pose storage type `poseIndex` : int 3 target position indices `speedValue` : double Motion speed value, unit: mm/s `smoothType` : SmoothType Smooth type `smoothDistance` : double Smooth distance, unit: mm, value range: 0~1000 `extraParam` : ExtraParam Extra parameter
Return Value	StatusCode: Motion instruction execution result

4.10.3 BasLogical Logical Instruction Subclass

BasLogical subclass contains all logic control-related methods, used for building logic control instruction sequences.

4.10.3.1 IF Conditional Instruction

Method Name	BasScript.BasLogical.IF(param1, index, param2, value, operatorType)
Description	Adds a logical IF statement to the script
Request Parameters	`param1` : First parameter, type RegisterType or IOType `index` : Index (integer) `param2` : Second parameter, type RegisterType, IOType, or OtherType `value` : Value, type index, number, string, or IOStatus `operatorType` : Boolean operator, default is equal
Return Value	StatusCode: Operation execution result

4.10.3.2 ELSE_IF Conditional Branch Instruction

Method Name	BasScript.BasLogical.ELSE_IF(param1, index, param2, value, operatorType)
Description	Adds a logical ELSE IF statement to the script
Request Parameters	`param1` : First parameter, type RegisterType or IOType `index` : Index (integer) `param2` : Second parameter, type RegisterType, IOType, or OtherType `value` : Value, type index, number, string, or IOStatus `operatorType` : Boolean operator, default is equal
Return Value	StatusCode: Operation execution result

4.10.3.3 ELSE Instruction

Method Name	BasScript.BasLogical.ELSE()
Description	Adds a logical ELSE statement to the script
Request Parameters	None
Return Value	StatusCode: Operation execution result

4.10.3.4 END_IF End Conditional Instruction

Method Name	BasScript.BasLogical.END_IF()
Description	Ends the logical IF statement
Request Parameters	None
Return Value	StatusCode: Operation execution result

4.10.3.5 WHILE Loop Instruction

Method Name	BasScript.BasLogical.WHILE(param1, index, param2, value, operatorType)
Description	Adds a logical WHILE statement to the script
Request Parameters	param1 : First parameter, type RegisterType or IOType index : Index (integer) param2 : Second parameter, type RegisterType, IOType, or OtherType value : Value, type index, number, string, or IOStatus operatorType : Boolean operator, default is equal
Return Value	StatusCode: Operation execution result

4.10.3.6 END_WHILE End Loop Instruction

Method Name	BasScript.BasLogical.END_WHILE()
Description	Ends the logical While statement
Request Parameters	None
Return Value	StatusCode: Operation execution result

4.10.3.7 SWITCH Multi-Branch Selection Instruction

Method Name	BasScript.BasLogical.SWITCH(param, index)
Description	Adds a logical SWITCH statement to the script
Request Parameters	`param` : Parameter, type RegisterType or IOType `index` : Index of the parameter
Return Value	StatusCode: Operation execution result

4.10.3.8 CASE Branch Instruction

Method Name	BasScript.BasLogical.CASE(param, value)
Description	Adds a logical CASE statement to the script
Request Parameters	`param` : Parameter, type RegisterType, IOType, or OtherType `value` : Value, type index, number, string
Return Value	StatusCode: Operation execution result

4.10.3.9 DEFAULT Branch Instruction

Method Name	BasScript.BasLogical.DEFAULT()
Description	Adds a logical DEFAULT statement to the script
Request Parameters	None
Return Value	StatusCode: Operation execution result

4.10.3.10 END_SWITCH End Multi-Branch Selection Instruction

Method Name	BasScript.BasLogical.END_SWITCH()
Description	Ends the logical SWITCH statement
Request Parameters	None
Return Value	StatusCode: Operation execution result

4.10.3.11 SKIP_CONDITION Skip Condition Instruction

Method Name	BasScript.BasLogical.SKIP_CONDITION(param1, index, param2, value, operatorType)
Description	Adds a logical SKIP CONDITION statement to the script
Request Parameters	param1 : First parameter, type RegisterType or IOType index : Index of parameter 1 param2 : Second parameter, type RegisterType, IOType, or OtherType value : Value, type index, number, string, or IOStatus operatorType : Boolean operator, default is equal
Return Value	StatusCode: Operation execution result

4.10.3.12 GOTO Jump Instruction

Method Name	BasScript.BasLogical.GOTO(index)
Description	GOTO jump statement
Request Parameters	`index` : Index of the target label
Return Value	StatusCode: Operation execution result

4.10.3.13 LABEL Instruction

Method Name	BasScript.BasLogical.LABEL(index)
Description	LABEL statement
Request Parameters	`index` : Index of the label
Return Value	StatusCode: Operation execution result

4.10.3.14 BREAK Break Out of Loop Instruction

Method Name	BasScript.BasLogical.BREAK()
Description	BREAK statement
Request Parameters	None
Return Value	StatusCode: Operation execution result

4.10.3.15 CONTINUE Skip Loop Instruction

Method Name	BasScript.BasLogical.CONTINUE()
Description	CONTINUE statement
Request Parameters	None

Method Name	BasScript.BasLogical.CONTINUE()
Return Value	StatusCode: Operation execution result

4.10.4 BasStructure Structural Instruction Subclass

BasStructure subclass contains all structure control-related methods, used for building structure control instruction sequences.

4.10.4.1 WAIT Wait Condition Instruction

Method Name	BasScript.BasStructure.WAIT(RegisterType `param1` , int `index` , ValueType `param2` , object `value` , BooleanOperator `operatorType` = BooleanOperator.EQ)
Description	Wait condition instruction, executes WAIT instruction only when the condition is met, the program can continue to execute downward, otherwise it waits until the condition is met, corresponds to WAIT COND statement in robot program writing
Request Parameters	`param1` : RegisterType First parameter (register or IO signal) `index` : int Index of parameter 1 `param2` : ValueType Second parameter (register, IO signal, or other type) `value` : object Index or value of parameter 2 `operatorType` : BooleanOperator Logical operator, default is equal
Return Value	StatusCode: Operation execution result

4.10.4.2 WAIT_TIME Wait Time Instruction

Method Name	BasScript.BasStructure.WAIT_TIME(ValueType `param` , double `value`)
Description	Wait time instruction, executes WAIT TIME instruction, the robot waits for the specified time before continuing to execute subsequent instructions, corresponds to WAIT TIME statement in robot program writing
Request Parameters	`param` : ValueType Parameter type (R register or value) `value` : double Time value, unit: sec
Return Value	StatusCode: Operation execution result

4.10.4.3 PAUSE Instruction

Method Name	BasScript.BasStructure.PAUSE()
Description	PAUSE statement
Request Parameters	None
Return Value	StatusCode: Operation execution result

4.10.4.4 ABORT Instruction

Method Name	BasScript.BasStructure.ABORT()
Description	ABORT statement
Request Parameters	None
Return Value	StatusCode: Operation execution result

4.10.4.5 CALL Synchronous Program Call Instruction

Method Name	BasScript.BasStructure.CALL(name)
Description	CALL synchronous program call
Request Parameters	`name` : Program name
Return Value	StatusCode: Operation execution result

4.10.4.6 RUN Asynchronous Program Call Instruction

Method Name	BasScript.BasStructure.RUN(name)
Description	RUN asynchronous program call
Request Parameters	`name` : Program name
Return Value	StatusCode: Operation execution result

4.10.4.7 LOAD Load Program Instruction

Method Name	BasScript.BasStructure.LOAD(param, value)
Description	LOAD load program
Request Parameters	`param` : Parameter, R register, SR register, number, or string `value` : Value of the parameter, number or string
Return Value	StatusCode: Operation execution result

4.10.4.8 UNLOAD Unload Program Instruction

Method Name	BasScript.BasStructure.UNLOAD(param, value)
Description	UNLOAD unload program

Method Name	BasScript.BasStructure.UNLOAD(param, value)
Request Parameters	`param` : Parameter, R register, SR register, number, or string `value` : Value of the parameter, number or string
Return Value	StatusCode: Operation execution result

4.10.4.9 EXEC Execute Program Instruction

Method Name	BasScript.BasStructure.EXEC(param, value)
Description	EXEC execute program
Request Parameters	`param` : Parameter, R register, SR register, number, or string `value` : Value of the parameter, number or string
Return Value	StatusCode: Operation execution result

4.10.5 BasSocket Socket Communication Subclass

BasSocket subclass contains all Socket communication-related methods, used for building Socket communication instruction sequences.

4.10.5.1 OPEN Open Socket Connection Instruction

Method Name	BasScript.BasSocket.OPEN(int `index`)
Description	Uses Socket Open instruction to create a Server and wait for Client connection, corresponds to SOCKET OPEN in robot program writing
Request Parameters	`index` : int SK register index
Return Value	StatusCode: Operation execution result

4.10.5.2 CLOSE Close Socket Connection Instruction

Method Name	BasScript.BasSocket.CLOSE(int `index`)
Description	Closes the specified socket connection, corresponds to SOCKET CLOSE in robot program writing
Request Parameters	`index` : int SK register index
Return Value	StatusCode: Operation execution result

4.10.5.3 CONNECT Socket Connection Instruction

Method Name	BasScript.BasSocket.CONNECT(int `index`)
Description	Uses Socket Connect instruction to connect to the specified socket server, corresponds to SOCKET CONNECT in robot program writing
Request Parameters	`index` : int SK register index
Return Value	StatusCode: Operation execution result

4.10.5.4 SEND Send Socket Data Instruction

Method Name	BasScript.BasSocket.SEND(int `index` , StrType `msgType` , object `value`)
Description	Uses Socket Send instruction to send data to the specified socket connection, corresponds to SOCKET SEND in robot program writing
Request Parameters	`index` : int SK register index `msgType` : StrType Message type `value` : object Message content or index
Return Value	StatusCode: Operation execution result

4.10.5.5 RECV Receive Socket Data Instruction

Method Name	BasScript.BasSocket.RECV(int `index` , int `msgLength` , StrType `msgType` , object `value`)
Description	Uses Socket Recv instruction to read characters from the specified socket connection, can specify maximum length, corresponds to SOCKET RECV in robot program writing
Request Parameters	`index` : int SK register index `msgLength` : int Message maximum length `msgType` : StrType Message type `value` : object Message content or index
Return Value	StatusCode: Operation execution result

4.10.6 BasModbus Modbus Communication Subclass

BasModbus subclass contains all Modbus communication-related methods, used for building Modbus register read/write instruction sequences.

4.10.6.1 READ_MH Read Modbus Holding Register Instruction

Method Name	BasScript.BasModbus.READ_MH(int `index` , int `id` , int `address` , int `length` , int `rIndex`)
Description	Read Modbus holding register instruction, corresponds to READ_MH in robot program writing
Request Parameters	`index` : int Channel index `id` : int Modbus ID `address` : int Register start address `length` : int Register length, i.e., number of registers to read `rIndex` : int R register start index to write result

Method Name	BasScript.BasModbus.READ_MH(int `index` , int `id` , int `address` , int `length` , int `rIndex`)
Return Value	StatusCode: Operation execution result

4.10.6.2 READ_MI Read Modbus Input Register Instruction

Method Name	BasScript.BasModbus.READ_MI(int `index` , int `id` , int `address` , int `length` , int `rIndex`)
Description	Read Modbus input register instruction, corresponds to READ_MI in robot program writing
Request Parameters	`index` : int Channel index `id` : int Modbus ID `address` : int Register start address `length` : int Register length, i.e., number of registers to read `rIndex` : int R register start index to write result
Return Value	StatusCode: Operation execution result

4.10.6.3 WRITE_MH Write Modbus Holding Register Instruction

Method Name	BasScript.BasModbus.WRITE_MH(int `index` , int `id` , int `address` , int `length` , ValueType `valueType` , int `value`)
Description	Write Modbus holding register instruction, corresponds to WRITE_MH in robot program writing
Request Parameters	`index` : int Channel index `id` : int Modbus ID `address` : int Register start address `length` : int Register length, i.e., number of registers to write `valueType` : ValueType Value type `value` : int Value or R register start index

Method Name	BasScript.BasModbus.WRITE_MH(int `index` , int `id` , int `address` , int `length` , ValueType `valueType` , int `value`)
Return Value	StatusCode: Operation execution result

4.10.7 BasVision Vision Instruction Subclass

BasVision subclass contains all vision-related methods, used for building vision program instruction sequences.

4.10.7.1 FIND Find Vision Program Instruction

Method Name	BasScript.BasVision.FIND(string `name`)
Description	Executes vision find program, corresponds to VISION FIND in robot program writing
Request Parameters	`name` : string Vision program name
Return Value	StatusCode: Operation execution result

4.10.7.2 GET_OFFSET Get Vision Program Offset Instruction

Method Name	BasScript.BasVision.GET_OFFSET(string `name` , int `index` , int `labelIndex`)
Description	Gets offset after vision program execution, corresponds to VISION GET OFFSET in robot program writing
Request Parameters	`name` : string Vision program name `index` : int Vision register index `labelIndex` : int Label index
Return Value	StatusCode: Operation execution result

4.10.7.3 GET_QUANTITY Get Vision Program Result Instruction

Method Name	BasScript.BasVision.GET_QUANTITY(string `name` , int `index`)
Description	Gets result quantity after vision program execution, corresponds to VISION GET QUANTITY in robot program writing
Request Parameters	`name` : string Vision program name `index` : int R register index to store result
Return Value	StatusCode: Operation execution result

4.10.8 AssignValue Assignment Instruction (Full Parameters)

Method Name	BasScript.AssignValue(AssignType `param1` , int `index` , AssignType `param2` , object `value` , int `optIndex` = 0, int `optValue` = 0)
Description	Assignment instruction, used to assign values to registers, IO signals, etc., corresponds to ASSIGN statement in robot program writing
Request Parameters	`param1` : AssignType First parameter (register or IO signal) `index` : int Index of parameter 1 `param2` : AssignType Second parameter (register, IO signal, or other type) `value` : object Index or value of parameter 2 `optIndex` : int Additional index for parameter 1 when param1 is PR_ELEMENT, default 0 `optValue` : int Additional index for parameter 2 when param2 is PR_ELEMENT or pulse value when value is IOStatus.PULSE, default 0
Return Value	StatusCode: Operation execution result

4.10.9 AssignValue Assignment Instruction (Simplified Parameters)

Method Name	BasScript.AssignValue(AssignType `param` , int `index` , object `value`)
Description	Assignment instruction (simplified parameters), used to assign values to registers, IO signals, etc., corresponds to ASSIGN statement in robot program writing
Request Parameters	`param` : AssignType Parameter type (register or IO signal) `index` : int Parameter index `value` : object Value (IOStatus, double, or string)
Return Value	StatusCode: Operation execution result

4.10.10 SetParam Set Parameter Instruction

Method Name	BasScript.SetParam(ParamType `type` , ValueType `valueType` , object `value`)
Description	Set parameter instruction, used to set various parameters of the robot, corresponds to SET PARAM in robot program writing
Request Parameters	`type` : ParamType Parameter type `valueType` : ValueType Value type `value` : object Value
Return Value	StatusCode: Operation execution result

4.11 Coordinate System Class

Overview

The `CoordinateSystem` class is used to manage robot User Frames (UF) and Tool Frames (TF), and provides unified interfaces for adding, deleting, updating, querying, and calculating coordinate systems.

Core Features

- Support getting user/tool coordinate system summary information list
- Support adding, deleting, updating, and querying user/tool coordinate systems
- Support calculating user/tool coordinate systems based on multiple input poses
- Provide a unified coordinate system management interface to facilitate maintenance of coordinate system lists in the controller

Use Cases

- Manage different tool coordinate systems during tool switching
- Maintain consistent spatial references during multi-station debugging
- Batch management and querying of robot coordinate systems
- Quickly calculate new user/tool coordinate systems from taught points

4.11.1 Get User/Tool Coordinate System List

Method Name	CoordinateSystem.GetCoordinateList(CoordinateType `type`)
Description	Gets all coordinate system summary information list for the specified coordinate type.
Request Parameters	`type` : CoordinateType Coordinate type (UF or TF)

Method Name	CoordinateSystem.GetCoordinateList(CoordinateType `type`)
Return Value	List<CoordSummary>: Coordinate summary list StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.11.2 Add User/Tool Coordinate System

Method Name	CoordinateSystem.Add(CoordinateType `type` , Coordinate `coordinate`)
Description	Adds a new coordinate system for the specified coordinate type.
Request Parameters	`type` : CoordinateType Coordinate type (UF or TF) `coordinate` : Coordinate Coordinate data to add
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.11.3 Delete User/Tool Coordinate System

Method Name	CoordinateSystem.Delete(CoordinateType `type` , int `index`)
Description	Deletes the coordinate system specified by coordinate type and index.
Request Parameters	`type` : CoordinateType Coordinate type (UF or TF) `index` : int Coordinate index
Return Value	StatusCode: Operation execution result

Method Name	CoordinateSystem.Delete(CoordinateType `type` , int `index`)
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.11.4 Update User/Tool Coordinate System

Method Name	CoordinateSystem.Update(CoordinateType `type` , Coordinate `coordinate`)
Description	Updates the coordinate system for the specified coordinate type and coordinate information.
Request Parameters	`type` : CoordinateType Coordinate type (UF or TF) `coordinate` : Coordinate Coordinate data to update
Return Value	StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.11.5 Get User/Tool Coordinate System Information

Method Name	CoordinateSystem.Get(CoordinateType `type` , int `index`)
Description	Gets coordinate system information by coordinate type and index.
Request Parameters	`type` : CoordinateType Coordinate type (UF or TF) `index` : int Coordinate index
Return Value	Coordinate: Coordinate data StatusCode: Operation execution result

Method Name	CoordinateSystem.Get(CoordinateType `type` , int `index`)
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

4.11.6 Calculate User/Tool Coordinate System

Method Name	CoordinateSystem.Calculate(CoordinateType `type` , List<Position> `pose`)
Description	Calculates a user/tool coordinate system from input pose points.
Request Parameters	`type` : CoordinateType Coordinate type (UF or TF) `pose` : List<Position> Input pose list. Supports 4-point or 7-point method. In 4-point method, provide 4 poses where the tool points to the same target point. In 7-point method, additionally provide origin, X-direction point, and Y-direction point. Angle unit: degree (°).
Return Value	Position: Calculated coordinate pose StatusCode: Operation execution result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Robot Jogging Motion

Overview

The `Jogging` class provides jog-control APIs in manual teaching mode, including single-axis stepping, single-axis continuous jog, multi-axis continuous jog, and stop operations.

Core Features

- Single-axis step jogging
- Single-axis continuous jogging
- Multi-axis continuous jogging
- Stop continuous jogging
- Direction control with positive/negative axis signs
- Configurable linear step length and rotational step angle

Use Cases

- Robot debugging and teaching process
- Fine position and posture adjustment
- Host-side remote manual pose adjustment
- Robot installation and calibration
- Position confirmation before executing complex trajectories

4.12.1 Single-Axis Step Jogging

Method Name	Jogging.Move(int `ajNum` , MoveMode `moveMode` , double `stepLength` = 0, double `stepAngle` = 0)
Description	Executes single-axis step jogging (`moveMode` = `MoveMode.Stepping`).

Method Name	Jogging.Move(int `ajNum` , MoveMode `moveMode` , double `stepLength` = 0, double `stepAngle` = 0)
Request Parameters	`ajNum` : int Axis index (1~6 maps to current coordinate-system axes; in Cartesian mode x/y/z/rx/ry/rz map to 1~6; sign indicates direction). `moveMode` : Move mode, fixed to `MoveMode.Stepping` for this scenario. `stepLength` : double Linear step length in mm (effective in stepping mode). `stepAngle` : double Rotational step angle in ° (effective in stepping mode).
Return Value	StatusCode: Jog operation result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Single-Axis Continuous Jogging

Method Name	Jogging.Move(int `ajNum` , MoveMode `moveMode` , double `stepLength` = 0, double `stepAngle` = 0)
Description	Executes single-axis continuous jogging (`moveMode` = `MoveMode.Continuous`).
Request Parameters	`ajNum` : int Axis index (1~6 maps to current coordinate-system axes; in Cartesian mode x/y/z/rx/ry/rz map to 1~6; sign indicates direction). `moveMode` : Move mode, fixed to `MoveMode.Continuous` for this scenario. `stepLength` : double Step parameter (typically unused in continuous mode). `stepAngle` : double Step-angle parameter (typically unused in continuous mode).
Return Value	StatusCode: Jog operation result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Multi-Axis Continuous Jogging

Method Name	Jogging.MultiMove(int[] `ajNums`)
Description	Executes simultaneous continuous jogging on multiple axes.
Request Parameters	`ajNums` : int[] Axis list (1~6 maps to current coordinate-system axes; in Cartesian mode x/y/z/rx/ry/rz map to 1~6; sign indicates each axis direction).
Return Value	StatusCode: Jog operation result
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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 Stop Continuous Jogging

Method Name	Jogging.Stop()
Description	Stops the current continuous jog movement.
Request Parameters	None
Return Value	void
Note	This method is required only in continuous jogging mode.
Compatible robot software version	Collaborative (Copper): v7.5.0.0+ Industrial (Bronze): v7.5.0.0+

Example Code

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.GetDescription()}"
                );
            }

            // [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 Successfully"
                );
                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 Robot Subscription & Publish Interface

Overview

The `SubPub` class provides WebSocket subscription/publish channel management for the robot controller. It is used to establish connections, subscribe to robot status/register/IO topics, and receive real-time data through callback or blocking APIs.

Core Features

- Support connecting to and disconnecting from WebSocket server
- Support subscribing to robot status data
- Support subscribing to register data
- Support subscribing to IO signal data
- Support continuously receiving messages via callback function
- Support blocking reception of the next message
- Support setting subscription frequency
- Support handling exceptions and troubleshooting during message reception

Use Cases

- Host-side real-time monitoring of robot operating status
- Implementing robot data visualization
- Achieving data linkage with external systems
- Real-time monitoring of register and IO states
- Building robot monitoring and control systems
- Implementing real-time alarm and notification for robot status

4.13.1 Connect to WebSocket Server

Method Name	SubPub.Connect()
Description	Connects to the robot controller WebSocket server
Request Parameters	None
Return Value	Task: Asynchronous connection operation result
Compatible robot software version	Collaborative (Copper): v7.7.0.0+ Industrial (Bronze): v7.7.0.0+

4.13.2 Disconnect from WebSocket Server

Method Name	SubPub.Disconnect()
Description	Disconnects from the robot controller WebSocket server
Request Parameters	None
Return Value	Task: Asynchronous disconnect operation result
Compatible robot software version	Collaborative (Copper): v7.7.0.0+ Industrial (Bronze): v7.7.0.0+

4.13.3 Add Robot Status Subscription

Method Name	SubPub.SubscribeStatus(RobotTopicType[] `topicTypes` , int `frequency` = 200)
Description	Adds robot status data subscription
Request Parameters	`topicTypes` : RobotTopicType[] Robot topic type list `frequency` : int Subscription frequency (unit: Hz, default 200)
Return Value	Task: Asynchronous subscription operation result
Compatible robot software version	Collaborative (Copper): v7.7.0.0+ Industrial (Bronze): v7.7.0.0+

4.13.4 Add Register Subscription

Method Name	SubPub.SubscribeRegister(RegTopicType `regType` , int[] `regIds` , int `frequency` = 200)
Description	Adds register data subscription
Request Parameters	`regType` : RegTopicType Register type `regIds` : int[] Register ID list `frequency` : int Subscription frequency (unit: Hz, default 200)
Return Value	Task: Asynchronous subscription operation result
Compatible robot software version	Collaborative (Copper): v7.7.0.0+ Industrial (Bronze): v7.7.0.0+

4.13.5 Add IO Subscription

Method Name	SubPub.SubscribeIO((IOTopicType, int)[] `ioList` , int `frequency` = 200)
Description	Subscribes to IO signal data, including digital input/output, etc.
Request Parameters	`ioList` : (IOTopicType, int)[] IO list (each element is `(IO Type, IO ID)`) `frequency` : int Subscription frequency (unit: Hz, default 200)
Return Value	Task: Asynchronous subscription operation result
Compatible robot software version	Collaborative (Copper): v7.7.0.0+ Industrial (Bronze): v7.7.0.0+

4.13.6 Start Receiving Messages

Method Name	SubPub.StartReceiving(Func<Dictionary<string, object>, Task> `onMessageReceived`)
Description	Starts receiving subscription messages and processes the received data via callback function.
Request Parameters	`onMessageReceived` : Func<Dictionary<string, object>, Task> Message receiving callback function
Return Value	Task: Asynchronous receiving task
Note	If the callback throws an exception, the receiving loop may stop. It is recommended to catch and log exceptions inside the callback.
Compatible robot software version	Collaborative (Copper): v7.7.0.0+ Industrial (Bronze): v7.7.0.0+

Example Code

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 Receive Next Message

Method Name	SubPub.Receive()
Description	Blocks to receive the next message and returns it.
Request Parameters	None
Return Value	Task<Dictionary<string, object>>: Received message dictionary
Note	Exceptions may be thrown when the connection is not established, when the connection is closed, or when message format is abnormal.
Compatible robot software version	Collaborative (Copper): v7.7.0.0+ Industrial (Bronze): v7.7.0.0+

Example Code

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 Update Notes

2.1.0.* Update (2026/4/9)

1. Added UDP feedback configuration related interfaces
2. `Motion.SetPositionTrajectoryParams` - Added `filterLayer` parameter to the interface for specifying UDP position control related parameters
3. `Motion.Payload.GetPayloadIdentifyState` - Updated message parsing for the payload identification status interface
4. Added `TerminatePayloadIdentify` interface for terminating payload identification
5. Added trajectory table and path table related interfaces
6. Added trajectory shaper related interfaces
7. Added path planning parameters related interfaces

2.0.3.* Update (2025/12/17)

1. Arm constructor now includes teachPanelIP parameter.
2. Removed System.Text.Json dependency.
3. Added synchronous connection interface ConnectSync.

2.0.2.* Update (2025/12/12)

1. Fixed the incorrect read/write order of data in the PR register struct.

2.0.1.* Update (2025/10/21)

1. Fixed the stuttering issue during continuous jogging.
2. Fixed a build-time error where the proxy executable might fail to copy.

2.0.0.* (2025-09-10)

1. Refactored the underlying request pattern and added a local controller proxy service.
2. Introduced subscription functionality.
3. Reorganized the BasScript class structure.
4. Full support for both .NET Framework and .NET (Core/5+/6+).

1.0.1.0 (July 7, 2025)

1. Added the old register interface class `RegistersOld` to be compatible with robot versions prior to 7.6.0.0
2. Added the Estop emergency braking interface
3. Fixed the example programs in the documentation

1.0.0.0 (May 30, 2025)

1. Implemented using RPC method.
2. Synchronized all interface definitions with the Python version.

Help

Agent Skills

Agent Skills extend AI agents (such as Codex and Claude Code) with standardized execution workflows and tooling for industrial or development scenarios.

Compatible Agent Types

Agent Skills currently support automatic detection and installation for the following AI tools and CLIs:

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

Quick Start

1. Environment Preparation

Before installing, make sure Node.js is installed locally. Guide: Node.js Download and Installation

2. Run the Install Command

Run the following command in your terminal to enter the interactive installer:

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

TIP

After running the command, the tool scans locally installed agents. Follow the prompts to select which agents to install the skill into (for example, Cursor or Claude Code).

Available Skills

agilebot-nlu-control (Natural Language Control for Agilebot)

This skill is designed for Agilebot robots. It uses NLU (Natural Language Understanding) so the agent can parse and execute control instructions directly.

- Environment requirements: system Python must be available and Conda must be installed. The skill checks required dependencies and guides installation if anything is missing.
- Trigger: when the conversation mentions "Agilebot" and provides a concrete control intent.
- Core features: natural-language command parsing and execution, robot status queries, basic motion control, parameterized motion commands (e.g., distance/angle/speed).

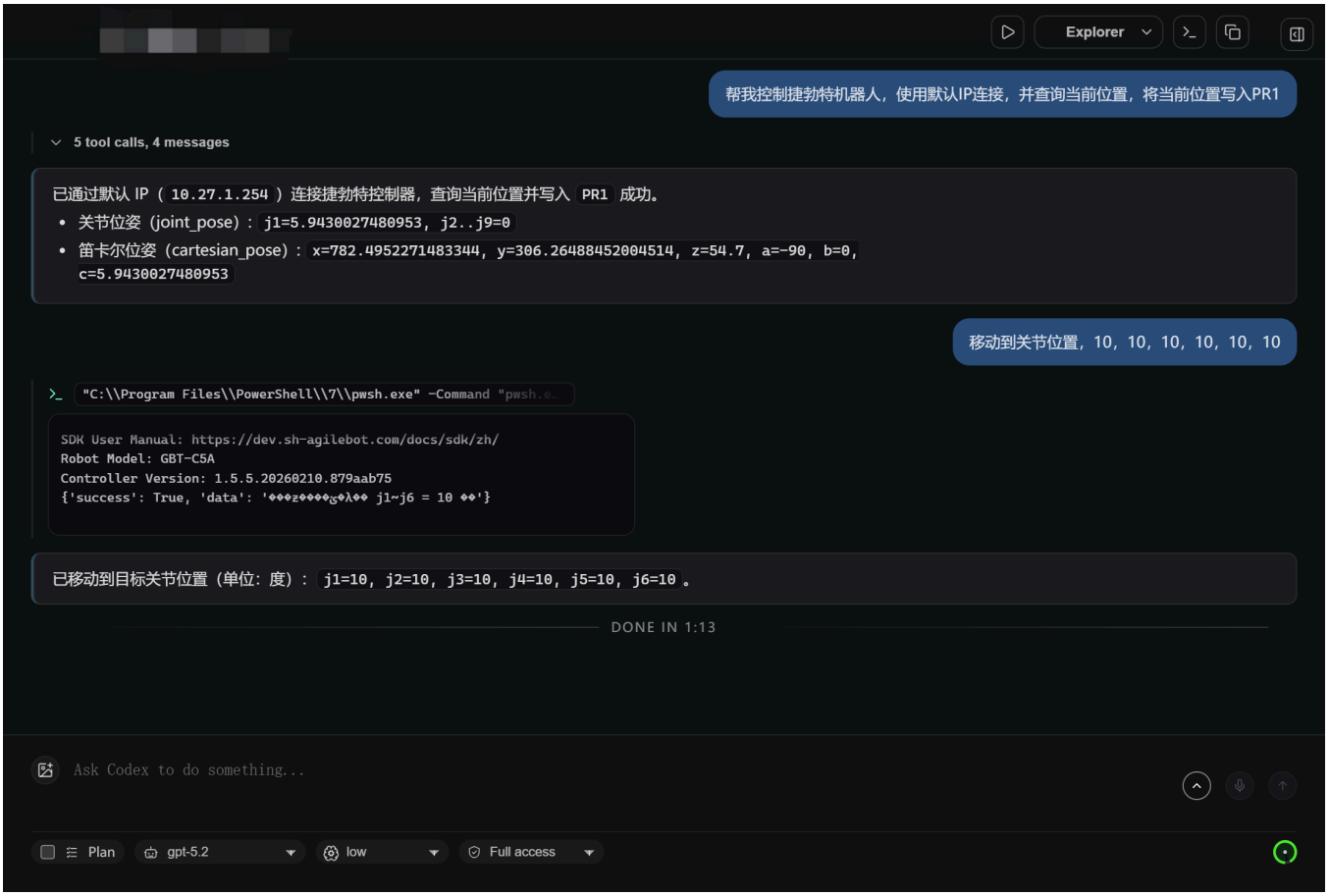
Warning

Installed skills run with full agent permissions. Make sure you are operating in a trusted environment before use.

Robot motion can be hazardous and may cause injury or equipment damage. Use in a safe environment. We are not responsible for any risks or losses arising from usage.

Usage Example

Using Codex as an example, after installation you can enter instructions directly in the chat:



AI Coding Support

This document describes how to use AI assistance tools (such as CodeBuddy, Codex, Cursor, etc.) to quickly develop robot programs.

Preparation

Before using AI coding, you need to prepare the reference documentation:

- **SDK Documentation**: https://dev.svfactory.com/docs/sdk/knowledge/docs.txt

> **Tip**: If your AI agent cannot read URLs well, download the txt document above to your local project directory and reference the local file path in your prompt.

Example Prompt

Here is a complete example for creating a Python program that reads robot status:

```
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.svfactory.com/docs/sdk/knowledge/docs.txt
```

If you rely on local documentation, you can modify it to:

```
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
```

Usage Tips

1. **Clear Requirements**: Clearly describe the functionality you want to implement
2. **Provide Context**: Reference relevant documentation and examples
3. **Stepwise Implementation**: Complex features can be generated step by step with AI

Notes

1. Code generated by AI needs to be verified and tested
2. Ensure code complies with project coding standards
3. Code involving robot control must undergo security review