Agito-AAmotion User Programming Manual v1.1.0

agito

www.agito-akribis.com

Member of Akribis Systems group

Version history

Version	Description	Author	Date
1.0.0	Agito-AAMotion User Programming Manual (for AAMotion versions 7.0.0 and later)	Jin YaShuang	2025/5/
1.1.0	Added AAMotionAPI vector motion method Add axis keywords Add vector keywords	Jin YaShuang	2025/6/

※The company reserves the right to update from time to time. According to the upgrade or update iteration of product hardware and software and market demand, this manual will be updated and adjusted from time to time without prior notice, if you need the latest version of the document, please contact Agito-Akribis for corresponding support.

Contents

Introduction

The AAMotion library is an advanced API designed specifically for Agito motion controllers for Windows and Linux platforms. This library provides a wealth of interfaces for the AAMotion series motion controllers, which greatly facilitates the development of motion control systems. Key features include controller connection, commutation, homing, IDE+ control, single-axis and multi-axis motion, I/O module management, CNC mode, position lock, position event (PEG), force control, vector motion, and configuration and management of position events.

AAMotion supports both object-oriented and process-oriented programming methods, which can be flexibly developed through classes and objects, or directly used the AAMotion API to quickly implement various functions in the manual in an object-oriented manner. Whether it's complex system integration or simple application development, AAMotion provides efficient and reliable support for your motion control projects.

1 AAMotion User Guide

AAMotion provides an object-oriented API interface with a clear structure and ease of use. The sample code prefers C#, but the API also supports multiple language interfaces such as Python, VB6, Java, and LabVIEW (please contact official support for the C++ version). In addition, users can choose ASCII code communication via RS232, RS485, Ethernet, or CAN buses.

1.1 Create a blank project

1. Open Visual Studio.
2. Select File > New > Project.
3. Select Console Application or another applicable project type (e.g., Library, WPF, etc.).
4. Name the project and select the storage path, then click Create.

1.2 Install the NuGet package

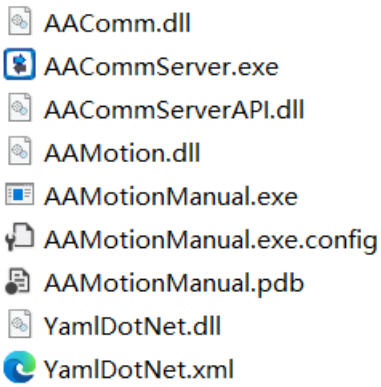
1. In the Tools menu, click NuGet Package Manager > Manage NuGet Packages for Solution.
2. In the Browse tab, search for **Agito.AAMotion**.
3. Once you find the package, tap "Install".
4. Wait for the installation to be complete and confirm that there are no abnormalities.

Or run the command in the Package Manager Console:

```
Install-Package Agito.AAMotion
```

1.3 Import library files

Once installed, the relevant DLL files are automatically added to the project or found in the project's output directory.



In the bin/Debug directory of your project, you will see the following DLL files:

Filename	Function Description
AAComm.dll	Communication-related API libraries for device communication.

Filename	Function Description
AAMotion.dll	Motion control related API libraries to operate mechanical motion.
YamDontNet.dll	Dependent standard libraries or extension libraries (used by .Net).

1.4 Reference DLL (if necessary)

If you are using a manual import method (not an automatic import of NuGet packages):
Right-click References in Solution Explorer > select Add References > Select dll file in bin/Debug directory and confirm the addition.

Suggestion:
Add a statement to the code file

```
using AAMotion;
```

1.5 Start programming

Once you have completed the above steps, you can use the motion control AAMotion API (object-oriented) in your code, example:

```
// Reference namespace
using AAMotion;
class Program
{
    static void Main(string[] args)
    {
        // Initialize the controller (example)
        var controller = AAMotionAPI.Initialize(ControllerType.AGM800);
        // Connect the controller
        bool isConnected = AAMotionAPI.Connect(controller,”172.1.1.101”);

        if (isConnected)
        {
            Console.WriteLine("Controller is connected!");
              AAMotionAPI.MotorOn(controller, AxisRef.A); // Enables A-axis
motors
            // Subsequent motion control code
        }
        else
        {
            Console.WriteLine("Connection failed, please check hardware and
network settings.");
        }
    }
}
```

1.6 Remark

- Make sure the DLL file is loaded correctly into the project.
- You need to adjust the parameters according to the actual hardware connection configuration.
- You can refer to the API documentation for more advanced operations.

2 Controller initialization and connection

This section describes how to use AAMotionAPI to initialize and connect controllers, including supported controller types and connection methods.

2.1 Initialize the controller

Before you do anything, you need to initialize the controller instance.

Code example

```
Public MotionController _controller =
AAMotionAPI.Initialize(ControllerType.AGM800);
```

Description

- `AAMotionAPI.Initialize(ControllerType controllerType)`: Initialization of controller instance based on the controller type.
- The entry parameter is the enumeration type `ControllerType`, which supports multiple hardware models.

Supported controller types

Enumerating ControllerType defines all supported controller types and their corresponding numbers.

Enum Value	Description	Number
AGD200	Controller model AGD200	5
AGD155	Controller model AGD155	9
AGM800	Controller model AGM800	10
AGD301	Controller model AGD301	11
AGD155EC	Controller model AGD155EC	12
AGD101EC	Controller model AGD101EC	13
AGD156EC	Controller model AGD156EC	14

2.2 Connect to the Controller API

After initialization, you need to connect the controller for the next step.

Connect (Connect to the controller)

Method 1: Connect to the Default IP (172.1.1.101)

	Bool Connect(MotionController controller)	
Description	Connect the controller	
Parameter	Controller	Controller instance
Return value	The connection successfully returns true, otherwise false	
Note	Connect to the default controller IP address "172.1.1.101"	

Method 2: Connect to the specified IP address

	Bool Connect(MotionController controller, string deviceAddress)	
Description	Connect the controller	
Parameter	Controller	Controller instance
	deviceAddress	The IP address of the controller which is connected
Return value	The connection successfully returns true, otherwise false	
Note	Connect the controller (IP address can be filled)	

Disconnect

	Bool Disconnect(MotionController controller)	
Description	Disconnect the controller	
Parameter	Controller	Controller instance
Return value	Disconnect successfully returns true, otherwise false	
Note	Disconnect the current controller connection	

2.3 Usage examples

```
// 1. Initializing the controller (using the AGM800)
MotionController                        controller                              =
AAMotionAPI.Initialize(ControllerType.AGM800);

 2. Connect the controller (use the default IP: 172.1.1.101).
AAMotionAPI.Connect(controller);
```

```
Alternatively, connect to a specific IP address
// AAMotionAPI.Connect(controller, "192.168.0.100");
```

2.4 Frequently Asked Questions

Q: What should I do if the connection fails?
A: Make sure the controller device is properly connected to the network, the IP address is correct, and the controller is powered on. You can try to reinitialize or check network connectivity.

Q: How do I switch controllers or change IP addresses?
A: In PCSuite-Configuration-Communications. After the change is saved to the flash memory, the device is powered back to use the new IP address.

3 General Function

3.1 Controller Exception Event Handling

Use the ErrorOccurred event to catch the error

If an error occurs during motion or runtime, or if a command fails to be sent using the AAMotion API, the error code and error message will be passed to the application internally through the ErrorOccurred event. Sample code:

```
{
    InitializeComponent();
    // Instantiate the controller
public              MotionController              _controller              =
AAMotionAPI.Initialize(ControllerType.AGM800);

    // Event binding: Device error message
    controller.ErrorOccurred += (errorCode, MsgSent, errorMsg) =>
    {
        ShowLog($"Message Sent: {MsgSent}");
        ShowLog($"Error Code: {errorCode}");
        ShowLog($"Error Message: {errorMsg}");
    };
}
```

Error handling suggestions

- **Real-time response**: Take immediate action based on errorCode and errorMsg (e.g. power-off, safe shutdown, etc.).
- **Condition monitoring**: Combined with motion status detection, determine whether there is any abnormality.
- **Error code checking:** ErrLog, ConFlt

Error type

```
There are usually 2 types of errors.
Interpreter error
When you send a command, the controller may reject it
For example, the command sent: BMotorOn = 1
To catch these types of errors, you can append a callback to your ctrl.
Errors occur
Ctrl.ErrorOccurred += (errCode, msgSent, errorMsg) =>
{
MessageBox.Show(errCode.ToString());
MessageBox.Show(msgSent);
MessageBox.Show(errorMsg);
};
You will get:
Error code: 159
msgSent's BMotorOn=1
errorMsg:
Error: 159
Command Sent:BMotorOn=1
Error: This request is valid only for the active (connected) Central-i port

Controller failure
```

When a motor control-related fault occurs.
For example, the motor current is too high.
The user needs to poll ConFlt and check when it is not 0

Misuse the scope

1. Special error type handling
- SyncErr: Prefixed with ERR Sync: Usually caused by multi-threaded concurrent calls to SendReceive (SendReceive non-threaded safety).
- PC Suite Error (PcSuiteErr): Prefix PC Suite: ERR, indicating an error detected by the API layer (e.g., the message format is illegal, not sent to the controller).
- Boot Mode Error (BootErr): Prefixed with BOOT ERR, it indicates an error returned when the controller is in Boot mode.

ErrorOccurred fully covers the entire process of error capture from command sending to response reception, and performs error code parsing and formatting through AAComm, which can accurately locate the error type (communication/controller/parameter problem) and its impact scope

3.2 Turning AACommServer on and off

Init(Open AACommServer)

	bool Init(MotionController controller)	
Description	Open AACommServer (Agito Communication App)	
Parameter	MotionController controller	Controller instance
Return value	true: The instruction was sent successfully, and the operation was completed. false: Instruction failed or communication was abnormal	
Remark	Used to initialize the controller communication environment, AACommServer will be automatically started establishing the communication connection between the PC and the controller	

Shutdown(Shut down AACommServer）

	bool Shutdown(MotionController controller)	
Description	Shut down AACommServer	
Parameter	MotionController controller	Controller instance
Return value	true: The instruction was sent successfully, and the operation was completed. false: Instruction failed or communication was abnormal	
Remark	It is used to turn off the controller communication environment, stop the operation of AACommServer, and disconnect the communication between the PC and the controller.	

User Guide:

- **Automatic management**: When calling connect to establish a connection, AACommServer will automatically start, without the user needing to manually call Init().

- **Manual control**: If you need to control the opening and closing of communication in specific scenarios, you can call Init() before connecting and call Shutdown() after completing communication to ensure resource release.
- `**Init()` **and** Shutdown() ** are management methods for the underlying resources of controller communication, ensuring timely closure when communication is no longer needed to save resources.
- When calling Connect(), if the AACommServer (underlying communication process) is started (started by Init() or previously enabled), there is no need to start it repeatedly.

3.3 Send command string method

SendCommandString

	bool SendCommandString(MotionController controller, string rawCommand, out string response)	
Description	Send the specified command string to the controller and wait for a response. Suitable for the transmission of common control commands.	
Parameter	MotionController controller	Controller instance
	string rawCommand	Command string to be sent (e.g., control command, query command, etc.)
	out string response	Output parameters used to receive the response information returned by the controller
Return value	true: The command transmits successfully and gets a response, no error detected. false: A communication error occurred, and the response contained an error message.	
Remark	Transmit individual control instructions (e.g., start, stop, status query, etc.). Suitable for fast, short-instruction communication.	

SendBulkCommandString

	bool SendBulkCommandString(MotionController controller, string rawCommand, out string response)	
Description	Bulk command strings are sent to the controller, supporting larger instruction sets or composite commands, and waiting for a response.	
Parameter	MotionController controller	Controller instance
	string rawCommand	The bulk command string to be sent
	out string response	Output parameters used to receive the response information returned by the controller
Return value	true: The batch command is successfully transmitted and the response is normal. false: A communication error occurred, and the ErrorOccurred response contains the	

	error details.
Remark	Multiple instructions or large blocks of data need to be transmitted at once, up to 50 commands. Used when the system needs to be configured in batches.

Example

```
string response;
bool success = SendCommandString(controller, "ASPeed", out response);
if (success)
{
    Console.WriteLine("Command succeeded. Response: " + response);
}
```

3.4 Set the position

SetPosition

	bool SetPosition(MotionController controller, AxisRef axis, int value)	
Description	Used to set a new position reading (position reference value) for the axis. A new position value can be assigned manually, affecting position control and subsequent motion references.	
Parameter	MotionController controller	Controller instance
	AxisRef axis	Axis reference
	int value	The new position value to set
Return value	true: The command was successfully transmitted and the response was normal. false: A communication error occurred, and the ErrorOccurred response contains error details.	
Remark	SetPosition cannot be called under the following conditions, otherwise the controller will return an error code: 1. When the axis is moving. 2. Error Mapping is activated. 3. Auto gain (Auto inertia) is activated. 4. The value of SetPosition exceeds the software position limit (RevPLim to FwdPLim). 5. The servo is enabled and the input shaping is activated. - SetPosition will update the relevant internal variables to ensure smooth operation without abrupt changes. - Use only in specific maintenance or debugging scenarios to avoid affecting normal motion control	

Sample code

```
// 1. Initialize the controller
MotionController controller =
AAMotionAPI.Initialize(ControllerType.AGM800);
```

```
// 2. Reset the A-axis position to 0
AAMotionAPI.SetPosition(controller, AxisRef.A, 0);
```

4 Commutation

Commutation is used to control the direction switching of motion axis and is a common operation in motion control. The API allows you to easily execute commutation commands on a specified axis and confirm the commutation status.

4.1 Commutation API

Commutate

	Void Commutate(MotionController controller , AxisRef axisRef, int? waitTime = null)	
Description	Perform commutation on the specified axis. Set the waiting time (optional).	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	Int? waitTime	Wait time (milliseconds, optional)
Return value	None	

IsCommutated

	bool IsCommutated(MotionController controller, AxisRef axisRef)	
Description	Check if axis has completed commutation.	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
Return value	true: Commutation completed； false: Commutation has not been performed.	

4.2 Example of commutation operations

Procedure:

1. Call the Commutate() method to perform commutation.
2. Use the IsCommutated() method to confirm the commutation is complete.

Detailed sample code

```
// Specify the axis for commutation operation and wait 5000 milliseconds (5 seconds)
AAMotionAPI.Commutate(controller, AxisRef.D, 5000);

// Check whether the axis has completed commutation
bool isCommutated = AAMotionAPI.IsCommutated(controller, AxisRef.D);

// Check the commutation state
if (isCommutated)
```

```
{
    Console.WriteLine("Commutation Successful, Axis Commutation
Completed.");
    // Can add additional actions later
}
else
{
    Console.WriteLine("Commutation is incomplete. Do you need to execute
the process again?");
    // Choose to re-communicate or Prompt the user
}
```

Notes:

- After calling Commutate(), call IsCommutated() to confirm whether the commutation is complete to ensure the operation is successful.
- If the commutation is not complete, consider increasing the waiting time or performing the operation again.
- Ensure that the controller and axes are in normal condition to avoid commutation failures due to abnormal equipment status.
- Make sure that the correct commutation can be performed in PCSuite, and that the correct commutation can be saved to Flash"

4.3 Axis (AxisRef) option list

Axis identification	Enum values	Description
A	1	Axis A
B	2	Axis B
C	4	Axis C
D	8	Axis D
E	16	Axis E
F	32	Axis F
G	64	Axis G
H	128	Axis H
I	256	Axis I
J	512	Axis J
K	1024	Axis K
L	2048	Axis L

5 Homing

Homing is a fundamental prerequisite for motion control by aligning the mechanical coordinate system with the physical origin.

5.1 Homing API

Home

	Void Home(MotionController controller, AxisRef axis, string filePath)	
Description	Axis homing operation, specify the homing file path	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	string filePath)	Homing Parameter Path (Exported by PCSuite)
Return value	None	

IsHoming (Verify homing success)

	bool IsHomed(MotionController controller, AxisRef axis)	
Description	Determine if the homing was successful	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
Return value	true: Homing successful; false: Homing failed	

5.2 Example of the operation process

Example of a homing code

```
// Set the homing file path
string homeFilePath = "your homing file path"; // Replace with the actual
file path

// Homing execution
AAMotionAPI.Home(controller, AxisRef.D, homeFilePath);
// Wait for some time before checking
bool isHomed = AAMotionAPI.IsHomed(controller, AxisRef.D);
if (isHomed)
{
    Console.WriteLine("Homing was successful!");
}
else
{
    Console.WriteLine("Homing is not complete or failed.");
}
```

Verify by the "HomingStat" status

In addition to calling the IsHomed() method, you can also read the value of the "HomingStat" keyword to determine whether the homing was successful.

Example:

```
string response;
string command = "AHomingStat"; // A axis homing status
bool success = AAMotionAPI.SendCommandString(controller, command, out response);
if (success) {
// Determine whether the response content contains "100"
if (response. Contains("100"))
 { Console.WriteLine("Homing was successful!"); }
else { Console.WriteLine("Homing was not successful , Response : " + response); } }
else { Console.WriteLine("Command sending failed!");}
```

Other considerations

- Ensure that homing file path is correct, the file exists and the format complies with the specifications.
- You can check the return value of IsHoming() immediately after homing to ensure that the homing is successful before performing subsequent operations.

5.3 HomingStat status value list

Status value	Explanation	Detailed Explanation	Remark
0	Not executed	The homing operation has not yet begun, and it may be in a waiting state or has not received a command.	It can be confirmed by status queries or commands.
100	Successfully completed	The axis has successfully homed and the homing operation has been completed.	Subsequent normal operations can be executed.
-1	Wrong parameters cause failure	The parameter setting in the HomingDef array parameter is incorrect and the homing process is aborted (the parameters are validated at the beginning of each step).	Check whether the parameter settings are correct and whether the parameter range is compatible with the device.
-2	Timeouts cause failure	A homing step did not complete within the preset timeout period, causing the homing operation to abort. Each step has a defined timeout setting, and the operation will be interrupted if the timeout is exceeded	Timeout can be optimized or device response time guaranteed.
-3	The motor fails due to unexpected	During a certain step, the motor was detected to be in a disabled state or unexpectedly lost power, resulting in a homing abort. Refer to the	Check the hardware power and drive status to ensure the motor is

Status value	Explanation	Detailed Explanation	Remark
	power loss or disable	status provided by MotorReason command.	working properly.
-4	Motion reason caused the failure.	At a certain step, the motion end reason (e.g., indexing, target position detected) is different from expectation, resulting in a homing abort.	Check the mechanical path and sensor status.
-5	The wrong step type caused the failure	The step type is not recognized or supported in the configuration. Homing has been aborted.	Reconfigure the correct step type.
-6	It cannot perform homing during motion	The axis is still in motion. Homing cannot be started until the motion is complete.	Wait for the motion to finish and try again.
-7	The number of steps is out of range	The number of defined homing steps exceeds the allowed limit, or the final stage is not marked as an end state.	Adjust parameters to ensure the scope and definition are satisfied.
-8	Mechanical limitation caused the failure	Mechanical or safety restrictions are activated (such as limit switches) to prevent homing action.	Unlock restrictions to ensure safety and try again.
-9	Set the position failed	If attempts to set a new position or target position fail, refer to the documentation for SetPosition command.	Verify target position parameters and configurations.
-10	Motion mode is out of range or not supported	The selected motion mode (or gear mode) is not present or allowed.	Select a supported motion type or modify the configuration.
-11	Error compensation is out of range	The configured error compensation parameter is out of the allowable range of the device.	Adjust the parameters to the allowable range.
-12	The auto phasing is not completed, and the operation failed	The auto phasing operation is not completed, which affects the subsequent homing process.	Complete the auto phasing first and then perform homing.

Explanation:

- The value "100" clearly indicates that the homing operation has been successfully completed.
- A negative value indicates that the operation failed, and the detailed reason is described **in** the "Detailed Explanation" section.
- The value of "0" indicates that it has not yet started homing or the state has not been detected.

5.4 Important reminder

- When using HomingStat to determine the homing state, it is recommended to combine the value returned by SendCommandString, especially when "100" is returned, indicating that the homing is complete.
- If the status is negative or unsuccessful, it is recommended to check the parameter configuration, hardware status, motion path, etc. to ensure that the conditions are met
- Ensure that machinery safety and equipment confirmation are ready before operation.

6 Motor enable/disable

Through the API interface, the motor of the mechanical device can be activated (MotorOn) or disabled (MotorOff). This function is used to start or stop the motor drive, making mechanical motion control more flexible and safer.

6.1 Motor Enable/Disable API

MotorOn

	void MotorOn(MotionController controller, AxisRef axis)	
Description	Activate the motor to enable control of the target axis and the motor drive is activated	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
Return value	None	

MotorOff

	void MotorOff(MotionController controller, AxisRef axis)	
Description	Disable the motor, turn off the motor, stop the driving force, and the power supply is no longer applied	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
Return value	None	

6.2 Function Explanation:

- **MotorOn**:
  - With the `MotorOn` set to `1`, power is applied to the motor and the drive starts working.
  - Allows motion commands to perform operations such as position control and homing.
- **Disable Motor (MotorOff)**:
  - Set the MotorOn to 0 to stop the motor drive.
  - The power supply is disconnected and the motor is not driven to ensure safety.

Precautions for Use

- **Safety first**: Before starting the motor, ensure that the machinery is safe and barrier-free to avoid injury caused by motion.
- **Fault Auto-Disable**: If the driver detects a fault (such as overtemperature, overload, etc.), the motor will be automatically disabled (`ConFlt` state), even the user manually enables it, the motor will not work properly, and the fault needs to be resolved first.

- **Enable the motor again**:
    - If there is a fault in the drive (ConFlt is not 0), the fault state should be cleared before enabling it again.
    - When MotorOn() is recalled, ConFlt will be cleared, but the motor may remain disabled if the hardware state is not restored (e.g., overheating does not dissipate).

- **Parameter Limits**:
    - Modification or setting of certain parameters is only allowed when the motor is disabled.

6.3 Example of Operation

Example 1: Enable the motor

```
// Enable the motor of axis 1
AAMotionAPI.MotorOn(controller, AxisRef.A);
Console.WriteLine("A Axis of Controller is enabled.");
```

Example 2: Disable the motor

```
// Disable the motor of axis 1
AAMotionAPI.MotorOff(controller, AxisRef.A);
Console.WriteLine("A Axis of Controller is disabled.")
```

7 Motion control

The AAMotion library supports multiple motion control modes to meet different automation needs:

Types of Motion	Features
Jogging	Keep moving until stop
PTP Absolute Motion	Move to the specified absolute position
PTP Relative Motion	Offset a specified distance from current position
Repeat the Motion	Repetitive motion between two points

7.1 Motion Control API

MoveRelative

	void MoveRelative(MotionController controller, AxisRef axis, int distance, int? speed = null, int? accel = null, int? decel = null)	
Description	Axis relative motion, that is, the distance that moves the target axis relative to the current position	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	int distance	Distance moved (Positive indicates forward direction; negative indicates reverse direction.)
	int? speed (optional)	Speed of motion
	Int? accel (optional)	Acceleration
	int? decel (optional)	Deceleration
Return value	No return value, the axis starts relative motion after execution	

MoveAbsRepetitive

	void MoveAbsRepetitive(MotionController controller, AxisRef axis, int position, int msDwellTime, int? speed = null, int? accel = null, int? decel = null)	
Description	The axis repeats absolute position motion	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	int distance	Target position (absolute coordinates)
	int msDwellTime	Number of milliseconds to stay at each position
	int? speed (optional)	Speed of motion

	Int? accel (optional)	Acceleration
	int? decel (optional)	Deceleration
Return value	No return value, and the target axis moves repeatedly according to the set position and stays at each position for a defined amount of time.	

MoveRelRepetitive

	void MoveRelRepetitive(MotionController controller, AxisRef axis, int position, int msDwellTime, int? speed = null, int? accel = null, int? decel = null)	
Description	The axis repeats relative motion	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	int distance	Target position (absolute coordinates)
	int msDwellTime	The number of milliseconds in each position
	int? speed (optional)	Speed of motion
	Int? accel (optional)	Acceleration
	int? decel (optional)	Deceleration
Return value	No return value. The target axis repeatedly moves by the specified relative distance and stays at each point for a defined amount of time.	

Jog (Jog Motion)

	Void Jog(MotionController controller, AxisRef axis, int velocity)	
Description	Continuous axis motion (Jog mode)	
Parameter	Controller	Controller instance
	deviceAddress	The IP address of the connected controller
	int velocity	Speed of motion (positive and negative represent different directions)
Return value	No return value, start Jog motion — stop must be issued later.	

MoveRel (Relative Motion)

	void MoveRel(MotionController controller,int accel, int speed, int decel, ISingleAxis a, int aPos, ISingleAxis b, int bPos)	
Description	Relative motion between two axes	
Parameter	MotionController controller	Controller instance
	int accel	Acceleration
	int decel	Deceleration
	ISingleAxis a	A-axis identification

	int aPos	A-axis motion target position
	ISingleAxis b	B-axis identification
	int bPos	B-axis motion target position
Return value	Return to bool, indicating whether the motion started successfully	

	void MoveRel(MotionController controller,int accel, int speed, int decel, ISingleAxis a, int aPos, ISingleAxis b, int bPos, ISingleAxis c, int cPos)	
Description	Three-axis relative motion	
Parameter	MotionController controller	Controller instance
	int accel	Acceleration
	int decel	Deceleration
	ISingleAxis a	A-axis identification
	int aPos	A-axis motion target position
	ISingleAxis b	B-axis identification
	int bPos	B-axis motion target position
	ISingleAxis c	C-axis identification
	int cPos	C-axis motion target position
Return value	Return to bool, indicating whether the motion started successfully	

	void MoveRel(MotionController controller,int accel, int speed, int decel, ISingleAxis a, int aPos, ISingleAxis b, int bPos, ISingleAxis c, int cPos)	
Description	Three-axis relative motion	
Parameter	MotionController controller	Controller instance
	int accel	Acceleration
	int decel	Deceleration
	ISingleAxis a	A-axis identification
	int aPos	A-axis motion target position
	ISingleAxis b	B-axis identification
	int bPos	B-axis motion target position
	ISingleAxis c	C-axis identification
	int cPos	C-axis motion target position
Return value	Return to bool, indicating whether the motion started successfully	

	void MoveRel(MotionController controller,int accel, int speed, int decel, ISingleAxis a, int aPos, ISingleAxis b, int bPos, ISingleAxis c, int cPos,ISingleAxis d, int dPos)	
Description	Four-axis relative motion	
Parameter	MotionController controller	Controller instance
	int accel	Acceleration
	int decel	Deceleration
	ISingleAxis a	A-axis identification
	int aPos	A-axis motion target position
	ISingleAxis b	B-axis identification
	int bPos	B-axis motion target position
	ISingleAxis c	C-axis identification
	int cPos	C-axis motion target position
	ISingleAxis d	D-axis identification
	int dPos	D-axis motion target position
Return value	Return to bool, indicating whether the motion started successfully	

	void MoveRel(MotionController controller,int accel, int speed, int decel, ISingleAxis a, int aPos, ISingleAxis b, int bPos, ISingleAxis c, int cPos,ISingleAxis d, int dPos，ISingleAxis e, int ePos)	
Description	Five-axis relative motion	
Parameter	MotionController controller	Controller instance
	int accel	Acceleration
	int decel	Deceleration
	ISingleAxis a	A-axis identification
	int aPos	A-axis motion target position
	ISingleAxis b	B-axis identification
	int bPos	B-axis motion target position
	ISingleAxis c	C-axis identification
	int cPos	C-axis motion target position
	ISingleAxis d	D-axis identification
	int dPos	D-axis motion target position
	ISingleAxis e	E-axis identification
	int EPos	E-axis motion target position
Return value	Return to bool, indicating whether the motion started successfully	

	void MoveRel(MotionController controller,int accel, int speed, int decel, ISingleAxis a, int aPos, ISingleAxis b, int bPos, ISingleAxis c, int cPos,ISingleAxis d, int dPos，ISingleAxis e, int ePos,ISingleAxis f, int fPos)	
Description	Six-axis relative motion	
Parameter	MotionController controller	Controller instance
	int accel	Acceleration
	int decel	Deceleration
	ISingleAxis a	A-axis identification
	int aPos	A-axis motion target position
	ISingleAxis b	B-axis identification
	int bPos	B-axis motion target position
	ISingleAxis c	C-axis identification
	int cPos	C-axis motion target position
	ISingleAxis d	D-axis identification
	int dPos	D-axis motion target position
	ISingleAxis e	E-axis identification
	int ePos	E-axis motion target position
	ISingleAxis f	F-axis identification
	int fPos	F-axis motion target position
Return value	Return to bool, indicating whether the motion started successfully	

MoveAbs (Absolute Motion)

	void MoveAbs(MotionController controller, AxisRef axis, int position, int msDwellTime, int? speed = null, int? accel = null, int? decel = null)	
Description	The axis is in absolute motion, that is, it moves to a specified absolute position	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	int distance	Target position (absolute coordinates)
	int? speed (optional)	Speed of motion
	Int? accel (optional)	Acceleration
	int? decel (optional)	Deceleration
Return value	No return value, after performing the axis motion to the target position	

	void MoveAbs(MotionController controller,int accel, int speed, int decel, ISingleAxis a, int aPos, ISingleAxis b, int bPos)

Description	Absolute motion on both axes	
Parameter	MotionController controller	Controller instance
	int accel	Acceleration
	int decel	Deceleration
	ISingleAxis a	A-axis identification
	int aPos	A-axis motion target position
	ISingleAxis b	B-axis identification
	int bPos	B-axis motion target position
Return value	Return to bool, indicating whether the motion started successfully	

	void MoveAbs(MotionController controller,int accel, int speed, int decel, ISingleAxis a, int aPos, ISingleAxis b, int bPos, ISingleAxis c, int cPos)	
Description	Three-axis absolute motion	
Parameter	MotionController controller	Controller instance
	int accel	Acceleration
	int decel	Deceleration
	ISingleAxis a	A-axis identification
	int aPos	A-axis motion target position
	ISingleAxis b	B-axis identification
	int bPos	B-axis motion target position
	ISingleAxis c	C-axis identification
	int cPos	C-axis motion target position
Return value	Return to bool, indicating whether the motion started successfully	

	void MoveAbs(MotionController controller,int accel, int speed, int decel, ISingleAxis a, int aPos, ISingleAxis b, int bPos, ISingleAxis c, int cPos)	
Description	Absolute motion on both axes	
Parameter	MotionController controller	Controller instance
	int accel	Acceleration
	int decel	Deceleration
	ISingleAxis a	A-axis identification
	int aPos	A-axis motion target position
	ISingleAxis b	B-axis identification
	int bPos	B-axis motion target position

	ISingleAxis c	C-axis identification
	int cPos	C-axis motion target position
Return value	Return to bool, indicating whether the motion started successfully	

	void MoveAbs(MotionController controller,int accel, int speed, int decel, ISingleAxis a, int aPos, ISingleAxis b, int bPos, ISingleAxis c, int cPos,ISingleAxis d, int dPos)	
Description	Four-axis absolute motion	
Parameter	MotionController controller	Controller instance
	int accel	Acceleration
	int decel	Deceleration
	ISingleAxis a	A-axis identification
	int aPos	A-axis motion target position
	ISingleAxis b	B-axis identification
	int bPos	B-axis motion target position
	ISingleAxis c	C-axis identification
	int cPos	C-axis motion target position
	ISingleAxis d	D-axis identification
	int dPos	D-axis motion target position
Return value	Return to bool, indicating whether the motion started successfully	

	void MoveAbs(MotionController controller,int accel, int speed, int decel, ISingleAxis a, int aPos, ISingleAxis b, int bPos, ISingleAxis c, int cPos,ISingleAxis d, int dPos，ISingleAxis e, int ePos)	
Description	Five-axis absolute motion	
Parameter	MotionController controller	Controller instance
	int accel	Acceleration
	int decel	Deceleration
	ISingleAxis a	A-axis identification
	int aPos	A-axis motion target position
	ISingleAxis b	B-axis identification
	int bPos	B-axis motion target position
	ISingleAxis c	C-axis identification
	int cPos	C-axis motion target position
	ISingleAxis d	D-axis identification

	int dPos	D-axis motion target position
	ISingleAxis e	E-axis identification
	int EPos	E-axis motion target position
Return value	Return to bool, indicating whether the motion started successfully	

	void MoveAbs(MotionController controller,int accel, int speed, int decel, ISingleAxis a, int aPos, ISingleAxis b, int bPos, ISingleAxis c, int cPos,ISingleAxis d, int dPos，ISingleAxis e, int ePos,ISingleAxis f, int fPos)	
Description	Six-axis absolute motion	
Parameter	MotionController controller	Controller instance
	int accel	Acceleration
	int decel	Deceleration
	ISingleAxis a	A-axis identification
	int aPos	A-axis motion target position
	ISingleAxis b	B-axis identification
	int bPos	B-axis motion target position
	ISingleAxis c	C-axis identification
	int cPos	C-axis motion target position
	ISingleAxis d	D-axis identification
	int dPos	D-axis motion target position
	ISingleAxis e	E-axis identification
	int ePos	E-axis motion target position
	ISingleAxis f	F-axis identification
	int fPos	F-axis motion target position
Return value	Return to bool, indicating whether the motion started successfully	

Precautions for use

- Confirm that all ISingleAxis axes have motors enabled (`MotorOn` = 1) before motion
- The target position array length should match the axis array length
- Motion parameters (speed, acceleration, deceleration) should be within the allowable limits of the device and axis
- Control to stop motion based on real-time monitoring

7.2 Motion Control Examples

Example of single-axis motion control

Jog Motion

Implementation:

```
// Start the Jog motion
AAMotionAPI.Jog(controller, AxisRef.A, 5000); // Forward direction, speed
5000 count

AAMotionAPI.Stop(controller, AxisRef.A);// Stop Jog motion
```

PTP Absolute Motion

Move to the specified position:

```
// No parameters, using the default speed, acceleration and deceleration of
the axis
AAMotionAPI.MoveAbs(controller, AxisRef.B, 100000);
// Specify the motion parameters
AAMotionAPI.MoveAbs(controller, AxisRef.C, 150000, speed: 20000, accel:
10000, decel: 8000);
```

PTP Relative Motion

Offset relative to current position:

```
// No parameters, use the default motion parameters
AAMotionAPI.MoveRel(controller, AxisRef.D, 5000);

// Specify the motion parameters
AAMotionAPI.MoveRel(controller, AxisRef.E, -3000, speed: 15000, accel:
5000);
```

Repeat the Motion

Reciprocating motion between two positions:

```
// Absolute position repetitive motion
AAMotionAPI.MoveAbsRepetitive(controller, AxisRef.F, 10000, 500);

// Relative position repetitive motion
AAMotionAPI.MoveRelRepetitive(controller, AxisRef.F, 10000, 500);
```

Multi-axis coordinated motion

Multi-axis PTP motion

Multi-axis motion synchronous control:

```
// The two axes synchronous absolute motion
AAMotionAPI.MoveAbs(accel: 10000, speed: 20000, decel: 8000,
                    axisA, 100000, axisB, 150000);

// Three-axis synchronous relative motion
```

```
AAMotionAPI.MoveRel(accel: 12000, speed: 18000, decel: 10000,
                    axisX, 5000, axisY, 8000, axisZ, 2000);
```

Explanation of motion parameters

Parameter	Unit	Description	Remark
speed	Count/sec	Speed of motion	MaxVel od the axis cannot be exceeded
accel	Count/sec²	Acceleration	MaxAcc cannot be exceeded
decel	Count/sec²	Deceleration	Recommended to be equal to the acceleration

7.3 In-Target status detection

To ensure that the motion is complete and the axis has reached the target position, the InTargetStat can be checked by polling. A value of 4 for InTargetStat indicates that the axis has reached the target position.

InTargetStat

The `InTargetStat` is used to indicate whether the axis has reached the expected target position, and its state values represent different motion states:

Value	State	Explanation
0	The motor is off	The motor drive is turned off and not in motion
1	Idle (servo state enable)	The axis servo has been enabled, but it is not moving
2	In motion	The axis is moving forward or backward
3	Wait for InTargetTime to complete	The motion has reached the target position and is waiting for the InTargetTime time to complete
4	The target has been reached	The axis has reached the target position and the motion is complete

Example

When checking whether an axis is in place, the value is usually polled:

```
// Initiate absolute motion first, move the axis to position 0 (example)
AAMotionAPI.MoveAbs(controller,  AxisRef.A,  100000,  1000);  //  Target
position 100000, speed 1000
while (InTargetStat != 4); // Until 4, means it is in place
```

Explanation

- In-position check: When the value of InTargetStat is 4, it means that the axis has reached the preset target position, and the motion can be considered as a successful completion.

- Waiting for the target to reach: In certain motion scenarios, InTargetStat will wait for the user-defined InTargetTime (milliseconds) after it is reached, ensuring motion stability.
- The code above continuously checks InTargetStat until its value is 4, indicating that the target has been fully reached.

InTargetTime

InTargetTime is the amount of time the motor should be within the target tolerance range in milliseconds to determine if the target has been reached. If the position error is less than InTargetTol in user unit and remains at least InTargetTime milliseconds, then the InTargetStat will receive a value indicating that the target has been reached.

InTargetTol

InTargetTol is an "in-target" tolerance. If the position error is less than InTargetTol in user unit and remains at least InTargetTime milliseconds, then InTargetStat receives a value indicating that the target has been reached.

8 Motion stops

In motion control, different stop commands meet different control purpose. The following describes the common stop methods and their applicable scenarios.

8.1 Motion Stop API

Stop

	bool Stop(MotionController controller, AxisRef axis)	
Description	Stop motion	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
Return value	None	
Explanation	Function: Gradually decelerate to stop the motion according to the deceleration defined in the Decel parameter. Features: Does not immediately interrupt the current motion but smoothly decelerates to avoids impact. Note: If there is no motion currently in progress, calling this command will have no effect. Application scenarios: Smooth and safe termination of motion under safe condition.	

StopRep (Stop Repetitive Motion)

	bool StopRep(MotionController controller, AxisRef axis)	
Description	Stop repetitive motion	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
Return value	None	
Explanation	Function: Stop repetitive motion (such as repetitive action) that is being performed. Features: Only stop the "repetitive motion", if no repeat motion is performed, the command has no effect. This method does not stop the current motion immediately but stops at the starting point after the current motion ends (similar to the effect of stopping gradually). Application scenarios: Used to interrupt the next cycle when performing multiple repetitive motion (such as oscillation, repetitive tasks).	

StopVec (Stop Vector Motion).

	bool StopVec(MotionController controller, AxisRef axis)	
Description	Stop vector motion	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
Return value	None	
Explanation	Function: Used to stop a motion vector (multiaxial linkage motion). Features: Similar to Stop() but designed for multiaxial vector motion. It is suitable for stopping multi-axis synchronous motion. Application scenarios: In multi-axis linkage motion, all motion need to be stopped safely.	

Abort (Emergency Stop)

	bool Abort(MotionController controller, AxisRef axis)	
Description	Emergency stop	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
Return value	None	
Explanation	Function: Immediately interrupt the current motion, without warning, stop immediately. Features: No buffering and deceleration, directly aborting the motion. Calling when there is no motion is also no effect. This is the most intense way to stop and is suitable for emergency situations (emergency stops). Application scenarios: In an emergency, all or partial motion needs to be stopped immediately to ensure safety.	

8.2 Remark

- In normal motion, it is recommended to give priority to Stop or StopVec to ensure a smooth end of the motion.
- In case of an emergency, Abort should be used to ensure a safe and quick stop of the device.
- After using any stopping method, it should be combined with monitoring condition to ensure that the motion has been completely terminated to avoid injury.

9 Control PCsuite IDE+

9.1 Program API

ProgRun (Run Thread Task)

	bool ProgRun(MotionController controller, AxisRef axis, int threadNumber, int taskNumber)	
Description	Run the specified thread task	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	threadNumber	thread number
	taskNumber	Mission number
Return value	Successful send returns true, failed send returns false	
Remark	Run the task in the specified thread; The task is taskNumber and the thread is threadNumber	

AutoExec (Power-On Automatic Motion IDE+)

	bool AutoExec(MotionController controller, AxisRef axis, bool enable)	
Description	Execute the task automatically	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	enable	Whether to enable automatic execution
Return value	Successful send returns true, failed send returns false	
Remark	Sets whether the axis automatically executes the predefined program after a restart	

ProgReset

	bool ProgReset(MotionController controller, AxisRef axis, int threadNumber)	
Description	Reset the program	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	threadNumber	thread number
Return value	Successful send returns true, failed send returns false	
Remark	Reset the corresponding thread program task to clear the paused or abnormal state	

ProgHalt (Pause the thread)

	bool ProgHalt(MotionController controller, AxisRef axis, int threadNumber)	
Description	Suspend the program of the specified thread	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	threadNumber	thread number
Return value	Successful send returns true, failed send returns false	
Remark	Pause the current thread	

ProgHaltAll (Pause all the threads)

	bool ProgHaltAll(MotionController controller, AxisRef axis, int threadNumber)	
Description	Stop all the threads	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	threadNumber	thread number
Return value	Successful send returns true, failed send returns false	
Remark	Stopping the program executed by all threads can be continued with ProgRun	

9.2 Keyword Explanation

Keywords	**Function Description**	**Example**
ProgRun	Run the task in the specified thread with the task number as taskNumber and the thread as threadNumber	AProgRun[3],5 —Start task 5 at Thread 3
1	Represents the “main program” or “main task”, which can be set in taskNumber to start the main program	AProgRun[3],1 — Runs the main program
AutoExec=1	The setup program is automatically executed after power-on or software restart, needs to be saved to flash memory	Set it first, then call the save command to make sure the settings are valid
ProgReset	Reinitialize the program, clear the paused state and exceptions, and return to the program starting point	After modifying the program, it is recommended to call to ensure that the exception is cleared
ProgHalt	Pause the program in the current thread, with support for resuming from the pause point	AProgHalt [3] Pause Thread 3

Keywords	Function Description	Example
ProgHaltAll	Stop all programs running, suitable for cleaning up system status or debugging	Make sure all procedures are paused during testing
ProgHaltAllLogHalt	Stop all programs and log them into troubleshoot issues	For automated failure analysis

9.3 API Usage Examples

4.1 Running Tasks

```
// Run the subprogram numbered 5 in thread 3
AAMotionAPI.ProgRun(controller, AxisRef.A, 3, 5);
```

Note: This command will start a task program with the number 5 on thread 3, which is suitable for specific task scheduling.

4.2 Run the main program

```
// Start the main program in thread 3 (use task number-1 for the main program)
AAMotionAPI.ProgRun(controller, AxisRef.A, 3, -1);
```

Note: Start the main program (usually the overall control process) in the specified thread (3).

4.3 Set up automatic execution

```
// Set the current axis to execute automatically, and the program will run automatically after the next power-on or restart
AAMotionAPI.AutoExec(controller, AxisRef.A, true);
```

Operation: Enable auto-execution. After the setting is completed, it is recommended to call the save command to save the settings to the flash memory (implemented according to the storage command supported by the device).

4.4 Program Reset

```
// Reset the program state of thread 3 and restore it to its initial state
AAMotionAPI.ProgReset(controller, AxisRef.A, 3);
```

Note: Reset the program state in thread 3, suitable for use when the program exception or needs to be restarted.

4.5 Pausing Threads

```
// Suspend the current program in thread 3
AAMotionAPI.ProgHalt(controller, AxisRef.A, 3);
```

Note: Pause the program in the specified thread, the pause point is the current position. Calling ProgRun again can resume execution from the pause point.

4.6 Stop all programs

```
// Stop programs in all threads on the controller
```

```
AAMotionAPI.ProgHaltAll(controller, AxisRef.A, threadNumber);
```

Note: Stop all running programs for security detection or system debugging.

9.4 Description

- **Make sure to save**: After setting AutoExec=1, be sure to use the save command to save it to flash memory to prevent loss after power-off.
- **Task priority**: Arrange threadNumber and taskNumber reasonably to avoid conflicts or waiting.
- **Pause and restart**: `ProgHalt can be resumed with ProgRun after pause, without affecting the program status.`
- **Abnormal reset**: If you get stuck for a long time, call ProgReset to reset.

10 CNC motion

This section describes how to use the host computer API to control motion groups (usually multiple axes synchronous control), including start, pause, stop, and trajectory motion.

10.1 CNC-API

Begin (Start CNC Motion)

	Bool Begin(MotionController controller)	
Description	Start CNC motion	
Parameter	MotionController controller	Controller instance
Return value	Successful send returns true, failed send returns false	
Remark	If successful, the CNC motion begins	

Pause (Pause CNC Motion)

	bool Pause(MotionController controller)	
Description	Pause CNC motion	
Parameter	MotionController controller	Controller instance
Return value	Successful send returns true, failed send returns false	
Remark	Pause the current motion and the state can be restored	

Resume (Restore CNC Motion)

	bool Resume(MotionController controller)	
Description	Restore CNC motion	
Parameter	MotionController controller	Controller instance
Return value	Successful send returns true, failed send returns false	
Remark	Continue the paused group motion	

ClearBuffer (Clear CNC Buffer)

	bool ClearBuffer(MotionController controller)	
Description	Clear the buffer	
Parameter	MotionController controller	Controller instance
Return value	Successful send returns true, failed send returns false	
Remark	Clear queued motion commands	

GroupStop (Stop CNC Motion)

	bool GroupStop(MotionController controller)	
Description	Stop group motion	
Parameter	MotionController controller	Controller instance
Return value	Successful send returns true, failed send returns false	
Remark	Stop group motion immediately, the current trajectory may not be completed	

LinearAbsolute (CNC Linear Absolute Motion)

	bool LinearAbsolute(MotionController controller, int? aPos, int? bPos, int? cPos, int velCruise, int velEnd)	
Description	CNC Linear Absolute Motion (ABC Axis)	
Parameter	MotionController controller	Controller instance
	int? aPos	CNC Motion A-Axis Position (Optional)
	int? bPos	CNC Motion B-Axis Position (Optional)
	int? cPos	CNC Motion C-Axis Position (Optional)
	int velCruise	CNC motion motion speed
	int velEnd	CNC motion end speed
Return value	Successful send returns true, failed send returns false	
Remark	Perform synchronous linear segments and can be used for up to 3 axes at the same time	

LinearAbsoluteX (X-axis linear motion)

	Bool LinearAbsoluteX(MotionController controller, int xPos, int velCruise, int velEnd)	
Description	X-axis linear motion	
Parameter	MotionController controller	Controller instance
	int xPos	CNC motion X-axis position
	int velCruise	CNC cruising speed
	int velEnd	CNC motion end speed
Return value	Successful send returns true, failed send returns false	
Remark	Only control X-axis linear motion	

LinearAbsoluteY (Y-axis Linear Motion)

	Bool LinearAbsoluteY(MotionController controller, int yPos, int velCruise, int velEnd)
Description	Y-axis linear motion

Parameter	MotionController controller	Controller instance
	int yPos	CNC motion Y-axis position
	int velCruise	CNC cruising speed
	int velEnd	CNC motion end speed
Return value	Successful send returns true, failed send returns false	
Remark	Only control Y-axis linear motion	

LinearAbsoluteZ (Z-axis Linear Motion)

	Bool LinearAbsoluteZ(MotionController controller, int yPos, int velCruise, int velEnd)	
Description	Z-axis linear motion	
Parameter	MotionController controller	Controller instance
	int zPos	CNC motion Z-axis position
	int velCruise	CNC cruising speed
	int velEnd	CNC motion end speed
Return value	Successful send returns true, failed send returns false	
Remark	Only control Z-axis linear motion	

LinearAbsoluteXY (XY axis Linear Motion)

	Bool LinearAbsoluteXY(MotionController controller, int xPos, int yPos, int velCruise, int velEnd)	
Description	XY axis linear motion	
Parameter	MotionController controller	Controller instance
	int xPos	CNC motion X-axis position
	int yPos	CNC motion Y-axis position
	int velCruise	CNC cruising speed
	int velEnd	CNC motion end speed
Return value	Successful send returns true, failed send returns false	
Remark	Perform synchronous linear segments to control the motion of the XY axis	

LinearAbsoluteXZ (XZ axis Linear Motion)

	Bool LinearAbsoluteXZ(MotionController controller, int xPos, int zPos, int velCruise, int velEnd)	
Description	XZ axis linear motion	
Parameter	MotionController controller	Controller instance

	int xPos	CNC motion X-axis position
	int zPos	CNC motion Z-axis position
	int velCruise	CNC cruising speed
	int velEnd	CNC motion end speed
Return value	Successful send returns true, failed send returns false	
Remark	Perform synchronous linear segments to control the motion of the XZ axis	

LinearAbsoluteYZ (YZ axis Linear Motion)

	Bool LinearAbsoluteYZ(MotionController controller, int yPos, int zPos, int velCruise, int velEnd)	
Description	YZ axis linear motion	
Parameter	MotionController controller	Controller instance
	int yPos	CNC motion Y-axis position
	int zPos	CNC motion Z-axis position
	int velCruise	CNC cruising speed
	int velEnd	CNC motion end speed
Return value	Successful send returns true, failed send returns false	
Remark	Perform synchronous linear segments to control the motion of the YZ axis	

LinearAbsoluteXYZ (XYZ axis Linear Motion)

	Bool LinearAbsoluteXYZ(MotionController controller, int xPos, int yPos, int zPos, int velCruise, int velEnd)	
Description	XYZ axis linear motion	
Parameter	MotionController controller	Controller instance
	int xPos	CNC motion X-axis position
	int yPos	CNC motion Y-axis position
	int zPos	CNC motion Z-axis position
	int velCruise	CNC cruising speed
	int velEnd	CNC motion end speed
Return value	Successful send returns true, failed send returns false	
Remark	Perform synchronous linear segments to control the motion of the XYZ axis	

ArcXY (XY axis Arc Motion)

	Bool ArcXY(MotionController controller, int xPos, int yPos, int iPos, int jPos, int dir, int

	velCruise, int velEnd)	
Description	XY Arc motion	
Parameter	MotionController controller	Controller instance
	int xPos	CNC motion X-axis end coordinates
	int yPos	CNC motion Y-axis end coordinates
	int iPos	CNC motion center position
	int jPos	CNC motion center position
	int dir	Direction (1-clockwise, 0-counterclockwise)
	int velCruise	Cruising speed
	int velEnd	CNC motion end speed
Return value	Successful send returns true, failed send returns false	
Remark	Perform synchronized arc motion, XY arc motion	

	Bool ArcXY(MotionController controller, int xPos, int yPos, int iPos, int jPos, int dir, int velCruise, int velEnd， int? additionalCycles)	
Description	XY arc motion	
Parameter	MotionController controller	Controller instance
	int xPos	CNC motion X-axis end coordinates
	int yPos	CNC motion Y-axis end coordinates
	int iPos	CNC motion center position
	int jPos	CNC motion center position
	int dir	Direction (1-clockwise, 0-counterclockwise)
	int velCruise	Cruising speed
	int velEnd	CNC motion end speed
	Int？ additionalCycles	Additional revolutions (optional)
Return value	Successful send returns true, failed send returns false	
Remark	Perform synchronized arc motion and support multi-turn motion	

ArcXZ (XZ axis Arc Motion)

	Bool ArcXZ(MotionController controller, int xPos, int zPos, int iPos, int jPos, int dir, int velCruise, int velEnd， int？ additionalCycles)	
Description	XZ arc motion	
Parameter	MotionController controller	Controller instance

	int xPos	CNC motion X-axis end coordinates
	int zPos	CNC motion Z-axis end coordinates
	int iPos	CNC motion center position
	int jPos	CNC motion center position
	int dir	Direction (1-clockwise, 0-counterclockwise)
	int velCruise	Cruising speed
	int velEnd	CNC motion end speed
	Int？ additionalCycles	Additional revolutions (optional)
Return value	Successful send returns true, failed send returns false	
Remark	Perform synchronized arc motion and support multi-turn motion	

ArcYZ (YZ axis Arc Motion)

	Bool ArcYZ(MotionController controller, int yPos, int zPos, int iPos, int jPos, int dir, int velCruise, int velEnd， int？ additionalCycles)	
Description	YZ Arc motion	
Parameter	MotionController controller	Controller instance
	int yPos	CNC motion Y-axis end coordinates
	int zPos	CNC motion Z-axis end coordinates
	int iPos	CNC motion center position
	int jPos	CNC motion center position
	int dir	Direction (1-clockwise, 0-counterclockwise)
	int velCruise	Cruising speed
	int velEnd	CNC motion end speed
	Int？ additionalCycles	Additional revolutions (optional)
Return value	Successful send returns true, failed send returns false	
Remark	Perform synchronized arc motion and support multi-turn motion	

Delay (CNC motion delay)

	Bool Delay(MotionController controller, int ms)	
Description	Delay waiting	
Parameter	MotionController controller	Controller instance
	int ms	Delay Time (ms)
Return	Successful send returns true, failed send returns false	

value	
Remark	Set a delay between segments

SetMotionProfile (Sets Motion Vector Parameters)

	Bool SetMotionProfile(MotionController controller, int percentage, int accel, int decel, double sfactor)	
Description	Set the motion vector parameters	
Parameter	MotionController controller	Controller instance
	int percentage	Speed percentage
	int accel	Acceleration
	int decel	Deceleration
	double sfactor	Smoothing factor
Return value	Successful send returns true, failed send returns false	
Remark	Setting vector motion parameters from this point can be used multiple times in CNC motion	

SetCornerParams (Set the Corner Parameters)

	Bool SetCornerParams(MotionController controller, int cornerType, int radiusMethod, int radiusOrError, int axisAccel, int minAngle, int accelLimitType)	
Description	Set the corner parameters	
Parameter	MotionController controller	Controller instance
	int cornerType	Corner type
	int radiusMethod	Corner radius method
	int radiusOrError	Corner radius
	int axisAccel	Corner axis acceleration
	int minAngle	Corner axis minimum speed
	int accelLimitType	Axis acceleration limit type (0--not used, 1-used)
Return value	Successful send returns true, failed send returns false	
Remark	Define general corner parameters (type, radius, etc.) to be used in all subsequent automatic corner motion segments	

SetStartPositions(Set the Starting Position)

	Bool SetStartPositions(MotionController controller, AxisRef axisRef, int xPos, int yPos, int zPos)
Description	Set the starting position

Parameter	MotionController controller	Controller instance
	AxisRef axisRef	axis reference
	int xPos	Set the X start position of the CNC motion
	int yPos	Set the Y start position of the CNC motion
	int zPos	Set the Z start position of the CNC motion
Return value	Successful send returns true, failed send returns false	
Remark	Sets the expected initial position for all member axes. Execute once per CNC move, and place it at the first segment.	

WriteDOutPort (Write Discrete Output)

	Bool WriteDOutPort(MotionController controller, int dOutPortValue)	
Description	Write discrete output	
Parameter	MotionController controller	Controller instance
	int dOutPortValue	Output value (decimal)
Return value	Successful send returns true, failed send returns false	
Remark	Write the specified value to DOutPort to the specified port (decimal)	

GroupDOutSetBit (Set Discrete Output)

	Bool GroupDOutSetBit(MotionController controller, AxisRef axisRef, int bitMask)	
Description	Set the discrete output (set 1)	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Axis reference (write A by default)
	int bitMask	Bit mask
Return value	Successful send returns true, failed send returns false	
Remark	Set the specified bits	

GroupDOutClearBit (Set Discrete Output)

	Bool GroupDOutClearBit(MotionController controller, AxisRef axisRef, int bitMask)	
Description	Clear discrete outputs (set to 0)	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Axis reference (write A by default)
	int bitMask	Bit mask
Return value	Successful send returns true, failed send returns false	

Remark	Clear the specified bits

GroupDOutToggleBit (Toggle Discrete Output)

	Bool GroupDOutToggleBit(MotionController controller, AxisRef axisRef, int bitMask)	
Description	Toggle discrete outputs	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Axis reference (write A by default)
	int bitMask	Bit mask
Return value	Successful send returns true, failed send returns false	
Remark	Toggle the specified bits	

WriteGenData (Write GenData[]array data)

	Bool WriteGenData(MotionController controller, int gendataIndex, int gendataValue)	
Description	Write GenData[] array data	
Parameter	MotionController controller	Controller instance
	int gendataIndex	GenData index
	int gendataValue	The value corresponding to the GenData index
Return value	Successful send returns true, failed send returns false	
Remark	Writes the specified value to the specified index of the GenData[] array	

WriteUserParam (Write UserParam[] array data)

	Bool WriteUserParam(MotionController controller, AxisRef axisRef, int userparamIndex, int userparamValue)	
Description	Write the UserParam[] array data	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Axis index
	int userparamIndex	UserParam index
	int userparamValue	The value corresponding to the UserParam index
Return value	Successful send returns true, failed send returns false	
Remark	Writes the specified value to the specified index of the UserParam[] array	

GenDataWait (Use GenData array segment to wait)

	Bool GenDataWait(MotionController controller, int gendataIndex, int trigType, int

	trigVal)	
Description	Use GenData array segment to wait	
Parameter	MotionController controller	Controller instance
	int gendataIndex	The array index used
	int trigType	Trigger type
	int trigVal	Trigger value
Return value	Successful send returns true, failed send returns false	
Remark	When this type of segment is reached, the CNC engine will wait until all member axes are able to meet the trigger conditions defined by the segment parameters	

UserParamWait (Use UserParam array segment to wait)

	Bool UserParamWait(MotionController controller, AxisRef axis, int userparamIndex, int trigType, int trigVal)	
Description	Use UserParam array segment to wait	
Parameter	MotionController controller	Controller instance
	AxisRef axis	axis index
	int userparamIndex	userparam index
	int trigType	Trigger type
	int trigVal	Trigger value
Return value	Successful send returns true, failed send returns false	
Remark	When this type of segment is reached, the CNC engine will wait until all member axes are able to meet the trigger conditions defined by the segment parameters	

SetCurrPositions(Set Position)

	Bool SetCurrPositions(MotionController controller, AxisRef axis, int? aPos, int? bPos, int? cPos)	
Description	Set (assign) position	
Parameter	MotionController controller	Controller instance
	AxisRef axis	Axis index
	int? aPos	Set the X-axis position (optional)
	int? bPos	Set Y-axis position (optional)
	int? cPos	Set Z-axis position (optional)
Return value	Successful send returns true, failed send returns false	
Remark	Assign a new position value to the selected member axis (must not be moving)	

AutoCorner (Auto Corner Motion)

	Bool AutoCorner(MotionController controller, AxisRef axis)	
Description	Automatic corner motion	
Parameter	MotionController controller	Controller instance
	AxisRef axis	Axis index
Return value	Successful send returns true, failed send returns false	
Remark	The corner is automatically calculated based on the last set corner parameter segment request	

SetMaxVelJumpParams (Sets the maximum speed jump parameter)

	Bool SetMaxVelJumpParams(MotionController controller, int? aMaxVJ, int? bMaxVJ, int? cMaxVJ, int jumpMode)	
Description	Set the maximum speed jump parameter	
Parameter	MotionController controller	Controller instance
	int? aMaxVJ	A-axis maximum jump speed (Can be empty)
	int? bMaxVJ	B-axis maximum jump speed (Can be empty)
	int? cMaxVJ	C-axis maximum jump speed (Can be empty)
	int jumpMode	Speed Jump Mode 0 - Ignore 1 - Use
Return value	Successful send returns true, failed send returns false	
Remark	Define the maximum speed jump for each axis, where a linear segment of motion is followed by another segment of motion (without an auto-cornering segment)	

SetMaxAccelParams (Sets the maximum acceleration of the axis)

	Bool SetMaxAccelParams(MotionController controller, int? aMaxAccel, int? bMaxAccel, int? cMaxAccel)	
Description	Set the axis maximum acceleration	
Parameter	MotionController controller	Controller instance
	Int? aMaxAccel	Maximum acceleration in the A-axis (Can be empty)
	Int? bMaxAccel	Maximum acceleration in the B-axis (Can be empty)
	Int? cMaxAccel	Maximum acceleration in the C-axis (Can be empty)
Return	Successful send returns true, failed send returns false	

value	
Remark	Only used when applied after a cornering parameter segment.

MultiWriteGenData (Multiple writes to GenData)

	Bool MultiWriteGenData(MotionController controller, int gendataIndex, int gendataValue1, int gendataValue2, int gendataValue3, int gendataValue4)	
Description	Write to GenData array multiple times	
Parameter	MotionController controller	Controller instance
	int gendataIndex	GenData[] data index
	int gendataValue1	GenData[x] data value
	int gendataValue12	GenData[X +1] data value
	int gendataValue13	GenData[X +2] data value
	int gendataValue14	GenData[X +3] data value
Return value	Successful send returns true, failed send returns false	
Remark	Write 4 values to 4 consecutive indexes of the GenData[] array	

MultiWriteUserParam (Multiple writes to UserParam)

	Bool MultiWriteUserParam(MotionController controller, AxisRef axis, int userparamIndex, int userparamValue1, int userparamValue2, int userparamValue3, int userparamValue4)	
Description	Write to UserParam array multiple times	
Parameter	MotionController controller	Controller instance
	AxisRef axis	axis index
	int userparamIndex	UserParam[] data index
	int userparamValue1	UserParam[x] data value
	int userparamValue2	UserParam[X +1] data value
	int userparamValue3	UserParam[X +2] data value
	int userparamValue4	UserParam[X +3] data value
Return value	Successful send returns true, failed send returns false	
Remark	Write 4 values to 4 consecutive indexes of the UserParam[] array	

MultiWriteGenDataWait (Multiple writes to GenData and wait)

	Bool MultiWriteGenDataWait(MotionController controller, int gendataIndex, int gendataValue1, int gendataValue2, int gendataValue3, int gendataValue4, int trigIndex, int trigTyp, int trigVal)
Description	Write to the GenData array multiple times and wait

Parameter	MotionController controller	Controller instance
	int gendataIndex	GenData[] data index
	int gendataValue1	GenData[x] data value
	int gendataValue12	GenData[X +1] data value
	int gendataValue13	GenData[X +2] data value
	int gendataValue14	GenData[X +3] data value
	int trigIndex	Trigger condition index
	int trigType	Trigger type
	int trigVal	Trigger value
Return value	Successful send returns true, failed send returns false	
Remark	Write 4 values to 4 consecutive indexes of the GenData[] or UserParam[] array, and wait at the position of all member axes until the trigger conditions defined by the segment parameters are met	

MultiWriteUserParamWait

	Bool MultiWriteUserParamWait(MotionController controller, AxisRef axis, int userparamIndex, int userparamValue1, int userparamValue2, int userparamValue3, int userparamValue4, int trigIndex, int trigTyp, int trigVal)	
Description	Write to UserParam array multiple times and wait	
Parameter	MotionController controller	Controller instance
	AxisRef axis	Axis index
	int userparamIndex	UserParam[]data index
	int userparamValue1	UserParam[x] data value
	int userparamValue2	UserParam[X +1] data value
	int userparamValue3	UserParam[X +2] data value
	int userparamValue4	UserParam[X +3] data value
	int trigIndex	Trigger condition index
	int trigType	Trigger type
	int trigVal	Trigger value
Return value	Successful send returns true, failed send returns false	
Remark	Write 4 values to 4 consecutive indexes of the UserParam[] array, and wait at the position of all member axes until the trigger condition defined by the segment parameter is met	

10.2 CNC-API（CiGroup）

LinearAbsolute (Ci-CNC Linear Absolute Motion)

	Bool LinearAbsolute(MotionController, AxisRef, int? aPos, int? bPos, int? cPos, int? dPos, int velCruise, int velEnd)	
Description	CNC Linear Absolute Motion (ABCD Axis)	
Parameter	MotionController controller	Controller instance
	AxisRef axis	Axis reference, when it is AxisRef.A, GroupA is selected. When it is AxisRef.B, GroupB is selected.
	int? aPos	CNC Motion A-Axis Position (Optional)
	int? bPos	CNC Motion B-Axis Position (Optional)
	int? cPos	CNC Motion C-Axis Position (Optional)
	int? dPos	CNC Motion D-Axis Position (Optional)
	int velCruise	CNC motion motion speed
	int velEnd	CNC motion end speed
Return value	Successful send returns true, failed send returns false	
Remark	Perform synchronous linear segments and can be used for up to 4 axes simultaneously	

	Bool LinearAbsolute(MotionController, int? aPos, int? bPos, int? cPos, int? dPos, int? ePos, int? fPos, int velCruise, int velEnd)	
Description	CNC Linear Absolute Motion (ABCD Axis)	
Parameter	MotionController controller	Controller instance
	int? aPos	CNC Motion A-Axis Position (Optional)
	int? bPos	CNC Motion B-Axis Position (Optional)
	int? cPos	CNC Motion C-Axis Position (Optional)
	int? dPos	CNC Motion D-Axis Position (Optional)
	int? ePos	CNC Motion E-Axis Position (Optional)
	int? fPos	CNC Motion F-Axis Position (Optional)
	int velCruise	CNC motion motion speed
	int velEnd	CNC motion end speed
Return value	Successful send returns true, failed send returns false	
Remark	Perform synchronous linear segments and can be used for up to 6 axes simultaneously	

	Bool LinearAbsolute(MotionController, int? aPos, int? bPos, int? cPos, int? dPos, int?

	ePos, int? fPos, int velCruise, int velEnd)	
Description	CNC Linear Absolute Motion (ABCDEF Axis)	
Parameter	MotionController controller	Controller instance
	AxisRef axis	Axis reference, when it is AxisRef.A, GroupA is selected. When it is AxisRef.B, GroupB is selected.
	int? aPos	CNC Motion A-Axis Position (Optional)
	int? bPos	CNC Motion B-Axis Position (Optional)
	int? cPos	CNC Motion C-Axis Position (Optional)
	int? dPos	CNC Motion D-Axis Position (Optional)
	int? ePos	CNC Motion E-Axis Position (Optional)
	int? fPos	CNC Motion F-Axis Position (Optional)
	int velCruise	CNC motion motion speed
	int velEnd	CNC motion end speed
Return value	Successful send returns true, failed send returns false	
Remark	Perform synchronous linear segments and can be used on up to 6 axes simultaneously	

LinearAbsoluteT (T-axis Linear Absolute Motion)

	Bool LinearAbsoluteT(MotionController, AxisRef, int tPos, int velCruise, int velEnd)	
Description	CNC Linear Absolute Motion (T-Axis)	
Parameter	MotionController controller	Controller instance
	AxisRef axis	Axis reference, when it is AxisRef.A, GroupA is selected. When it is AxisRef.B, GroupB is selected.
	int tPos	CNC motion T-axis position
	int velCruise	CNC motion motion speed
	int velEnd	CNC motion end speed
Return value	Successful send returns true, failed send returns false	
Remark	Perform synchronous linear segments that can be used on up to 1 axis simultaneously	

LinearAbsoluteXT (XT axis Linear Absolute Motion)

	Bool LinearAbsoluteXT(MotionController, AxisRef, int xPos, int tPos, int velCruise, int velEnd)	
Description	CNC Linear Absolute Motion (XT Axis)	
Parameter	MotionController controller	Controller instance

	AxisRef axis	Axis reference, when it is AxisRef.A, GroupA is selected. When it is AxisRef.B, GroupB is selected.
	int xPos	CNC motion X-axis position
	int tPos	CNC motion T-axis position
	int velCruise	CNC motion motion speed
	int velEnd	CNC motion end speed
Return value	Successful send returns true, failed send returns false	
Remark	Perform synchronous linear segments and can be used for up to 2 axes simultaneously	

LinearAbsoluteYT (YT axis Linear Absolute Motion).

	Bool LinearAbsoluteYT(MotionController, AxisRef, int yPos, int tPos, int velCruise, int velEnd)	
Description	CNC Linear Absolute Motion (YT Axis)	
Parameter	MotionController controller	Controller instance
	AxisRef axis	Axis reference, when it is AxisRef.A, GroupA is selected. When it is AxisRef.B, GroupB is selected.
	int yPos	CNC motion Y-axis position
	int tPos	CNC motion T-axis position
	int velCruise	CNC motion motion speed
	int velEnd	CNC motion end speed
Return value	Successful send returns true, failed send returns false	
Remark	Perform synchronous linear segments and can be used for up to 2 axes simultaneously	

LinearAbsoluteZT (ZT axis Linear Absolute Motion).

	Bool LinearAbsoluteZT(MotionController, AxisRef, int zPos, int tPos, int velCruise, int velEnd)	
Description	CNC Linear Absolute Motion (ZT Axis)	
Parameter	MotionController controller	Controller instance
	AxisRef axis	Axis reference, when it is AxisRef.A, GroupA is selected. When it is AxisRef.B, GroupB is selected.
	int zPos	CNC motion Z-axis position
	int tPos	CNC motion T-axis position
	int velCruise	CNC motion motion speed

	int velEnd	CNC motion end speed
Return value	Successful send returns true, failed send returns false	
Remark	Perform synchronous linear segments and can be used for up to 2 axes simultaneously	

LinearAbsoluteXYT (XYT axis Linear Absolute Motion).

	Bool LinearAbsoluteXYT(MotionController, AxisRef, int xPos, int yPos, int tPos, int velCruise, int velEnd)	
Description	CNC Linear Absolute Motion (XYT Axis)	
Parameter	MotionController controller	Controller instance
	AxisRef axis	Axis reference, when it is AxisRef.A, GroupA is selected. When it is AxisRef.B, GroupB is selected.
	int xPos	CNC motion X-axis position
	int yPos	CNC motion Y-axis position
	int tPos	CNC motion T-axis position
	int velCruise	CNC motion motion speed
	int velEnd	CNC motion end speed
Return value	Successful send returns true, failed send returns false	
Remark	Perform synchronous linear segments and can be used for up to 3 axes simultaneously	

LinearAbsoluteXZT (XZT axis Linear Absolute Motion).

	Bool LinearAbsoluteXZT(MotionController, AxisRef, int xPos, int zPos, int tPos, int velCruise, int velEnd)	
Description	CNC Linear Absolute Motion (XZT Axis)	
Parameter	MotionController controller	Controller instance
	AxisRef axis	Axis reference, when it is AxisRef.A, GroupA is selected. When it is AxisRef.B, GroupB is selected.
	int xPos	CNC motion X-axis position
	int zPos	CNC motion Z-axis position
	int tPos	CNC motion T-axis position
	int velCruise	CNC motion motion speed
	int velEnd	CNC motion end speed
Return value	Successful send returns true, failed send returns false	
Remark	Perform synchronous linear segments and can be used for up to 3 axes simultaneously	

LinearAbsoluteYZT (YZT axis Linear Absolute Motion).

	Bool LinearAbsoluteYZT(MotionController, AxisRef, int yPos, int zPos, int tPos, int velCruise, int velEnd)	
Description	CNC Linear Absolute Motion (YZT Axis)	
Parameter	MotionController controller	Controller instance
	AxisRef axis	Axis reference, when it is AxisRef.A, GroupA is selected. When it is AxisRef.B, GroupB is selected.
	int yPos	CNC motion Y-axis position
	int zPos	CNC motion Z-axis position
	int tPos	CNC motion T-axis position
	int velCruise	CNC motion motion speed
	int velEnd	CNC motion end speed
Return value	Successful send returns true, failed send returns false	
Remark	Perform synchronous linear segments and can be used for up to 3 axes simultaneously	

LinearAbsoluteXYZT (XYZT axis Linear Absolute Motion).

	Bool LinearAbsoluteXYZT(MotionController, AxisRef, int xPos, int yPos, int zPos, int tPos, int velCruise, int velEnd)	
Description	CNC Linear Absolute Motion (XYZT Axis)	
Parameter	MotionController controller	Controller instance
	AxisRef axis	Axis reference, when it is AxisRef.A, GroupA is selected. When it is AxisRef.B, GroupB is selected.
	int xPos	CNC motion X-axis position
	int yPos	CNC motion Y-axis position
	int zPos	CNC motion Z-axis position
	int tPos	CNC motion T-axis position
	int velCruise	CNC motion motion speed
	int velEnd	CNC motion end speed
Return value	Successful send returns true, failed send returns false	
Remark	Perform synchronous linear segments and can be used for up to 4 axes simultaneously	

ArcXT (XT Arc Motion)

	Bool ArcXT(MotionController, AxisRef, int xPos, int tPos, int iPos, int lPos, int dir, int velCruise, int velEnd)

Description	XT Arc Motion	
Parameter	MotionController controller	Controller instance
	AxisRef axis	Axis reference, when it is AxisRef.A, GroupA is selected. When it is AxisRef.B, GroupB is selected.
	int xPos	CNC motion X-axis end coordinates
	int tPos	CNC motion T-axis end coordinates
	int iPos	CNC motion center position
	int lPos	CNC motion center position
	int dir	Direction (1-clockwise, 0-counterclockwise)
	int velCruise	Cruising speed
	int velEnd	CNC motion end speed
Return value	Successful send returns true, failed send returns false	
Remark	Perform synchronized arc motion, XT arc motion	

ArcYT (YT Arc Motion).

	Bool ArcYT(MotionController, AxisRef, int yPos, int tPos, int jPos, int lPos, int dir, int velCruise, int velEnd)	
Description	YT Arc Motion	
Parameter	MotionController controller	Controller instance
	AxisRef axis	Axis reference, when it is AxisRef.A, GroupA is selected. When it is AxisRef.B, GroupB is selected.
	int yPos	CNC motion Y-axis end coordinates
	int tPos	CNC motion T-axis end coordinates
	int iPos	CNC motion center position
	int lPos	CNC motion center position
	int dir	Direction (1-clockwise, 0-counterclockwise)
	int velCruise	Cruising speed
	int velEnd	CNC motion end speed
Return value	Successful send returns true, failed send returns false	
Remark	Perform synchronized arc motion, YT arc motion	

ArcZT (ZT Arc Motion).

	Bool ArcZT(MotionController, AxisRef, int zPos, int tPos, int kPos, int lPos, int dir, int velCruise, int velEnd)	
Description	ZT Arc Motion	
Parameter	MotionController controller	Controller instance
	AxisRef axis	Axis reference, when it is AxisRef.A, GroupA is selected. When it is AxisRef.B, GroupB is selected.
	int ZPos	CNC motion Z-axis end coordinates
	int TPos	CNC motion T-axis end coordinates
	int iPos	CNC motion center position
	int lPos	CNC motion center position
	int dir	Direction (1-clockwise, 0-counterclockwise)
	int velCruise	Cruising speed
	int velEnd	CNC motion end speed
Return value	Successful send returns true, failed send returns false	
Remark	Perform synchronized arc motion, ZT arc motion	

WriteICiGroupDOutPort（Write Ci Group Dout）

	Bool WriteICiGroupDOutPort(MotionController, AxisRef, int dOutPortValue)	
Description	Write discrete output	
Parameter	MotionController controller	Controller instance
	int dOutPortValue	Output value (decimal)
Return value	Successful send returns true, failed send returns false	
Remark	Write the specified value to DOutPort to the specified port (decimal)	

10.3 CNC-API（AGM800-CiGroup）Example

```
// 1. Initialize the controller instance
MotionController controller =
AAMotionAPI.Initialize(ControllerType.AGM800);

// 2. Set the motion mode of each axis to CNC (11 for CNC mode)
controller. GetAxis(AxisRef.A). MotionMode = 11;
controller. GetAxis(AxisRef.B). MotionMode = 11;
controller. GetAxis(AxisRef.C). MotionMode = 11;
controller. GetAxis(AxisRef.D). MotionMode = 11;

// 3. Clear the motion buffer and prepare for the next motion command
AAMotionAPI.ClearBuffer(controller);
```

```
// 4. Issue the first set of motion commands (A, B, and C axes move synchronously to the target position)
AAMotionAPI.LinearAbsolute(controller, AxisRef.A, 10000, 10000, 10000, null, 10000, 0);
// Description:
// - A: Move to position 10000
// - B: Move to position 10000
// - C: Move to position 10000
// - Speed: 10000
// - End speed: 0

// 5. Issue the second set of motion commands (B-axis motion to the target position)
// Note: The parameter configuration should comply with the API definition, the demonstration here assumes that you want to move the B-axis to position 8888, the second parameter "AxisRef."A" uses GroupA, and for AGM800, AxisRef.B is supported to use GroupB.
AAMotionAPI.LinearAbsolute(controller, AxisRef.A, null, 8888, null, null, 10000, 10000, 10000, 0);
// - Other axes remain unchanged or no target is set (null)

// 6. Start motion, execute commands
AAMotionAPI.Begin(controller);

// 7. Wait for the A-axis motion to complete
while (controller. GetAxis(AxisRef.A). InTargetStat != 4) // State 4: Position reached
{
    Thread.Sleep(10); // Check every 10ms
}

// 8. Wait for the B-axis motion to complete
while (controller. GetAxis(AxisRef.B). InTargetStat != 4)
{
    Thread.Sleep(10);
}
```

10.4 CNC-API Example

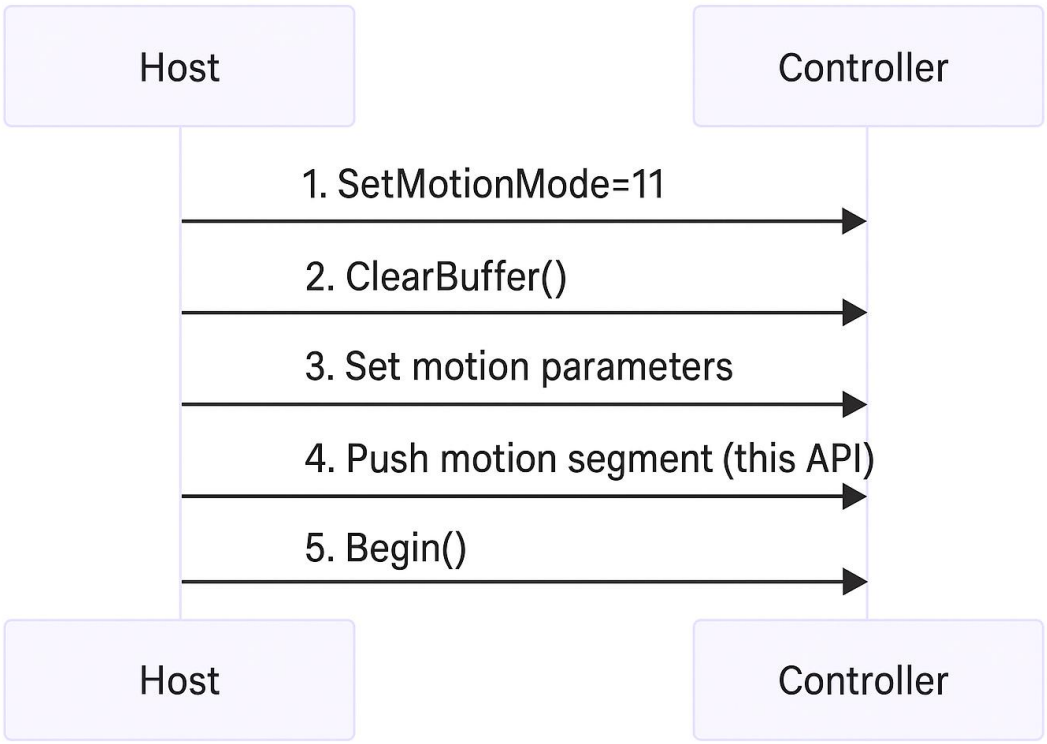
Brief introduction

This example shows how to set the controller to CNC motion mode, clear the buffer, configure the motion parameters, start position, and motion path, and finally start the motion. The examples involve X-axis, Y-axis motion, and XY coordinated motion, showing the basic process of CNC motion.

Example

```
// 1. Set the axis motion mode to CNC mode (value 11)
controller. GetAxis(AxisRef.A). MotionMode = 11;
controller. GetAxis(AxisRef.B). MotionMode = 11;

// 2. Clear the motion buffer
AAMotionAPI.ClearBuffer(controller);

// 3. Set the motion configuration parameters: speed ratio 100%, maximum acceleration/deceleration 200000, smoothing factor 0
AAMotionAPI.SetMotionProfile(controller, 100, 200000, 200000, 0);
```

```
// 4. Set the starting position, the starting point of the X and Y axes are
0, and the C axis is ignored (null)
AAMotionAPI.SetStartPositions(controller, 0, 0, null);

// 5. Example of a motion command
AAMotionAPI.LinearAbsoluteX(controller, 500, 50000, 10000); // The X-axis
moves to 500, with a starting speed of 50000 and an end speed of 10000
AAMotionAPI.LinearAbsoluteY(controller, 800, 50000, 50000); // The X-axis
moves to 800, with a starting speed of 50000 and an end speed of 50000
AAMotionAPI.LinearAbsoluteXY(controller, 1000, 1000, 50000, 50000); // The
XY axis moves to (1000, 1000), with a starting speed of 50000 and an end
speed of 50000

AAMotionAPI.Delay(controller, 1000); // Delay 1000ms and wait for the
motion to complete
// 6. Start the motion execution
AAMotionAPI.Begin(controller);

// 7. The end point of motion, the end of the speed is set to 0, which
means stop
AAMotionAPI.LinearAbsoluteXY(controller, 0, 0, 50000, 0);

// 8. Start the motion execution
AAMotionAPI.Begin(controller);
```



Explanation of Critical Steps

Set the motion mode:
Call MotionMode=11 to set the corresponding axis to the CNC motion mode. The controller will perform the corresponding motion type according to this mode on subsequent calls to Begin.
Note: MotionMode cannot be changed until the current motion ends.

Buffer cleanup:
Call ClearBuffer to ensure there are no residual motion commands that could interfere with the current motion.

Motion parameter setting:
Call SetMotionProfile to configure the speed ratio, maximum acceleration and deceleration, and smoothing factor to affect the motion dynamic response.

Segmented trajectory planning:
Use APIs such as LinearAbsoluteX/Y/XY to write the motion path segment by segment, and the speed at the final segment needs to be set to 0 to ensure that the motion stops smoothly.

Start Execution:
Call Begin to start all the above configured motion commands.

10.5 Notes on Using ArcXY with Additional Revolutions

When using AAMotionAPI.ArcXY for arc motion with additional revolutions, the following critical points require special attention to, otherwise it will cause motion errors or delay failure.

1. The end speed must be set to 0

API Example call:

```
controller. ErrorOccurred += (errorCode, msgSent, errorMsg) =>
{
    Console.WriteLine($"Message Sent: {msgSent}");
    Console.WriteLine($"Error Code: {errorCode}");
    Console.WriteLine(errorMsg);

};
AAMotionAPI.ArcXY(controller, 100, 100, 500, 500, 1, 10000, 0, 3);
```

- Description:
  - The end coordinates are (100, 100)
  - The center coordinates is (500, 500)
  - Direction of rotation: clockwise (`dir=1`).
  - The starting speed is 10000
  - **The end speed must be 0**
  - The additional number of revolutions is 3, and the actual number of revolutions in action is 1 + 3 = 4 revolutions

Required parameters Note:
When using additional parameters, the end speed must not be non-zero, otherwise an error will be reported:

```
Error:204
ERR 204
Sent: 83886079,2000
Error: The last motion segment has a non-zero end speed and this segment
can't continue it (not the same involved axes or a motion-blocking segment)
```

2. The starting point, center coordinates, and end point must form a valid arc

Make sure that the start point, center coordinates and end point can form a correct arc, otherwise the command will fail to execute.

3. Delay Operation Notes

If the delay command is received after the arc motion and the end speed is not set to 0, the delay command does not take effect.

4. Summary

- **The end speed must be 0**, otherwise an error will be reported and the subsequent delay will be invalid
- The arc parameters (start, center, end) must comply with circular geometry
- The additional number of revolution parameter is used to achieve multi-turn motion, and the actual number of revolutions = 1 + additional revolution

10.6 MotionMode keyword description

Value	Motion Types	Remark
0	Jog	Jog – The motor will reach the velocity set in Speed and continue at a constant velocity until the Stop command is received. Acceleration and deceleration used by motion are defined by Accel and Decel. Motion can also be smoothed according to the Jerk value.
1	PTP (Point to Point)	PTP – Move to a position defined by RelTrgt or AbsTrgt (if RelTrgt = 0). Other parameters of motion are defined by Accel, Speed, Decel and Jerk.
2	PTP Repetitive	PTP Repetitive – Reciprocating motion from the current position to the position defined by RelTrgt or AbsTrgt (if RelTrgt = 0). Other parameters of motion are defined by Accel, Speed, Decel and Jerk. The motor will wait for a time defined by RptWait at each end of the motion. Motion stops using StopRep.
3	Pulse direction direct mode	Pulse Direction Direct Mode – The reference position of the position loop is received directly from the pulse direction input.
4	Pulse direction indirect mode	Pulse Direction Indirect Mode – The position signal for this motion is determined based on the pulse direction input. However, the number of pulses received is not immediately used as the reference for the position loop. The number of pulses received is added to the current position target. The profiler uses the values of all motion parameters to build the trajectory of the new target: Accel, Decel, Speed, Jerk. This means that even if a large number of pulses are input at once, the

Value	Motion Types	Remark
		resulting motion will not be like a step response, but smoother. Note that the response to quick input will be smoother, but also slower.
5	Gear direct motion	The gears move directly
6	Gear indirect motion	
7	ECAM direct motion	
8	ECAM indirect motion	
9	FIFO motion	
10	Slave position reference	
11	CNC group A motion	This example is used for multi-axis linkage CNC control
12	Joystick direct motion	Direct motion of the joystick (analog signal position reference)
13	Joystick indirect motion	Rocker indirect motion (analog signal position reference, user configurable)
14	Joystick direct motion	The joystick moves directly, the input analog signal represents the speed reference
15	Joystick indirect motion	The joystick moves indirectly, and the input analog signal represents the speed reference, and this profiler can be set by the user
16	Vector motion	
17	CNC Group B motion	AGM800 supported
18	Spline buffer motion	

11 Digital IO output

This section describes the API interfaces related to digital output (DOutPort), which support clearing, setting, and switching operations for the digital output bits of a specified axis IO module. The underlying layer of the interface sends commands through the communication interface and waits for a response.

11.1 Digital IO-API

DOutClearBit (Clear Digital Output Bits)

	bool DOutClearBit(MotionController controller, AxisRef axis, int index)	
Description	Clear the digital output bits	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	int index	The digital output port index starts from 0
Return value	true: The command was sent successfully, and the operation was completed. false: Command transmission failed or communication was abnormal	
Remark	Clear (reset) the index bit of the digital output port of the specified axis	

DOutSetBit (Set the digital output bits)

	bool DOutSetBit(MotionController controller, AxisRef axis, int index)	
Description	Set the digital output bits	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	int index	The digital output port index starts from 0
Return value	true: The instruction was sent successfully, and the operation was completed. false: Instruction transmission failed or communication was abnormal	
Remark	Set the index bit of the digital output port of the specified axis to 1	

DOutToggleBit (Toggle digital output bits)

	bool DOutToggleBit(MotionController controller, AxisRef axis, int index)	
Description	Toggle the digital output bits	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	int index	The digital output port index starts from 0
Return value	true: The instruction was sent successfully, and the operation was completed. false: Instruction transmission failed or communication was abnormal	

Remark	Toggle the index bit of the digital output port of the specified axis

GetDOutPort (Get the digital output bits)

	int GetDOutPort(MotionController controller, AxisRef axis , int bit)	
Description	Get the digital output bits	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	int bit	The digital output port index starts from 0
Return value	Gets the current state of a specific digital output bit	
Remark	State of the specified bit (0 = Off, 1 = On). It is also necessary to refer to DOutLog	

GetDInPort (Get the digital input bits)

	int GetDInPort(MotionController controller, AxisRef axis , int bit)	
Description	Get the digital input bits	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	int bit	The digital input port index starts from 0
Return value	Get the current state of a specific digital input bit	
Remark	State of the specified bit (0 = Off, 1 = On). It is also necessary to refer to DInLog	

11.2 Digital IO example

```
// Clear the 3rd digit of the A-axis digital output
AAMotionAPI.DOutClearBit(controller, AxisRef.A, 3);

// Set the 1st position of the B-axis digital output
AAMotionAPI.DOutSetBit(controller, AxisRef.B, 1);

// Switch the 0th position of the C-axis digital output
AAMotionAPI.DOutToggleBit(controller, AxisRef.A, 0);

// Get the status of the A-axis digital input port 2
AAMotionAPI.GetDInPort(controller, AxisRef.A, 2);

// Obtain the status of the A-axis digital output port 3
AAMotionAPI.GetDOutPort(controller, AxisRef.A, 3);
```

11.3 Notes:

- The index parameter must be within the range supported by the hardware to avoid out-of-bounds access.
- The operation successfully returns true, and it is recommended to check the return value after the call to ensure that the execution is correct.
- The digital output operation is usually used to control external devices or indicator lights, please confirm that the hardware is connected correctly.

12 Event Triggering (PEG)

This section describes the API interfaces related to the PEG function, which support triggering one-time or fixed-interval pulse events for a specified axis based on the current position. It is mainly used to achieve external signal synchronization; marker point detection and other application scenarios. The interface is configured and controlled via the motion controller axis object.

12.1 PEG-API

SetSingleEventPEG (Set a single PEG event)

	Void SetSingleEventPEG(MotionController controller, AxisRef axis, int eventBegPos, int eventSelect, int? eventPulseRes = null, int? eventPulseWid = null)	
Description	Set the specified axis at the eventBegPos position to trigger a single event	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	int eventBegPos	The absolute position where the event is triggered
	int eventSelect	Event Selection (refer eventSelect for details)
	int? eventPulseRes	(Optional) Event pulse resolution 0-Low (microseconds) 1-High (nanosecond)
	int? eventPulseWid	(Optional) Event pulse width
Return value	None	
Remark	After successful configuration, the corresponding trigger event will be responded to.	

SetEventFixedGapPEG (Set Fixed-Interval PEG Events)

	void SetEventFixedGapPEG(MotionController controller, AxisRef axis, int eventBegPos, int eventGap, int eventEndPos, int eventSelect, int? eventPulseRes = null, int? eventPulseWid = null)	
Description	Fixed distance interval events are triggered	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	int eventBegPos	The absolute position where the event is triggered
	int eventGap	Event interval distance
	int eventEndPos	End position
	int eventSelect	Event Selection (refer eventSelect for details)

	int? eventPulseRes	(Optional) Event pulse resolution 0-Low (microseconds) 1-High (nanosecond)
	int? eventPulseWid	(Optional) Event pulse width
Return value	None	
Remark	Configure the specified axis to trigger an event at every eventGap distance, starting from eventBegPos and ending at eventEndPos.	

EventEnable (Enable Event Trigger)

	void EventEnable(MotionController controller, AxisRef axis)	
Description	Enable event triggering	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
Return value	None	
Remark	Enable the event trigger function of the specified axis and set EventOn to 1	

Note: After each event trigger, the status of EventOn will be set to 0, and the event trigger will be disabled. Before the next use, EventOn must be enabled.

EventDisEnable

	void EventDisEnable(MotionController controller, AxisRef axis)	
Description	Disable event triggers	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
Return value	None	
Remark	Disable the event trigger function of the specified axis and set EventOn to 0	

12.2 eventSelect Description

The event selection parameter is used to trigger the event number or combination, which is mapped as follows:

eventSelect value	**Meaning**
0	No incident
1	Use Event #1
2	Use Event #2

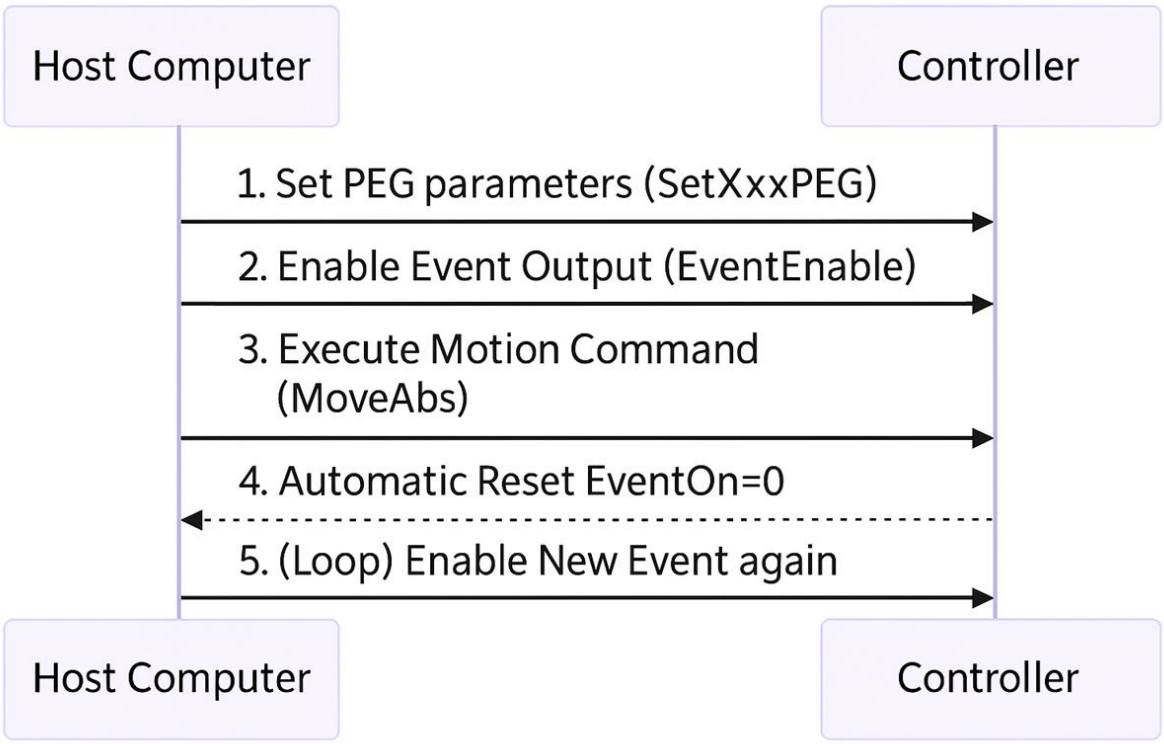
eventSelect value	Meaning
3	Use events #1 and #2
4	Use Event #3
5	Use events #1 and #3
6	Use events #2 and #3
7	Use events #1, #2, and #3
other	Unknown or reserved

The current IO needs to be set to the corresponding event number. You can pass the corresponding eventSelect value on demand when the API is called. For example, if you want to trigger both event 1 and event 3, pass eventSelect = 5.

12.3 PEG-API Example

Correct example

```
// Configure fixed-interval PEG events
// Start position 300, event interval 100, end position 1000
// Event Selection 1 (using Event #1), Pulse Resolution 1 (High, Nanoscale),
Pulse Width 500
AAMotionAPI.SetEventFixedGapPEG(controller, AxisRef.A, 300, 100, 1000, 1,
1, 500);

// Enable PEG event output
AAMotionAPI.EventEnable(controller, AxisRef.A);

// The axis moves to 1000 positions and the speed is 1000
AAMotionAPI.MoveAbs(controller, AxisRef.A, 1000, 1000);

// Wait for the motion to be in place, and InTargetStat of 4 indicates that
the motion is complete
while (controller. GetAxis(AxisRef.A). InTargetStat != 4);

// Configure a single PEG event, position 200, event selection 1 (use event
#1)
// Pulse resolution and pulse width use the default parameters
AAMotionAPI.SetSingleEventPEG(controller, AxisRef.A, 200, 1);

// Enable PEG event output, after the last PEG is completed, need to enable
PEG again
AAMotionAPI.EventEnable(controller, AxisRef.A);


// Move back to position 0 with a speed of 1000
AAMotionAPI.MoveAbs(controller, AxisRef.A, 0, 1000);

// Wait for back to position 0 to complete
int iIntargetStat;
do
{
    iIntargetStat = controller. GetAxis(AxisRef.A). InTargetStat;
    Thread.Sleep(50);
} while (iIntargetStat != 4);
```




```
AAMotionAPI.SetSingleEventPEG(controller, axis, eventBegPos, eventSelect);
AAMotionAPI.EventEnable(controller, axis); // The event must be explicitly
enabled
AAMotionAPI.MoveAbs(controller, axis, targetPos);
```

Description

- **SetEventFixedGapPEG**: Sets the trigger Event 1 every 100 units distance in the range between 300 to 1000, pulse resolution 1, and pulse width 500.
- **EventEnable**: Enable event trigger function.
- **MoveAbs**: The drive shaft moves to a specified position, supporting position and speed parameters.
- **Wait for motion to complete**: Synchronous wait is achieved by checking whether the target state InTargetStat is 4.
- **SetSingleEventPEG**: Set a single event trigger at position 200.
- Move back to the starting point at second time and wait for the motion to end.

This example shows a typical usage process for PEG functions: configure multi-point fixed-interval event, execute motions, then configure one-time single-point event, and wait for the motion to complete to ensure proper synchronization of event triggers.

12.4 Precautions for Use

Auto Reset Mechanism
After each PEG event is triggered, the system will automatically set the event enable status EventOn to 0 (disabled).

Manual re-enable requires
EventEnable**() must be** explicitly called to re-enable the event before each new PEG motion is executed.

Event override rules
The new configured PEG parameters will take effect immediately, and there is no need to wait for the pre-order motion to complete.

12.5 Typical problem troubleshooting table

Phenomenon	**Possible reason**	**Solution**
The event is not triggered	EventEnable was not called	Confirm that it is enabled before MoveAbs
Occasional missed triggers	Moving speed too fast	Reduce the motion speed or increase the pulse width
Pulse width deformation	The pulse resolution is set too high	Adjust to the appropriate resolution
Event delay	The system is overloaded	Increase thread sleep time appropriately

13 Position Lock

13.1 Lock-API

SetLock (Configure Lock)

	void SetLock(MotionController controller, AxisRef axis, bool enableLock, int lockSignalPolarity, int lockSource)	
Description	Configure the lock state of the specified axis, set the position lock pin to enable or disable.	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	bool enableLock	Whether to enable lock (true is lock enable, false is disable).
	int lockSignalPolarity	Lock signal polarity: 1 indicates a rising edge trigger, 0 indicates a falling edge trigger
	int lockSource	For more information, refer to the keyword lockSource
Return value	None	

LockEnable (Enable Lock)

	void LockEnable(MotionController controller, AxisRef axis)	
Description	Enable the Lock function of the target axis	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
Return value	None	
Remark	This call activates the lock state by setting the axis lock enable register to 1	

LockDisEnable (Disable Lock)

	void LockDisEnable(MotionController controller, AxisRef axis)	
Description	Disable the Lock function of the target axis	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
Return value	None	
Remark	After calling, set the lock register to 0 and disable the lock state	

13.2 Lock-API example

```
// Configure the lock parameter: the rising edge is triggered, and the source is the rising edge
AAMotionAPI.SetLock(controller, AxisRef.A, true, 1, 1);

// Enable lock (use the SetLock configuration to enable the lock function, then LockEnable is no longer needed).
// AAMotionAPI.LockEnable(controller, AxisRef.A);

// Move to the target position and the lock state is triggered
AAMotionAPI.MoveAbs(controller, AxisRef.A, 100000, 1000); // Target position 100000, speed 1000

// The lock state remains until it disables again
// Disable lock (example)
AAMotionAPI.LockDisEnable(controller, AxisRef.A);
```

13.3 Lock function keyword description

LockEn

Definition:
Set LockEn = 1 to enable the position lock function.

Function:
When position lock is enabled, the specified input changes (e.g., clockwise motion increases, counterclockwise decreases) will lock the current main encoder position to ensure position stability.

Note:
Lock and event functions are mutually exclusive. When Lock (LockEn=1) is enabled, related events are automatically disabled to avoid conflicts.

LockVal

Definition:
Locks the current position recorded by the input change during the input change.

Detailed description:

- For incremental encoders, the position is locked directly by the hardware and the values are extremely accurate.
- For absolute encoders, after the input changes (on the next sample), the lock position read is stored in LockVal.

Update frequency:

- LockVal is updated only once at the sampling time.
- In multiple valid input changes, only the last value change is recorded.

LockCntr (Lock Counter)

Definition: Records the number of input changes detected since lock (LockEn=1) was enabled.

Function:
It can be used for status monitoring to determine the number of input changes, and even for dead zone detection.

Reset:

- LockCntr will be reset when the LockEn state is changed (e.g., LockEn=0 or reassigned to 0).
- The count can also be reset by explicitly assigning 0.

LockSrc (Lock Source Input)

Definition:
The input signal number used for the lock function.

Function:
Specifies which input triggers the position lock, and input changes are reflected in LockVal and LockCntr.

Extended roles:
Inputs designated as LockSrc can also be configured as other functional roles (such as limits, etc.) in DInMode to achieve multi-role coexistence.

Example:
Input 3 can be used as a left limit switch (normal limit detection) or as a lock trigger source.

13.4 Keyword usage examples

```
// Enable position locking
AAMotionAPI.SetLock(controller, AxisRef.A, true, 1, 3); // LockEn=1, rising edge polarity, lock source is input 3

// Turn off position lock
AAMotionAPI.SetLock(controller, AxisRef.A, false, 1, 3);

// Force reset lock values and counters
controller. GetAxis(AxisRef.A). LockVal = 0; // Position value reset
controller. GetAxis(AxisRef.A). LockCntr = 0; // Counter reset
```

14 Force

This chapter introduces the API interfaces related to Force Control and how to use them to help users implement force control adjustment, motion, and parameter configuration of axis. The force control function allows the mechanical shaft to be configured and adjusted through software, support force feedback, pressure control and other application scenarios. Users can switch between position mode and force mode (open loop is current mode), adjust force parameters, set target position and motion state.

14.1 Force Control - API

GoToForceMode (Enter Force Mode)

	bool GoToForceMode(MotionController controller, AxisRef axis)	
Description	Enter force control mode and switch the axis to force control state	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
Return value	None	
Remark	After calling, the axis enters a force control state, ready for force adjustment, usually before motion	

GoToCurrMode (Enter Current Mode)

	bool GoToCurrMode(MotionController controller, AxisRef axis)	
Description	Enter current control mode	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
Return value	None	
Remark	Exit force control mode and switch back to current control	

GoToPosMode (Enter Position Mode)

	bool GoToPosMode(MotionController controller, AxisRef axis)	
Description	Return to position control mode	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
Return value	None	
Remark	Exit force control mode and switch back to position control.	

SetPosition (Set Target Position value)

	bool SetPosition(MotionController controller, AxisRef axis, int value)	
Description	Set the target position value	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	int value	Target position value
Return value	None	
Remark	Set the target position for position control	

ForceCurr_SetForceCmdSource (Set the source of force command)

	void ForceCurr_SetForceCmdSource(MotionController controller, AxisRef axis, int forceCmdSrc, int[] forceCurrSlope, int[] forceCmdHTime, int[] forceCmdVal)	
Description	Set the source parameter of the force command	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	int forceCmdSrc	Force command source number (e.g., 1 represents a signal channel)
	int[] forceCurrSlope	Force command slope array (adjust the speed of change)
	int[] forceCmdHTime	Command duration array (controls pulse width)
	int[] forceCmdVal	Force Command Value Array (Adjust Force Magnitude)
Return value	None	
Remark	Configure the input signal or command source for force control, including the speed, duration, and magnitude of the force change.	

ForceCurr_SetCurrModeSWTrigSrc (Set trigger sources and conditions)

	void ForceCurr_SetCurrModeSWTrigSrc(MotionController controller, AxisRef axis, int currPosTh, int currPosThDir, int? forcePosErrTh = null, int? forceAInTh = null)	
Description	Set trigger sources and conditions to automatically switch to force mode	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	int currPosTh	Position threshold

	int currPosThDir	Direction (1: forward, 0: reverse)
	int? forcePosErrTh	Position error threshold (optional)
	int? forceAInTh	Analog Input Threshold (Optional)
Return value	None	
Remark	Set trigger conditions that automatically switch to force control when conditions are met.	

ForceCurr_SetForceTunePID (Adjust the force control PID parameters)

	void ForceCurr_SetForceTunePID(MotionController controller, AxisRef axis, int? fForceGain = null, int? fForceKi = null, int? fForceKd = null, int? fForceFFW = null, int? fForceVelFFW = null)	
Description	Adjust the force control PID parameters	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	Int? fForceGain	Proportional gain (Kp)
	Int? fForceKi	Integral Gain (Ki)
	int? fForceKd	Differential gain (Kd)
	Do you? fForceFFW	Feedforward force parameters
	Do you? fForceVelFFW	Feedforward Speed parameters
Return value	None	
Remark	Adjust the force control PID parameters for smoother and more precise force response.	

ForceCurr_SetMotion (Adjust the force control PID parameters)

	void ForceCurr_SetMotion(MotionController controller, AxisRef axis, int absTrgt, int? relTrgt = null, int? speed = null, int? accel = null, int? decel = null)	
Description	Adjust the force control PID parameters	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	int absTrgt	Absolute target position
	int? relTrgt	Relative target position (optional, null means not set)
	int? speed	Acceleration (optional)
	int? accel	Deceleration (optional)
Return	None	

value	
Remark	Set the target position and parameters of the target motion and perform the motion

ForceCurr_SetSlowApproache (Set slow approach parameters)

	void ForceCurr_SetSlowApproache(MotionController controller, AxisRef axis, int speedChgNew, int speedChgPos, int speedChgDir, int speedChgOn)	
Description	Set the buffer gradual approximation parameter	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	int speedChgNew	New speed
	int speedChgPos	Position threshold
	int speedChgDir	Direction (1: forward, 0: reverse)
	int speedChgOn	Open sign (1 on, 0 off)
Return value	None	
Remark	The axis is slowly adjusted according to the set parameters as it approaches the target position	

ForceCurr_SetRetractMotion (Set Retract Motion Parameters)

	void ForceCurr_SetRetractMotion(MotionController controller, AxisRef axis, int fRetractTarget, int fRetractSpeed, int? fBeginOnToPos = null)	
Description	Set retract motion parameters	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	int fRetractTarget	New speed
	int fRetractSpeed	Position threshold
	int? fBeginOnToPos	Whether to start retracting after being in place (1: Yes, 0: No)
	int speedChgOn	Whether to enable (1: Enable)
Return value	None	
Remark	Set the retract action after the motion is complete.	

ForceCurr_SetPosThForAutoSwToPosMode (Set the threshold parameter for automatic switching to position mode)

	void ForceCurr_SetPosThForAutoSwToPosMode(MotionController controller, AxisRef axis, int posPosFlag, int posPosTh)

Description	Set the threshold parameters for automatic switching to position mode	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	int posPosFlag	Position threshold trigger flag (usually set to 1)
	int posPosTh	Position difference threshold
Return value	None	
Remark	When the threshold conditions are met, it automatically switches to position control mode	

14.2 Force Control API example

```
// 1. Enter force control mode
AAMotionAPI. GetAxis(axis).GoToForceMode();

// 2. Set the target position (specific target position and speed can be adjusted according to actual needs)
AAMotionAPI. GetAxis(axis).ForceCurr_SetMotion(3350, 0, 5000);

// 3. Configure the slow approach parameter
AAMotionAPI. GetAxis(axis).ForceCurr_SetSlowApproache(1000, 3000, 0, 1);

// 4. The PID parameters are set to adjust the response characteristics of force control
AAMotionAPI. GetAxis(axis).ForceCurr_SetForceTunePID(80000, 30, 2000, 400, 100);

// 5. Set trigger conditions for automatic switching to force control mode (example threshold)
AAMotionAPI. GetAxis(axis).ForceCurr_SetCurrModeSWTrigSrc(3100, 1, 5);

// 6. Configure force command sources and parameters (e.g., acting on a force source channel)
AAMotionAPI. GetAxis(axis).ForceCurr_SetForceCmdSource(
    1, // forceCmdSrc: force command source number
    new int[] { 2000, 2000, 2000, 2000 }, // force command slope (rate of change)
    new int[] { 500, 500, 500, 500 }, // command duration (width)
    new int[] { 500, 200, 300, 0 } // command value (size)
);

// 7. Set a retract (backward) action, such as retract after reaching a target
AAMotionAPI. GetAxis(axis).ForceCurr_SetRetractMotion(0, 20000);

// 8. Start Action (Commit Settings)
AAMotionAPI. GetAxis(axis).Begin();

// 9. The PID parameters or target position can be continuously adjusted as needed during the control process
// For example, dynamically adjust PID parameters to optimize responses
```

```
AAMotionAPI. GetAxis(axis). ForceCurr_SetForceTunePID(60000, 20, 1500, 300,
80);

// 10. Continuously monitor the status and adjust operations according to
actual needs
// For example, obtain the current force value and position to determine
whether the target has been reached, etc
```

Description

- **Step 1**: Switch to force control mode and prepare for force adjustment.
- **Step 2-4**: Set the target position, buffer parameters, and PID parameters to ensure the stability and response speed of force control.
- **Step 5**: Configure the trigger conditions to automatically switch to the force control state.
- **Step 6**: Define source of applied force (command source), set the slope, time, and value of the force command.
- **Step 7**: Set the retract action to ensure that you can automatically exit to a safe state after the motion is completed.
- **Step 8**: Start Control and commit all configurations by calling Begin().
- **Step 9-10**: In practical operation, you can continue to adjust the parameters or read the state to achieve precise control.

14.3 Notes:

- **Parameter adjustment**: In practical applications, PID parameters and motion parameters need to be debugged and optimized according to factors such as equipment rigidity and environmental noise.
- **Motion status monitoring**: It is recommended to combine status query interface (such as position, current, force sensor readings) with dynamic adjustment.
- **Safety control**: Limits should be set during operation to trigger a timely shutdown or alarm when the system exceeds the predefined safety threshold.
- **API call sequence**: Make sure to call GoToForceMode() first, then set the parameters, and finally call Begin() to start execution.

14.4 Common problems and solutions

Problem description	Possible causes	Solution
Force control is inactive or has delayed response.	PID parameters are not optimized	Adjust PID parameters step by step to reduce response time or oscillation
Motion no response or unstable control	The parameter is misconfigured	Check the parameter configuration to ensure that the target position and parameters are reasonable
The automatic switching condition is not triggered	The trigger condition is set incorrectly or the threshold is slightly high	Adjust the threshold to ensure that the trigger conditions are met
The output force value deviates from	Incorrect force source configuration or limited output	Modify the force source configuration

Problem description	Possible causes	Solution
expectations	range	

15 Vector Motion

15.1 Vector motion API

StopVec (Stop Vector Motion)

	bool StopVec(MotionController controller, AxisRef axis)	
Description	Stop vector motion on the specified axis.	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
Return value	Successful sending returns true, sending fails returning false	

VectorPause (Pause Vector Motion)

	bool VectorPause(MotionController controller, AxisRef axis)	
Description	Pause vector motion on the specified axis.	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
Return value	Successful sending returns true, sending fails returning false	

VectorContinue (Resume Vector Motion)

	bool VectorContinue(MotionController controller, AxisRef axis)	
Description	Resume paused vector motion.	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
Return value	Successful sending returns true, sending fails returning false	

VectorLinear (Linear Vector Motion)

	bool VectorLinear(MotionController controller, AxisRef axis, int? aPos, int? bPos, int? cPos, int? dPos, int? ePos, int? fPos, int? gPos, int? hPos, int speed, int accel, int decel, int emrgdec, int smooth)	
Description	Linear vector motion	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	int? aPos	The absolute position of the A-axis vector motion. It can be null

	int? bPos	The absolute position of the B-axis vector motion. It can be null
	int? cPos	The absolute position of the C-axis vector motion. It can be null
	int? dPos	The absolute position of the D-axis vector motion. It can be null
	int? ePos	The absolute position of the E-axis vector motion. It can be null
	int? fPos	The absolute position of the F-axis vector motion. It can be null
	int? gPos	The absolute position of the G-axis vector motion. It can be null
	int? hPos	The absolute position of the H-axis vector motion. It can be null
	int speed	Vector motion speed
	int accel	Vector motion acceleration
	int decel	Vector motion deceleration
	int emrgdec	Vector motion emergency stop speed
	int smooth	Vector motion smoothing parameters
Return value	Successful sending returns true, sending fails returning false	
Remark	Initiated by the axis with the lowest index among the axes involved in the motion.	

	bool VectorLinear(MotionController controller, AxisRef axis, AxisRef axisMask, int speed, int accel, int decel, int emrgdec, int smooth, params int[] pos)	
Description	Linear vector motion.	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	AxisRef axisMask	Participate in vector motion bit masks
	int speed	Vector motion speed
	int accel	Vector motion acceleration
	int decel	Vector motion deceleration
	int emrgdec	Vector motion emergency stop speed
	int smooth	Vector motion smoothing parameters
	params int [] pos	Vector motion axis absolute position array
Return value	Successful sending returns true, sending fails returning false	
Remark	Initiated by the axis with the lowest index among the axes involved in the motion.	

VectorArc (Vector Arc Motion)

	bool VectorArc(MotionController controller, AxisRef axis, AxisRef axis1, AxisRef axis2, int center1, int center2, int pos1, int pos2, int dir, int circles, int speed, int accel, int decel, int emrgdec, int smooth)	
Description	Vector Arc Motion	
Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	AxisRef axis	Participating in vector arc motion bit mask (lowest axis number)
	AxisRef axis1	X-axis Vector arc motion
	AxisRef axis2	Y-axis Vector arc motion
	int center1	The center point of X-axis vector arc motion
	int center2	The center point of Y-axis vector arc motion
	int pos1	The end position of the X-axis vector arc motion
	int pos2	The end position of the Y-axis vector arc motion
	int dir	Direction of vector arc motion (0-counterclockwise, 1-clockwise)
	int circles	Vector arc motion additional revolutions
	int speed	Vector motion speed
	int accel	Vector motion acceleration
	int decel	Vector motion deceleration
	int emrgdec	Vector motion emergency stop speed
	int smooth	Vector motion smoothing parameters
	params int [] pos	Vector motion axis absolute position array
Return value	Successful sending returns true, sending fails returning false	
Remark	Initiated by the axis with the lowest index among the axes involved in the motion.	

SetVectorParam (configure vector motion parameters).

	void SetVectorParam(MotionController controller, AxisRef axis, int speed, int accel, int decel, int emrgdec, int smooth)
Description	It is used to configure parameters such as smoothing, speed, and acceleration of the axis

Parameter	MotionController controller	Controller instance
	AxisRef axisRef	Identification of the axis
	int speed	Vector motion speed
	int accel	Vector motion acceleration
	int decel	Vector motion deceleration
	int emrgdec	Vector motion emergency stop speed
	int smooth	Vector motion smoothing parameters
Return value	None	
remark	Initiated by the axis with the lowest index among the axes involved in the motion.	

15.2 Vector motion API usage Examples

1. Initialize the controller

```
MotionController controller =
AAMotionAPI.Initialize(ControllerType.AGM800);
controller. ErrorOccurred += (errorCode, msgSent, errorMsg) => // Configure
error handling
{
    Console.WriteLine($"Message Sent: {msgSent}");
    Console.WriteLine($"Error Code: {errorCode}");
    Console.WriteLine(errorMsg);
};
if (! AAMotionAPI.Connect(controller))
{
    Console.WriteLine("Controller connection failed");
    return;
}
Console.WriteLine("Connection successful");
```

2. Turn on the motor

```
AAMotionAPI.MotorOn(controller, AxisRef.A); // Turn on A-axis motor
AAMotionAPI.MotorOn(controller, AxisRef.B); // Turn on B-axis motor
```

3. Absolute position motion

```
// Move axes A and B to position 0 and speed 10,000
AAMotionAPI.MoveAbs(controller, AxisRef.A, 0, 10000); // Moves A-axis to 0
AAMotionAPI.MoveAbs(controller, AxisRef.B, 0, 10000); // Moves B-axis to 0
// Wait for the motion to complete
while (controller. GetAxis(AxisRef.A). InTargetStat != 4 || controller.
GetAxis(AxisRef.B). InTargetStat != 4)
{
    Thread.Sleep(10);
}
Console.WriteLine("A and B axes have reached position 0");
```

4. Set vector linear motion example

1. Vector linear motion of a single target position (example)

```
// Take the A-axis as an example, the target position is 30000 and the
speed is 10000
AAMotionAPI.VectorLinear(
    controller,
    AxisRef.A, // Start with the A-axis
    30000,     // Target position aPos
    null,      // B-axis target position (left blank not set)
    null, null, null, null, null, null, // Target of others axes not set
    10000,     // Motion speed
    100000,    // Acceleration
    100000,    // Deceleration
    1000000,   // Emergency braking deceleration
    0          // smoothing factor
);
controller. GetAxis(AxisRef.A).Begin(); // Start motion
```

2. Vector linear motion through axis mask and position array (example)

Let's say you want to set the target position of both axes A and B at the same time, use AxisRef.A | AxisRef.B as a mask, position array positions = { 10000, 5000 }, the code is as follows:

```
// Define the target position array in the order specified by the mask
int[] positions = new int[] { 10000, 5000 };

// Perform vector linear motion of the multi-axis target position
AAMotionAPI.VectorLinear(
    controller,
    AxisRef.A,              // Start with the A-axis
    AxisRef.A | AxisRef.B,  // Axis mask: Simultaneous operation of A and B
    10000,                  // speed
    100000,                 // Acceleration
    100000,                 // Deceleration
    1000000,                // Emergency braking deceleration
    0,                      // smoothing factor
    positions               // target position array
);
controller. GetAxis(AxisRef.A). Begin(); // Start motion
```

Description

- "Axis Mask" AxisRef.A | AxisRef.B means controlling both axes A and B simultaneously, and their target positions correspond in order in the position array (the first is A, the second is B).
- You can adjust the speed and parameters according to the actual axis and motion needs.
- Make sure that the Begin() function is called after the motion command to start the motion.
- The number of elements in the target position array should be consistent with the number of axes in the mask, and the order should correspond.

5. Pause and Resume the motion

Pause A-axis motion

```
AAMotionAPI.VectorPause(controller, AxisRef.A);
Console.WriteLine("Pause A-axis motion");
Thread.Sleep(1000); // Wait for one second
// Keep moving
AAMotionAPI.VectorContinue(controller, AxisRef.A);
```

```
Console.WriteLine("Resume A-axis motion");
```

6. Examples of arc motion

Create an arc motion path

```
AAMotionAPI.VectorArc(
    controller,
    AxisRef.A,
    AxisRef.A, // The axis involved in the arc motion (starting from the
lowest index axis)
    AxisRef.B,
    5000, // center1
    5000, // center2
    0, // pos1
    0, // pos2
    1, // Direction (0: clockwise, 1: counterclockwise)
    0, // Number of revolutions
    50000, // Speed
    100000, // Acceleration
    100000, // Deceleration
    1000000, // Emergency deceleration
    0 // smoothing factor
);
controller.GetAxis(AxisRef.A).Begin(); // Start arc motion
```

7. Set the motion parameters

Set motion parameters (speed, acceleration, etc.) for a specified axis

```
AAMotionAPI.SetVectorParam(controller, AxisRef.A, 20000, 150000, 150000,
1000000, 0);
Console.WriteLine("A-axis motion parameters set")
```

Full example process

```
// Initialize and connect
// ... (Previous step)
AAMotionAPI.MotorOn(controller, AxisRef.A);
AAMotionAPI.MotorOn(controller, AxisRef.B);

// Move to the origin
AAMotionAPI.MoveAbs(controller, AxisRef.A, 0, 10000);
AAMotionAPI.MoveAbs(controller, AxisRef.B, 0, 10000);
while (controller. GetAxis(AxisRef.A). InTargetStat != 4 || controller.
GetAxis(AxisRef.B). InTargetStat != 4)
{
    Thread.Sleep(10);
}
Console.WriteLine("Reached the origin");

// Set the linear motion
AAMotionAPI.VectorLinear(controller, AxisRef.A, 30000, 30000, null, null,
null, null, null, null, 10000, 100000, 100000, 1000000, 0);
controller. GetAxis(AxisRef.A). Begin();
Thread.Sleep(2000);

// Pause and Resume
AAMotionAPI.VectorPause(controller, AxisRef.A);
Console.WriteLine("Pause motion");
```

```
Thread.Sleep(1000);
AAMotionAPI.VectorContinue(controller, AxisRef.A);
Console.WriteLine("Resume motion");

// Arc motion
AAMotionAPI.VectorArc(controller, AxisRef.A, AxisRef.A, AxisRef.B, 5000,
5000, 0, 1, 0, 50000, 100000, 1000000, 1000000, 0);
controller. GetAxis(AxisRef.A).Begin();

// Wait for the motion to end
while (controller. GetAxis(AxisRef.A). InTargetStat != 4||controller.
GetAxis(AxisRef.B). InTargetStat != )
{
    Thread.Sleep(10);
}
Console.WriteLine("Arc motion completed");

Turn off the motor
AAMotionAPI.MotorOff(controller, AxisRef.A);
AAMotionAPI.MotorOff(controller, AxisRef.B);
```

16 Keywords

16.1 Axis Keywords

Pos (Position)

Keyword	Pos
Type	int (read-only attribute).
Unit	The unit set by User Units (UsrUnits) is the user-defined unit by default, and if UsrUnits=1, it is the encoder count value.
Description	Represents the current position of the target axis, based on encoder counts, with range limits ranging from -2,147,483,648 to 2,147,483,647. After the reset, the Pos value is 0.
Function	Obtain the position of the current axis (encoder count or user unit).
Limit	- The range of values is limited to 32-bit signed integers. - Out of range causes position rollback (wraparound loop). - Use ModRev to avoid overflow issues.
Special Instructions:	- Set position for autonomous motion or calibration - UsrUnits control display units, with 1 indicating encoder values.

Usage examples

```
// 1. Initialize the controller
MotionController controller = AAMotionAPI.Initialize(ControllerType.AGM800);

// 2. Get the current position of an axis (e.g. axis A)
int currentPos = controller. GetAxis(AxisRef.A). Pos;

// 3. Output current position
Console.WriteLine($"Current axis A position (Encoder count) : {currentPos}");
```

AuxPos (Aux Encoder Position)

Keyword	**AuxPos**
Type	int (read-only attribute).
Unit	User units (AuxUsrUnits); If AuxUsrUnits=1, it is expressed as an encoder count value.
Description	Represents the current position value of the auxiliary encoder (auxiliary axis or secondary encoder), based on the encoder count. The range is limited, ranging from -2,147,483,648 to 2,147,483,647. After the Reset, the AuxPos value is 0.
Function	Obtain the current position of the auxiliary encoder (encoder count or user units) to assist in user position adjustment or calibration.
Limit	- The range of values is limited to 32-bit signed integers. - Out of range causes position rollback (wraparound loop). - Use the AuxModRev modular function to prevent data overflow.

Description:

- When the motor is turned off, the user can manually set the AuxPos value (for calibration).

- Suitable for auxiliary encoder position detection and adjustment.

PDPos (Pulse Direction Position)

Keyword	PDPos
Type	int (read-only attribute).
Unit	The number of pulses represented by user-defined units (PDUsrUnits). If PDUsrUnits=1, the unit is in pulses.
Description	Report pulse input position command reads value, reflecting the current position. After resetting, the value is 0.
Range	Between -2,147,483,648 and 2,147,483,647. When the read command is out of range, the position value is rolled back (wrap loop).
Restrictions	If the pulse input exceeds the range, the read position is rolled back, causing the position to wrap around.

Description:

PDPos reports pulse direction input position command readings, expressed in PD user units (*PDUsrUnits*). If *PDUsrUnits* = 1, then PDPos are in pulses. PDPos have a value of 0 when reset.

The PDPos count position is between -2147483648 cnts and 2147483647. If command input from PDPos exceeds these limits, position command reads are rolled back.

PDUsrUnits are only used to report the number of pulses that have been counted. The number of input pulses can also be multiplied by a factor, with the product used as a reference.

Reference = $[Number\ of\ input\ pulses] * \frac{PDFact}{PDFactDen}$

The PDPos value displayed will be:

PDPos = $\text{Reference} * PDUsrUnits$

Example:

PDUsrUnits = 5

(e.g.: 5 pulses represent 1 mm of motion, and the user wants to read commands in millimeters)

After 20 pulses are entered on the input port:

PDPos -> 4

After entering 3 more pulses (23 in total):

PDPos -> 4

Other references： PDfact , PDFactDen , PDUsrUnits , PDFiltFact , PDEncFilt , PDEncDir

Vel (speed)

Keyword	Vel
CAN code	5
Type	Parameter
Array Index Range	1 : 4
Access	Read only
Related axes	Yes
Value type	int(32 bits)
User units	User Units (UsrUnits/sec).
Allowed in motion	Yes
Allow on the motor	Yes
Whether to save to storage	No
Minimum value	-2,147,483,648
Maximum value	2,147,483,647
Default value	0

Vel array elements and descriptions

Index	Description
Vel[1]	The filter value of the speed (the average speed of smoothing processing, which can be used for graph display and noise suppression) The filter factor is determined by the VelFilt parameter.
Vel[2]	The raw velocity reading (the unfiltered velocity value calculated from the position derivative) Reflects the actual instantaneous velocity measured by the encoder.
Vel[3]	The average velocity of the 16 samples (the average of a certain sampling point in the past, reducing the effect of instantaneous variations) Based on multiple samples, providing a smoother velocity estimation.
Vel[4]	1/T velocity (reciprocal velocity value, representing the inverse of velocity, used for specific control algorithms)

Vel[] is an array that reports the master encoder speed in three different formats:

Vel[1] is the filter value for the velocity. The factors of the filter are determined according to VelFilt.

Vel[2] is the raw velocity reading.

Vel[3] is the average velocity of 16 samples.

All of the above are in UsrUnits / second.

Why do we need to filter velocity?

The raw velocity reading is the derivative of the position. Position is an integer value that represents the number of encoder counts read. What can and often happens is that the required speed corresponds to a non-integer number of encoder pulses.

For example, if the actual velocity is 10.5 counts per sample time, the position after 1 sample time would be:

$Post_1 = 10$

(Because 0.5 count cannot be detected)

The calculation speed per second for 16384 samples is 10 * 16384 = 163840 counts/second.

In the next sample time, the motor passed 10.5 counts. Since it starts at the actual position of 10.5, it will reach position 21.

$Post_2 = 21$

The controller calculates the position difference to detect the speed. The calculated velocity is 11 counts per sample time, or 11 * 16384 = 180224

We receive a difference of 18384 between the velocity readings of the two sample times, even though in fact the velocity is constant.

Filtering for velocity has two benefits:

1. Users can see smooth values when viewing graphs and better indicate the actual average speed.
2. The controls also use filter values to eliminate unwanted high-frequency noise.

Moving averages are used for user display purposes only. It is not used for control.

AuxVel (Auxiliary Speed)

Keyword	AuxVel
Description	Auxiliary velocity parameters (specific use depends on equipment design)
CAN Code	6
Type	Parameter
Access Rights	Read only
Axis Related	Yes
Numeric Type	32-bit integer (int)
User Units	Aux user units
Allowed in Motion	Yes
Allowed with motor on	Yes
Save To Flash	No
Minimum Value	-2,147,483,648
Maximum Value	2,147,483,647
Default Value	0

Description:

AuxVel reports the speed of the auxiliary encoder in auxiliary user units. Auxiliary speeds are reported in user units/seconds.

PDVel (Pulse Direction Velocity)

Keyword	PDVel
CAN Code	7
Type	Parameter
Access Rights	Read only
Axis Related	Yes
Numeric Type	32-bit integer (int)
User Units	Pulse direction user units
Allowed In Motion	Yes
Allowed with motor on	Yes
Save To Flash	No
Minimum Value	-2,147,483,648
Maximum Value	2,147,483,647
Default Value	0

Description:

Reports the speed of pulse direction input in PDUsrUnits/second. It is calculated by the derivative of PDPos.

IaErr (Phase A Current Error)

Keyword	IaErr
CAN Code	20
Type	Parameter
Access Rights	Read only
Axis Related	Yes
Numeric Type	32-bit integer (int)
User Units	milliamps (mA)
Allowed In Motion	Yes
Allowed with motor on	Yes
Save To Flash	No
Minimum Value	-2,147,483,648
Maximum Value	2,147,483,647
Default Value	0

Description

- **IaErr** represents the current error of phase "A", which reflects the deviation between the actual current and the command current, and is used to detect and control current errors.
- **Example**: If `IaRef = 1000, Ia = 990`, then `IaErr = 100 mA`.

IbErr (Phase B Current Error)

Keyword	IbErr
CAN code	21
Type	Parameter
Access rights	Read only
Axis Related	Yes
Numeric Type	32-bit integer (int)
User units	milliamps (mA)
Allowed In Motion	Yes
Allowed with motor on	Yes
Save to flash	No
Minimum value	-2,147,483,648
Maximum value	2,147,483,647
Default value	0

Description

- **IbErr** represents the current error of phase "B", reflecting the deviation of the actual current from the command current, and is used to monitor and control the current difference.
- **Example**: If `IbRef = 1000, Ib = 990`, then `IbErr = 100 mA.`

IdErr (Id Vector Current Control Error)

Keyword	IdErr
CAN code	22
type	Parameter
Access rights	Read only
Axis Related	Yes
Numeric type	32-bit integer (int)
User units	No user units
Allowed In Motion	Yes
Allowed with motor on	Yes
Save to flash	No
Minimum value	-2,147,483,648
Maximum value	2,147,483,647
Default value	0

Description

In vector control mode (refer ControlMode keyword), the current control loop is closed on Id and Iq. The Iq closed loop uses CurrRef as its reference, while the Id closed loop uses zero as its reference.

IdErr is (IdRef-Id) in [mA].

This is the error of the Id control loop in vector control mode.

Note that IdErr (like IdRef and Id) is calculated even if the controller is not in vector control mode.

Note:

In older firmware versions (prior to version 1.3.0), Id and Iq were swapped. Starting with firmware version 1.3.0, this issue has been corrected and now complies with the documentation above.

Others reference： IdRef, Id, IqRef, Iq, IqErr, Control Mode, CurrGain, CurrKi。

IqErr (Iq Vector Current Control Error)

Keyword	IqErr
CAN code	23
type	Parameter
Access rights	Read only
Axis Related	Yes
Numeric type	32-bit integer (int)
User units	No user units
Allowed In Motion	Yes
Allowed with motor on	Yes
Save to flash	No
Minimum value	-2,147,483,648
Maximum value	2,147,483,647
Default value	0

Description

In vector control mode (refer ControlMode keyword), the current control loop is closed on Id and Iq. The Iq closed loop uses CurrRef as its reference, while the Id closed loop uses zero as its reference.

IqErr is (IqRef-Iq) in [mA].

It is the error of the Iq control loop in vector control mode.

Note that IqErr (just like IqRef and Iq) is calculated even when the controller is not in vector control mode.

Note:

In older firmware versions (prior to version 1.3.0), Id and Iq were swapped. Starting from firmware version 1.3.0, this issue has been corrected and now complies with the documentation above.

Others reference： IdRef, Id, IdErr, IqRef, Iq, CurrGain, CurrKi。

PosRef (Position Reference Value)

Keyword	PosRef
CAN code	24
type	Parameter
Access rights	Read only
Axis Related	Yes
Numeric type	32-bit integer (int)
User units	User Units (UsrUnits)
Allowed In Motion	Yes
Allowed with motor on	Yes
Save to flash	No
Minimum value	-2,147,483,648
Maximum value	2,147,483,647
Default value	0

Description

- **PosRef** represents a position reference generated by the internal profiler, converted into the user-defined unit.
- This value is fed directly into the position control loop to guide the moving target.

VelRef (Speed Reference Value)

Keyword	VelRef
CAN code	25
type	Parameter
Access rights	Read only
Axis Related	Yes
Numeric type	32-bit integer (int)
User units	User Units (UsrUnits)
Allowed In Motion	Yes
Allowed with motor on	Yes
Save to flash	No
Minimum value	-1,300,000,000
Maximum value	1,300,000,000
Default value	0

Description

- **VelRef** represents a speed reference generated by an internal profiler that has been converted into user units in **UsrUnits/s**.
- This value serves as the target input for velocity control.

CurrRef (Current Reference)

Aspect	Detail
Mnemonic	CurrRef
CAN code	26
type	Parameter
Access	Read-only
Allowed in motion	Yes
Allowed with Motor On	Yes
Save to flash	No
Axis Related	Yes
Numerical range	Dynamic (depending on the configuration of the remote unit)
Default value	Dynamic (depending on the configuration of the remote unit)
User units	No user units
Implementation state	Implemented

Note:
CurrRef represents the real-time current target value used by the control loop in milliamps, which is determined by the configuration of the remote unit.

IaRef (Phase A Current Reference)

Aspect	detail
Mnemonic	IaRef
CAN Code	27
Type	Parameter
Access	Read only
Allowed In Motion	No
Allowed with Motor On	Yes
Save To Flash	No
Axis Related	Yes
Numerical Range	Dynamic (depending on remote unit configuration)
Default	Dynamic (depending on remote unit configuration)
User Units	No user units
Implementation State	Implemented

Note:

IaRef represents the current reference value applied to "Phase A" in milliamps, and the range is dynamically set by the remote unit.

IbRef (Phase B current reference)

Aspect	Detail
Mnemonic	IbRef
CAN code	28
type	Parameter
Access rights	Read only
Allowed in motion	No
Allowed with Motor On	Yes
Save to flash	No
Axis Related	Yes
Minimum value	Dynamic (dependent on remote units)
Maximum value	Dynamic (dependent on remote units)
Default value	Dynamic (dependent on remote units)
User units	No user units
Implementation state	Implemented

Description

- **IbRef** represents the current reference value of "phase B", generated in real-time by the control loop in milliamps.
- The numerical range and default values depend on the configuration of the remote-control unit.

IdRef (Vector Id Current Reference)

Aspect	Detail
Mnemonic	IdRef
CAN code	29
type	Parameter
Access rights	Read only
Allowed in motion	No
Allowed with Motor On	Yes
Save to flash	No
Axis Related	Yes
Minimum value	Dynamic (dependent on remote units)
Maximum value	Dynamic (dependent on remote units)
Default value	Dynamic (dependent on remote units)
User units	No user units
Implementation state	Firmware version FW1.3.0 and above are supported

Description

In vector control mode (refer ControlMode keyword), the current control loop is closed on Id and Iq. The Iq closed loop uses CurrRef as its reference, while the Id closed loop uses zero as its reference.

IdRef is the reference to the Id control loop in units of [mA]. In fact, it is always equal to zero.

Note that the IdRef is calculated even if the controller is not in vector control mode.

Note:

In older firmware versions (prior to version 1.3.0), Id and Iq were swapped. Starting with firmware version 1.3.0, this issue has been corrected and now complies with the documentation above.

Other reference： CurrRef, Id, IdErr, IqRef, Iq, IqErr, ControlMode, CurrGain, CurrKi.

IqRef (Vector Iq Current Reference)

Aspect	Detail
Mnemonic	IqRef
CAN code	30
type	Parameter
Access rights	Read only
Allowed in motion	No
Allowed with Motor On	Yes
Save to flash	No
Axis Related	Yes
Minimum value	Dynamic (dependent on remote units)
Maximum value	Dynamic (dependent on remote units)
Default value	Dynamic (dependent on remote units)
User units	No user units
Implementation version	1.3.0

Description

In vector control mode (refer ControlMode keyword), the current control loop is closed on Id and Iq. The Iq closed loop is referenced by CurrRef, and the Id closed loop is referenced by zero.

IqRef is the reference to the Iq control loop in [mA]. It is actually always equal to CurrRef.

Note that the IqRef is calculated even if the controller is not in vector control mode.

Note:

In older firmware versions (prior to version 1.3.0), Id and Iq were swapped. Starting with firmware version 1.3.0, this issue has been corrected and now complies with the documentation above.

Other reference： CurrRef, IdRef, Id, IdErr, Iq, IqErr, ControlMode, CurrGain, CurrKi.

ConFlt (Controller Fault Code)

Aspect	Detail
Mnemonic	ConFlt
CAN Code	31
Type	Parameter
Access Rights	Readable in motion (Yes)
Allowed With Motor On	Yes
Save To Flash	No
Axis Related	Yes
Numerical Range	-2,147,483,648 to 2,147,483,647(int range)
Default Value	0
User Units	No user units
Implementation State	Implemented

ConFlt reports an error that causes the motor to be disabled. Each failure that is input to ConFlt is also logged in ErrLog. ConFlt is cleared at the next time the motor is enabled.
The following table shows the values of ConFlt and their meanings:

value	Meaning
0	No malfunctions occurred
1001	An abort signal is detected
1002	The motor phase is shorted to ground
1003	The encoder is disconnected
1004	FPGA watchdog did not receive
1005	PWM dead time is too short
1006	Hall input disconnected
1007	Motor is stuck
1008	The bus voltage is too high
1009	The bus voltage is too low
1010	The logic voltage is too high
1011	The logic voltage is too low
1012	The bus current is too high
1013	Phase A current is too high
1014	Phase B current is too high
1015	Phase C current is too high
1016	The motor current is too high
1017	Drive power exceeds limit
1018	IPM temperature is too high
1019	Velocity too high
1020	Position error exceeds limit
1021	One of the 5V output pins is used for the encoder or I/O to activate the current limit. Check your hardware connections!
1022	CPU temperature too high

1023	The supply voltage is too low
1024	The motor current is too high
1025	Drive power exceeds limit
1026	IPM temperature is too high
1027	Velocity too high
1028	Position error exceeds the limit
1029	Encoder failure
1030	Motor overload
1031	The motor temperature is too high
1032	The motor current is unbalanced
1033	The motor voltage is unbalanced
1034	The motor phase is unbalanced
1035	Motor phase loss
1036	Motor phase short circuit
1037	The motor phase is shorted to ground
1038	Motor phase-to-phase short circuit
1039	The motor phase shorts the power supply
1040	The motor phase pair logic shorts
1041	The motor phase is shorted to ground
1042	Motor phase-to-phase short circuit
1043	The motor phase shorts the power supply
1044	Motor phase to logic short circuit
1045	The motor phase is shorted to ground
1046	Motor phase-to-phase short circuit
1047	The motor phase shorts the power supply
1048	The other axes member turns off the motor
1049	The speed difference in the full closed loop is too large
1050	External fault inputs (e.g., external drives) are activated
1051	The internal relay remains off. Try waiting longer after the power is turned on before enabling the motor
1052	STO2 is activated
1053	AmpType values are not allowed for this product
1054	At least one of the required AC power input phases is cut off

Description

- ConFlt stores the current fault code, and the fault is represented by different values for different causes.
- This value is set when a fault occurs and cleared after the next motor enables.
- The meaning of specific fault codes is detailed in the table, including hardware failure, communication interruption, overvoltage, and more.

Related information

- ErrLog: Each fault code can be found in ErrLog in detail.

MotionStat (Motion State)

Aspect	Content
Mnemonic	MotionStat
CAN code	32
type	Parameter
Access Rights	Read only
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	-2,147,483,648
Maximum value	2,147,483,647
Default value	0
User units	No user units
Implementation state	Implemented

MotionStat reports the status of the current motion. Each bit in MotionStat represents a motion state. It is possible that more than one bit will be on in certain cases. When the motor is not in motion MotionStat = 0. The table below shows the meanings of MotionStat bits:

0	In motion
1	Waiting (during repetitive motion)
2	In repetitive stop (following StopRep command)
3	Stop requested
4	In acceleration
5	In deceleration
6	Waiting for smoothing to end
7	In ECAM stop (following StopECAM command)
8	In FIFO stop (following StopFIFO command)
9	In wait for Input (motion is suspended till rising edge at the user defined input)
10	CNCA member
11	CNCA involved now
12	In the CNCA stop

StatReg (Status Register)

Aspect	Content
CAN code	33
Type	Parameter
Access	Read only
Axis Related	Yes
Value type	int（32-bit）
User units	No
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Min value	-2,147,483,648
Max value	2,147,483,647
Default value	0

StatReg is a status register. The bits of StatReg show the following status:

Bit	Status
0	Commutation done bit
1	Regeneration is on
2	Reserved
3	Maximum bus voltage exceeded (MaxVBus)
4	Under minimum bus voltage (MinVBus)
5	Main relay is on (for products with main relay)
6	Absolute maximum bus voltage exceeded (MaxVBusAbs)
8..7	Bus voltage warning : 00 – No warning. Bus voltage is deeply within its range. 01 – Low warning. VBus ≥ (0.88 * MaxVBus), or VBus ≤ (1.12 * MinV Bus) 10 – Medium warning. VBus ≥ (0.92 * MaxVBus), or VBus ≤ (1.08 * MinV Bus) 11 – High warning. VBus ≥ (0.96 * MaxVBus), or VBus ≤ (1.04 * MinV Bus) Bit 7 is the LSB.
10..9	Current reference warning: 00 – No warning. Current reference (CurrRef) is deeply within its range. 01 – Low warning. Abs(CurrRef) > (0.88 * PeakCL)

	10 – Medium warning. Abs(CurrRef) > (0.92 * PeakCL) 11 – High warning. Abs(CurrRef) > (0.96 * PeakCL) Bit 9 is the LSB.
12..11	Power unit temperature warning: 00 – No warning. PwrTemp is deeply within its range. 01 – Low warning. PwrTemp > (0.88 * MaxPwrTemp) 10 – Medium warning. PwrTemp > (0.92 * MaxPwrTemp) 11 – High warning. PwrTemp > (0.96 * MaxPwrTemp) Bit 11 is the LSB.
14..13	General control saturation warning: This warning bits indicates how deep the control algorithms are in one of these saturations: Velocity saturation (MaxVel) or Current saturation (PeakCL or ContCL when continuous current limitation is active or limits by analog inputs when activated or by fixed values when activated) or voltage saturation (MaxPWM). In case that any of these saturations is activated (maximal or minimal values) at a given control cycle (sample), a counter is increased. If this counter, over a period of 1 second, is higher than the following defined thresholds, the proper warning state is signaled at this status bits. This warning state is updated once per second and stays unchanged for a period of 1 second (so the relevant LEDs at the PC Suite can be easily observed by the user). This warning is extremely important, as once the control algorithms enter one of its saturations, it means that the tracking performance are limited to some level (depend how deep is the saturation). The depth of the saturation is measured as the period of time at which at least one of the saturations was activated, within a period of 1 second. For example, a saturation depth of 100msec means that at least during an accumulated period of 100ms, within a period of 1 second, the control algorithms hit one of the saturations. This means: 10% of the control cycles were in saturation (one of them, not necessarily the same one at all cycles), or at least 1638 cycles (typically with Agito controllers the sampling frequency is 16384 Hz). Note that the warning here is signaled starting from very low saturation depths. This sis because saturation is a critical state for motion performance and that sometimes the user is not aware to the saturation state. With these status bits, the event of saturation can be easily noticed. 00 – No warning. The saturation depth is less than 0.2% . 01 – Low warning. The saturation depth is more than 0.2%. 10 – Medium warning. The saturation depth is more than 1%. 11 – High warning. The saturation depth is more than 5%. Bit 13 is the LSB. Note that bits 21 to 23, as described below, provides the momentary status of each saturation (velocity, current and voltage).
16..15	Motor temperature warning:

	00 – No warning. MotorTemp is deeply within its range. 01 – Low warning. MotorTemp > (0.88 * MaxMotorTemp) 10 – Medium warning. MotorTemp > (0.92 * MaxMotorTemp) 11 – High warning. MotorTemp > (0.96 * MaxMotorTemp) Bit 15 is the LSB. (These bits are in use only from FW version 1.4.0)
17	Reverse hardware limit is activated. This is identical to the status at LimitsStat.
18	Forward hardware limit is activated. This is identical to the status at LimitsStat.
19	The position reference (PosRef) is smaller than the software position forward limit (RevPLim).
20	The position reference (PosRef) is greater than the software position forward limit (FwdPLim).
21	Current saturation is activated. The current saturation is activated if abs(CurrRef) is larger than one of: PeakCL or ContCL when continuous current limitation is active, or limits by analog inputs when activated, or by fixed values when activated. Note that in contrast to bits 14.. 13 that are described above, this bit reflects the momentary state of the saturation (as was activated or not activated during the recent control cycle).
22	Voltage saturation is activated. Voltage saturation refers to Va or Vb or Vc, compared to ± MaxPWM. Note that with bit 14 described above: 13 Unlike this bit, this bit reflects the instantaneous state of saturation (whether it was activated during the most recent control cycle).
23	Velocity saturation is activated. The v elocity saturation refers to VelRef , as compared to ± MaxVel . Note that in contrast to bits 14.. 13 that are described above, this bit reflects the momentary state of the saturation (as was activated or not activated during the recent control cycle).
25..24	Other warning . This is a general warning, for various purposes as follows: 00 – No warning. 01 – Low warning. Currently not in use. 10 – Medium warning. I^2t current limit is activated. 11 – High warning. Currently not in use. Bit 24 is the LSB.
26	The definition of at least one of the high order (bi-quad) filters, at the velocity control loop or at the position control loop, was modified by the user. This bit is set after one of the following parameters is assigned with a new value: VelFiltDef[], VelFiltOn[], PosFiltDef[] or PosFiltOn[]. This bit reflects a non-valid state, at which the definitions of the filters are not correlated to the internal coefficients of the filters. Note that as a result, the controller will not allow to enable the axis (MotorOn=1) before CalcFilters is successfully executed.
27	CalcFilters execution was failed. This means that the internal coefficients of the bi-quad filters do not match the definition of the filters. Note that as a result, the controller will not allow to enable the axis (MotorOn=1) before CalcFilters is successfully executed.

28	Dynamic brake status bit. It is set to “1” when the dynamic brake is activated.
29	Static brake lock request. It is set to "1" when the static brake is commanded to Lock state and to "0" when the static brake is (requested to) release.
30	Home switch is enabled. Valid only if a discrete input has been programmed as Home input for this axis.

The “Warning” bits, as described above, can be used by the user to identify an “almost” fault situation.

Note that “Agito PC Suite” software uses the bits values of this status parameter to control the LEDs at its status panel. The warning statuses, which have 4 values (none, low, medium and high) are reflected as a multi-color LED at the PC Suite status panel (off, yellow, orange and red, respectively).

Refer Also:

MotionStat, MotionReason, ConFlt, MotorOn, VBus, MaxVBus, MinVBus, CurrRef, PeakCL, PwrTemp, MaxPwrTemp, MaxVel, ContCL, CurrLimMode, CurrLimFwd, CurrLimRev, LimitsStat, RevPLim, FwdPLim, PosRef, Va, Vb, Vc, MaxPWM, VelRef, PeakTime, VelFiltDef[], VelFiltOn [], PosFiltDef[], PosFiltOn[] and . CalcFilters

MasterPos (Gear Master Position)

Aspect	Content
CAN code	44
type	Parameter
Access rights	Read only
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
minimum	-2,147,483,648
maximum	2,147,483,647
Default	0
User units	In user units
Implementation state	Implemented
Associate parameters	See details in： MasterFact, MotionMode

Description

- The value returned by **MasterPos** is the main axis position in the gear transmission configuration for closed-loop control and position synchronization.
- It is a read-only parameter that cannot be modified by the user but can be read in motion.
- Ideal for monitoring or synchronizing main axis position in multi-axis motion.

IndexStat（Index Status）

Items	Content
Mnemonic	IndexStat
CAN code	45
type	Parameter
Access rights	Read only
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	0
Maximum value	1
Default value	0
User units	No user units
Implementation state	Implemented

Description

IndexStat saves the current state ("0" or "1") of the main encoder's index input (only available for incremental and SinCos encoders).

If you need to detect the encoder's index event in your application, you can read the parameter value periodically to achieve response actions.

Refer Also:

StopOnIndex , IndexPos , AuxIndexStat and AuxIndexPos.

AuxIndexStat (Auxiliary Index Status)

Items	Content
Mnemonic	AuxIndexStat
CAN code	46
type	Parameter
Access rights	Read only
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	0
Maximum value	1
Default value	0
User units	No user units
Implementation state	Implemented

Description

AuxIndexStat is used to reflect the current state of the index signal of the auxiliary encoder.

A value of "1": The index signal is triggered.

Value of "0": Not triggered.

It is mainly used for calibration or special synchronization needs of auxiliary encoders.

Since it is a read-only parameter, the user can use it to monitor the real-time status of the index signal, but it cannot be modified.

This parameter value can be read periodically when you need to implement position calibration or monitor the status of auxiliary index.

Refer Also:

AuxIndexPos, IndexStat and IndexPos.

IndexPos(Index Position)

Items	Content
Mnemonic	IndexPos
CAN code	47
type	Parameter
Access rights	Read only
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	-2,147,483,648
Maximum value	2,147,483,647
Default value	0
User units	In user units
Implementation state	Implemented

Description

IndexPos saves the last main encoder position where the encoder index was detected.

IndexPos is only available for incremental and SinCos encoders.

Note that, to have a proper detection of the index signal and its position, the encoder speed must be less than the controller sampling frequency so that the index pulse width is at least as wide as the individual sample.

Refer Also:

StopOnIndex , IndexStat , AuxIndexStat and AuxIndexPos.

AuxIndexPos (Auxiliary Index position)

Items	Content
Mnemonic	AuxIndexPos
CAN code	48
type	Parameter
Access rights	Read only
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	-2,147,483,648
Maximum value	2,147,483,647
Default value	0
User units	Additional Aux user units

Description

AuxIndexPos saves the last auxiliary encoder position where the auxiliary encoder index was detected.

AuxIndexPos is only available for incremental encoders.

Note that, to have a proper detection of the index signal and its position, the encoder speed must be less than the controller sampling frequency so that the index pulse width is at least as wide as the individual sample.

Refer Also:

AuxIndexStat, IndexStat and IndexPos.

LimitsStat(Limit Status)

Items	content
Mnemonic	LimitsStat
CAN code	49
type	Parameter
Access rights	Read only
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	-2,147,483,648
Maximum value	2,147,483,647
Default value	0
User units	No user units
Implementation state	Implemented

Detailed description

The LimitsStat parameter is an integer value that stores the state of two important limit inputs.

Bit 0 (least significant bit): Represents the state of the Reverse Limit Switch (RLS).

- 0: Not active
- 1: Activate (trigger limit).

Bit 1: Represents the state of the Forward Limit Switch (FLS).

- 0: Not active
- 1: Activate

Other bits are not defined or reserved. The specific limit status can be determined using a bitwise AND operation.

Example: If the LimitsStat is 3 (binary 11), it means that both limits are activated.

This parameter is read-only and can be used by the user to monitor whether the current machinery has reached the limit state and ensure the safety of motion.

If you need to monitor the limit state in motion control, you can read this parameter to get the real-time situation.

MotorType

Items	Content
Mnemonic	MotorType
CAN code	50
type	Parameter
Access rights	Read / Write
Allowed in motion	No
Allowed with motor on	No
Save to flash	Yes
Axis Related	Yes
minimum	0
maximum	7
Default	0 (Unknown)
User units	No user units
Implementation state	Implemented

MotorType is the type of motor that is connected to the controller. This variable determines how voltage is applied to the motor. Selecting the wrong MotorType can lead to serious consequences.

The values that can be assigned to a MotorType are:

Value	Motor type
0	Unknown
1	DC brush
2	Voice Coil
3	Linear DC brushless
4	Rotary DC brushless
5	Simulation

The meaning of each motor type is described below:

Unknown

This is the default for new controllers. No voltage will be applied to the power stage output until the user sets another valid motor type.

DC brush

If the DC brush motor type is set, the full voltage is applied to both power terminals. Refer to the hardware manual to determine which terminals to use.

Voice Coil

This motor type is the same as DC brush.

Linear/Rotary DC brushless

Voltage is applied to the three motor phases, and commutation is used to determine the voltage applied to each motor power terminal. The actual difference between motor types lies in the feedback configuration and the handling of motor poles. Separating the types helps with configuration in PCSuite.

Refer to the hardware manual for the correct phase connection.

Simulation

This motor type is used for simulation during development. It allows for the generation of simulated profiles, inputs, and output behaviors without actual connecting the motor to the controller.

ContCL (Continuous Current Limit)

Items	Content
Mnemonic	ContCL
CAN code	51
type	Parameter
Access rights	Read / Write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
Minimum value	Dynamic (according to the remote unit) – the actual value is dynamically determined by the control system
Maximum value	Dynamic (based on remote units)
Default value	Dynamic (based on remote units)
User units	No user units

Precautions and Description

ContCL (Continuous Current Limiting in [mA]) is used with the PeakCL and PeakTime parameters to define the behavior of the amplifier/motor's I^2 power limiting scheme.

- **Dynamic value**: Since the value depends on the remote hardware/unit, the actual settings will change dynamically and need to be adjusted according to the specific configuration of the system.
- **Function**: Together with PeakCL and PeakTime, limit the motor current during continuous operation to avoid overheating or damage.

Related parameters and references

- **PeakCL:** Peak current limit.
- **PeakTime**: The peak duration.
- **MotorCurr**: Current motor current value.
- **StatReg**: A state register parameter.

PeakCL (Peak Current Limit)

Items	Content
Mnemonic	PeakCL
CAN code	52
type	Parameter
Access rights	Read / Write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
Minimum value	Dynamic (dependent on remote units, actual values are controlled by hardware)
Maximum value	Dynamic (dependent on remote units, actual values are controlled by hardware)
Default value	Dynamic (dependent on remote units)

Basically, PeakCL is the peak current limit (in [mA]). The current command (CurrRef) never exceeds this value (± PeakCL for saturation CurrRef).

The maximum allowable value for PeakCL (see table above) is the maximum allowable current for this product.

Note that PeakCL refers to the peak value of the sinusoidal signal of the current when sinusoidal commutation, not its RMS value.

However, the PeakCL parameter, along with the ContCL and PeakTime parameters, is used by the controller to implement the I^2t amplifier/motor power limit scheme.

The maximum values of PeakCL, ContCL, and PeakTime define the I^2t limit scheme to ensure that the amplifier's capacity is not exceeded. However, users can modify these parameters to define a new limit scheme, such as matching the power limit of the motor.

For example, a product can have the following maximums:

PeakCL = 16000

ContCL = 8000

PeakTime = 1000

This means that the product can drive up to 16000mA (16A) in 1000 milliseconds (1 second) with a continuous current limit of 8000mA (8A). This is a limitation of the product itself.

However, for a motor/application, the user can set:

PeakCL = 5000

ContCL = 3000

PeakTime = 500

It should not exceed the motor's power capability, or—if the application does not provide rated cooling conditions—should also stay within the amplifier's power capacity.

How does an amplifier/motor I2T power limit scheme work?

This limit uses the following parameters: ContCL (continuous current limit), PeakCL (peak current limit), and PeakTime (maximum allowable peak current time).

The limit code always calculates I^2 (using the overall motor current: MotorCurr) over time (I^2 integral) If it becomes higher than $ContCL^2$, the limit will be activated.

Once the limit is activated, the CurrRef is usually limited (saturated) at ± PeakCL and is now limited at ±ContCL. Therefore, in practice the amplifier is now limited to deliver current not exceeding ContCL.

At any time, I^2 calculates over time, using a first-order filter to simulate the heat dissipation of the motor (or more precisely, the motor temperature). In this way, I^2 "integrates" over time to estimate the temperature generated inside the amplifier/motor. The coefficients of the first-order filter are functions of PeakTime, PeakCL, and ContCL, therefore:

After a long period of no current to the motor, if a PeakCL current is driven now ($PeakCL^2$ is the input of the filter), during the PeakTime time, the output of the filter will reach the $ContCL^2$ value, which will trigger the limit and limit the current to ContCL.

Of course, if the current driven to the motor is lower than PeakCL, the available time will be longer if the limit is not triggered. In fact, if the current is equal to or lower than ContCL, the limit will never be activated.

Note that the availability of PeakCL current, during the PeakTime time, is only after there is no current to the motor for a long time (hence the filter output is 0). In other cases (e.g., 0.9*ContCL for a long time), the actual peak time will be (significantly) shortened, because the filter output will reach $ContCL^2$ faster (this is equivalent to the motor temperature behavior, since in this case the motor is already heated due to the presence of non-zero current over time).

Once I^2 falls below 0.9 x $ContCL^2$ over time, the limit is lifted. This is to create a hysteresis so that the limit does not turn on/off/on/off quickly around continuous current.

Note that all parameters of this restriction can be modified dynamically.

Note that the limit only applies to CurrRef saturation, usually it is PeakCL. It has no effect on other limitations supported by the controller: current limits or fixed values using analog inputs (refer: CurrLimMode).

Note that, although the user can set values such as PeakCL=100 and ContCL=200 (i.e., Peak value < Continuous or Peak=Continuous), but this condition is not valid and the controller will use

a different continuous current value internally, equal to PeakCL/2. Once the user has set both parameters correctly, the controller will return values using PeakCL and ContCL.

The limit status is reflected in the StatReg parameter, at the 24th and 25th bits. In PCSuite, when the limit is activated, you will see the "Other Warnings" LED turn "Orange".

Note:
This I^2t power limiting scheme only works when the current control loop is active (refer ControlMode parameters) or if an external amplifier is used and CurrRef is used to drive the analog output.
In the others case, CurrRef is used in the limit equation instead of MotorCurr (see explanation above).

Refer Also:

ContCL, PeakTime, MotorCurr and StatReg.

Notes:

- **Dynamic Parameter Adjustment**: These parameters can be adjusted during runtime to match different load and cooling conditions.
- **Parameter relationship**: If the PeakCL is set lower than ContCL, the system will automatically adjust, and the actual continuous current will become 1/2 of the PeakCL to ensure safety.
- **Limit Impact**: This limit only affects the peak value command (CurrRef) saturation and does not affect other hardware or simulation limitations (such as analog input limits).
- **Prerequisites**: Make sure the current control is activated (ControlMode parameter) or the external amplifier is enabled.

PeakTime

Items	Content
Mnemonic	PeakTime
CAN code	53
type	Parameter
Access rights	Read / Write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
Minimum value	1 millisecond
Maximum value	3000 ms (i.e. 3 seconds)
Default value	500 milliseconds
User units	No user units

Detailed description

Action: PeakTime is defined as how long it is allowed to apply after peak current (PeakCL) starts (maximum peak current dürfen is applied within this time). If it is exceeded, the thermal power limit will be triggered but actually it is limited to the ContCL (continuous current) level.

Functional relationship:

- Combined with PeakCL and ContCL, the controller utilizes the I^2t algorithm to perform thermal management of motor power.
- The longer the PeakTime, the longer it is allowed to run at peak current, slowing down the thermal limit.

Impact:

- During the peak current duration (not exceeding PeakTime), the current command can reach PeakCL.
- If PeakTime is exceeded, the limit will be activated and the current will be limited to the ContCL range.

Practical Application:

- Users can set PeakTime to balance performance with thermal protection.

PolePrs (Pole pairs)

Items	Content
Mnemonic	**PolePrs**
CAN code	54
type	Parameter
Allowed in motion	No
Allowed with motor on	No
Save to flash	Yes
Axis Related	Yes
Minimum value	1
Maximum value	50
Default value	4
User units	No user units

Precautions and Description

PolePrs is the number of magnetic pole pairs per revolution of a controlled motor. This number, along with EncRes, is used to determine the voltage pattern applied to the motor's phase. Please use the motor datasheet to enter the correct PolePrs value.

For linear motors, enter PolePrs = 1, EncRes = number of encoders counts per revolution

Warning:

If PolePrs is wrong, unexpected behavior will occur. This can lead to serious damage to the controller, motor, or any other system component connected to the motor.

Refer Also:

- EncRes**: Encoder resolution (number of pulses per electrical cycle),** used in conjunction with PolePrs.

EncType

Items	Content
Mnemonic	EncType
CAN code	55
type	Parameter
Access rights	Read only (No)
Allowed in motion	No
Allowed with motor on	No
Save to flash	Yes
Axis Related	Yes
Minimum value	0
Maximum value	6
Default value	1 (Incremental Encoder)
User units	No user units

EncType sets the encoder feedback type connected to the controlled motor. The values that can be assigned to EncType are shown below. Note that some of these encoder types are not available on all products, and some require special options.

value	Encoder type
0	Unknown
1	Incremental encoders
2	Sin/Cos
3	Endat
4	SSI
5	Nikon 17-bit encoder

Relevant parameters

- **EncRes**: Encoder resolution (number of pulses or resolution, to be configured for accurate feedback).

EncRes (Encoder Resolution)

Items	Content
Mnemonic	EncRes
CAN code	56
type	Parameter
Access rights	Read only (No)
Allowed in motion	No
Allowed with motor on	No
Save to flash	Yes
Axis Related	Yes
Minimum value	1
Maximum value	2,000,000
Default value	1
User units	No user units

EncRes is the number of encoders counts in mechanical rotation. This number, along with PolePrs, is used to determine the voltage pattern applied to the motor's phase. Please use the motor datasheet to enter the correct EncRes value.

For linear motors, enter PolePrs = 1 and EncRes = Encoder count per electrical cycle.

Warning:

If wrong EncRes, unexpected behavior will occur. This can lead to serious damage to the controller, motor, or any other system component connected to the motor.

Relevant parameters

- **PolePrs**: The number of magnetic pole pairs per mechanical rotation
- **EncType**: Encoder type (e.g., Incremental, Sin/Cos, etc.).

Items	Content
Mnemonic	EncFilt
CAN code	57
type	Parameter
Access rights	Read/Write (Yes)
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
Minimum value	0
Maximum value	255
Default value	10

Specifies the digital filter to be applied to the incremental main encoder input channels (A, B, and Index).

The filter is implemented by hardware.

Suitable for AG300 controllers

The characteristics of the filter are as follows:

- The input is sampled by the filter mechanism. The input level is confirmed only after 6 consecutive samples with the same value. This means that the signal's "1" logic needs to be sampled at least 6 times, and the same "0" logic also needs to be sampled, for a total of (2 * 6) samples.
- The filter frequency (sample frequency) is determined by the EncFilt parameter.
- If EncFilt == 0, the filter frequency is equal to 300MHz, which is the clock frequency of the DSP.
- If EncFilt is not equal to 0, the filter frequency is:

$$Filter\ Frequency = \frac{300}{2 \cdot EncFilt} \quad [MHz]$$

- Therefore, the maximum theoretical input frequency (in the absence of noise) for A, B, and Index signals can be calculated using the following formula:

$$Max\ Input\ Frequency = \frac{FilterFrequency}{2 * 6} \quad [MHz]$$

$\therefore if\ EncFilt = 0,$

$$Max\ Input\ Frequency = \frac{300}{2 \cdot 6} \quad [MHz]$$

$Else,$

$$Max\ Input\ Frequency = \frac{300}{(EncFilt \cdot 2)(2 \cdot 6)} \quad [MHz]$$

- Perhaps more applicable is the associated maximum speed of the axis (after x4 counts/sec), as follows:

$$Max\ Axis\ Speed = \frac{FilterFrequency}{3} \quad [\frac{counts}{sec}]$$

$\therefore$ *If EncFilt = 0:*

$$Max\ Axis\ Speed = \frac{300 * 10^6}{3} \quad [\frac{counts}{sec}]$$

Else:

$$Max\ Axis\ Speed = \frac{300 * 10^6}{EncFilt \cdot 6} \quad [\frac{counts}{sec}]$$

Note:

- The "maximum input frequency" (or maximum axis speed) set by EncFilt is the theoretical upper limit of the input frequency (axis speed), assuming an ideal square wave signal (infinite slope) and ideal electronics (The receiving chip in the controller has no delay or slope). The actual "maximum input frequency" will be smaller, depending on the quality of the signal.

- Also, if noise on the signal "interferes" with the filter counting a given logic level for 6 consecutive samples. Therefore, it is recommended not to set EncFilt too high.

- In summary, we recommend setting EncFilt to a calculated theoretical maximum axis speed which is twice the actual maximum speed in the system.

- Setting the actual value of EncFilt in a given system can be preliminarily set up based on the formula/considerations above but must be carefully tested and adjusted in the system to filter out the noise that occurs on the encoder signal and the quality of the signal itself.

- From a filter perspective, the fastest input signal can be 25MHz (EncFilt = 0 and assuming no noise), which correlates with a maximum speed of 100 x 10^6 counts/second.

- The fastest filter frequency is 300 MHz (EncFilt = 0) and the slowest is about 0.59 MHz (EncFilt maximum value is 255).

The characteristics of the filter are as follows:

- The input is sampled by the filter mechanism. The input level is confirmed only after 4 consecutive samples have the same value. This means that the signal's "1" logic needs to be sampled at least 4 times, and the same "0" logic also needs to be sampled, for a total of (2 * 4) samples.

- The filter frequency (sample frequency) is determined by the EncFilt parameter.

$$Filter\ Frequency = \frac{100}{2^{EncFilt\ +\ 1}} \quad [MHz]$$

- Therefore, the maximum theoretical input frequency (in the absence of noise) for encoder signals A, B, and Index can be calculated using the following formula:

$$Max\ Input\ Frequency = \frac{100}{2^{EncFilt\ +\ 1} \cdot (2 \cdot 4)} \quad [MHz]$$

Or through this table:

EncFilt	Filter clock	Maximum encoder frequency
0	CLK/2 (50MHz)	CLK/16 (6.25MHz)
1	CLK/4 (25MHz)	CLK/32 (3.125MHz)
2	CLK/8 (12.5MHz)	CLK/64 (1.5625MHz)
3	12.5 *10 6	
4	6.25 *10 6	
5	781.25 *10 3	
6	390.625 *10 3	
7	195.3125 *10 3	

Note:

- The "maximum input frequency" (or maximum axis speed) set by EncFilt is the theoretical upper limit of the input frequency (axis speed), assuming an ideal square wave signal (infinite slope) and ideal electronics (no delay and slope in the receiving chip in the controller). The actual "maximum input frequency" will be smaller, depending on the quality of the signal.

- Also, if noise on the signal "interferes" with the filter counting a given logic level for 6 consecutive samples. Therefore, it is recommended not to set EncFilt too high.

- In summary, we recommend setting EncFilt to a calculated theoretical maximum axis speed which is twice the actual maximum speed in the system.

- Setting the actual value of EncFilt in a given system can be preliminarily set up based on the formula/considerations above but must be carefully tested and adjusted in the system to filter out the noise that occurs on the encoder signal and the quality of the signal itself.

- From a filter perspective, the fastest input signal can be 6.25MHz, meaning $25 \cdot 10^6$ counts/second (EncFilt = 0 and assuming no noise). If you need faster input, consult Agito.

- The fastest filter frequency is 50 MHz (EncFilt = 0) and the slowest is 781.25 kHz (EncFilt maximum value is 7).

EncDir (Encoder Direction)

Items	Content
Mnemonic	EncDir
CAN code	58
type	Parameter
Access rights	Read only (No)
Allowed in motion	No
Allowed with motor on	No
Save to flash	Yes
Axis Related	Yes
Minimum value	0
Maximum value	1
Default value	0 (represents the default forward direction, that is, when the forward direction is clockwise, the encoder value increases)

EncDir can be used to change the direction of the encoder's reading. If the encoder reading is moving forward in a positive direction when the motor is rotating clockwise, and EndDir = 0, set EncDir = 1 to make the reading move in a negative direction as the motor rotates clockwise.

AuxEncType (Auxiliary Encoder Type)

Items	Content
Mnemonic	AuxEncType
CAN code	59
type	Parameter
Access rights	Read only (No)
Allowed in motion	No
Allowed with motor on	No
Save to flash	Yes
Axis Related	Yes
Minimum value	0
Maximum value	6
Default value	1 (Incremental Encoder)

AuxEncType sets the auxiliary encoder feedback type. The values that can be assigned to AuxEncType are listed below. Note that some of these encoder types are not available on all products, and some require special options.

Value	Encoder type
0	Unknown
1	Incremental encoders
2	Sine/cosine
3	Endat
4	SSI
5	Nikon 17-bit encoder
6	Absolute BiSS-C
7	Analog position feedback
8	Tamagawa

Relevant parameters

AuxEncFilt (Auxiliary Encoder Digital Filter)

Items	Content
Mnemonic	AuxEncFilt
CAN code	60
type	Parameter
Access rights	Read/Write (Yes)
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
Minimum value	0
Maximum value	255
Default value	0

AuxEncFilt defines the behavior of the hardware digital filter for the auxiliary incremental encoder input signal (A, B, and Z).

For more information on filter implementations, refer to the EncFilt manual page.

The filter on the main encoder (controlled by EncFilt) and the filter on the auxiliary encoder (controlled by AuxEncFilt) are two separate filters. However, they are implemented in the same way.

AuxEncDir (Auxiliary Encoder Direction)

Items	Content
Mnemonic	AuxEncDir
CAN code	61
type	Parameter
Access rights	Read Only (No)
Allowed in motion	No
Allowed with motor on	No
Save to flash	Yes
Axis Related	Yes
Minimum value	0
Maximum value	1
Default value	0 (No reverse by default)

Enables reverse reading in the direction of the auxiliary encoder (suitable for all types of encoders).

If AuxEncDir == 0, the auxiliary encoder is read based on its wiring configuration to the drive.

If AuxEncDir == 1, the auxiliary encoder will read in the reverse direction.

PDEncFilt (Position Feedback Filter)

Items	Content
Mnemonic	PDEncFilt
CAN code	62
type	Parameter
Access rights	Read / Write
Axis Related	Yes
Numeric type	int 32 bit
User units	No user units
Allowed in motion	No
Allowed with motor on	Yes
Save to flash	Yes
Minimum value	-2,147,483,648 (Maximum negative integer value)
Maximum value	2,147,483,647 (Maximum positive integer value)
Default value	0

Definition:
This parameter is used to control whether the position feedback signal (usually is the encoder position feedback) is digitally filtered, and the specific parameters of the filtering.

General use:
Adjust the digital filter parameters of PD feedback to optimize the system's ability to suppress high-frequency noise to obtain smoother and more stable position feedback.

Suggested to use:

- In practical applications, it is generally recommended to start with the default value of 0, and if the noise reduction is needed, it can be set to a positive integer (the larger the filtering, the stronger the filtering), but it should be adjusted under the condition that the system response speed is sufficient.
- Setting is too large can lead to feedback delays, affecting dynamic responses.

PDEncDir (Position Feedback Direction Reversal)

Items	Content
Mnemonic	PDEncDir
CAN code	63
type	Parameter
Access rights	Read / Write
Axis Related	Yes
Numeric type	int 32 bit
User units	No user units
Allowed in motion	No
Allowed with motor on	Yes
Save to flash	Yes
Minimum value	-2,147,483,648 (Maximum negative integer value)
Maximum value	2,147,483,647 (Maximum positive integer value)
Default value	0

Definition:
This parameter is used to control whether the position feedback signal (usually provided by the encoder) is reversed. When set to a non-zero value, the direction of the feedback signal is inverted, helping to match the actual mechanical motion or the reverse of the hardware wiring.

Application Scenarios:

- Cooperate with the actual hardware wiring to ensure that the feedback direction is consistent with the motion control logic.
- Adjust when the system is debugged or hardware is installed to ensure that the position feedback is in the correct direction.

Setup Suggestions:

- The default value is 0 (no inversion).
- If the actual system feedback direction is opposite to the control logic, the value can be set to 1 to invert.

Notes:

- This parameter cannot be modified during motion; it is recommended to adjust it during initialization or shutdown.

UsrUnits

Items	Content
Mnemonic	UsrUnits
CAN code	64
type	Parameter
Access rights	Read / Write
Allowed in motion	No
Allowed with motor on	No
Save to flash	Yes
Axis Related	Yes
Minimum value	1
Maximum value	2,147,483,647 (32-bit maximum integer value)
Default value	65,536

UsrUnits allow users to read positions and their derivatives in units other than counting encoder. UsrUnits is the ratio between the required units and the encoder count. Full instructions on how user units work can be found at the beginning of this document.

Example:

If the user wants to see position readings in millimeters and every 5 encoder counts equate to 1 mm, set UsrUnits to 5.

Position readings are now received in millimeters, velocity in millimeters per second, and acceleration in millimeters per second^2.

Refer Also:

AuxUsrUnits , PDUsrUnits

AuxUsrUnits (Auxiliary Feedback Position Units)

Items	Content
Mnemonic	AuxUsrUnits
CAN code	65
type	Parameter
Access rights	Read / Write
Allowed in motion	No
Allowed with motor on	No
Save to flash	Yes
Axis Related	Yes
Minimum value	1
Maximum value	2,147,483,647
Default value	65,536
User units	No user units
Implementation state	Implemented
Example	If the user wants the auxiliary position to be expressed in millimeters (mm) and the encoder count corresponds to 1mm for every 5 encoder counts, the UsrUnits can be set to 5. The auxiliary position is displayed in millimeters, with velocity in mm/sec and acceleration in mm/sec².
Associate parameters	UsrUnits, PDUsrUnits

Description

Aux User Units (AuxUsrUnits) allow the user to read the auxiliary feedback position and its derivatives in non-encoder count units. AuxUsrUnits is the ratio between the required units and the encoder count. Full instructions on how user units work can be found at the beginning of this document.

Example:

If the user wishes to see the auxiliary position readings in millimeters and the count of every 5 encoder counts on the auxiliary encoder is equivalent to 1 mm, set the UsrUnits to 5.

Auxiliary position readings will now be received in millimeters, velocity in mm/s, and acceleration in mm/s².

Refer Also: UsrUnits, PDUsrUnit

PDUsrUnits (Pulse Direction Command Units)

Items	Content
Mnemonics	PDUsrUnits
CAN code	66
type	Parameter
Access rights	Read / Write
Allowed in motion	No
Allowed with motor on	No
Save to flash	Yes
Axis Related	Yes
Minimum value	1
Maximum value	2,147,483,647
Default value	65,536
User units	No user units
Implementation state	Implemented

Description

PDUsrUnits allow users to read pulse direction commands and their derivatives in non-pulse units. PDUsrUnits are the ratio between the desired unit and the pulse. A full explanation of how user units work can be found at the beginning of this document.

Example:

If the user wants to view the position command in millimeters and every 5 pulses equals 1 mm, set UsrUnits to 5.

The position commands will be received in millimeters now.

Refer Also:

AuxUsrUnits , UsrUnits

EmulRat (Analog Encoder Ratio)

Items	Content
Mnemonic	EmulRat
CAN code	69
type	Parameter
Access rights	Read / Write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
Minimum value	0
Maximum value	65,536
Default value	0
User units	No user units
Implementation state	Implemented

EmulRat determines the ratio between the main encoder input and the encoder emulation output.

There are 4 types of emulated encoder signals:

1. A – Logic signal on the A side of the digital incremental encoder signal
2. B – Logic signal on the B side of the digital incremental encoder signal
3. Z – Index signal
4. Pulse – For each edge conversion of the analog encoder signal, a 5-microsecond pulse is generated

The output of the emulation encoder signal is routed only to the differential output signal, as shown below:
- Emulation encoder A is connected to differential output 1
- Simulation encoder B is connected to differential output 2
- Analog encoder Z is connected to differential output 3
- Emulation encoder pulse connected to differential output 4

They can be configured by selecting DOutSelect for a specific output with a value of 1.

Example:

If EmulRat = 8, an emulation encoder output pulse will be generated for every 8 input pulses.

Associate parameters

- DOutPort: Controls the differential output port

- DOutSelect: Select the type of analog signal corresponding to the output

ModRev

Items	Content
Mnemonic	ModRev
CAN code	70
type	Parameter
Access rights	Read / Write
Allowed in motion	No
Allowed with motor on	No
Save to flash	Yes
Axis Related	Yes
Minimum value	0
Maximum value	2,000,000,000
Default value	0 (Close Modulus)
User units	In user units

Description

ModRev is used for the modulo mode of the controller. In the modulo mode the main position (Pos) is between 0 and ModRev. When the actual position exceeds ModRev the value of Pos is calculated as part of ModRev. This allows a rotary axis to move in the same direction indefinitely without exceeding any numerical limits.

ModRev = 0 turns off the Modulo mode.

Notes:

1. Auto-Phasing (the power on process to align the controller with the motor electrical cycle) may fail if ModRev is not zero, and/or is not high enough. Please set ModRev to zero before performing the Auto-Phasing and set it to the desired value once the Auto-Phasing is completed.
2. Do not use Modulo together with input shaping.
3. If you manually set the position using SetPosition to a value that exceeds the range of ModRev unexpected behavior may occur.
4. Endless motion (ModRev not 0) can be used together with smoothing (Jerk not 0), but the user must take care to properly set ModRev and Jerk, according to the following guidelines:

    a. The time (in number of samples) for a full revolution of the modulus (moving ModRev distance), at the maximal speed, should be longer than the time associated with the Jerk value (2^{Jerk} is the number of samples).

This can be easily solved by properly setting ModRev (large enough).

For example, if the maximal speed is 10,000,000 [counts/sec] and Jerk = 9 (i.e. 2^9 = 512 samples), and the controller sampling rate is 16,384 [samples/sec], then ModRev must be higher than 10,000,000 / 16,384 * 512 = 312,500 [counts].

b. The multiplication of the number of samples associated with the Jerk (it is 2^{Jerk} [samples]) and ModRev (in [counts]) must be smaller than (2^{31}-1).

This can be easily solved by properly setting ModRev (small enough).

Continuing the above example, if Jerk = 9, then the ModRev must be smaller than:

(2^{31}-1) / 512 = 4,194,304 [counts]

P lease consult Agito for the proper way to set ModRev and Jerk, for a given application that requires smoothing and endless motion together , if you, from some reason, can't set ModRev as high as needed or as low as needed, according to the above guidelines.

Relevant parameters

- **Jerk**: Motion smoothing parameter

MotorOn

Items	Content
Mnemonic	MotorOn
CAN code	130
type	Parameter
Access rights	Read / Write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	0
Maximum value	1
Default value	0 (motor off)
User units	No user units
Implementation state	Implemented

Description:

MotorOn: Enable/disable the motor.

MotorOn = 0 disables the motor.

MotorOn = 1 Enable the motor.

When the motor is disabled, power is not applied to the motor and is not controlled by the driver.

The motor can be disabled internally by the driver due to a fault (refer ConFlt).

When the user re-enabled the motor, the ConFlt was cleared. If the actual state causing the failure is not corrected (e.g., overtemperature causes the motor to be disabled, the user tries to enable the motor before cooling), the motor will remain disabled.

Some command and parameter changes are only allowed when the motor is disabled. Refer to the properties of each parameter.

InjectType (Inject signal type)

Items	Content
Mnemonic	InjectType
CAN code	112
type	Parameter
Access rights	Read / Write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	0
Maximum value	6
Default value	0
User units	No user units
Implementation state	Implemented

InjectType determines which signal type will be injected when using the injection function. Possible injection types are:

Value	Injection type
0	No injection
1	Direct sine signal
2	Add sine signal
3	Direct square signal
4	Add square signal
5	Direct pulse

The required signal type is injected to the control loop that is selected by InjectPoint

Direct sine signal

A sinusoidal signal is injected instead of any other reference that is supposed to be used in the injection point. For example, if the injected signal is injected to the position loop, the position reference that is generated by the profiler is ignored and replaced by a sinusoidal reference.

Add sine signal

A sinusoidal signal is added any other reference that is supposed to be used in the injection point. For example, if the injected signal is injected to the position loop, a sinusoidal signal is added to the position reference that is generated by the profiler.

Direct square signal

A square wave signal is injected instead of any other reference that is supposed to be used in the injection point. For example, if the injected signal is injected to the position loop, the position reference that is generated by the profiler is ignored and replaced by a square wave reference. This can be used as a simulated step response.

Add square signal

A square wave signal is added any other reference that is supposed to be used in the injection point. For example, if the injected signal is injected to the position loop, a square wave signal is added to the position reference that is generated by the profiler.

Direct pulse

A single pulse is injected instead of any other reference that is supposed to be used in the injection point.

Refer Also:

InjectPoint , InjectCurrAmp , InjectVelAmp , InjectPosAmp , InjectFreq , InjectValue , InjectTimeOn , InjectCurrDC

ScheduleSet

Items	Content
Mnemonic	ScheduleSet
CAN code	261
type	Parameter
Access rights	Read / Write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	1
Maximum value	5
Default value	1
User units	No user units
Implementation state	Implemented

Additional Notes:

- `ScheduleSet` is primarily used to configure the parameter set of the Gain Scheduling feature.
- For detailed scheduling parameters and behavior definitions, refer to `ScheduleMode` keyword page.
- Scheduling can dynamically adjust control parameters to adapt to different working conditions and improve system performance.

Example:

- Set `ScheduleSet = 3` indicates that a specific gain configuration is applied using Scheduling Scheme No. 3.

MotionSamples

Items	Content
Mnemonic	MotionSamples
CAN code	267
type	Read only
Access rights	Allow reading in motion (Yes)
Allowed with motor on	Yes
Array index range	1 : 4
Save to flash	No
Axis Related	Yes
Minimum value	-1
Maximum value	2,147,483,647 (32-bit maximum)
Default value	-1 (means no exercise has been played)
User units	No user units
Implementation state	Implemented

Description:

Index 1: Sample analyzer

Index 2: Sample up to target

Index 3: Sample until the target is reached

Index 4: Sample settling time

MotionSamples[] is an array parameter that reports the time duration of the last (recently completed) motion.

The time is reported in number of controller samples (typically for Agito's controllers: 1/16384=61 μs per sample).

Three times are reported, as follows:

MotionSamples[1]: The time of the motion profile itself.

Independent of the actual motion of the motor/axis.

MotionSamples[2]: The time of the motion profile itself plus the time it takes for the motor to settle into the target (**not including** the InTargetTime).

MotionSamples[3]: The time of the motion profile itself plus the time it takes for the motor to settle into the target (**including** the InTargetTime).

After power on or reset, all MotionSamples[ ] elements are set to -1, to indicate that no motion was performed yet.

Others reference: InTargetTime, InTargetTol, InTargetStat.

PosErr (Position Error)

Items	Content
Mnemonic	PosErr
CAN code	18
type	Read only
Access rights	Allow reading in motion (Yes)
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	- 2,147,483,648 (minimum of 32-bit signed integers)
Maximum value	2,147,483,647 (maximum of 32-bit signed integers)
Default value	0
User units	In user units
Implementation state	Implemented

Description:

PosErr returns the value of the position error, expressed in user units (UsrUnits). Positional error is the difference between the position command (PosRef) and the actual position. If the value of PosErr exceeds the maximum allowable position error defined in MaxPosErr, the motor is disabled.

Example:

If PosRef = 1000 and Pos = 990，then PosErr = 10

Refer Also:

N/A

VelErr (Velocity Error)

Items	Content
Mnemonic	VelErr
CAN code	19
type	Read only
Access rights	Allow reading in motion (Yes)
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	- 2,147,483,648 (minimum of 32-bit signed integers)
Maximum value	2,147,483,647 (maximums of 32-bit signed integers)
Default value	0
User units	In user units/sec
Implementation state	Implemented

Description:

VelErr returns the value of the speed error in user units (UsrUnits)/second. The velocity error is the difference between the velocity command (VelRef) and the actual velocity (Vel). If the value of VelErr exceeds the maximum allowable position error defined in MaxVel Err, the motor will be disabled.

Example:

If VelRef = 10000 and Vel = 9900, then VelErr = 100

Refer Also:

UsrUnits , MaxVel Err

DPosRef (Deviation of Target Position)

Items	Content
Mnemonic	DPosRef
CAN code	155
type	Read only
Access rights	Allow reading in motion (Yes)
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	- 2,147,483,648 (32-bit signed integer minimum)
Maximum value	2,147,483,647 (32-bit signed integer maximum)
Default value	0
User units	In user units
Implementation state	Implemented

Description:

- `DPosRef` represents the deviation value of the target position from the current position (Pos) in user units.
- It is often used for deviation detection in feedback regulation or control algorithms.

ComtStatus（Commutation Status）

Items	Description
Mnemonic	ComtStatus
CAN code	143
type	Parameter
Array index range	1 : 2
Access rights	Read only
Axis Related	Yes
Value type	32-bit integer (int).
User units	No user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Minimum value	0
Maximum value	2,147,483,647
Default value	0

Status Value Meaning (Index 1)

Value	Status Description
0	Not yet completed (waiting or auto-phasing not started)
1	In Process (auto-phasing running)
100	Successful commutation (auto-phasing succeeded)
200	Auto-phasing not required
300	Successful completion (using Hall sensor)
400	Successful completion (using Hall sensor)
500	Successfully completed (parameters modified!!)
600	Virtual phasing in aging mode
-1	Failed: Motor unexpectedly disabled (see MotorReason command)
-2	Failed: Unknown auto-phasing method
-3	Failed: Search exceeded 360 degrees
-4	Failed: Encoder disconnection detected
-5	Phasing required: Related parameters were modified
-6	Failed: Hardware not ready. Please power cycle

Value	Status Description
-7	Failed: Invalid Hall signal. Please power cycle
-8	Phasing required: Central-i remote unit disconnected
-9	Failed: Invalid Hall value (0 or 7) during self-learning
-10	Failed: Illegal Hall sequence during self-learning
-11	Failed: Invalid Hall value. EncDir was modified
-12	Failed: Illegal Hall sequence. EncDir was modified

Ia (Phase "A" Current)

Items	Content
Mnemonic	Ia
CAN code	9
type	Parameter
Access rights	Read only
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	- 2,147,483,648 (32-bit signed minimum)
Maximum value	2,147,483,647 (32-bit signed maximum)
Default value	0
User units	No user units

Description:

Ia represents the current value of the phase "A" in milliamps (mA), defined according to the hardware connection scheme.

Example:
If the detected current is 1500 mA, the value of Ia is 1500.

This parameter is commonly used to monitor the real-time value of each phase current to ensure that the motor operates within a safe range.

If other phase current parameters (e.g. Ib, Ic, Id) are required

Ib (phase "B" current)

Items	Content
Mnemonic	Ib
CAN code	10
type	Parameter
Access rights	Read only
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	- 2,147,483,648 (32-bit signed minimum)
Maximum value	2,147,483,647 (32-bit signed maximum)
Default value	0
User units	No user units

Description:

Ib represents the current value of the phase "B" in milliamps (mA), determined by the hardware connection configuration.

Example:
If the detected current is 1600 mA, the value of Ib is 1600.

This parameter is used to monitor the current status of the phase "B", helping to detect anomalies or optimize control.

Id (Linear Axis Current)

Items	Content
Mnemonic	Id
CAN code	11
type	Parameter
Access rights	Read only
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	- 2,147,483,648 (32-bit signed minimum)
Maximum value	2,147,483,647 (32-bit signed maximum)
Default value	0
User units	No user units

Description:

In vector control mode (refer ControlMode keyword), the current control loop is closed on Id and Iq. The Iq closed loop uses CurrRef as its reference, while the Id closed loop uses zero as its reference.

Id is calculated from Ia and Ib and is a non-effective current, meaning a current that does not produce torque (as opposed to Iq that produces torque).

Id is in [mA].

Note that the Id is calculated even if the controller is not using vector control. Users can monitor and analyze the Id and Iq to estimate the efficiency of the motor operation.

Normally Iq should be equal to CurrRef and Id should be equal to zero.

Note:

In older firmware versions (prior to version 1.3.0), Id and Iq were swapped. Starting from firmware version 1.3.0, this issue has been corrected and now complies with the documentation above.

Others reference:IdRef, IdErr, IqRef, Iq, IqErr, ControlMode.

Iq (Effective Current/Torque Current)

Items	Content
Mnemonic	Iq
CAN code	12
type	Parameter
Access rights	Read only
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	- 2,147,483,648 (32-bit signed minimum)
Maximum value	2,147,483,647 (32-bit signed maximum)
Default value	0
User units	No user units
Implementation version	Firmware version 1.3.0 and above

Description:

In vector control mode (refer ControlMode keyword), the current control loop is closed on Id and Iq. The Iq closed loop is referenced by CurrRef, and the Id closed loop is referenced by zero.

Iq is calculated from Ia and Ib and is the effective current, meaning the current that produces torque (as opposed to Id not producing torque).

Iq is measured in [mA].

Note that the Iq is computed even if the controller is not using vector control. Users can monitor and analyze the Id and Iq to estimate the efficiency of the motor operation.

Normally Iq should be equal to CurrRef and Id should be equal to zero.

Note:

In older firmware versions (prior to version 1.3.0), Id and Iq were swapped. Starting from firmware version 1.3.0, this issue has been corrected and now complies with the documentation above.

Others reference: IdRef, Id, IdErr, IqRef, IqErr, ControlMode.

Va (phase "A" voltage)

Items	Content
Mnemonic	Va
CAN code	13
type	Parameter
Access rights	Read only
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	Dynamic (dependent on remote control unit, varies with device)
Maximum value	Dynamic (dependent on remote control unit, varies with device)
Default value	Dynamic (remote control unit dependent)

Description:

- `Va` represents the voltage on the phase "A" in percentage (%) of full PWM cycles.
- "Full PWM" refers to the maximum modulation amplitude, and the `Va` value range varies with different hardware remote units and is determined dynamically.
- **Purpose:** Real-time reflection of the voltage level of phase "A", helping to monitor the drive voltage status.

Va returns the voltage value applied to phase "A", expressed as a percentage of 0.1% full PWM.

Example: The value of Va is 53, which means that the PWM command for phase A is 5.3% (this value is given in 0.1% increments).

Va-related parameters (e.g., `Vb`, `Vc`).

Vb (Phase "B" Voltage)

Items	Content
Mnemonic	Vb
CAN code	14
type	Parameter
Access rights	Read only
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	Dynamic (dependent on remote control unit, varies with device)
Maximum value	Dynamic (dependent on remote control unit, varies with device)
Default value	Dynamic (remote control unit dependent)

Description:

- `Vb` represents the voltage value of the phase "B" in percentage (%) over the full PWM cycle.
- "Full PWM" represents the maximum modulation amplitude, and the range of `Vb` varies with the hardware remote control unit and is determined dynamically.
- **Purpose**: Used to monitor the voltage status of phase "B" in real time, aiding in diagnosis and debugging.

Vc (Phase "C" Voltage)

Items	Content
Mnemonic	Vc
CAN code	15
type	Parameter
Access rights	Read only
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	Dynamic (dependent on remote units, changing with hardware)
Maximum value	Dynamic (dependent on remote units, changing with hardware)
Default value	Dynamic (dependent on remote units)

Description:

- `Vc` represents the voltage value of the phase "C", and the percentage represents the corresponding full PWM amplitude.
- "Full PWM" refers to the maximum modulation amplitude, which varies with hardware and is determined dynamically.
- **Usage**: Used to monitor the voltage status of phase "C" in real time, facilitating debugging and status monitoring.

Vd (Direct-axis voltage)

Items	Content
Mnemonic	Vd
CAN code	16
type	Parameter
Access rights	Read only
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	Dynamic (dependent on remote units, changing with hardware)
Maximum value	Dynamic (dependent on remote units, changing with hardware)
Default value	Dynamic (dependent on remote units)

Description:

The Vd parameter is only used when vector control is activated (refer ControlMode keyword).

Vd is the output of the PI controller in the id current control loop.

Vd and Vq are used to calculate Va and Vb (required motor phase voltage).

Vd is expressed in internal units (usually in Agito's controller: 100% PWM when Vd value is 4577).

When the vector control is not activated, Vd is equal to zero.

Others reference:Vq, IdRef, IqRef, Id, Iq, IdErr, IqErr, ControlMode, VA, Vb, Vc。

Vq (Alternating Voltage)

Items	Content
Mnemonic	Vq
CAN code	17
type	Parameter
Access rights	Read only
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	Dynamic (dependent on remote units, changing with hardware)
Maximum value	Dynamic (dependent on remote units, changing with hardware)
Default value	Dynamic (dependent on remote units)

Description:

The Vq parameter is only used when vector control is activated (refer ControlMode keyword).

Vq is the output of the PI controller in the Iq current control loop.

Vd and Vq are used to calculate Va and Vb (required motor phase voltage).

Vq is expressed in internal units (usually in Agito controllers: 100% PWM when the Vq value is 4577).

When vector control is not activated, Vq is equal to zero.

Others reference:vd, idRef, IqRef, id, iq, idErr, IqErr, ControlMode, va, vb, vc.

HomingStat (Homing Status)

Items	Content
Mnemonic	HomingStat
CAN code	342
type	Parameter
Access rights	Read only
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	- 2,147,483,648 (32-bit signed minimum)
Maximum value	2,147,483,647 (32-bit signed maximum)
Default value	0
User units	No user units

Description:

- `HomingStat` represents the status or code of the current Homing process.
- For a detailed description of the homing function, please refer to the HomingOn keyword page, which involves the homing process, state machine, and control parameters.
- **Use**: Monitors the real-time status of the homing operation to determine whether it is completed or abnormal.

Value	Description
0	Not executed
100	Successfully completed
-1	Failed due to wrong parameters
-2	Failed due to timeout
-3	Failed due to motor unexpectedly disabled (refer to the MotorReason command).
-4	Failure due to wrong motion
-5	Failed due to wrong step type
-6	Failure due to in-motion
-7	Because the number of steps is out of range
-8	Failure due to unexpected limitations (obstacles, etc.).
-9	Unable to set position (see SetPosition command)

Value	Description
-10	Motion /gear mode out of range
-11	The error compensation value is out of range
-12	The auto-phasing is not executed, the operation failed

MotionReason (The reason for the termination of the most recent motion

Items	Content
Mnemonic	MotionReason
CAN code	43
type	Read only
Access rights	Read only
Axis Related	Yes
Numeric type	Int 32 bit
User units	No user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Value range	0 to 2,147,483,647
Default value	0

Function description:

MotionReason returns a value indicating the reason that caused the most recent motion to end. The Begin command resets the MotionReason to 0.

The following table lists the possible values of MotionReason and what they mean:

- Value = 0: Normal Current motion still not ended or Motion Ended normally
- Value = 1: Motion ended due to Stop command
- Value = 2: Motion ended due to Abort command
- Value = 3: Motion ended due to StopRep command
- Value = 4: Motion ended due to reverse limit switch detection
- Value = 5: Motion ended due to forward limit switch detection
- Value = 6: Motion ended due to reverse software limit
- Value = 7: Motion ended due to forward software limit
- Value = 8: Motion ended due to motor disable (refer MotorReason command)
- Value = 9: Motion ended due to a StopECAM command
- Value = 10: Motion ended due to a StopFIFO command
- Value = 11: Motion ended due to detected index
- Value = 12: Motion ended due to a StopCNCA command
- Value = 13: Motion ended due to timeout at homing
- Value = 14: Motion ended due to GoToCurrMode command
- Value = 15: Motion ended due to hard stop at homing
- Value = 16: Motion ended due to home switch change
- Value = 17: Motion ended due to GoToForceMode command
- Value = 18: Motion ended due to CNC participating axis disabled

- Value = 19: Motion ended due to CNC involved axis is stopped
- Value = 20: Motion ended due to CNC axis is aborted
- Value = 21: Motion ended due to stop by input signal
- Value = 22: Motion ended due to abort by input signal
- Value = 23: Motion ended due to one member of CNCA hitting the forward/reverse limit switch detection
- Value = 24: Motion ended due to one member of CNCA hit the forward/reverse software limit
- Value = 25: Motion ended due to a StopCNCB command
- Value = 26: Motion ended due to one member of CNCB hit the forward/reverse software limit
- Value = 27: Motion ended due to one member of CNCB hit the forward/reverse limit switch detection
- Value = 28: Motion ended due to digital input-controlled stop
- Value = 29: Motion ended due to user request to stop vector motion
- Value = 30: Motion ended due to vector participating axis being disabled
- Value = 31: Motion ended due to vector participating axis being stopped
- Value = 32: Motion ended due to vector participating axis being aborted
- Value = 33: Motion ended due to vector participating axis hit the forward/reverse limit switch detection
- Value = 34: Motion ended due to vector participating axis hit the forward/reverse software limit
- Value = 35: Motion ended due to user request to stop interpolation motion
- Value = 36: Motion ended due to interpolation participating axis being disabled
- Value = 37: Motion ended due to interpolation participating axis being stopped
- Value = 38: Motion ended due to interpolation participating axis being aborted
- Value = 39: Motion ended due to interpolation participating axis hit the forward/reverse limit switch detection
- Value = 40: Motion ended due to interpolation participating axis hit the forward/reverse software limit
- Value = 41: Motion ended due to jog motion exceeds forward/reverse software limit

Example:

If the motion was ended by an abort command, but during deceleration the forward software limit was exceeded, and then the limit switch was encountered, MotionReason will have a value of 2, indicating the original reason to stop and ignoring any following events that could have stopped the motion.

InTargetStat (Target Arrival State)

Items	Content
Mnemonic	InTargetStat
CAN code	268
type	Read only
Access rights	Read only
Axis Related	Yes
Numeric type	Int 32 bit
User units	No user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Value range	0 to 4
Default value	0

Status codes and meanings

Value	Status description	Explanation
0	The motor is off	The motion system is not started or the motor is not activated
1	Servo enabled	The servo is enabled, ready or in motion
2	In motion	Currently performing a motion
3	Wait for InTargetTime to end	The target position has been reached, but the InTargetTime is still waiting for the confirmation time (InTargetTime) to complete to confirm the target has arrived
4	The target has been reached	Confirm that the target has been reached after the duration exceeds the InTargetTime within the allowable tolerance (InTargetTol).

Additional Notes

How it works:
When the position error is less than the InTargetTol in user units and the duration reaches the InTargetTime milliseconds, the InTargetStat is set to **4**, indicates that the target has been reached.

Related parameters:

- **InTargetTol**: Position error threshold
- **InTargetTime**: Duration of continuous confirmation

MotionMode

Items	Content
Mnemonic	MotionMode
CAN code	141
type	Read / Write
Axis Related	Yes
Numeric type	Int 32 bit
User units	No user units
Allowed in motion	No (cannot be changed during motion)
Allowed with motor on	Yes
Save to flash	Yes
range	-1 to 19
Default value	-1 (indicates not set or invalid)

Description

MotionMode determines the type of motion that will be executed after the next Begin command. MotionMode cannot be changed until the current motion ends.

The values of MotionMode and the related motion type are:

MotionMode	Resulting Motion
0	Jog – The motor will reach the velocity set in Speed and continue in constant speed until a Stop command is received. The acceleration and deceleration used for the motion are as defined by Accel and Decel . The motion may also be smoothed according to the value of Jerk.
1	Point to Point – Move to the position defined by RelTrge of AbsTrgt (if RelTrgt = 0). The other parameters of the motion are as defined by Accel, Speed, Decel and Jerk.
2	Point to point repetitive – A repetitive motion from the current position to the the position defined by RelTrge of AbsTrgt (if RelTrgt = 0) and back. The other parameters of the motion are as defined by Accel, Speed, Decel and Jerk. The motor will wait at each end of the motion for the duration determined by RptWait . The motion is stopped using StopRep.
3	Pulse Direction direct mode – The position reference to the position loop is received directly from the pulse direction input.
4	Pulse Direction indirect mode – The position command to this motion is determined according to the pulse – direction input. However, the number of pulses received is not immediately used as the reference to the position loop. The number of pulses that were received is added to the current position target. The profiler builds a motion trajectory to the new target using the values of all the motion parameters: Accel, Decel, Speed, Jerk.

	This means that even if a large number of input pulses is entered at once, the resulting motion will not look like a step response but will be smoother. Note that the response to fast input will be smoother, but also slower.
5	Gear direct motion
6	Gear indirect motion
7	ECAM direct motion
8	ECAM indirect motion
9	FIFO motion
10	Slave position reference
11	CNC Group A motion
12	Joystick direct motion, input analog in signal representing position reference
13	Joystick indirect motion, input analog in signal representing position reference, and this profiler can be setting by user
14	Joystick direct motion, input analog in signal representing velocity reference
15	Joystick indirect motion, input analog in signal representing velocity reference, and this profiler can be set by user
16	Vector motion
17	CNC Group B motion
18	Spline buffer motion

**Note: Some motion modes are available when the firmware upgrade to some version.

Refer Also:

Accel, Decel, Speed, Jerk, AbsTrgt, RelTrgt, Begin, RptWait, StopRep

Jerk (Smoothing parameter)

Items	Content
Mnemonic	Jerk
CAN code	139
type	Read / Write
Axis Related	Yes
Numeric type	Int 32 bit
User units	No user units
Allowed in motion	No - cannot be modified during exercise
Allowed with motor on	Yes
Save to flash	Yes
range	0 to 13
Default value	0

Jerk defines the time it takes to build the desired acceleration (or deceleration) in all profiler-based motions.

The time to build the acceleration is calculated by:

Jerk Time = SAMPLE_TIME * 2^{Jerk}

SAMPLE_TIME is the sampling time of the controller, typically 1/16384=61 ms with Agito controllers.

So, for example, if JERK = 7, the jerk time is 7.808ms.

Note that the Jerk is the only motion profile parameter that cannot be modified during the motion.

The Jerk Time is used whenever acceleration starts or ends, and similarly for the deceleration.

Clearly, with higher values of Jerk, the motion time is longer. However, it is smoother and reduced overshoots are expected. The jerk shall be tuned for optimal performance per specific application.

Notes:

1. Smoothing (Jerk not 0) can be used together with endless motion (ModRev not 0), but the user must take care to properly set Jerk and ModRev, according to the following guidelines:
    1. The time (in number of samples) for a full revolution of the modulus (moving ModRev distance), at the maximal speed, should be longer than the time associated with the Jerk value (2^{Jerk} is the number of samples).

       This can be easily solved by properly setting ModRev (large enough).

       For example, if the maximal speed is 10,000,000 [counts/sec] and Jerk = 9 (i.e. 2 9 = 512 samples), and the controller sampling rate is 16,384 [samples/sec], then ModRev must be higher than 10,000,000 / 16,384 * 512 = 312,500 [counts].

2. The multiplication of the number of samples associated with the Jerk (it is 2 Jerk [samples]) and ModRev (in [counts]) must be smaller than (2 31 -1).

   This can be easily solved by properly setting ModRev (small enough).

   Continuing the above example, if Jerk = 9, then the ModRev must be smaller than:

   $(2^{31}-1) / 512 = 4,194,304$ [counts]

Please consult Agito for the proper way to set ModRev and Jerk, for a given application that requires smoothing and endless motion together, if you, for some reason, can't set ModRev as high as needed or as low as needed, according to the above guidelines.

2. The Jerk (smoothing) value appears on the PC Suite at almost all motion windows. However, please note that the Jerk value is not written to the controller when a motion button is pressed (unlike other parameters like acceleration and speed). This is because the Jerk cannot be modified while motor is on. So, if a new value for smoothing (jerk) is needed, the user should change the smoothing value, and then disable the motor, use “Apply Changes” to write the new Jerk value (it will be now written, as the motor is disabled) and only then use the motion buttons (the motion will now use the new smoothing/Jerk value).

Refer also:

Accel, Decel, Speed, RelTrgt, AbsTrgt and ModRev.

Accel (Acceleration)

Items	Content
Mnemonic	Accel
CAN code	136
type	Read / Write
Axis Related	Yes
Numeric type	int 32 bit
User units	User units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	100 to 2,000,000,000
Default value	100,000

Accel is the acceleration used for all motions in user units / sec^2. This acceleration is also used with pulse and direction command input when the indirect mode is applied.

Accel can be changed during motion and the latest value will be effective immediately. If the current motion is still accelerating, the acceleration rate will be changed. If constant velocity was reached the new acceleration value will be used the next time acceleration is required.

Refer Also:

AbsPosTrgt, RelPosTrgt, Vel, Speed, Decel, MaxAcc

Decel (Deceleration)

Items	Content
Mnemonic	Decel
CAN code	137
type	Read / Write
Axis Related	Yes
Numeric type	Int 32 bit
User units	User units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	100 to 2,000,000,000 (User Units/s²)
Default value	100,000

Definition and function description

Decel is the deceleration used for all motions in user units / sec². This deceleration is also used with pulse – direction command input when the indirect mode is applied.

Decel can be changed during motion and the latest value will be effective immediately. If the current motion is still decelerating, the deceleration rate will be changed. Otherwise, the new acceleration value will be used the next time acceleration is required.

Decel is also used for deceleration after a Stop command was entered.

For emergency stops EmrgDec is used.

Refer Also:

EmrgDec, Stop

EmrgDecel(Emergency Deceleration)

Items	Content
Mnemonic	EmrgDec
CAN code	140
type	Read / Write
Axis Related	Yes
Numeric type	int 32 bit
User units	User units/sec^2
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	100 to 2,000,000,000 (User Units/s^2)
Default value	100,000

EmergDec is the emergency deceleration in UsrUnits /Sec 2. This parameter should normally be set to a much higher deceleration than decal. The value in EmrgDec is used when the motion has to be stopped for one of the following reasons:

- Hardware limit
- Software limit

Refer Also:

RevPlim , FwdPlim , Decel , DInMode

Speed

Items	Content
Mnemonic	Speed
CAN code	138
type	Read / Write
Axis Related	Yes
Numeric type	int 32 bit
User units	User Units per Second (UsrUnits/Sec)
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	-1,300,000,000 to 1,300,000,000 user units/sec (-130 million to 130 million)
Default value	0

Definition and role

The value of Speed determines the desired speed of the motion in UsrUnits /Sec. Speed can be changed on the fly during motion. If the current motion is in the acceleration phase or in the constant speed phase it will change to the updated speed.

Example:

Spee d = 10,000

… (all other motion parameters and mode)

Begin

The above sequence starts a motion with desired speed of 10,000.

If after the actual velocity reaches 10,000 Speed is changed:

Speed = 20,000

(No Begin needed)

The motor will accelerate to 20,000 using the current set acceleration.

Refer Also:

Accel , Decel , AbsTrgt , RelTrgt , Start

MaxVel (Max Speed)

Items	Content
Mnemonic）	MaxVel
CAN code	80
type	Read / Write
Axis Related	Yes
Numeric type	int 32 bit
User units	User Units per Second (UsrUnits/Sec)
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	0 to 1,300,000,000 user units/sec
Default value	100,000

Definition and role

MaxVel determines the maximum allowed velocity for the controlled motor. The velocity reference from the velocity loop is limited by MaxVel. If the actual reading from the feedback exceeds MaxVel by more than 25% the motor is disabled.

MaxAcc (Maximum Acceleration)

Items	Content
Mnemonic	MaxAcc
CAN code	81
type	Read / Write
Axis Related	Yes
Numeric type	int 32 bit
User units	No user units
Allowed in motion	No - only set before exercise, not dynamically changed during motion
Allowed with motor on	Yes
Save to flash	Yes
range	-2,147,483,648 to 2,147,483,647 (full range)
Default value	0

Definition and role

MaxAcc is the maximum acceleration allowed by Agito PC suite motion window. In this window the user can enter the desired acceleration as a percentage of this value. It is saved in the controller but not used by it, so it is possible for the user to set a higher acceleration while ignoring this value.

InTargetTol(In Target Tolerance)

Items	Content
Mnemonic	InTargetTol
CAN code	265
type	Read / Write
Axis Related	Yes
Numeric type	int 32 bit
User units	In user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	0 to 2,147,483,647 (32-bit signed integer maximum)
Default value	10

Definition and role

Function: InTargetTol is the “In target” tolerance. If the position error in user units is less than InTargetTol for at least InTargetTime milliseconds then InTargetStat receives the value that indicates that the target was reached.

During motion, if the current position error (the difference between the target position and the current position) is less than that tolerance and stays at least the InTargetTime milliseconds, the target state is marked as reached (InTargetStat will receive a value indicating that the target has been reached).

Usage Scenarios:

Target achievement criteria in motion control: the error is within the tolerance range and remains stable for a certain duration.

Improve the accuracy of motion completion determination to avoid misjudgments caused by excessive error or transient deviations.

Relevant parameters

Parameter	Description
InTargetTime	The time the error stays within tolerance range (milliseconds)
InTargetStat	Status indication to determine whether the motion goal has been achieved (reached or not reached)

Tolerance Values:

Set according to the actual motion accuracy requirements.

The smaller the value, the stricter the judgment, and the more accurate the end of the motion, but it may fail to meet the standard multiple times due to small vibrations or noise.

If the value is larger, the judgment is looser, which is suitable for occasions where the error is difficult to eliminate completely.

Time setting:

Set the InTargetTime to ensure the stability of the target.

Typical value range: 10 to 100 milliseconds, adjusted according to the speed and smoothness of the motion.

InTargetTime

Items	Content
Mnemonic	InTargetTime
CAN code	266
type	Read / Write
Axis Related	Yes
Numeric type	int 32 bit
User units	Milliseconds (milliseconds are units of time, no user conversion required)
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	0 to 9,999 ms
Default value	3

Definition and role

Function: InTargetTime specifies the duration for which a target position is considered "achieved" in milliseconds.

When the current position error is within the InTargetTol tolerance, the system will start timing until the error continues within the tolerance range for milliseconds or more than the InTargetTime milliseconds (InTargetStat trigger).

This can avoid misjudgment caused by transient errors or oscillations during motion and improve the reliability of motion determination.

Application Scenarios:

Target completion determination in precision control systems.

Wait for a certain amount of time after the position is reached to ensure stable motion.

Time setting:

The longer the time, the more "conservative" the judgment will be, which will avoid misjudgment but delay the target arrived identification.

It is usually set between 1 and 10 milliseconds, adjusted according to actual control needs.

PTPKeepMoving (Keep Motion from Point to Point)

Items	Content
Mnemonic	PTPKeepMoving
CAN code	625
type	Read / Write
Axis Related	Yes
Numeric type	int 32 bit
User units	No user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
range	0 to 1
Default value	0

Definition and role

Function:

PTPKeepMoving controls special behavior in point-to-point (PTP) motion:

When set to 1, the motion continues during motion if it has not yet reached the target position and is not interrupted by the "Stop" command at the target position.

When set to 0, the motion will pause or stop when the target position is reached, end the motion.

LockEn

Items	Content
Mnemonic	LockEn
CAN code	170
type	Read / Write
Axis Related	Yes
Numeric type	int 32 bit
User units	No user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
range	0 or 1
Default value	0

LockEn = 1 will enable position lock. When position lock is enabled, the change in the specified input (clockwise motion rises, counterclockwise motion fall) will lock the main encoder position.

Locks and events are mutually exclusive, so when locking is enabled, events are automatically disabled.

The position value during the input change is saved in "LockVal". The number of a position that is locked is saved in "LockCntr".

For incremental encoders, the position is locked by hardware and is therefore very accurate. For absolute encoders, the position read in the next sample after the lock input change is saved in "LockVal".

LockVal and LockCntr are updated only once per sample time. If multiple valid input changes occur in a single sample time, only the last change is recorded.

Refer Also:

LockVal, LockCntr, LockSrc

LockCntr (Position Lock Counter)

Items	Content
Mnemonic	LockCntr
CAN code	172
type	Read / Write
Axis Related	Yes
Numeric type	int 32 bit
User units	No user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
range	- 2,147,483,648 to 2,147,483,647
Default value	0

Definition and role

Function:

LockCntr records the number of input changes detected that match the lock source (LockSrc) condition since the last time Position Lock was enabled (LockEn was triggered by 0 to 1 or 1 to 0).

Key points:

Lock by changing LockEn from 0 to 1 or reset the counter when it changes from 1 to 0.

Users can also reset this counter by directly assigning it to 0.

Relevant parameters

Parameter	Function Description
LockEn	Enable or disable position lock
LockVal	The current lock position value
LockSrc	Input source that triggers position lock
LockCntr	The number of input changes detected since the last lock/unlock

LockVal

Items	Content
Mnemonic	LockVal
CAN code	173
type	Read only
Axis Related	Yes
Numeric type	int 32 bit
User units	In user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
range	- 2,147,483,648 to 2,147,483,647
Default value	0

Function:

LockVal saves the latest locked position information after input changes during position lock.

For incremental encoders: this value is saved directly by the hardware and is extremely accurate.

For absolute encoders: the current position is automatically saved in the next sampling cycle after the input changes.

Application Scenarios:

Used to record or trace back the current position value of the lock point as a position reference.

LockEn and LockSrc can be used in combination to monitor the impact of input changes on the lock position.

LockSrc (Lock Function Source Input Number)

Items	Content
Mnemonic	LockSrc
CAN code	171
type	Read / Write
Axis Related	Yes
Numeric type	int 32 bit
User units	No user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	-38 to 38
Default value	1

Definition and role

Function:

LockSrc specifies the input signal number (usually numerically encoded, representing a specific digital input) to be used as a position lock.

When this input signal changes (up or down), the system saves the current position value to LockVal and the number of changes to LockCntr.

Inputs set to LockSrc can be defined as different roles in DInMode (Digital Input Mode), such as limit switches or other functions.

Application Scenarios:

Input number 3 can be used as both a left limit switch and a locking source. This setup allows for different functions such as trigger locking or limit detection on the same physical input.

Relevant parameters

Parameter	Function Description
LockEn	Enable/disable position lock
LockVal	The current lock position value
LockCntr	The number of changes detected since the last lock/unlock
DInMode	Role definition for digital inputs (different roles used to assign input signals)

LockValTable

Items	Content
Mnemonic	LockValTable
CAN code	386
type	Array with index range of: 1 : 65000
Access rights	Read only
Axis Related	Yes
Numeric type	int 32 bit
User units	In user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
range	- 2,147,483,648 to 2,147,483,647
Default value	0

Definition and role

Function:

LockValTable is an array that holds multiple lock position values, with indexes ranging from 1 to 65,000.

These values can be used to store lock positions in different states, such as different lock points, position points, or history record.

Since it is a read-only array, the system automatically maintains and the user can read the lock position values of a particular index.

Application Scenarios:

In complex motion control, it is used to store the position values of multiple locking points.

Implementing multi-point locking or backup lock position management.

Indexing facilitates quick access to different lock states.

LockTimeTable

Items	Content
Mnemonic	LockTimeTable
CAN code	387
type	Array with index range of: 1 : 65000
Access rights	Read only
Axis Related	Yes
Numeric type	int 32 bit
User units	No user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
range	-2,147,483,648 to -147,483,647 (Note: The range is negative, the range setting should be confirmed)
Default value	0

Definition and role

Function:

LockTimeTable is an array that stores multiple sets of position lock times, with indexes ranging from 1 to 65,000.

Each time value defines the time of the corresponding lock point (usually in milliseconds or maybe a specific unit of time, need to confirm by following the system definition).

Application Scenarios:

Multi-point locking: Each lock point corresponds to a different lock time, which helps to achieve time synchronization or time management of different lock states.

UserParam (user-defined parameters)

Items	Content
Mnemonic	UserParam
CAN code	624
type	Array with index range of: 1 : 250
Access rights	Read / Write
Axis Related	Yes
Value type	int 32 bit
User units	No user units
Allowed in motion	Yes
Allowed with motor on	Yes
Flash memory is available	Yes
range	- 2,147,483,648 to 2,147,483,647
Default value	0

Function

- This is an array of parameters that users can define themselves, supporting up to 250 elements for storing custom information.
- User-specific parameters, preferences, or status information can be retained during motion and shutdown.

UserPWM (User PWM Control)

Items	Content
Mnemonics (keywords)	UserPWM
CAN code	626
type	Parameter
Access rights	Read / Write
Allowed in motion	Yes
Allows servo enable	Yes
Index range (array)	1:2 (supports two user PWM channels: UserPWM1 and UserPWM2)
Save to flash	Yes
Axis Related	Yes
Minimum value	0
Maximum value	4095
Default value	0
User units	No user units
Implementation status	Implemented (Valid from FW1.3.0 with AG300)

There are two parameters used to set the user PWM module, UserPWM and UserPWMDiv.

UserPWM controls the duty cycle of each user PWM output. This is the compare value such that when the internal counter < UserPWM[N] the output value is 1 else the output value is 0 . This allows us to generate a PWM signal where we can vary the duty cycle with resolution of 1/4096(12bit).

There are two UserPWM—UserPWM1 and UserPWM2. UserPWM1 and UserPWM2 have the same PWM frequency and are fully synchronized. Yet, the duty cycle of each one of them is set separately.

UserPWMDiv controls the clock frequency for the user PWM module. The va lue of UserPWMDiv is between 0 to 15, where 0 makes the user PWM clock count at 40 M Hz , 1 will make it 20 MHz, 2 will make it 10 MHz etc. It counts 4096 counts (we will call this internal counter) before going back to 0 .

The PWM clock Frequency = 80MHz / 2^ (UserPWMDiv + 1)

And the frequency of the PWM signal would be:

PWM Frequency = PWM clock frequency / 4096

After the setting of these two parameters, users can use the discrete outputs selector to output the PWM signal to any output. Please refer to *D OutSelect* manual page for full description.

Note: UserPWM is not supported for now at Central-I systems, but it can be easily added upon request.

Examples :

AUserPWM[1]=1000; Then it will generate a PWM with duty cycle of 1000/4096=24.4%.

AUserPWMDiv= 3 ; Then PWM clock Frequency=80/ 2 ^ (3 + 1) MHz =5 MHz and the resulted PWM frequency will be:

5MHz / 4096 = 1.22 KHz

With these settings, the selector output would send out 1.22 KHz PWM output with 24.4%duty cycle.

Refer Also:

UserPWM Div , *D OutSelect* .

UserPWMDiv (User PWM Frequency Division Parameter)

Items	Content
Mnemonics	UserPWMDiv
CAN code	627
type	Parameter
Access rights	Read / Write
Allowed in motion	Yes
Allows servo enable	Yes
Save to flash	Yes
Axis Related	Yes
Minimum value	0
Maximum value	15
Default value	9
User units	No user units
Implementation status	Implemented (supported in this version)

Function description:

Example:

AUserPWMDiv= 3 ; Then PWM clock Frequency=80/ 2^ (3 + 1) MHz =5 MHz and the resulted PWM frequency will be:

5MHz / 4096 = 1.22 KHz.

Refer Also:

UserPWM **,** ***DoutSelect*** **.**

Application Description:

- By adjusting **UserPWMDiv**, you can flexibly adjust the frequency of PWM.
- When combined with the UserPWM setting to define the duty cycle, this forms a PWM signal with different frequencies and duty cycles.
- The PWM signal can be routed to any digital output via DOutSelect (refer to the relevant manual).

Items	Content
Mnemonics (keywords)	MapEncoder
CAN code	322
type	Parameter
Array index range	1 : 3
Access rights	Read / Write
Axis Related	Yes
Value type	32-bit integer (int).
User units	No user units
Whether in-motion modifications are allowed	No
Whether to allow modification when the servo is turned on	No
Save to flash	Yes
Minimum value	1
Maximum value	24
Default value	1

Function description

Error Mapping Configuration:

Activate different mapping modes (0: Disabled, 1:1D, 2:2D, 3:3D) by setting the MapType parameter.
Before homing or when auto-phase, the MapType must be 0 to avoid misuse of error tables.
MapType will not be stored in flash memory and will return to zero after reboot until system calibration is complete.

Configuration parameters:

MapEncoder[1..3]: Defines error correction of encoder, supporting:
1: Main encoder
2: Auxiliary encoder
3 and above: encoders for other axes (e.g. B-axis main/auxiliary encoders)
MapTable[]: A table that stores error correction values.
MapStartIndex: Starting index of the error table.
MapStartPos[]: Starting position of the corresponding encoder input, used as the reference point for the error table.
MapLength[]: The number of error table points in each direction.
MapPosGap[]: Interval between consecutive error points in the input encoder, affecting interpolation accuracy.

Error mapping correction principle

1D Error Mapping:
Reads the specified encoder input (primary or auxiliary).
If the input value is less than the table start point (MapStartPos[1]), use the first error value.
If it exceeds the maximum range, use the last error value.
Within the range, perform linear interpolation to calculate the error.
Add the error value to the encoder reading for correction.

2D Error Mapping:
First apply error correction to the first encoder (row direction).
Then apply error correction to the second encoder (column direction).
Calculate the final error correction value using 2D linear interpolation.

3D error mapping:
Perform 2D correction for the first and second encoders.
Based on the third encoder position, interpolate between layers to obtain the 3D error correction value.
Apply the error to the position reading for multi-dimensional precision correction.

Notes:
During auto-phasing (commutation) and homing operations, set MapType to 0 to disable error mapping.
After a system restart or reset, MapType will automatically initialize to 0.
It is recommended to enable error mapping (MapType = 1, 2, or 3) only after the system has completed commutation and homing.

Example description:
MapEncoder[1] = 1: Use the first encoder (main encoder) as the error correction input.
MapEncoder[2] = 2: The second encoder, possibly the secondary encoder of the system.
MapEncoder[3] = 3: Probably the main encoder for the B-axis for 3D error mapping.

MapEncoder (Encoder Error Mapping)

Items	Content
Mnemonic	MapEncoder
CAN code	322
type	Parameter
Array index range	1 to 3
Access rights	Read / Write
Axis Related	Yes
Value type	32-bit integer (int)
User units	No user units
Allowed in motion	No
Allowed with Motor On	No
Save to flash	Yes
Minimum value	1
Maximum value	24
Default value	1

Remark

- Only supports modification when stationary (modification is not allowed during motion)
- This parameter supports configuration of 1D, 2D, and 3D error mappings.
- Typical use: Specify which encoder to use for error correction (e.g., main encoder, secondary encoder, etc.).

MapLength(Number of error-mapping points)

Items	Content
Mnemonic	MapLength
CAN code	324
type	Parameter
Array index range	1 to 3
Access rights	Read / Write
Axis Related	Yes
Value type	32-bit integer (int)
User units	No user units
Allowed in motion	No
Allowed with Motor On	No
Save to flash	Yes
Minimum value	1
Maximum value	3,000
Default value	10

Description

- Represents the number of sampled points in each error mapping direction.
- Affect the interpolation accuracy and smoothness of the error-compensation curve.
- Setting the appropriate number of points when configuring the error mapping parameters can help to obtain more precise correction results.

MapPosGap (Error Point Input Encoder Interval)

Items	Content
Mnemonic	MapPosGap
CAN code	325
type	Parameter
Array index range	1 to 3
Access rights	Read / Write
Axis Related	Yes
Value type	32-bit integer (int)
User units	No user units
Allowed in motion	No
Allowed with Motor On	No
Save to flash	Yes
Minimum value	1
Maximum value	8,000,000
Default value	1,000

Description

- Controls the input encoder value interval between consecutive error points.
- Affect the interpolation accuracy and the distance between data points.
- Setting reasonable interval values helps optimize the smoothness and accuracy of error correction.

MapStartIndex

Items	Content
Mnemonic	MapStartIndex
CAN code	321
type	Parameter
Array index range	1 to 5,000
Access rights	Read / Write
Axis Related	Yes
Value type	32-bit integer (int)
User units	No user units
Allowed in motion	No
Allowed with Motor On	No
Save to flash	Yes
Minimum value	1
Maximum value	5,000
Default value	1

Description

- Specifies the starting index of the error table for managing and locating error-correction data.
- Used with MapTable, define the starting position of error correction data in the table.
- Set appropriate indexes to ensure the correctness and consistency of error mapping.

MapStartPos (Error Mapping Start Position)

Items	Content
Mnemonic	MapStartPos
CAN code	323
type	Parameter
Array index range	1 to 3
Access rights	Read / Write
Axis Related	Yes
Value type	32-bit integer (int)
User units	No user units
Allowed in motion	No
Allowed with Motor On	No
Save to flash	Yes
Minimum value	-2,147,483,648
Maximum value	2,147,483,647
Default value	0

Description

- Define the encoder input position of the error mapping starting point as a reference for the error table.
- Used with MapTable for the starting point of error correction.
- Normal positive and negative range is allowed, and absolute or relative position correction is supported.

MapTable

Items	Content
Mnemonic	MapTable
CAN code	326
type	Parameter
Array index range	1 to 3000
Access rights	Read / Write
Axis Related	Yes
Value type	32-bit integer (int)
User units	No user units
Allowed in motion	No
Allowed with Motor On	No
Save to flash	Yes
Minimum value	-2,147,483,648
Maximum value	2,147,483,647
Default value	0

Description

- Stores the specific correction values of the error mapping, supports up to 3000 points, and supports a wide range of error compensation.
- Combined with parameters such as `MapStartIndex` and `MapStartPos`, the specific data position of the error mapping is defined.
- It supports positive and negative values to meet various error correction needs.

MapErrOffset

Items	Content
Mnemonic	MapErrOffset
CAN code	411
type	Parameter
Access rights	Read / Write
Axis Related	Yes
Value type	32-bit integer (int)
User units	No user units
Allowed in motion	No
Allowed with Motor On	No
Save to flash	No
Minimum value	-2,147,483,648
Maximum value	2,147,483,647
Default value	0

Description

- Offset compensation for error correction, adjusting the overall bias of the error map.
- Not stored to flash memory and is typically used for debugging or temporary calibration.
- Supports adjustment during operation (if needed), but it is recommended to proceed with caution.

MapErrOffRamp (Error Mapping Offset Ramp)

Items	Content
Mnemonic	MapErrOffRamp
CAN code	454
type	Parameter
Access rights	Read / Write
Axis Related	Yes
Value type	32-bit integer (int)
User units	No user units
Allowed in motion	Yes
Allowed with Motor On	Yes
Save to flash	Yes
Minimum value	1
Maximum value	2,147,483,647
Default value	16,384

Description

- Function: Used in error mapping calibration to provide a ramp parameter for dynamically adjusting the offset, controlling the rate at which the offset changes with the error.
- Supports adjustment during motion to facilitate on-site tuning.
- Setting appropriate values helps refine the real-time dynamic changes of error compensation.

MapErrOnStep

Items	Content
Mnemonic	MapErrOnStep
CAN code	476
type	Parameter
Access rights	Read / Write
Axis Related	Yes
Value type	32-bit integer (int)
User units	No user units
Allowed in motion	Yes
Allowed with Motor On	Yes
Save to flash	Yes
Minimum value	0
Maximum value	16,384
Default value	0

Description

- Define the error step threshold at which the error mapping starts to do error correction when this value is reached.
- Supports adjustment during motion to facilitate on-site tuning.
- It is suitable for controlling the starting point of error mapping when errors accumulate gradually.

GantryOn

Items	Content
Mnemonic	GantryOn
CAN code	650
type	Parameter
Access rights	Read / Write
Axis Related	Yes
Value type	32-bit integer (int)
User units	No user units
Allowed in motion	No
Allowed with Motor On	Yes
Save to flash	No
Minimum value	0
Maximum value	1
Default value	0

Description

- 0 indicates the "Disable" status and 1 indicates the "Enable" status.
- Typical application: Control the start/stop status of the gantry as a discrete (on/off) control signal.
- It can only be modified with the servo drive on.

GantryAccFFW (Gantry Acceleration Feedforward)

Items	Content
Mnemonic	GantryAccFFW
CAN code	655
type	Parameter
Access rights	Read / Write
Axis Related	Yes
Value type	32-bit integer (int)
User units	No user units
Allowed in motion	Yes
Allowed with Motor On	Yes
Save to flash	Yes
Minimum value	0
Maximum value	500,000
Default value	0

Description

- Control the acceleration feed-forward of the gantry and adjust this parameter to improve the acceleration control response.
- Support in-motion adjustment, suitable for dynamically adjusting acceleration feedback.

GantryPosGain (Gantry Position Gain)

Items	Content
Mnemonic	GantryPosGain
CAN code	654
type	Parameter
Access rights	Read / Write
Axis Related	Yes
Value type	32-bit integer (int)
User units	No user units
Allowed in motion	Yes
Allowed with Motor On	Yes
Save to flash	Yes
Minimum value	0
Maximum value	100,000
Default value	100

Description

- The gain value of position control, by adjusting this parameter can affect the sensitivity and stability of the gantry's position response.
- Supports in-motion adjustment, suitable for dynamic optimization control.

GantryVelGain(Gantry Velocity Gain)

Items	Content
Mnemonic	GantryVelGain
CAN code	656
type	Parameter
Access rights	Read / Write
Axis Related	Yes
Value type	32-bit integer (int)
User units	No user units
Allowed in motion	Yes
Allowed with Motor On	Yes
Save to flash	Yes
range	0 to 100,000
Default value	100

Description

- Adjust the gain parameters in the velocity control, affect the sensitivity of system response to the velocity input.
- Supports adjustment during motion, suitable for dynamic optimization.

GantryVelKi (Gantry Velocity Integral Gain)

Items	Content
Mnemonic	GantryVelKi
CAN code	657
type	Parameter
Access rights	Read / Write
Axis Related	Yes
Value type	32-bit integer (int)
User units	No user units
Allowed in motion	Yes
Allowed with Motor On	Yes
Save to flash	Yes
range	0 to 100,000
Default value	100

Description

- This is the integral gain parameter of velocity control to improve the response and stability of the system in velocity control.
- Supports adjustment during motion and helps dynamic optimization.

GantryFdbk(Gantry Feedback)

Items	Content
Mnemonic	GantryFdbk
CAN code	652
type	Read-only
Access rights	Read only
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	-2,147,483,648
Maximum value	2,147,483,647
Default value	0
User units	By user unit

Brief Description:

This parameter is read-only and reflects the feedback value of MIMO type gantry control. - Calculation Formula:

- AGantryFdbk = (APos + BPos) / 2
- BGantryFdbk = (APos – BPos)
- GantryOffset is included in the calculation, although the formula is not shown.
- Parameter names that are not "A" or "B" (starting with "?" at the beginning) have no practical use and always returns 0.

- This parameter is used to measure and monitor the feedback status of the gantry.
- The calculation results are continuously updated in any mode (on or off).

GantryOffset (Gantry Offset)

Items	Content
Mnemonic	GantryOffset
CAN code	653
type	Read-only
Access rights	Read only
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	-2,147,483,648
Maximum value	2,147,483,647
Default value	0
User units	By user unit

Summary:

- **`AGantryOffset`** is calculated only once when the user switches **`AGantryOn`** from 0 to 1.
- Calculation formula:

  ```
  AGantryOffset = APosRef – BPosRef
  ```

- This offset is used in subsequent **GantryFdbk** calculations to eliminate the initial bias of both encoders.
- Actual calculation formula:
- AGantryFdbk = (APos + BPos + AGantryOffset) / 2
- BGantryFdbk = (APos – BPos – AGantryOffset)
- ?GantryOffset (with "?" arguments at the beginning) are not used and always return 0

- This parameter represents the initial deviation between the two encoders and is used to optimize the accuracy of the feedback.
- **Read-only**, automatically updated during system operation, and cannot be manually modified.

GantryYawRef (Gantry Yaw Reference Value)

Items	Content
Mnemonic	GantryYawRef
CAN code	651
type	Read / Write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	-20,000
Maximum value	20,000
Default value	0
User units	By user unit
Implementation state	was realized

Description:

- This parameter is used to set the yaw reference value of the gantry and adjust the yaw angle.
- Supports adjustment and dynamic tuning during motion.

EventCntr (Event Counter)

Items	Content
Mnemonic	EventCntr
CAN code	186
type	Read / Write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	-2,147,483,648
Maximum value	2,147,483,647
Default value	0
User units	No user units

Description:

- **EventCntr** is used to accumulate the number of events that have occurred since the last EventOn.
- Switching **EventOn** will reset the counter.
- Users can also reset this counter manually.

EventSelect (Event Select)

Items	Content
Mnemonic	EventSelect
CAN code	317
type	Parameter
Access rights	Read / Write
Axis Related	Yes
Value type	32-bit integer (int)
User units	No user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	0 to 7
Default value	1

Description:

- This parameter is used to select a specific event, ranging from 0 to 7.
- It can be adjusted during motion and when the servo is on.
- By changing this value, different event conditions can be controlled or triggered.

EventOn

Items	Content
Mnemonic	EventOn
CAN code	178
type	Parameter
Access rights	Read / Write
Axis Related	Yes
Value type	32-bit integer (int)
User units	No user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
range	0 to 1
Default value	0

Function description:

- When EventOn = 1 is set, the first position is loaded into the position comparator (based on `EventType`, along with related parameters such as `EventBegPos` and `EventEndPos`, and trigger the start of event.
- **Recommendation**: Set EventOn = 1 when the motor position is less than the first requested event position to avoid abnormal behavior.
- **Mutually exclusive:** `Event` and `Lock` functions are mutually exclusive. When `EventOn` = 1, the system automatically assigns `LockEN` to `0`.

EventNextPos (Event Next Position)

Items	Content
Mnemonic	EventNextPos
CAN code	319
type	Read only
Axis Related	Yes
Value type	32-bit integer (int)
User units	In terms of user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
range	- 2,147,483,648 to 2,147,483,647
Default value	0

Description:

- This is a read-only parameter that displays the current “Next Event Position” value.
- Used in combination with other event parameters such as `EventEndPos` and `EventCntr` to control event triggering and position management.

EventTableBeg (Event Table Start Position)

Items	Content
Mnemonic	EventTableBeg
CAN code	184
type	Read / Write
Axis Related	Yes
Value type	32-bit integer (int)
User units	No user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	1 to 65,000
Default value	1

Description:

- This parameter defines the starting position of the event table.
- Supports adjustment in motion.
- The settings are stored in the device flash memory for persistent configuration.

EventTableEnd (The end position of event table)

Items	Content
Mnemonic	EventTableEnd
CAN code	185
type	Read / Write
Axis Related	Yes
Value type	32-bit integer (int)
User units	No user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	1 to 65,000
Default value	1

Description:

- Define the end position of event table.
- It can be set in motion and the configuration is stored to flash memory to ensure persistence.

EventTableSrc (Event Table Source)

Items	Content
Mnemonic	EventTableSrc
CAN code	313
type	Read / Write
Axis Related	Yes
Value type	32-bit integer (int)
User units	No user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	0 to 1
Default value	0

Description:

- Control the source type of the event table with a value of 0 or 1.
- It can be adjusted during motion and when the servo is on, and the configuration is stored in flash memory for long-term setups.

EventTableSel (Event Table Selection)

Items	Content
Mnemonic	EventTableSel
CAN code	318
type	Array with index range
Index range	1 to 65,000
Access rights	Read / Write
Axis Related	Yes
Value type	32-bit integer (int)
User units	No user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
range	0 to 7
Default value	1

Description

- This is an index array parameter that selects the corresponding event table.
- It can be adjusted during motion and when the servo is on, but the configuration is not stored to flash memory.

Items	Content
Mnemonic	EventTable
CAN code	316
type	Parameter
Index range	1: 65000
Access rights	Read / Write
Axis Related	Yes
Value type	32-bit integer ('int')
User units	In user units
Allowed in motion	Yes
Relationship with motor status	Allowed in motion
Save to flash	No
Minimum value	-2,147,483,648
Maximum value	2,147,483,647
Default value	0

Definition and Function:

An `EventTable` is an array used in the application to store a set of position points that trigger an event (such as a pulse output) when a motion or feedback position reaches a certain position.

Supported indexes ranging from 1 to 65,000, with large storage space for complex path planning and multipoint triggering.

Access Restrictions & Storage:

- It can read and write dynamically during motion and supports coexistence with the axis motion state.
- If it is not stored in the device's flash memory, the `EventTable` will need to be reconfigured after the system restarts.

EventPulseWid (Event Pulse Width)

Items	Content
Mnemonic	EventPulseWid
CAN code	179
type	Parameter
Access rights	Read / Write
Axis Related	Yes
Value type	32-bit integer ('int')
User units	No user units
Allowed in motion	Yes
Relationship with motor status	Allow (allow modification in motion, allow modification when servo is on)
Save to flash	Yes
range	-10,000,000 to 10,000,000
Default value	50

Parameter description:

Definition and Role:

`EventPulseWid` is used to set the width of the event signal pulse, which is the duration of the output pulse signal (in integers, no user units) when the event is triggered.

Usage scenario:

Adjust the pulse width to achieve different signal durations, suitable for synchronous control, signal strength adjustment, and other occasions.

EventType

Items	Content
Mnemonic	EventType
CAN code	180
type	Parameter
Access rights	Read / Write
Allowed in motion	Yes
Allowed with Motor On	Yes
Save to flash	Yes
Axis Related	Yes
range	0 to 2
Default value	0
User units	No user units

An event is a pulse generated on a specified output, generated when the actual feedback position is equal to the desired comparison position. EventType determines different comparison options:

- EventType = 0: A pulse is generated when the feedback position is equal to the position in the EventBegPos.
- EventType = 1: By interval event. The first pulse is generated when the position is equal to EventBegPos. Another pulse is generated each time it passes the distance defined by the EventGap. When the position exceeds the EventEndPos, the pulse stops generating.
- EventType = 2: Use table event. Input the position table where the event should be generated into the GenData[] array. ETStart is the index at which the event table starts. ETEnd is the index at the end of the event table. The positions in the table must be sorted from lowest to highest.

Related parameters:

- `EventOn`: Controls Event On or off.
- `EventGap`: Defines the distance between two events when EventType=1.
- `EventEndPos`: Defines where the event termination triggers.
- `EventTableBeg` and `EventTableEnd`: Define the start and end indexes of the table position.

EventBegPos (Event Start Position)

Items	Content
Mnemonic	EventBegPos
CAN code	181
type	Parameter
Access rights	Read / Write
Allowed in motion	Yes
Allowed with Motor On	Yes
Save to flash	Yes
Axis Related	Yes
range	- 2,147,483,648 to 2,147,483,647
Default value	0
User units	In terms of user units

Parameter description:

Definition and Use:

- `EventBegPos` specifies the feedback position generated by the first event, for event type 1 **(via interval) and type 2 (via position table).**
- In event type 1, the first event is generated when the feedback position reaches `EventBegPos`.
- In event type 2, the base position where the event begins is defined as the reference value for the first trigger point in the table.

Practical Application:

- Combine the `EventType` parameter to achieve precise event trigger control.

Suggestion:

- Set a reasonable starting position to avoid events that cannot be triggered or triggered too early.
- The position value can be negative or positive and is configured according to the actual motion target.

EventEndPos (Event End Position)

Items	Content
Mnemonic	EventEndPos
CAN code	183
type	Parameter
Access rights	Read / Write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
range	- 2,147,483,648 to 2,147,483,647
Default value	0
User units	In user units

Definition and Use:

- `EventEndPos` defines the maximum feedback position for an event. When using `EventType=2` (driven by the table), no more events are generated after this position.
- For example, when the event type is set to 2, `EventBegPos is set to 1000`, `EventGap is set to 2000`, and `EventEndPos is set to 8000`, the system will generate events when positions reach 1000, 3000, 5000, and 7000 (depending on `EventPulseWid` sets), after 8000 positions, no more events will be generated.

Suggested use:

- Before the motion, set `the EventOn` to a position smaller than `the EventBegPos` to ensure that the system does not misfunction before the event starts.
- After reaching `the EventEndPod`, it is recommended to turn off or reset EventOn to restart event generation.

Example Description:

- Hypothesis:
  - `EventType=2`
  - `EventBegPos=1000`
  - `EventGap=2000`
  - `EventEndPos=8000`
  - `EventOn=1` is set at a point where position is less than 1000
- The system will appear at positions: `1000`, `3000`, `5000`, `7000` until it exceeds `8000`, after which the event will not be generated again until it is manually turned back on.

Items	Content
Mnemonic	EventGap
CAN code	182
type	Parameter
Access rights	Read / Write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
range	- 2,147,483,648 to 2,147,483,647
Default value	0
User units	In user units

Parameter description:

Definition and Function:

- `EventGap` represents the distance (or position difference) value between two event generations.
- Valid when `EventType=1` (via interval) or `EventType=2` (via position table).
- Setting a small `EventGap` value can cause events to occur frequently, especially at high speeds, which can overlap with the pulse width set by the `EventPulseWid` and affect signal accuracy.

Suggested use:

- Choose the appropriate `EventGap` and avoid triggering or pulse overlays too frequently.
- When setting up high-speed motion or long pulse widths, consider increasing `the EventGap` to ensure a clear signal.

Storage & Conditioning:

- Supports dynamic adjustment in motion, and configurations are stored in flash memory.
- Adjustment of this parameter affects the frequency and response of the event trigger.

Example:

- If you set:
  - `EventType=1`
  - `EventBegPos=1000`

- `EventGap=2000`
- `EventEndPos=8000`
- `EventOn=1`

- During motion, the system will generate events at positions: `1000`, `3000`, `5000`, `7000`, and the pulse duration is determined by `the EventPulseWid` configuration.

EventTableCor(Event Table Correction)

Items	Content
Mnemonic	EventTableCor
CAN code	315
type	Read only
Index range	1 to 65000
Axis Related	Yes
Value type	32-bit integer (int)
User units	The user unit shall prevail
Allowed in motion	Yes
Save to flash	No
Minimum value	-2,147,483,648
Maximum value	2,147,483,647
Default value	0

Description

Function:
This parameter is used to provide offset correction regarding the position of the event table, helping to make fine adjustments to ensure the precision of the event trigger position.

Usage scenarios:
Commonly used for debugging or calibrating event trigger points in motion or dynamically adjusting trigger deviations.

Note:
Since it is a read-only parameter, it cannot be modified in motion but can only be read for monitoring or debugging.

OperationMode

Items	Content
Mnemonic	OperationMode
CAN code	78
type	Parameter
Access rights	Read / Write
Allowed in motion	No
Allowed with motor on	No
Save to flash	Yes
Axis Related	Yes
range	1 to 4
Default value	3
User units	No user units

Detailed explanation of the operation mode

Value	Mode of operation	Description
1	Current control only	Only the current is controlled, and the current reference value is set by analog input.
2	Speed control	Control speed and current, input speed reference values via analog input.
3	Position Control (Default)	This is the default motion mode. In this mode, all control loops are active. The position profiler generates position commands based on user-set motion requirements. Users can use different motion modes to input position, target, command, or speed command.
4	Reserve or set aside	(No details, often set to default 3)

Configuration Recommendation:
The default operating mode is Position Control (value 3), which is suitable for most motion applications.
If current control or speed control is required, the corresponding value (1 or 2) can be switched as needed.

ModeSwitchPos

Items	Content
Mnemonic	ModeSwitchPos
CAN code	438
type	Read only
Index range	1 to 2
Axis Related	Yes
Value type	32-bit integer (int)
User units	In user units
Allowed in motion	Yes
Save to flash	No
Minimum value	-2,147,483,648
Maximum value	2,147,483,648
Default value	0

The position where it was last switched to force (current) operating mode. A value of 0 may mean that no switching has occurred since the power on

PosPosFlag (Position vs. Flag)

Items	Content
Mnemonic	PosPosFlag
CAN code	328
type	Read / Write
Axis Related	be
Value type	32-bit integer (int)
User units	No user units
Allowed in motion	Yes
Allowed when servo on	Yes
Save to flash	Yes
range	0 to 2
Default value	0

Description

Force mode -> position mode

; Define automatic switching to position operation mode when a position threshold is passed

0-disable

1- When position < threshold

2- When position > threshold

PosPosTh

Items	Content
Mnemonic	PosPosTh
CAN code	329
type	Read / Write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
range	- 2,147,483,648 to 2,147,483,647
Default value	0
User units	In user units

When the PosPosFlag is not 0, **PosPosTh reaches the position where it automatically switches to position mode**

CurrGain

Items	Content
Mnemonic	CurrGain
CAN code	104
type	Read / Write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
range	0 to 200,000
Default value	0
User units	No user units

Agito controllers implement PI control filter for the current closed loop control.

Two control loops are implemented. One for the current of motor phase A (Ia) and one for the current of motor phase B (Ib).

CurrGain is the gain of this PI.

The relevant equations are:

CurrRef = … the output of the velocity control filter. Refer to the documentations of VelGain , VelKi .

IaRef = commutation(CurrRef)

IaErr = IaRef – Ia

IaIntegral = IaIntegral + IaErr * CurrKi * 0.001

IaIntegral is then saturated by the parameter MaxPWM

Temp = IaIntegral + IaErr

Va = Temp * CurrGain * 0.001

Va is then saturated by the parameter MaxPWM

(a similar control loop is executed for the current of motor's B phase – of course, with properly shifted commutation).

CurrRef , IaRef , Ia , IaErr are all in [mA]

Va is in internal units, with a product dependent value that represents 100% PWM duty cycle. Please consult Agito if the exact internal value for the product you are using is required.

Refer Also:

CurrKi, Ia, Ib, CurrRef, IaRef, IbRef, IaErr, IbErr, Va, Vb, Vc, MaxPWM and ControlMode。

Note to Agito Product1 users:

In this product, the current control algorithms are implemented with slightly different equations (the idea is the same, but there is different scaling and slightly different equations organization). Please address Agito in case you need the exact equations for this case.

CurrKi (Current Integration)

Items	Content
Mnemonic	CurrKi
CAN code	105
type	Read / Write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
range	0 to 200,000
Default	0
User units	No user units

Agito controller s implement PI control filter for the current closed loop control.

Two control loops are implemented. One for the current of motor phase A (Ia) and one for the current of motor phase B (Ib).

CurrKi is the gain of the integral term of this PI.

The relevant equations are:

CurrRef = … the output of the velocity control filter. Refer to the documentations of VelGain , VelKi .

IaRef = commutation(CurrRef)

IaErr = IaRef – Ia

IaIntegral = IaIntegral + IaErr * CurrKi * 0.001

IaIntegral is then saturated by the parameter MaxPWM

Temp = IaIntegral + IaErr

Va = Temp * CurrGain * 0.001

Va is then saturated by the parameter MaxPWM

(a similar control loop is executed for the current of motor's B phase – of course, with properly shifted commutation).

CurrRef , IaRef , Ia , IaErr are all in [mA]

Va is in internal units, with a product dependent value that represents 100% PWM duty cycle. Please address Agito if the exact internal value, for the product you are using, is required.

Refer Also:

CurrGain, Ia, Ib, CurrRef, IaRef, IbRef, IaErr, IbErr, Va, Vb, Vc, MaxPWM and ControlMode。

Note to Agito Product1 users:

In this product, the current control algorithms are implemented with slightly different equations (the idea is the same, but there are different scaling and slightly different equations organization). Please address Agito in case you need the exact equations for this case.

MotorCurr (total motor current)

Items	Content
Mnemonic	MotorCurr
CAN code	8
type	Read only
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
range	- 2,147,483,648 to 2,147,483,647
Default value	0
User units	No user units

Function:

- `MotorCurr` represents the total current of the motor, including the sum of all phases, in milliamps (mA).
- It is mainly used to monitor the working status of the motor and detect overload or abnormal current.

CurrPosErrTh (Position Error Threshold)

Items	Content
Mnemonic	CurrPosErrTh
CAN code	337
type	Read / Write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
range	-327,680 to 327,680
Default value	0
User units	In user units

Global thresholds that must be met in order to switch to current or force operation mode. If the position reference (PosRef) is greater or less than the threshold, you can switch (if any other threshold is triggered).

Requires reference to CurrPosThDir; 'int' 32 bit; Define the threshold direction (greater or less). A value of 0 bypasses this condition

-1 - when PosRef < threshold

0 - Bypass the condition

1 – when PosRef > threshold

CurrCurrTh (Current Threshold)

Items	Content
Mnemonic	CurrCurrTh
CAN code	339
type	Read / Write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
range	-64,000 to 64,000
Default value	0
User units	No user units

Open-loop force-controlled switching force: The threshold for switching to current operating mode if the current reference (CurrRef) is greater or less than the threshold.

CurrCurrThDir (Open Loop Force Current Threshold Direction)

Items	Content
Mnemonic	CurrCurrThDir
CAN code	343
type	Read / Write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
range	0 to 1
Default value	0
User units	No user units

Open-loop force control: CurrCurrThDir; 'int' 32 bit; Define the threshold direction (greater or less). A value of 0 bypasses this condition

0 - when CurrRef > threshold

1 - when CurrRef < threshold

CurrAInTh (Open-Loop Force Feedback Threshold)

Items	Content
Mnemonic	CurrAInTh
CAN code	338
type	Read / Write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
range	-100,000 to 100,000
Default value	0
User units	No user units

Open-loop force control: Achieve a force feedback value of 0 to automatically switch to force mode to avoid this type of switching. Positive or negative values define the threshold direction

CurrCmdSrc (Open-Loop Force-Controlled Current Command Source)

Items	Content
Mnemonic	CurrCmdSrc
CAN code	330
type	Read / Write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
range	0 to 2
Default value	0
User units	No user units

Open-loop force control current command source

Function:

- Controls the source of the current command, available values include:

`0`: Analog input (input defined by AInMode).

`1`: User-defined value or time

`2`: User defined value/time (interpolation, future supported, currently same with 1).

Application Description:

- By setting different values, the source of current commands can be dynamically switched, enabling multiple control strategies.
- In practice, it is usually set to 0 (analog input) or 1 (user-defined).

CurrCmdSlope (Open-Loop Force Controlled Current Command Slope)

Items	Content
Mnemonic	CurrCmdSlope
CAN code	568
type	Array parameters (index range 1..20)
Access rights	Read and write
Axis Related	Yes
Value type	int 32 bit
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	1 to 2,147,483,647
Default value	100
User units	No user units

Open-loop force control:

Current command source: 1-segmented current command value

Function:

- Define the maximum slope (i.e., change rate limit) of the current command change during each control cycle.
- By limiting the rate of current change, it helps reduce current surge, protect hardware, and improve motion smoothness.

Application Scenarios:

- Control the gradual change of current, so as to control the smoothness of acceleration and deceleration.
- Synchronized motion in multiple axes to avoid sudden changes in current cause unstable motion.

Characteristics:

- The array length of 20 means that different slopes can be defined for up to 20 control points or multiple segments, enhancing flexibility.

CurrCmdHTime (Open Loop Force Controlled Current Command Hold Time)

Items	Content
Mnemonic	CurrCmdHTime
CAN code	332
type	Array parameters (index range 1: 20）
Access rights	Read and write
Axis Related	Yes
Value type	int 32 bit
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	-122,070,306 to 122,070,306
Default value	0
User units	None (Not defined)

Open-loop force control:

Current command source: 1-segmented current command value

Function:

- Sets the duration (in unspecified units, usually milliseconds or control cycles) for a current command to hold (hold time).
- Controls the duration of the current, affecting the smoothness and responsiveness of the motion.

Application Scenarios:

- Adjust the duration of current commands to optimize control strategies.
- Combined with slope parameters, smooth and gradual current changes are achieved.

CurrCmdVal (Open-Loop Force Control Current Command Value)

Items	Content
Mnemonic	CurrCmdVal
CAN code	331
type	Array parameters (index range 1..20)
Access rights	Read and write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
range	Based on remote units (dynamic, dependent on actual hardware and configuration)
Default value	According to remote unit (dynamic)
User units	No user units
Implementation state	Implemented

Open-loop force control:

Current command source: 1-segmented current command value

Function:

- Set the current command value at the corresponding index (1 to 20) position to control the actual current applied.
- A very critical parameter for specific current command in motion control.

Application Scenarios:

- Dynamically set multiple current values in motion control for precise current regulation and motion optimization.
- It cooperates with the control state machine or motion mode switching to achieve complex motion logic.

SpeedChgOn

Items	Content
Mnemonic	SpeedChgOn
CAN code	345
type	Read / Write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
range	0 (disabled) to 1 (enabled)
Default value	0
User units	No user units

Force Control default slow-approach switch

Function:

- Force control whether to automatically switch to a new preset speed at a specific position.
- When set to 1 (enabled), the speed will automatically change when it reaches the set position.
- If it is off, automatic speed switching will not occur.

Reference parameters:

- `SpeedChgNew`: Define new speeds.
- `SpeedChgPos`: Defines the trigger position.
- `SpeedChgDir`: Define the switching direction.

SpeedChgPos (Force Control Slow Approach Position)

Items	Content
Mnemonic	SpeedChgPos
CAN code	346
type	Read / Write
Axis Related	Yes
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
User units	In user units
range	- 2,147,483,648 to 2,147,483,647
Default value	1,000
Implementation state	Implemented

Function:

Force control defaults to slow approach position

- **Function**: Sets the speed change to be triggered at a specific position to the speed specified by `SpeedChgNew`.
- **Usage**: When you specify a position on the motion path, the speed changes to the new preset speed.
- Use in combination with `SpeedChgNew`.

SpeedChgDir (Force Control Slow Approach Direction)

Items	Content
Mnemonic	SpeedChgDir
CAN code	347
type	Read / Write
Axis Related	Yes
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
User units	No user units
range	0 or 1
Default value	0
Implementation state	Implemented

Function:

Force control defaults to slow approach direction

- **Function**: Determines the direction in which the speed change is triggered.

0: When the current position is greater than the set position (SpeedChgPos), change the speed to `SpeedChgNew`.

1: When the current position is less than the set position, change to the new speed.

- Use in combination with `SpeedChgPos` and `SpeedChgNew` for automatic speed switching.

SpeedChgNew

Items	Content
Mnemonic	SpeedChgNew
CAN code	344
type	Read / Write
Axis Related	Yes
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
User units	In user units
range	-60,000,000 to 60,000,000
Default value	10,000
Implementation state	Implemented

Function

Force control defaults to the slow approach speed value

- **Function**: Set the new speed value in the variable speed operation.
- Combined with `SpeedChgPos`, `SpeedChgDir`, automatic or manual speed switching.

ForceRef

Items	Content
Mnemonic	ForceRef
CAN code	581
type	Read-only
Axis Related	Yes
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
User units	No user units
range	- 2,147,483,648 to 2,147,483,647
Default value	0
Implementation state	Implemented

Function:

- **Function**: Displays or reads the current mandatory reference value, usually used for debugging or monitoring.
- **Note**: This parameter is read-only and cannot be modified directly.

Force

Items	Content
Mnemonic	Force
CAN code	582
type	Read-only
Axis Related	Yes
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
User units	No user units
range	- 2,147,483,648 to 2,147,483,647
Default	0
Implementation state	Implemented

Function description:

- **Function**: Display or monitor the current force value, used for force control adjustment, cannot be directly modified.
- **Purpose**: Combine force control algorithm to achieve precise adjustment of load as internal state parameter.

ForceErr(Force Error)

Items	Content
Mnemonic	ForceErr
CAN code	583
type	Read-only
Axis Related	Yes
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
User units	No user units
range	- 2,147,483,648 to 2,147,483,647
Default value	0
Implementation state	Implemented

Function description:

- **Function**: Displays the current force error for monitoring and adjusting force control accuracy.
- **Note**: This is a monitoring parameter and cannot be modified manually.

ForceGain

Items	Content
Mnemonic	ForceGain
CAN code	577
type	Read / Write
Axis Related	Yes
Value type	32-bit integer (int)
User units	No user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	0 to 2,147,483,647
Default value	0

Function description:

- **Function**: Adjust the gain parameter in the force feedback control, the higher the value, the more sensitive the force control response.
- **Application**: Combine force sensor and force control algorithm to optimize the adjustment effect.

Items	Content
Mnemonic	ForceKi
CAN code	578
type	Read / Write
Axis Related	Yes
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	0 to 2,147,483,647
Default value	0

Function description:

- **Function**: Adjust the integration coefficient in force control, which is used to slow down the accumulation of errors and improve system stability.
- **Application**: Combined force error (ForceErr) for closed-loop control to improve the adjustment effect.

ForceKd

Items	Content
Mnemonic	ForceKd
CAN code	588
type	Read / Write
Axis Related	Yes
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	0 to 1,000,000
Default value	0

Function description:

- **Function**: Adjust the differential coefficient of force control, which is used to suppress oscillations and improve response stability.
- **Applications**: Differential terms help systems slow down mutations or oscillations and improve the stability of force control.

ForceFFW(Force Feedforward)

Items	Content
Mnemonic	ForceFFW
CAN code	589
type	32-bit integer (int)
Axis Related	Yes
User units	No user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	0 to 1,000,000
Default value	0

Function description:

- **Function**: Adjust the force feedforward compensation coefficient, which is used to compensate for the expected load or impedance in advance, and improve the control response speed and accuracy.
- **Applications**: Adjust this parameter in scenarios requiring quick response and precise force control.

ForceVelFFW(Force Velocity Feedforward)

Items	Content
Mnemonic	ForceVelFFW
CAN code	580
type	32-bit integer (int)
Axis Related	Yes
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	- 2,147,483,648 to 2,147,483,647
Default value	0

Function description:

- **Function**: Set the speed feedforward compensation to improve the response speed and stability of the force control in the state of variable speed.
- **Applications**: Suitable for dynamic scenarios that require quick response to load changes.

ForceRefFilt(Force Reference Filter)

Items	Content
Mnemonic	ForceRefFilt
CAN code	586
type	Read / Write
Axis Related	Yes
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	1 to 500,000
Default value	10,000

Function description:

First-order low-pass filter applied to force commands,

- **Function**: Filter out high-frequency noise, smooth force feedback reference signal, and ensure stable control.
- **Application**: Adjust filter intensity, balance response speed and noise suppression.

MaxForceErr (Closed-loop force control maximum force error)

Items	Content
Mnemonic	MaxForceErr
CAN code	585
type	Read / Write
Axis Related	Yes
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	0 to 32,768
Default value	2,000

Function description:

- **Function**: Set the maximum allowable force error value to prevent excessive deviation from causing abnormalities or protect the system.
- **Application**: Used to limit the force deviation range and improve system safety and stability.

MaxForceErrOL (Maximum Force Error Open Loop)

Items	Content
Mnemonic	MaxForceErrOL
CAN code	591
type	32-bit integer (int)
Axis Related	Yes
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	0 to 327,680
Default value	50,000

Function description:

- **Function**: Limit the maximum force error in the open loop state to ensure safety and system stability.
- **Applications**: Error thresholds are set especially in the case of open-loop control or no feedback.

ForcePosErrTh (Closed Loop Force Control Force Position Error Threshold)

Items	Content
Mnemonic	ForcePosErrTh
CAN code	576
type	Read / Write
Axis Related	Yes
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
range	-327,680 to 327,680
Default value	0
User units	In user units

Function description:

Reach the position error at which automatic switching to force mode occurs. When either this or ForceAInTh is satisfied, switch to force control.

0 - Avoid dragging in this kind of dragging.

Positive or negative values define the threshold direction.

ForceAInTh (Closed-loop Force Control Force Feedback Threshold)

Items	Content
Mnemonic	ForceAInTh
CAN code	584
type	Read / Write
Axis Related	Yes
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
range	-100,000 to 100,000
Default value	0
User units	No user units

Function description:

Closed-loop force control

Achieve the force error that automatically switches to force mode. When either this or ForcePosErrTh is satisfied, switch to force control.

0 - Avoid dragging in this kind of dragging.

Positive or negative values define the threshold direction.

ForceCmdSrc (Closed-loop Force Control Command Source)

Items	Content
Mnemonic	ForceCmdSrc
CAN code	570
type	Read / Write
Axis Related	Yes
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	0 to 2
Default value	0
User units	No user units

ForceCmdSrc (Force Command Source) Numerical Definition:

- 0：Analog Input
- 1：Scheduled Force Command
- 2：Interpolated Scheduled Force Command

ForceCmdSlope (Closed Loop Force Command Slope)

Items	Content
Mnemonic	ForceCmdSlope
CAN code	569
type	32-bit integer (int)
Array index range	1 to 20
Axis Related	Yes
Allowed in motion	Yes
Allowed with Motor On	Yes
Save to flash	Yes
range	1 to 2,147,483,647
Default value	100

Function description:

- **Function**: Limit the maximum rate (slope) of force command change to prevent system instability caused by sudden changes.
- **Application**: Adjustment to ensure smooth transition of force control.

ForceCmdHTime (Closed Loop Force Command Hold Time)

Items	Content
Mnemonic	ForceCmdHTime
CAN code	572
type	Array parameters (index range 1..20)
Access rights	Read and write
Axis Related	Yes
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	-122,070,306 to 122,070,306
Default value	0
User units	No user units
Implementation state	Implemented

Feature Description

Function:

- Set the hold time of the force command, i.e. the time (units not specified, usually control periods or milliseconds) after the command change occurs.
- Multiple segments can define different times for greater control flexibility.

ForceCmdHTime (Closed Loop Force Command Hold Time)

Items	Content
Mnemonic	ForceCmdHTime
CAN code	572
type	Array parameters (index range 1..20)
Access rights	Read and write
Axis Related	Yes
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	-122,070,306 to 122,070,306
Default value	0
User units	No user units
Implementation state	Implemented

Feature introduction

Function:

- Set hold time for the force command to ensure that the forced operation lasts for a certain amount of time to take effect.
- Arrays support multi-time definition for enhanced control flexibility.

ForceCmdVal (Closed-Loop Force Command Value)

Items	Content
Mnemonic	ForceCmdVal
CAN code	571
type	Array parameters (index range 1: 20）
Access rights	Read and write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
range	-16,000 to 16,000
Default value	0
User units	No user units
Implementation state	Implemented

Function:

- Set the force command value in the case of non-analog input (`ForceCmdSrc non-zero`).
- The array format supports setting multiple values (up to 20 segments) at different stages or under different conditions, making it easy to achieve phased control.

Description:

- This parameter works with the `ForceCmdSrc` control method to apply the actual force (such as current or torque, depending on the hardware implementation).

Notes:

- The force value ranges from -16000 to 16000, and the magnitude and direction of the force can be adjusted.

HomingOn

Items	Content
Mnemonic	HomingOn
CAN code	340
type	32-bit integer (int)
Access rights	Read/write
Axis Related	Yes
Allowed in motion	No
Allowed with motor on	Yes
Save to flash	No
range	0 (Did not start homing) or 1 (Start homing))
Default value	0

Brief description

Function:

- Controls whether to start the Homing process.
- When the user needs to perform a homing operation, set this parameter to 1, and the controller will define the corresponding homing policy based on HomingDef[].

Notes:

- Only when set to 1, the controller starts the homing process.
- After homing is completed or aborted, the system automatically clears this parameter.
- This parameter cannot be set in motion (it cannot be activated back to homing during motion).

Agito's controllers support a built-in homing process that can be "programmed" by the user to create any homing process that is optimally suitable for the application.

The Homing process is controlled by two parameters: the HomingOn parameter and the HomingDef[] array parameter.

The status of the Homing process is reported at the HomingStat parameter.

The HomingOn parameter is cleared to "0" upon power on or reset. Once it is set to "1" by the user, the controller will start the homing process according to the definitions at the HomingDef array parameter, while reporting its status, during the process, at the HomingStat parameter.

The HomingOn parameter is cleared by the controller upon completion of the homing process (whether successfully or due to some error, as described below).

The Homing process consists of steps. The number of steps and the action to perform at each step (move to limit, move to index, set position…) are defined by the values at the HomingDef[] array parameter, as described in detail below.

HomingStep keyword can be used to check the currently active step of the homing process. It is 0 after power on before any attempt to perform homing. Upon homing completion (successful or not), it will be equal to the last executed step (can be used to analyze the reason for failure to complete the homing process).

Upon successful completion of the last step, the homing is completed and the HomingOn parameter is cleared by the controller.

At each step, some errors can occur (depending on the step type/action). In case of an error, the homing process will be aborted, HomingOn will be cleared, and HomingStat will report a suitable value, as listed below.

The maximal number of homing process steps is the size of the HomingDef[] array, divided by 10. Typically, at Agito's controllers, the size of HomingDef[] is 150, which means that the user can create a homing process that includes up to 15 steps.

The HomingStat may report one of the following statuses:

HomingStat value	Meaning
0	No homing was done after power on or reset
Positive (not 100)	Homing is in process. HomingStat value reflects the number of the currently processed step in the homing process. Refer also to HomingStep above.
-1	The homing process failed (and aborted) due to wrong parameter at HomingDef[] array parameter (the parameters related to each step are checked when each step starts during the homing process).
-2	The homing process failed (and aborted) due to timeout during one of the homing steps. Each step (if relevant) is defined with a timeout. In case this timeout passes and the step has not yet completed, it is an error.
-3	The homing process failed (and aborted) due to unexpected motor off. This means that during one of the homing steps, the axis was disabled due to some fault (reflected at the value of ConFlt) and the step could not be completed.
-4	The homing process failed (and aborted) due to wrong motion reason. This means that a given step at the homing process, which expects a given reason for end of motion (RLS, index, reached target…) encounters a different reason for ending the motion.
-5	The homing process failed (and aborted) due to wrong step type. This means that the homing process reached a step whose type (as defined in the HomingDef[] array parameter) is not recognized.
-6	The homing process failed (and aborted) due to in-motion. This means that the homing process detected that the axis is in-motion, while it is starting a new step.
-7	The homing process failed (and aborted) due to too many steps. This error will happen if the homing process reaches the last step defined in the HomingDef[] array parameter, and it will not be an end of homing step.
-8	The homing process failed (and aborted) due to unexpected limit. This error is relevant only to one of the homing steps types: Check that out of limits' . See more details below.
-9	The homing process failed (and aborted) due to inability to perform SetPosition (Error Mapping activated? Auto Gain activated? Value out of software position limits?). This error is relevant only to one of the homing steps types: 'Set position'. See more details below.
100	The homing process has been successfully completed.

As explained above, the HomingDef[] array parameter is used to define the homing process, by definitions of the homing steps and the type and parameters for each step.

HomingDef[1] defines the type of the first step.

HomingDef[2-10] defines the parameters for this step.

The number of actually used parameters, and the meaning of each parameter, depend on the step type.

Similarly, HomingDef[11-20] defines that 2 nd step. HomingDef[21-30] defines the 2 nd step.

It is extremely convenience to define the homing process parameters at a text file (*.par, for parameters file) that can be downloaded (using the PC Suite) to controller to define the homing process. Here is an example for such file (definition of homing process for a given application), which will be followed by a detailed description of the supported step types and the parameters for each type:

```
//
// 1. Go fast into reverse limit
//
AHomingDef[1]=1
AHomingDef[2]=-10000
AHomingDef[3]=10000000
AHomingDef[4]=10000000
AHomingDef[5]=163840
//
// 2. Wait 100ms
//
AHomingDef[11]=7
AHomingDef[12]=1638
//
// 3. Jog forward slow to index
//
AHomingDef[21]=4
AHomingDef[22]=1000
AHomingDef[23]=1000000
AHomingDef[24]=1000000
AHomingDef[25]=163840
//
// 4. Wait 100ms
//
AHomingDef[31]=7
AHomingDef[32]=1638
//
// 5. Set position to 0
//
AHomingDef[41]=6
AHomingDef[42]=0
//
// 6. End of homing
//
AHomingDef[51]=0
```

Users can also use the Homing tool (window) of the PC Suite which provides easy configuration of the Homing process and parameters, as well as built-in homing scenarios.

Now it is needed to define the details of the HomingDef[] content, which defines the homing process:

The description below will refer to HomingDe f [1-10]. However, the same is relevant for HomingDef[11-20] (2 nd step) and so on.

HomingDef[1] defines the type of the step. The following table shows the available types:

HomingDef[1] (or [11],[21] ...) value	Step Type
0	End homing. This must be the last step.
1	Jog (jogging) into limit.
2	Check that out of limits
3	Move relative PTP (point to point)
4	Jog to index
5	Move to index position
6	Set position (***)
7	Wait N samples
8	Enable (or disable) the motor
9	Move to hard stop (detected by motor stuck) (***)
10	Move to hard stop (detected by high position error) (***)
11	Jog to a change in the Home discrete input
12	Move absolute PTP (point to point)
13	Set position software limits (RevPLim and FwdPLim parameters)
14	Configure Position Lock
15	Jog to Lock (index by HW)
16	Move to Lock position

(***) These homing step types have some limitations regarding position value setting. Please read the description below carefully.

The following table presents the description for each type, and its parameters:

Step type	Steps described	Step parameters of HomingDef [2-10] (or [12-20], [22-30] ...）
End homing. This must be the last step.	The homing process will stop. Reaching this step means that the homing is completed successfully.	No parameters
Jog (jogging) into limit.	Move in jogging mode, in the speed and direction as defined at the step parameters. Successfully completed when the motion stops sue to the relevant limit switch.	HomingDef[2]: Jogging speed. The sign is the direction and also defines which limit switch to look for. HomingDef[3]: Acceleration/Deceleration. HomingDef[4]: Emergency deceleration. HomingDef[5]: Maximum step time, in [samples]
Check that out of limits	Check if both RLS and FLS are not activated. If one of them is activated, terminate the homing process with a suitable error.	No parameters
Move relative PTP (point to point)	Move with the provided motion parameters to a given relative distance.	HomingDef[2]: Maximum speed. HomingDef[3]: Acceleration/Deceleration. HomingDef[4]: Relative distance (can be positive or negative). HomingDef[5]: Maximum step time, in [samples]

Step type	Steps described	Step parameters of HomingDef [2-10] (or [12-20], [22-30] ...）
Jog to index	Jog with the provided motion parameters till an index is detected. The jogging speed shall be low enough to ensure that the index is detected. Theoretically, low enough should be smaller than 16384 counts/sec. Recommended values are smaller than 8000 counts/sec.	HomingDef[2]: Jogging speed. The sign is the direction of motion. HomingDef[3]: Acceleration/Deceleration. HomingDef[4]: Emergency deceleration. HomingDef[5]: Maximum step time, in [samples]
Move to index position	Move (using the provided motion parameters) to the last recorded index position.	HomingDef[2]: Jogging speed. The sign is the direction of motion. HomingDef[3]: Acceleration/Deceleration. HomingDef[4]: Emergency deceleration. HomingDef[5]: Maximum step time, in [samples]
Set position	Set the current position to the provided value. Note that this step will do nothing in case the conditions for SetPosition are not met. Refer to the SetPosition keyword manual page.	HomingDef[2]: Desired position value to set at the current position.
Wait N samples	Wait N (as provided by the step parameters) samples.	HomingDef[2]: Number of cycle to wait before advancing to the next step.

Step type	Steps described	Step parameters of HomingDef [2-10] (or [12-20], [22-30] ...) 〉
Enable (or disable) the motor	Enables or disables the motor	HomingDef[2]: 0 to disable the motor. 1 to enable the motor. HomingDef[3]: Maximum step time, in [samples]
Move to hard stop (detected by motor stuck)	Move (using the provided motion parameters) till hard stop is detected using the provided parameters to detect motor stuck. Motor stuck is detected when the motor velocity (absolute value) is lower than the given velocity threshold and the motor current (absolute value) is higher than the given motor current threshold, both for consecutive period of the given stuck time. Once motor stuck is detected, the motor position is set to the given "position to set" value. The sign of the Maximum speed parameter defines the direction of the motion. Note that this step will not set the desired position in case the conditions for SetPosition are not met. Refer to the SetPosition keyword manual page.	HomingDef[2]: Maximum speed. HomingDef[3]: Acceleration/Deceleration. HomingDef[4]: Emergency deceleration. HomingDef[5]: Velocity threshold to detect motor stuck. HomingDef[6]: Motor current (in mA) threshold to detect motor stuck. HomingDef[7]: Stuck time (in samples). HomingDef[8]: Position to set when the hard stop is detected. HomingDef[9]: Maximum step time, in [samples]

Step type	Steps described	Step parameters of HomingDef [2-10] (or [12-20], [22-30] ...) 〉
Move to hard stop (detected by high position error)	Move (using the provided motion parameters) till hard stop is detected using the provided parameters to detect high position error. Motor stuck is detected when the position error (absolute value) is higher than the provided maximal position error threshold. Once motor stuck is detected, the motor position is set to the given "position to set" value. The sign of the Maximum speed parameter defines the direction of the motion. Note that this step will not set the desired position in case the conditions for SetPosition are not met. Refer to the SetPosition keyword manual page.	HomingDef[2]: Maximum speed. HomingDef[3]: Acceleration/Deceleration. HomingDef[4]: Emergency deceleration. HomingDef[5]: Maximal position error threshold to detect motor stuck. HomingDef[6]: Position to set when the hard stop is detected. HomingDef[7]: Maximum step time, in [samples]

Step type	Steps described	Step parameters of HomingDef [2-10] (or [12-20], [22-30] ...) 〉
Jog to a change in the Home discrete input	Move (using the provided motion parameters) till a change in the Home discrete input is detected. Once detected, stop the motion. The sign of the given Maximum speed parameter and the state of the Home discrete input define the direction of the motion. If the Home discrete input is "0" the sign of the Maximum speed parameter is the direction of the motion. If the Home is "1", the direction is inverted.	HomingDef[2]: Maximum speed. HomingDef[3]: Acceleration/Deceleration. HomingDef[4]: Emergency deceleration. HomingDef[5]: Maximum step time, in [samples]
Move absolute PTP (point to point)	Move with the provided motion parameters to a given absolute target position.	HomingDef[2]: Maximum speed. HomingDef[3]: Acceleration/Deceleration. HomingDef[4]: Absolute target position. HomingDef[5]: Maximum step time, in [samples]

Step type	Steps described	Step parameters of HomingDef [2-10] (or [12-20], [22-30] ...）
Set software position limits	Provides the means to optionally set value to the reverse software position limit (RevPLim parameter) and similarly also for the forward limit.	HomingDef[2]: Set value to RevPLim? 1 to set, 0 to avoid setting. HomingDef[3]: New value for RevPLim (ignored if HomingDef[2] is 0). HomingDef[4]: Set value to FwdPLim? 1 to set, 0 to avoid setting. HomingDef[5]: New value for FwdPLim (ignored if HomingDef[4] is 0).
Configure Position Lock	Configures the Position Lock feature	HomingDef[2]: 0 to disable the Lock . 1 to enable the Lock . Similar to LockEn. HomingDef[3]: Defines the Lock source and polarity. Similar to LockSrc. HomingDef[4]: Maximum step time, in [samples]
Jog to Lock (Index by HW)	Jog with the provided motion parameters till Lock happens and Lock position is latched. Then decelerate and stop.	HomingDef[2]: Jogging speed. The sign is the direction of motion. HomingDef[3]: Acceleration/Deceleration. HomingDef[4]: Emergency deceleration. HomingDef[5]: Maximum step time, in [samples]

Step type	Steps described	Step parameters of HomingDef [2-10] (or [12-20], [22-30] ...）
Move to Lock position	Move (using the provided motion parameters) to the last recorded Lock position (LockVal).	HomingDef[2]: Maximum speed. HomingDef[3]: Acceleration/Deceleration. HomingDef[4]: Emergency deceleration. HomingDef[5]: Maximum step time, in [samples]

Most of the step types include built-in error detection mechanism. For example, timeout, or reaching an unexpected limit switch and so on. When such an error is detected during the homing process, the process is aborted and HomingStat is properly set to reflect the error type (see details above).

Note that upon entering the homing process, the controller motion parameters (Speed, Acceleration, deceleration and Emergency deceleration) are temporarily saved, and are restored when the homing process is completed. This is required since the homing process may change the values of these parameters.

Refer Also:

HomingDef, HomingStat, MotionReason , SetPosition 和 ConFlt。

HomingDef（Homing Definition）

Items	Content
Mnemonic	HomingDef
CAN code	341
type	Array parameters (index range 1: 150）
Access rights	Read/write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
range	- 2,147,483,648 to 2,147,483,647
Default value	0
User units	No user units
Implementation state	Implemented

Function:

- Define and store several different homing policies.
- Each element represents a specific homing configuration, including step sequence, parameters, and more.
- Detailed configuration can involve steps such as limit detection, position setting, waiting, enabling/disabling the motor, etc.

Refer also:

- For more details, check the `HomingOn` keyword page, which describes the overall process and configuration of homing.

RetractSpeed

Items	Content
Mnemonic	RetractSpeed
CAN code	608
type	32-bit integer (int)
Access rights	Read/write
Axis Related	Yes
User units	The user unit shall prevail
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	-1,300,000,000 to 1,300,000,000
Default value	1,000

Function:

- Control the speed parameters used for the "return operation".
- It is often used for speed settings in retraction or reverse operation, especially as the default speed in force control.

RetractTarget

Items	Content
Mnemonic	RetractTarget
CAN code	609
type	32-bit integer (int)
Access rights	Read/write
Axis Related	Yes
User units	The user unit shall prevail
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	- 2,147,483,648 to 2,147,483,647
Default value	0

Function:

- Set in force control mode to return to the target position of the operation.
- This position represents the target point when the device is retracted or calibrated.

CurrDir (current direction)

Items	Content
Mnemonic	CurrDir
CAN code	76
type	32-bit integer (int)
Access rights	Read/write
Axis Related	Yes
User units	No user units
Allowed in motion	No
Allowed with motor on	No
Save to flash	Yes
range	0 (forward) or 1 (reverse)
Default value	0

Function:

- Controls the direction of the current (or poles).
- It can only be modified during the setup or debugging stage, cannot be changed in motion.
- It is often used for calibration or debugging to define the forward and reverse directions of motion.

Application Scenarios:

- Adjust the direction of the motor poles to ensure the correct direction of motion.

RevPLim (Reverse Position Restrictions)

Items	Content
Mnemonic	RevPLim
CAN code	82
type	32-bit integer (int)
Access rights	Read/write
Axis Related	Yes
User units	In terms of user units
Allowed in motion	No
Allowed with motor on	Yes
Save to flash	Yes
range	- 2,147,483,648 to 2,147,483,647
Default value	-2,000,000,000

Function:

- Define the reverse position limit (minimum position) of the motor.
- If the current position exceeds this limit, the motor will stop moving and cannot start a new motion in the reverse direction.
- Forward motion is allowed only if the current position returns to the limit.

Related parameters:

- **FWDPlim**: Forward position limit, which defines the range of motion when used together.

FwdPLim (Forward Position Limit)

Items	Content
Mnemonic	FwdPLim
CAN code	83
type	32-bit integer (int)
Access rights	Read/write
Axis Related	Yes
User units	In terms of user units
Allowed in motion	No
Allowed with motor on	Yes
Save to flash	Yes
range	- 2,147,483,648 to 2,147,483,647
Default value	2,000,000,000

Function:

- Define the forward maximum position limit for the motor.
- When the current position exceeds this limit, the motor stops and cannot start a new forward motion.
- Reverse motion is allowed only if the current position returns to the limit.

Related parameters:

- Combined with RevPLim, the range of motion is defined to ensure safe motion.

MaxPosErr (Maximum Position Error)

Items	**Content**
Mnemonic	MaxPosErr
CAN code	84
type	32-bit integer (int)
Access rights	Read/write
Axis Related	Yes
User units	In user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	0 to 80,000,000
Default value	20

Function:

- Define the maximum position deviation (error) allowed.
- Position error is the difference between the target position (PosRef) and the actual position (Pos).
- Beyond this error, the motor is disabled as a fail-safe measure.

MaxVelErr (Maximum Speed Error)

Items	Content
Mnemonic	MaxVelErr
CAN code	85
type	32-bit integer (int)
Access rights	Read/write
Axis Related	Yes
User units	In user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	0 to 1,300,000,000
Default value	32,768

Function:

- Control the maximum speed error allowed.
- If the actual speed error (the difference between the target speed and the actual speed) exceeds this value, the motor will be disabled as safety protection.

HomingStep(Homing Step)

Items	**Content**
Mnemonic	HomingStep
CAN code	385
type	32-bit integer (int)
Access rights	Read Only
Axis Related	Yes
User units	No user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
range	- 2,147,483,648 to 2,147,483,647
Default value	0

Function:

- Indicates the specific step number or the current step in the homing operation.
- As a read-only parameter, it is used to monitor and diagnose the status of the homing process.

Application Scenarios:

- In the automatic homing process, it is used to track the status of the homing step.
- Direct modifications are not allowed and are controlled by the system.

ComtMode

Items	Content
Mnemonic	ComtMode
CAN code	72
type	Parameters (array, indexes 1 to 24)
Access rights	Read and write
Axis Related	Yes
User units	No user units
Allowed in motion	No
Allowed with motor on	No
Save to flash	Yes
range	- 2,147,483,648 to 2,147,483,647

Detailed explanation of the index

index	Description	Example/description of taking values
1	Autophasing Method	• 0: Jump to zero phase • 1: Reserved • 2: Absolute encoder with predefined offset • 3: Special encoder with Halls Encoder switching • 4: Encoder with Halls/Encoder Switching • 5: Minimal jump search • 6: Halls-only commutation
2	Defines the voltage of each auto phasing step at jump-to-zero method	Voltage parameters for auto-phase
3	Defines the duration of each auto phasing step at jump-to-zero method	Duration parameters for auto-phase
4	Encoder readings at zero electric angle in absolute encoder autophase	Reference value
5	Trigger Values for Starting Auto Phasing (1282)	When this value is set, the autophasing start is triggered
6	Process configuration of jump to zero auto-phase	0: Use jump; 1: Use the smoothing process (recommended)
7-16	Reserved	Expand the reserved space in the future
18	Accuracy requirements for the end of autophasing	Set the tolerance range for error

index	Description	Example/description of taking values
19	Trigger event for Autophasing	• 0: Automatic upon power on • 1: Only upon manual request • 2: Automatic upon MotorOn (if needed) • 3: Automatic upon power on and MotorOn
20-23	Configuration parameters for the minimum jump method	Program adjustment for optimizing autophasing

Uses and application scenarios

- Set detailed parameters for automatic homing and phase adjustment.
- Control automatic homing behavior, precision, trigger conditions, etc.
- Implement different automatic calibration strategies through multi-index parameters.

Commutation is the process of alternating the current between the different motor phases to generate motion (for DC Brushless motors only) . To commutate the current correctly the controller must have information about the electrical angle of the motor. This information can be received from Hall sensors, for example.

In the absence of Hall sensors, the electrical angle of the motor must be derived from the encoder readings, which are generally not aligned with the motor electrical angle. As a result, an initialization process is required in order to align the encoder reading and the motor's electrical phase.

This can be done in one of several methods.

ComtMode [ ] is an array that controls the electrical angle detection process.

ComtMode[1] selects which method will be used to detect the electrical angle:

- 0 – Jump to zero. The motor will jump from its present location to the electrical zero angle of one of the phases. This method usually involves a large, sudden movement.
- 1 – Minimal motion. In this method the motor will only perform a very small movement and detect the electrical angle in its present location.
- 2 – Absolute encoder. For absolute encoder, perform angle detection once per each motor + controller set using one of the methods above. After the angle is detected use the Save command to save all the parameters. The absolute location of angle 0 will be saved in the controller's flash memory. In all the following uses of the same motor – controller set, use ComtMode[1]=2 to calculate the current electrical angle using the number saved in memory without any motion.

When using the "Jump to zero" method the user can determine the amount of voltage that will be applied to the motor during the process. If the voltage (and therefore the resulted current) is too low, mechanical factors in the system (such as friction) may prevent the motor from moving and cause

failure of the process. High currents that are applied to the same phase for a long time can cause damage. The user should apply a current that is enough to move but not too much.

The voltage (and thus the current) to the motor is increased gradually over a period of time so the maximum current is only applied for a short time. Every voltage (current) is applied for a step duration of 10mSec.

ComtMode[2] : Used only in “jump to zero” mode. Determines by how much the voltage to the motor is increased after each step. The resulting current depends on the bus voltage and the motor resistance. To prevent damage, it is recommended to set this number low during the first experiments and raise it only if necessary. If needed change ComtMode[3].

ComtMode[3] determines how many voltage steps will be applied during the angle detection process. Each voltage value is applied to the motor for 10 mSec and then the voltage is increased. It is recommended to start with a 1 second detection process (ComtMode[3] = 100). After the detection is done, with motor disabled, move the motor manually to a different position and repeat the process. The motor is expected to move. If you don’t see any motion, increase ComtMode[3] and move the motor again until you see a detection process that causes movement. As long as you don’t see motion it is possible that the current is not enough to overcome friction, or the motor was in the 0 position in the first place.

About the “Minimal motion” method:

With incremental encoder (or first using a motor with absolute encoder), there is no information about the electrical angle of the motor after power on. The “Jump to zero” method overcomes this difficulty by forcing the motor to jump into a known electrical angle (0 degrees). But this involves a relatively large motor’s jump. The “Minimal motion” method performs a different process, to minimize this motor movement.

The idea in general is to temporarily, during the process (about 2-3 seconds), to close a control loop that tries to maintain the motor at its current position, while commanding the current control loop with a fixed current command (set be the user) and an electrical angle that is the output of the control loop. As a result, the control loop (trying to keep the motor at its position when the process was started) will find the suitable electrical angle that will keep the motor in this location although current is applied to the motor.

This is, by definition, the electrical angle of the motor position, minus 90 degrees.

So, the control loop finds the electrical angle of the motor, at its current position.

Since this angle is not known initially (a zero angle is assumed), the motor will slightly move, till the control loop will be stabilized into steady state, where the motor will be back in its original position.

In order to optimally overcome frictions and to ensure the accuracy of the process, after this phase (in closed loop) is completed, a second phase is performed. In this phase, a user defined current (higher than the one used in the closed loop) is applied in open loop to the motor, to force it into the position that is related to the detected angle.

Knowing the detected angle, and its related position, the drive can initiate the commutation variables (where is the “zero” of the electrical cycle) and is ready for motions.

When using the “ Minimal motion ” method the user shall determine the amount of current that will be used during the process, as follows:

ComtMode[6]: Is the current to use, in [mA], during the process of looking for the electrical angle in closed loop, as explained above . Typically, it is around 10% to 20% of the motor’s peak current. It must be above the current that is required to overcome the system friction.

ComtMode[7]: Is the current to use, in [mA] , during the last phase of the minimal motion commutation process . In this phase, the current is applied in open loop, so that the motor is forced to the position that is related to the detected electrical angle. Typically, it is around 15% to 3 0% of the motor’s peak current. It must be above the current that is required to overcome the syst em friction and for optimal performance, it shall be higher than ComtMode[6].

Note that while ComtMode[6] and [7] are used to set the current reference (CurrRef) during the process, the actual value of CurrRef is subjected to the drive current limitations, as defined by the user.

As the “Minimal motion” method involves a closed loop control to look for the electrical angle at the current position, it involves gain and integral terms parameters for the closed loop, as explained above. These parameters shall be set by the user as follows:

ComtMode[8]: Is the gain of the PI control filter. Typical value is 5000. Changes may be required depending on the system, the encoder resolution and the motor’s number of poles.

ComtMode[9]: Is the integral gain of the PI control filter. Typical value is 20 . Changes may be required depending on the system, the encoder resolution and the motor’s number of poles.

ComtMode[4] holds the position of the electrical zero of a motor with absolute encoder. If the motor or controller need to be replaced, the electrical angle of the motor must be detected again and saved. The value of ComtMode[4] is automatically assigned following a commutation process with ComtMode[1]=0 or 1. This means that for a motor with absolute encoder, you shall first perform the detection process using ComtMode[1] = 0 (or 1) and then, after the process is successfully completed and ComtMode[4] is automatically updated by the controller, you can change to ComtMode[1]=2 (“Absolute encoder” mode) and save to the Flash, so that there will be no need for any moti on in the next times.

ComtMode[5] is used to request a new commutation process. To repeat the commutation process, enter ComtMode[5] = 1282. A new commutation process will begin and the value of ComtMode[5] will be cleared.

Commutation process is automatically performed following power on or reset.

Specific products notes:

1. Product 1:

a. Currently, only the “Jump to zero” method is supported.
b. Repeating the commutation process is done by ComtMode[1]=1282 (and not ComtMode[5]).

ComtAng

Items	Content
Mnemonic	ComtAng
CAN code	73
type	Read-only parameters (32-bit integers)
Axis Related	Yes
User units	No user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
range	0 to 35,999
Default value	0

ComtAng is the commutation angle in encoder counts, or actually the distance in encoder counts between the current position and the actual electrical zero position of the motor. The commutation angle is the angle of the motor within the current electric cycle.

Example:

Assume a 2 pole pairs motor with 4000 cnts encoder. Each electric cycle is 2000 counts. If the motor position is currently 500 counts from the location detected as the electrical zero then ComtAng = 500, and this means that the motor is 90 degrees from the electrical zero.

Note:

Electrical zero position is defined as the motor location where the sinusoidal BEMF (or Torque/Force) "constant" of motor's phase A is 0.

RelTrgt (relative target)

Items	Content
Mnemonic	RelTrgt
CAN code	135
type	32-bit integer (int)
Access rights	Read and write
Axis Related	Yes
User units	In terms of user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
range	- 2,147,483,648 to 2,147,483,647
Default value	0

Function:

- Define the relative motion distance in PTP (point-to-point) and repetitive motion.
- As the target position of the motion, the deviation is relative to the current position.

Detailed description:

- **PTP Motion**: Represents the offset from the current position to the target position.
- **Repeat Motion**: Define the stop position, relative to the current or initial position.
- If `RelTrgt ≠ 0`, AbsTrgt is ignored and RelTrgt is used directly as the motion target.
- RelTrgt can be changed dynamically during motion, and the target adjusts instantly.

Example:

1. Current Pos=1000, AbsTrgt=5000, RelTrgt=0: The end point of the next PTP is 5000.
2. Current Pos=1000, AbsTrgt=5000, RelTrgt=7000: The next PTP end point is Pos = 1000 + 7000 = 8000.
3. Current Pos=1000, AbsTrgt=5000, RelTrgt=7000: Continuous repetition will go to Pos=8000, then back to Pos=1000, and continue to loop.

AbsTrgt (Absolute Goal)

Items	Content
Mnemonic	AbsTrgt
CAN code	134
type	32-bit integer (int)
Access rights	Read and write
Axis Related	Yes
User units	In terms of user units
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
range	- 2,147,483,648 to 2,147,483,647
Default value	0

Function:

1. Define the absolute target position for PTP (point-to-point) and repetitive motion.
2. The target position reached at the end of the motion, regardless of the current position.

Detailed description:

1. **PTP Motion**: This is the end position.
2. **Repetitive motion**: Defines the position where the electromechanical stops and performs reciprocating motion between this position and the starting position during continuous motion.
3. If `RelTrgt ≠ 0`, AbsTrgt is ignored and `RelTrgt` is used as the actual target.
4. AbsTrgt can be changed dynamically during motion, and the target is updated immediately.

Example:

1. Current Pos=1000, AbsTrgt=5000, RelTrgt=0: The next PTP ends at Pos=5000.
2. Current Pos=1000, AbsTrgt=5000, RelTrgt=7000: The next PTP is at Pos=8000 (1000 + 7000).
3. Current Pos=1000, AbsTrgt=5000, RelTrgt=0: Repeat the motion to Pos=5000, and then return Pos=1000, as in a repeated loop.

RptWait (Repeating Wait Time)

Items	Content
Mnemonic	RptWait
CAN code	147
type	32-bit integer (int)
Access rights	Read and write
Axis Related	Yes
User units	Milliseconds (ms)
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	0 to 131,071,993
Default value	0

Function:

- In the repetitive motion mode, define the time spent on each endpoint in milliseconds.
- Support adjusting the waiting time for the motor to move reciprocally between two points.

Application Scenarios and Significance:

- Convenient debugging: The time separation between the end of the motion and the next start is obvious, which is convenient for observing and recording the motion trajectory.
- Let the system slow down and attenuate during motion to avoid sudden changes and facilitate monitoring.
- When the motor is in motion for long periods of time, adding waiting time can prevent the hardware from overheating.

Suggestions for use:

- In complex or high acceleration/deceleration motions, a waiting time buffer system can be set to ensure smooth motion.

RptMode

Items	Content
Mnemonic	RptMode
CAN code	712
type	32-bit integer (int)
Access rights	Read and write (available and set)
Axis Related	Yes
User units	None (belongs to a pattern number or code)
Default **value**	712 (if your requirement is set to 712)

`RptMode` defines the repetition mode of the motion, i.e., how the motion will be repeatedly executed. Common patterns may include:

- **0**: No repetition (single motion).
- **1**: Circular round trip (reciprocating motion repeats non-stop).

RptCycles (number of repetitions executed)

Items	Content
Mnemonic	RptCycles
CAN code	713
type	32-bit integer (int).
Access	Read and write
Axis Related	Yes
User units	Frequency (number of repetitions)
Motion is allowed	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	0 to 2,147,483,647 (maximum is a 32-bit integer maximum)
Default value	0

Function:

- `RptCycles` is used to set the total number of repetitions to be performed in the multi-repetition motion mode.
- When `RptMode` is set to "Fixed number of repetitions".

RptCounter (Repeat Count)

Items	Content
Mnemonic	RptCounter
CAN code	714
type	32-bit integer (int).
Access	Read-only (cannot be modified, can only be read)
Axis Related	Yes
User units	Frequency (Number of completed repetitions)
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
range	0 to 2,147,483,647 (maximum is a 32-bit integer maximum)
Default value	0 (default value before the start of the exercise)

Function:

- `RptCounter` is used to read the number of completed repetitions in real time.

CurrPosTh (Switching Force Control Position Threshold)

Items	Content
Mnemonic	CurrPosTh
CAN code	426
type	32-bit integer (int).
Access	Read and write
Axis Related	Yes
User units	User-defined units (possibly position units)
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
range	- 2,147,483,648 to 2,147,483,647
Default value	0

Function:

Global thresholds that must be met in order to switch to current or force operation mode. If the position reference (PosRef) is greater or less than the threshold, it can be toggled (if any other threshold is triggered)

CurrPosThDir (Switching Force Control Position Threshold Direction)

Items	Content
Mnemonic	CurrPosThDir
CAN code	427
type	32-bit integer (int).
Access	Read and write
Axis Related	Yes
User units	None (direction flags only, i.e. direction parameters)
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
range	-1 to 1
Default value	0

Function:

- `CurrPosThDir` indicates the direction of the current position threshold, with values of `-1`, `0`, `1 only`.
  - `-1`: Means that the threshold is in the opposite direction of the current position.
  - `0`: No specific orientation setting, default behavior (usually means not biased in any direction).
  - `1`: The positive direction of the threshold at the current position.

Items	Content
Mnemonic	StartOnToPos
CAN code	587
type	32-bit integer (int).
Access	Read and write
Axis Related	Yes
User units	None (0 or 1) represents a boolean value that controls whether or not the motion starts automatically
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
range	0 (Does not start automatically) to 1 (starts automatically)
Default value	0

Used for the controller's behavior when switching to position mode and falling back to the set position:

- **Value 1**: Auto start motion (automatically start motion when switching modes for quick and continuous operation).
- **A value of** 0: No motion is initiated

Application Scenarios:

- In force control mode or position control switching scenarios, enabling this parameter can enable the fallback function to achieve automatic start.

16.2 Vector keywords

VecAbsTrgt (Vector Motion Target Position)

Items	Content
Mnemonic	VecAbsTrgt
CAN code	642
type	Read only
Access	Read-only (cannot be set)
Axis Related	Yes
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Value range	-2,147,483,648 to 2,147,483,647
Default value	0
User units	None (no user units, direct values)

Function description:

- `VecAbsTrgt` represents the target distance of the current vector motion (from start to end) It is just a state parameter used to monitor the distance of motion.
- **It is not used to define the motion path**; the actual motion definition is defined by the associated `RelTrgt` (relative target distance) or the AbsTrgt of the member axes.
- **Note**: `VecAbsTrgt` is always positive, indicating absolute distance.

VecAccel (Vector Motion Acceleration)

Items	Content
Mnemonic	VecAccel
CAN code	636
type	Read / Write
Access	Can be set in motion
Axis Related	Yes
Save to flash	Yes
Allowed in motion	Yes
Allowed with motor on	Yes
Value range	100 to 2,000,000,000
Default **value**	100,000
User units	User unit (corresponding to the actual application of the motion unit)

Function description:

- `VecAccel` sets the acceleration in vector motion. The higher the value, the faster the acceleration.
- This parameter is valid for both the start of the motion or the dynamic adjustment during the motion.
- The specific value should be set reasonably according to the hardware capabilities and motion needs.

VecArcCenter (Vector Motion Arc Motion Center Point)

Items	Content
Mnemonic	VecArcCenter
CAN code	633
type	Parameter
Access	Read / Write
Allowed in motion	No (cannot be modified during motion)
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
Minimum value	-2,147,483,648
Maximum value	2,147,483,647
Default value	0
User units	In user unit

Function description:

- `VecArcCenter` defines the center position of the arc in the arc's motion.
- Suitable for arc vectors, the controller can calculate the radius by setting the arc center coordinates.
- This parameter works in combination with all other vector arc motion parameters to ensure the correctness of the trajectory of motion.
- It can only be modified in the non-motion state (i.e., the parameter cannot be changed while moving).
- Note: The `VecArcCenter` for each axis defines the coordinates of the arc center.

Related parameter references

- `VecAccel`: Motion acceleration
- `VecAbsTrgt`: Target Absolute Distance
- `VecArcDir`: Arc direction
- `VecEncRatio`: Encoder Ratio

Items	Content
Mnemonic	VecArcDir
CAN code	634
type	Parameter
Access	Read / Write
Allowed in motion	No (cannot be modified during motion)
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
Minimum value	0
Maximum value	1
Default value	0
User units	None (no user units)

Function description:

`VecArcDir` defines the direction of arc motion:

`"0"` indicates counterclockwise (CCW).

`"1"` indicates clockwise (CW).

Motion is only based on the `VecArcCenter` of that parameter when using the `Begin` command.

Two axes are defined to make arc motion:

The plane of motion is defined by two axes: the first axis is the X axis and the second axis is the Y axis.

The arc is in the plane of these two axes, and the third axis remains unchanged.

Order matters:

For example: `B, C` vs. `C, B,` different orders represent different arc directions.

By default, the first axis is the X axis and the second axis is the Y axis, and the direction of motion of the arc is determined by `VecArcDir`.

Note:

Motion definitions cannot be modified after they are set before motion.

The design needs to align with the arc definition of CNC to ensure accurate motion.

Related parameter references

- `VecAccel`: Motion acceleration
- `VecAbsTrgt`: Target distance
- `VecArcCenter`: arc-center coordinates
- `VecEncRatio`: Encoder Ratio
- Other event parameters: `oEventOn`, `oEventGap`, `etc`

VecDecel (Vector Motion Deceleration)

Items	Content
Mnemonic	VecDecel
CAN code	637
type	Read / Write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
Minimum value	100
Maximum value	2,000,000,000
Default value	100,000
User units	User Units (Same as VecAccel)

Function description:

- `VecDecel` defines deceleration in vector motion, and the higher the value, the faster the deceleration.
- It can be dynamically adjusted during motion.
- Works with `VecAccel` to control the acceleration and deceleration characteristics of the motion.

Related parameter references

- `VecAccel`: Acceleratiom
- `VecAbsTrgt`: Target Absolute Distance
- `VecArcCenter`: arc-center coordinates
- `VecArcDir`: Arc direction
- `VecEncRatio`: Encoder Ratio

VecdPosRef (derivative of the position of the vector motion)

Items	Content
Mnemonic	VecdPosRef
CAN code	644
type	Read only
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	-2,147,483,648
Maximum value	2,147,483,647
Default value	0
User units	None (the value has no user units, expressed as the derivative rate)

Function description:

- `VecdPosRef` represents the derivative of the reference position of the vector motion, which is the numerical value of the speed of motion.
- It is a state parameter, always positive (representing the speed magnitude), to monitor the speed at the moment of motion.
- It can be read in real time during motion.

VecEmrgDec (Vector Motion Emergency Deceleration)

Items	Content
Mnemonic	VecEmrgDec
CAN code	638
type	Read / Write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
Minimum value	100
Maximum value	2,000,000,000
Default value	100,000
User units	User unit (corresponding to the actual unit of the motion)

Function description:

- `VecEmrgDec` defines the emergency deceleration of the motion, with higher values indicating faster deceleration.
- In the event of a sudden emergency stop or other situations where rapid deceleration is required, the controller uses the value of this parameter to ensure a safe response.

Relevant parameters

- `VecAccel`: Acceleration
- `VecDecel`: Deceleration
- `VecAbsTrgt`: Target distance
- `VecArcCenter`、`VecArcDir`: Arc motion parameter
- `VecEncRatio`: Encoder Ratio

VecEncRatio (Vector Motion Encoder Resolution)

Items	Content
Mnemonic	VecEncRatio
CAN code	632
type	Read / Write
Allowed in motion	No (cannot be modified during exercise)
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
Minimum value	256
Maximum value	25,600
Default value	256 (scale=1)
User units	None (numerical value is a scale factor)

Function description:

- `VecEncRatio` is used to compensate for encoder resolution differences for each axis.
- The actual scale inside the controller is its value divided by 256, for example:
- `VecEncRatio=256` indicates a ratio of 1
- `VecEncRatio=260` indicates a ratio of 260/256 ≈ 1.016
- The motion position is synchronized with the encoder reading, ensuring the accuracy of the motion.
- Modifications are prohibited in motion to avoid accumulation of errors.

Relevant parameters

- `VecAccel`: Motion acceleration
- `VecDecel`: Deceleration
- `VecAbsTrgt`: Target distance
- `VecArcCenter`、`VecArcDir`： Arc motion parameter
- `VecEncRatio`: Encoder Ratio

VecJerk (vector motion smoothing coefficient)

Items	Content
Mnemonic	VecJerk
CAN code	639
type	Read / Write
Allowed in motion	No (cannot be modified during motion)
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
minimum	0
maximum	9
Default	0
User units	None (Numeric range)

Function description:

- `VecJerk` defines the Jerk parameter in motion that controls the smoothness of the motion.
- Larger values change the motion faster but may introduce oscillations or reduced smoothness.
- It can only be set before the start of the motion and cannot be modified during the motion.

Related parameter references

- `VecAccel`: Acceleration
- `VecDecel`: Deceleration
- `VecJerk`: Jerk
- `VecAbsTrgt`: Target distance
- The leap value in the motion parameters is directly related to the smoothness of the motion

VecMemberAxes

Items	Content
Mnemonic	VecMemberAxes
CAN code	631
type	Read / Write
Allowed in motion	No (cannot be modified during motion)
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
Minimum value	0
Maximum value	255
Default value	0
User units	None

Function description:

`VecMemberAxes` uses 8-bit binary bits to indicate whether different axes are involved in vector motion. For example:

- `bit 0` represents axis A (first axis).
- `bit 1` represents axis B (second axis).
- And so on

By setting different bit values, multiple axes can be combined to participate in the motion. As:

- Select Axis A and Axis B: `1 | 2 = 3`
- Axis A only: `1`
- Axis B only: `2`

Example:
Set the participation axes A and B: `VecMemberAxes = 3` (binary `00000011`).

Relevant parameters

- `VecAccel`: Acceleration
- `VecDecel`： Deceleration
- `VecJerk`: Jerk
- `VecAbsTrgt`: Target position
- `VecArcCenter`： Arc Centre Coordinates
- Other motion parameters depend on the axis of participation settings

VecMotionStat (Vector Motion State)

Items	Content
Mnemonic	VecMotionStat
CAN code	641
type	Read only
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	-2,147,483,648
Maximum value	2,147,483,647
Default value	0
User units	None

Function description:

- `VecMotionStat` is used to obtain the current state of vector motion and can dynamically read the state information of the motion.
- Since it is a read-only parameter, it cannot be modified and is only used for monitoring and feedback.
- For specific status codes, please refer to the status code definition table of the device (usually composed of a variety of signs or status codes used to indicate the state of motion in progress, pause, abnormality, etc.).

VecPause (Vector Motion Pause)

Items	Content
Mnemonic	VecPause
CAN code	640
type	Read / Write
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	0
Maximum value	1
Default value	0
User units	None

Function description:

- **With a value of 1**: Pause the vector motion and the motion reduces the speed to 0 until it starts again.
- **With a value of 0**: Normal motion resumes (if previously paused, the motion accelerates to the original set speed `VecSpeed`).

VecPosRef (Vector Position Reference Value)

Items	Content
Mnemonic	VecPosRef
CAN code	643
type	Read only
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	No
Axis Related	Yes
Minimum value	-2,147,483,648
Maximum value	2,147,483,647
Default value	0
User units	None

Function description:

- `VecPosRef` represents the real-time position reference value of the current vector motion (current position along the path).
- It starts at 0 and reaches the target position, `VecAbsTrgt` after the end of the motion.
- **Always positive**, reflecting the current position of the motion.
- It is often used to monitor motion progress.

VecSpeed

Items	Content
Mnemonic	VecSpeed
CAN code	635
type	Read / Writ
Allowed in motion	Yes
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
Minimum value	0
Maximum value	60,000,000
Default value	10,000
User units	User unit (specific unit is set according to the motion)

Function description:

- `VecSpeed` is used to set the target speed of the motion, which affects the determination of acceleration and the smoothness of the motion.
- The higher the value, the faster the motion.
- Can be modified during motion.

Relevant parameters

- `VecAccel`: Acceleration
- `VecDecel`: Deceleration
- `VecAbsTrgt`: Target position
- `VecEncRatio`: Encoder Ratio
- `VecJerk`: Jerk
- `VecMotionStat`: Motion status monitoring

VecType

Items	Content
Mnemonic	VecType
CAN code	630
type	Read / Write
Allowed in motion	No
Allowed with motor on	Yes
Save to flash	Yes
Axis Related	Yes
Minimum value	0
Maximum value	1
Default value	0
User units	None

Function description:

`VecType = 0`: Linear motion.

`VecType = 1`: Arc motion.

Possible future expansion support:

`VecType = 2`: Combination arc and linear motion (Combination of main arc motion and linear motion of other axes).

It cannot be modified during motion to ensure the motion path and motion precision.

Related parameter references

- `VecAccel`: Acceleration
- `VecAbsTrgt`: Target position
- `VecArcCenter`： Arc-center coordinates
- `VecArcDir`: Arc direction
- `VecSpeed`: Speed of motion
- `VecEncRatio`: Encoder Ratio

VecNumCircles

Items	Content
Mnemonic	VecNumCircles
CAN code	646
type	32-bit integer (int).
Access rights	Read / Write
Axis Related	Yes
Allowed in motion	No (cannot be modified during motion)
Allowed with motor on	Yes
Save to flash	Yes
Minimum value	0
Maximum value	100
Default value	0

Function description:

- `VecNumCircles` are primarily used to define the number of circles in an arc or arc path, as a motion path can consist of multiple circles, each "circle" representing a complete arc.
- Setting to 0 means that there is no loop path or just a linear line.

Relevant parameters

- `VecArcCenter`: Arc-center coordinates
- `VecArcDir`: Arc direction
- `VecType`: The path type (linear or arc).
- `VecAbsTrgt`: Target position
- `VecSpeed`: Speed of motion

VecPosFOn (Vector Motion Position Filter Switch)

Items	Content
Mnemonic	VecPosFOn
CAN code	648
type	32-bit integer (int).
Access rights	Read / Write
Axis Related	Yes
Allowed in motion	No
Allowed with motor on	Yes
Save to flash	Yes
Minimum value	0
Maximum value	1
Default value	0

Function description:

- `VecPosFOn` controls whether position feedback is enabled:
  - **0: Off** (default).
  - **1: Turn it on**
- It can only be set when the motion is stopped and cannot be modified during motion.
- It is mainly used to enable or disable the function of position feedback during motion.

VecPosFDef (Vector Motion Position Filter Type)

Items	Content
Mnemonic	VecPosFDef
CAN code	647
type	32-bit integer array (range: 1 to 5)
Access rights	Read / Write
Axis Related	Yes
Allowed in motion	No
Allowed with motor on	Yes
Save to flash	Yes
Value range	- 2,147,483,648 to 2,147,483,647
Default value	0

Function description:

- VecPosFDef is an array with an index range of 1 to 5 that is used to preset position feedback defaults in different directions or at different stages.

The function is mainly to set or reset target value or reference value of position feedback.

- None (0 - None): No filtering effect.
- First-order General Filter: Basic first-level filtering for simple filtering needs.
- Second-order low-pass filter: Smoother low-pass filtering, which can remove high-frequency noise.
- First-order lead-lag filter: Phase lead is introduced at the same time as filtering to compensate for system delay.
- First-order lag filter: Introduces a lag effect after filtering to improve system response.
- Second-order lead-lag filter: More complex lead-lag filter, combining multi-stage filtering with phase lead functionality
- Second-order lag filter: Similar to the above, but with more advanced response adjustments.
- Notch: Filtering of a specific frequency range used to suppress a certain frequency component.
- Complex lead-lag filter: A composite filter used for highly complex filtering and phase adjustment requirements.

17 Appendix

Through the above documentation, you can learn more about how to use Agito-AAMotion and its features. If you have any other questions, please consult the source code of the library or related documentation.