NEXCOBOT
Open Robot & Machines

NexMotion Library
User Manual

Version: 1.6
Date: 2018-11-15

Copyright Statement and Disclaimer

The contents contained in this document are the proprietary property of NexCOBOT Co., Ltd. (NexCOBOT hereafter) and is subject to the protection of intellectual property law (including, but not limited to the Copyright Act). The use of any material in relation to this document without the prior authorization of NexCOBOT is considered infringement. Without the written approval of NexCOBOT in advance, this document or any part of it shall not be photocopied, sold, distributed, modified, published, stored or otherwise used.

To keep this document and its contents correct and complete, NexCOBOT reserves the right to change or revise the document at any time without further notification.

Operating machine or equipment has a certain level of danger. It is the user's responsibility to pay special attention and have safety protection in place before operating any machine or equipment. NexCOBOT shall not be held for any and all direct or indirect damage or loss to the equipment mentioned in this document due to the use for a purpose other than the intended.

Revision History

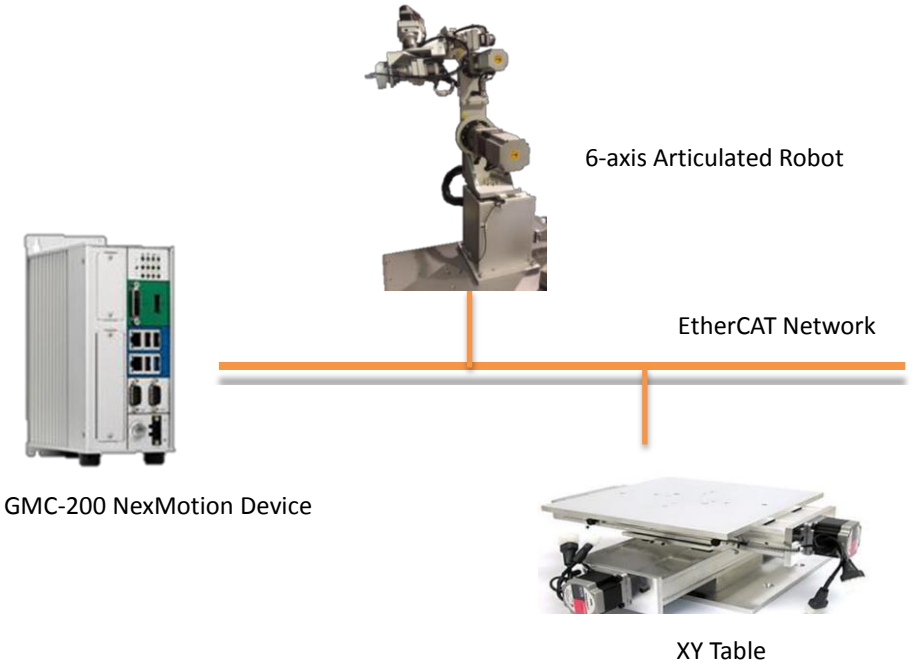
Rev.	Description
1.0	First released.
1.1	Add Programming theory
1.2	1. Insert a chapter 3.2.8 get description APIs. 2. Add an axis parameter. Chapter 2.2
1.3	Modify description of NMC_GroupGetMotionBuffSpace()
1.4	1.Add Ch1.4.2.4~7 for tool/base setting and teaching 2.Add Ch2.3 Tool/Base parameters for group 3.Add Ch3.4.13~14 tool/base teaching APIs
1.5	1.Add Ch3.4.15 3D simulation view APIs 2.Add Ch3.2.6.6~17 I/O functions.
1.6	1.Add Ch3.2.6 new I/O functions 2.Enhance I/O function descriptions 3.Correct the descriptions of group parameters

Table of Contents

1. Programming Principles

The basic development procedure of the NexMotion application is described as follows:

1. Plan the control system
2. Configure and test the controlled system with the NexMotion Studio
3. Generate the configuration file (NCF file) of the controlled system with the NexMotion Studio
4. Develop the control program
5. Compile and debug the control program
6. Test and fine tune the control program



Control System Example (Device + Controlled System)

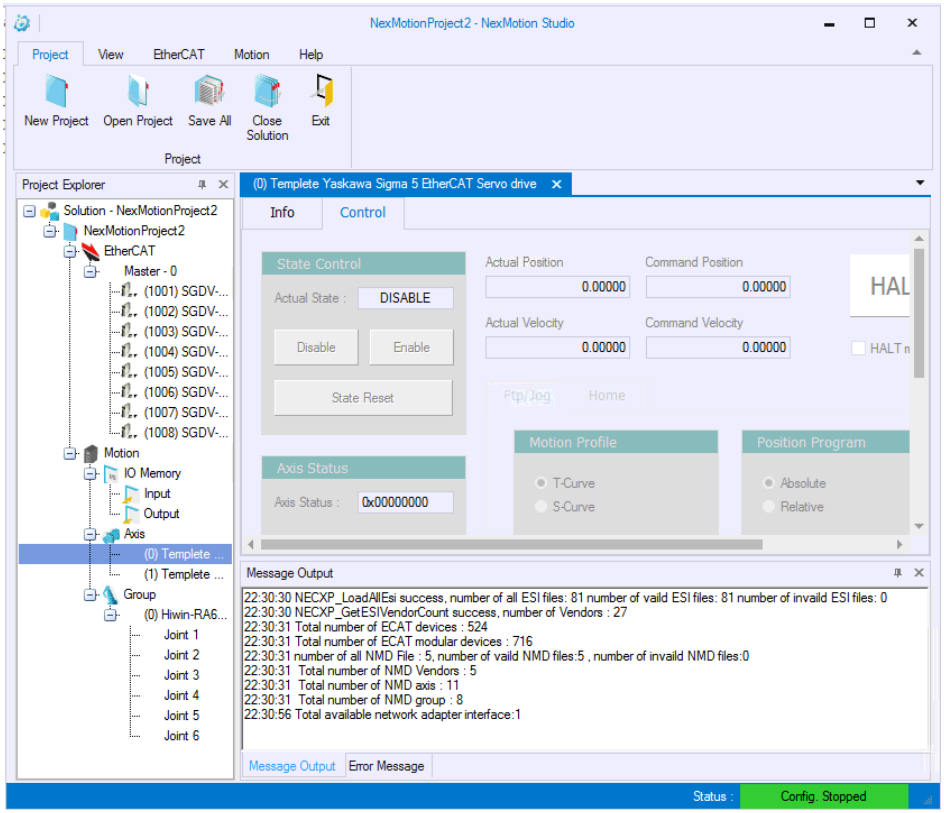
The first 3 steps aim to confirm the following items:

1. The installation and wiring of the controlled system, such as the installation of the servo motor, the operations of the I/O devices, etc., and
2. The axis control parameters controlled with the NexMotion Studio for the equipment operation. For example, a controlled system composed of a 6-axis articulated robot and a set of 2-axis XY table can be controlled with the NexMotion Studio not only to set the mechanism parameters, units, acceleration/deceleration, and other configurations, but also to perform the homing, point-to-point and other motions to check if the motion direction, unit moving distance, and others are in compliance with the system plan. Please refer to the NexMotion Studio User Manual for the usage of the NexMotion Studio.

After the above steps are completed, the configuration files will be generated in the default system path (C:\Nexcom\). The files in the path shall not be modified to avoid any undesired system error. The 4th step, develop the control program, indicates the control program developments based on the library provided by the NexMotion. The program can issue commands to the device by calling APIs to complete each applications. The following sections will describe the library functionalities systematically, including:

1. System operations,
2. I/O control,
3. Axis control, and
4. Group control.

These classes of these APIs can be understood based on the naming rule of such APIs. For example: the APIs with the prefix NMC_Device are the System Operations class, the APIs with the prefix NMC_Axis are related to the Axis Control, the APIs with the prefix NMC_Group are related to the Group Control, and so on. Moreover, the extension applications can be developed by referring to the sample programs provided under the installation folder.



NexMotion Studio Development and Configuration Tools

1.1. System Operations

The section describes the following items:

1. System initialization,
2. Watch dog, and
3. Debugging.

1.1.1. System Initialization

The system initialization shall be performed before starting to use the Library, the simplest way is to call the function NMC_DeviceOpenUp(). A device identification (ID) will be returned after the successful initialization, and it be used to control the device.

A C sample program shows the system open up and shut down as follows:

```
#include "NexMotion.h"
#include "NexMotionError.h"

int main()
{
	RTN_ERR ret;
	I32_T		devType	= NMC_DEVICE_TYPE_ETHERCAT;
	I32_T		devIndex = 0;
	I32_T		devID;
	ret = NMC_DeviceOpenUp( devType, devIndex, &devID );
	if( ret != ERR_NEXMOTION_SUCCESS )
	{
		// Error handling...
	}

	// Device is start up successfully.
	// Do something...
	// Support to write codes for control

	NMC_DeviceShutdown( devID );

	return 0;
}
```

NMC_DeviceOpenUp() is a blocked function. It will perform the system initialization processes, including:

1. Establish a device, and generate the device identification (ID),
2. Load the configuration file (NCF file), and
3. Open up the device communication.

The above processes may cost several seconds. After the function returns successfully, the following processes can send commands to the axes or groups. In addition, the Library also supports a non-blocked function, NMC_DeviceOpenUpRequest(). The function can return immediately after called. The device can also receive the same commands and perform the above open up processes. After completing these processes, the device state will transfer to 「OPERATION」, and NMC_DeviceGetState() can be used to check the state by polling or NMC_DeviceWaitOpenUpRequest() can be used to wait for the open up completion.

1.1.2. Device Shutdown

NMC_DeviceShutdown() can be used to shut down the device. After the function is called, the device will close communications immediately, so that users shall pay attention to the shutdown process. For example, users can call NMC_DeviceShutdown() to shut down the device completely after confirming all device motions are stopped.

1.1.3. Advanced System Initialization

The aforementioned procedure is easy and quick for the general initialization. However, some parameters may be configured before the system opens up. The system initialization can be achieved by separating the procedure as follows:

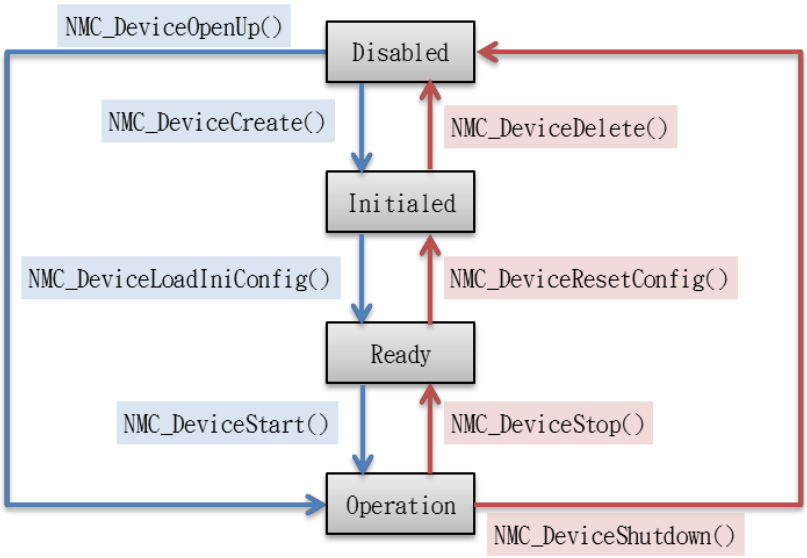
```
#include "NexMotion.h"
#include "NexMotionError.h"

int main()
{
	RTN_ERR ret;
	I32_T		devType = NMC_DEVICE_TYPE_ETHERCAT;
	I32_T		devIndex = 0;
	I32_T		devID;

	ret = NMC_DeviceCreate( devType, devIndex, &devID );
	if( ret != ERR_NEXMOTION_SUCCESS )
	{
		// Error handling...
	}

	ret = NMC_DeviceLoadIniConfig( devID );
	if( ret != ERR_NEXMOTION_SUCCESS )
	{
		// Error handling...
	}

	// Do something here...
	// Parameter setting...

	ret = NMC_DeviceStart( devID );
	if( ret != ERR_NEXMOTION_SUCCESS )
	{
		// Error handling...
	}


	// Device is start up successfully.
	// Do something...

	// ...

	NMC_DeviceShutdown( devID );
```

```
    return 0;
}
```

The separation initialization procedure shall call the 3 APIs in order:

1. NMC_DeviceCreate()
2. NMC_DeviceLoadIniConfig()
3. NMC_DeviceStart()

The purpose is transfer the device state to 「OPERATION」. The below figure shows the device state transitions and related functions.



Device State Transitions

The device states can be getting by function NMC_DeviceGetState().

1.1.4. Watch Dog Timer

The watch dog timer is a dedicated timer of the device. After the timer starts, the application shall reset the value of timer within a specific period. Otherwise, if the timer reaches the configured time, the device will perform the corresponding actions for system shutdown automatically.

The function is a protective measure to avoid the application crash or the system failure due to other programs. In case of the system failure, the device will enable the shutdown process because the timer is timeout when application cannot be reset and.

If the timer is enabled during the debugging phase of development stage, the debug process may interrupt the program. Some undesired(unexpected?) cases may occurred since the timer is timeout and the system is shut down. Therefore, it is recommended to disable the function at the development stage and enable the function after the development completion for additional protection.

Generally, the watch dog timer is used by enabling a system timer interrupt or creating an independent thread. NMC_DeviceWatchdogTimerEnable() is used to enable the watch dog timer, and NMC_DeviceWatchdogTimerReset() is used to reset such timer. The reset may be two times or more as frequent as the configured time. NMC_DeviceWatchdogTimerDisable() is used to disable the timer.

An example is shown as follows:

```
#include "NexMotion.h"
#include "NexMotionError.h"
```

```
#include <Windows.h>

I32_T gThreadCtrl = 1;

DWORD WINAPI WatchDogTimerResetThread(_In_ LPVOID lpParameter )
{
	I32_T		devId		= *(I32_T *)lpParameter;
	I32_T		timeoutMs = 1000;
	DWORD sleepMs		= timeoutMs / 2;

	NMC_DeviceWatchdogTimerEnable( devId, timeoutMs, 0 );

	while( gThreadCtrl == 1 )
	`{
		NMC_DeviceWatchdogTimerReset( devId );
		Sleep( sleepMs );
	}

	NMC_DeviceWatchdogTimerDisable( devId );

	return 0;
}

int main()
{
	RTN_ERR ret;
	I32_T		devType = NMC_DEVICE_TYPE_ETHERCAT;
	I32_T		devIndex = 0;
	I32_T		devID;
	HANDLE threadHandle;

	ret = NMC_DeviceOpenUp( devType, devIndex, &devID );
	if( ret != ERR_NEXMOTION_SUCCESS )
	{
		// Error handling...
	}

	threadHandle = CreateThread( NULL, 0, WatchDogTimerResetThread, &devID,
0, NULL );
	if( threadHandle == NULL )
	{
		// Error handling...
	}

	// Device is start up successfully.
	// Do something...
	// ...
```

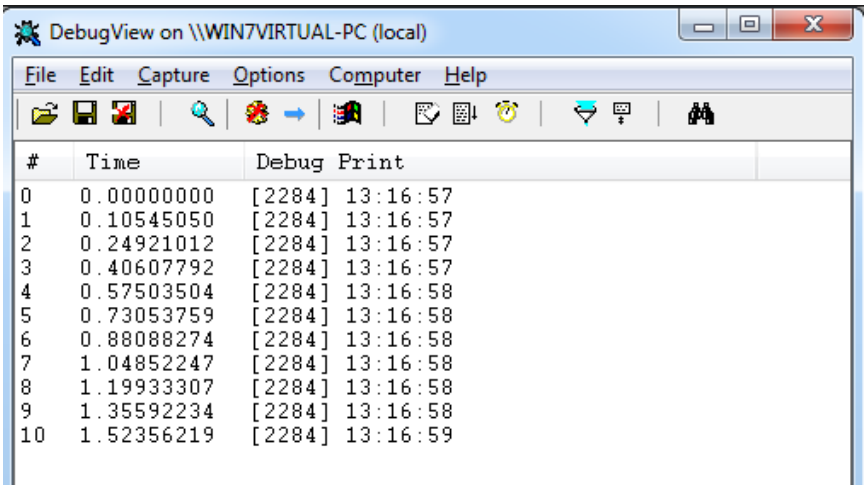
```
    // Try to stop WDT thread
    gThreadCtrl = 0;
    WaitForSingleObject( threadHandle, INFINITE );

    NMC_DeviceShutdown( devID );

    return 0;
}
```

1.1.5. Debugging

To assist developers to debug the application, there are two measures, system messages and API trace.

1.1.5.1. System Message

The system messages will be issued by the device during the operating and stored in the message queue. NMC_MessagePopFirst() can be used to get and remove the system messages from the message queue. To facilitate the debugging, NMC_MessageOutputEnable() can be used to copy the message and to transfer it to the MS Windows system message (OutputDebugString). Developers can download the tool, DebugView, to get the MS Windows system message from the website: https://docs.microsoft.com/zh-tw/sysinternals/downloads/debugview.



DebugView

1.1.5.2. API Trace

The API Trace can trace the calling of NexMotion Library by applications. NMC_DebugSetTraceMode() can used to enable the API Trace. The called function name, input parameters and return values can be output to MS Windows system message (OutputDebugString) in accordance with the configured Trace mode. Developer can observe the calling of NexMotion with the software, DebugView.

Moreover, a hook function (or a callback function) can be used to hook the self-defined function into the NexMotion functions. NMC_DebugSetHookFunction() can register the self-defined function into the system, and the NexMotion APIs will call the hook function before returning.

The registered hook function is global, and all called NexMotion APIs will call the same hook function. NMC_DebugGetApiAddress() can be used to inquiry the names of called APIs.

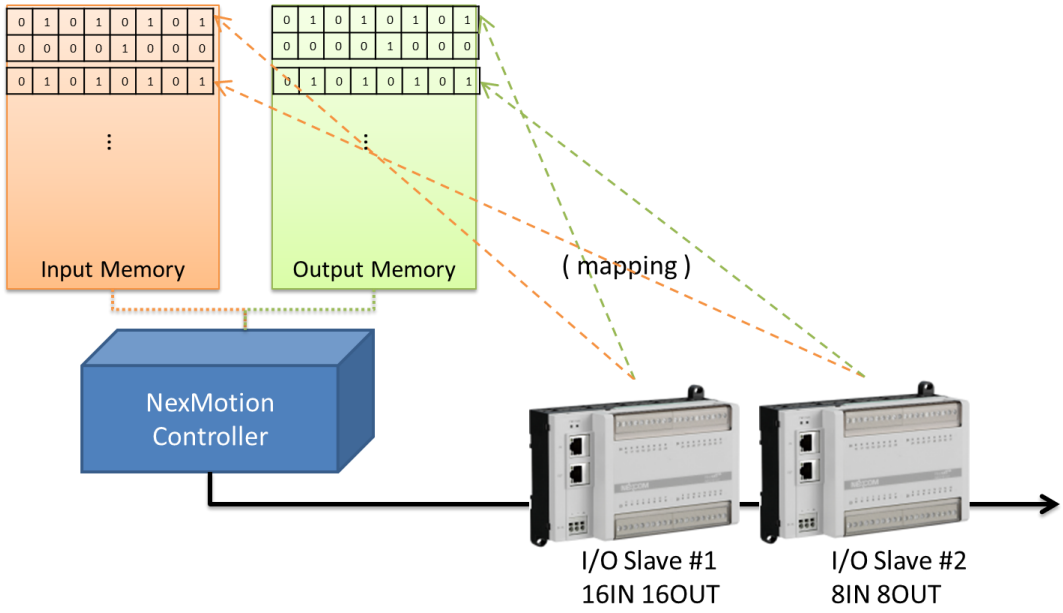
An example is shown as follows:

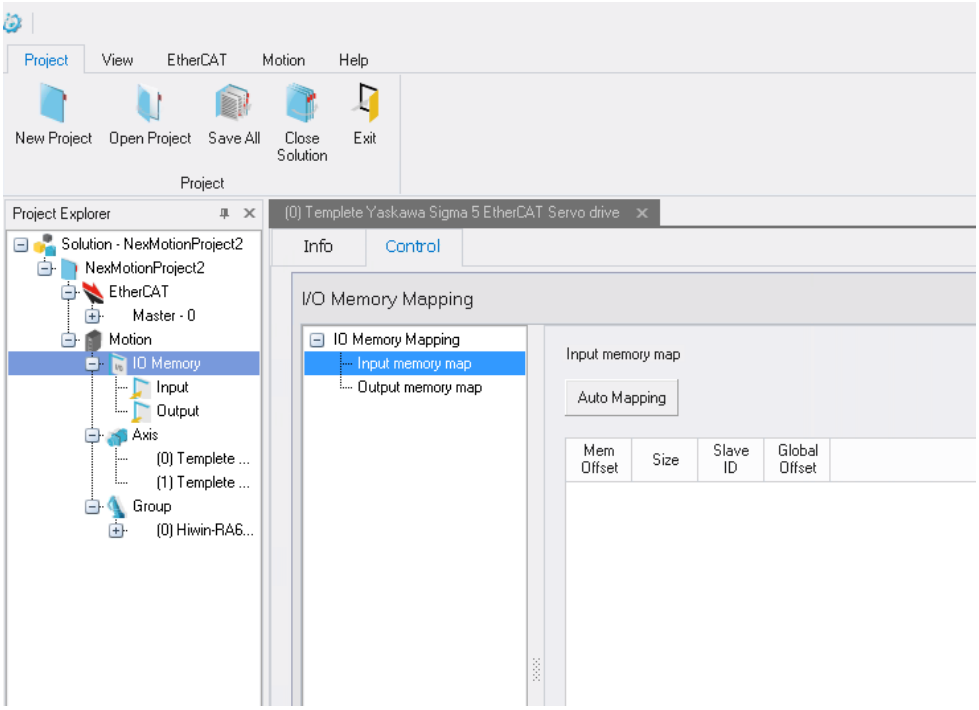
```
#include "NexMotion.h"
#include "NexMotionError.h"
#include <stdio.h>

void MyHookFunction(
      const void      *PFuncAddress      // [i] Function name call this hook function.
      , const char    *PFuncName  // [i] Pass API Name to hook function.
      , RTN_ERR       ReturnCode    // [i] function return call
      , void      *PUserData    // [i] User data.
)
{
```

```
	const void *pFunAddr =
NMC_DebugGetApiAddress( "NMC_DeviceOpenUp" );
	I32_T *pCounter	= (I32_T *)PUserData;

	if( PFuncAddress == pFunAddr )
	{
		(*pCounter)++;
		printf( " NMC_DeviceOpenUp is called %d times \n", *pCounter );
	}

	printf( "Hook: %s is called.\n", PFuncName );
}

int main()
{
	RTN_ERR ret;
	I32_T	devType = NMC_DEVICE_TYPE_ETHERCAT;
	I32_T	devIndex = 0; I32_T devID;
	I32_T	counter = 0;

	// Set Hook function
	NMC_DebugSetHookData( &counter );
NMC_DebugSetHookFunction( MyHookFunction );

	ret = NMC_DeviceOpenUp( devType, devIndex, &devID );
	if( ret != ERR_NEXMOTION_SUCCESS )
	{
		// Error handling...
	}

	// Device is start up successfully.
	// Do something...
	// ...

	// Disable hook function
	NMC_DebugSetHookFunction( 0 );
	NMC_DebugSetHookData( 0 );

	NMC_DeviceShutdown( devID );

	return 0;
}
```

1.2. I/O Control

NexMotion provides flexible and high-velocity I/O control as the memory access. Developers shall map the memory address of I/O devices onto the virtual memory with NexMotion Studio, and then they can control or get/set the I/O devices by accessing the memory. Please refer to the NexMotion Studio user manual for the configurations.





NexMotion Studio I/O Memory setting screen

After configurations, the following functions can be used to access or control the I/O devices:

1. NMC_ReadInputMemory(),
2. NMC_ReadOutputMemory(), and
3. NMC_WriteOutputMemory().

Note: The update rate of I/O memory is once per 10 ms (100Hz). Therefore, the functions may not support responses immediately if the frequency of digital output calling is higher than 100Hz.

1.3. Axis Control

Different from the group axis for coordination operations in the mechanism, the single axis described in the section is independent for the corresponding to the mechanical structure. Functionally, the contents of this section are mainly divided into three major parts. The first part is the axis configurations, including the unit and the software limit protection configurations. The second part is the motion control functions, including excitation, point-to-point motion, JOG motion, stop motion and the change on fly function. Finally, the third part mainly focuses on reading the information related to the axis motion, including reading the current status and motion information of axis.

1.3.1. Axis Unit Configuration

The required parameters set with the APIs for the axis motion, such as the position of point-to-point motion or the velocity of the JOG motion. These motions are operating on these units defined by users (user unit). To establish the relationships between these user units and the command counts actually set to the device, users shall set the related unit parameters first. The axis parameters related to unit configurations are listed in the below table:

Param. Num	Sub Index	Description	Remark
0x00	0	Mechanical pitch (unit/rev)	(*1)
0x01	0	Mechanical revolution	(*1)
0x02	0	Motor revolution	(*1)
0x03	0	Encoder resolution (pulse/rev)	(*1)

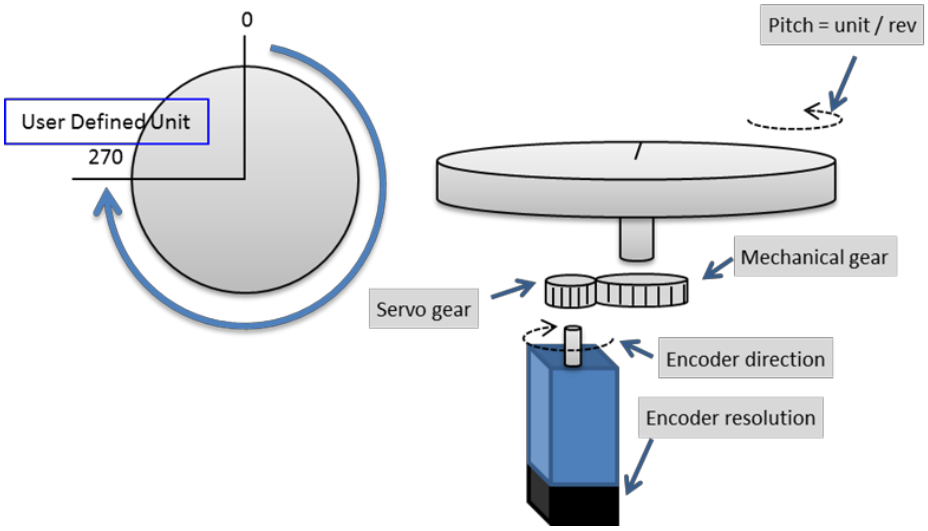
(*1) The parameter cannot be modified after the system is enabled.

- 0x00: Mechanical pitch (user unit/rev)
  The parameter mainly describes the user unit required for a mechanism revolution.

- 0x01: Mechanical revolution
  0x02: Motor revolution
  The two parameters must be set coordinately. They are used to describe the gear ratio relationship between the motor and the mechanism. For a rotating mechanism, there is usually a velocity reducer between the motor and the mechanism. Assume that the gear ratio of the velocity reducer is 80. It means that the motor rotates 80 revolutions while the mechanism rotates one revolution. Therefore, the 0x01 shall be set to 1 and the 0x02 shall be set to 80. For the linear mechanism with the lead screw, if there is only a coupling between the motor and the mechanism and there is no velocity reduction mechanism between them, the 0x01 and 0x02 shall be set to 1. However, if there is a velocity reduction mechanism between the motor and the mechanism, the 0x01 and 0x02 must be set in accordance with the gear ratio of the velocity reduction mechanism.

- 0x03: Encoder resolution (pulse/rev)
  The parameter is mainly used to set the revolution of encoder. It represents how many pulse counts the encoder will return for a revolution of motor. For an encoder with a 20-bit resolution, the parameter shall be set to 1,048,576 because of 2^{20}.
  The unit conversion formula is as follows:
  $$1\ \text{User unit} = \text{EncoderResolution} \times \frac{Motor\ revolution}{Mechanical\ revolution} \div Pitch\ (Pulse)$$

Unit configuration example 1: A disk mechanism



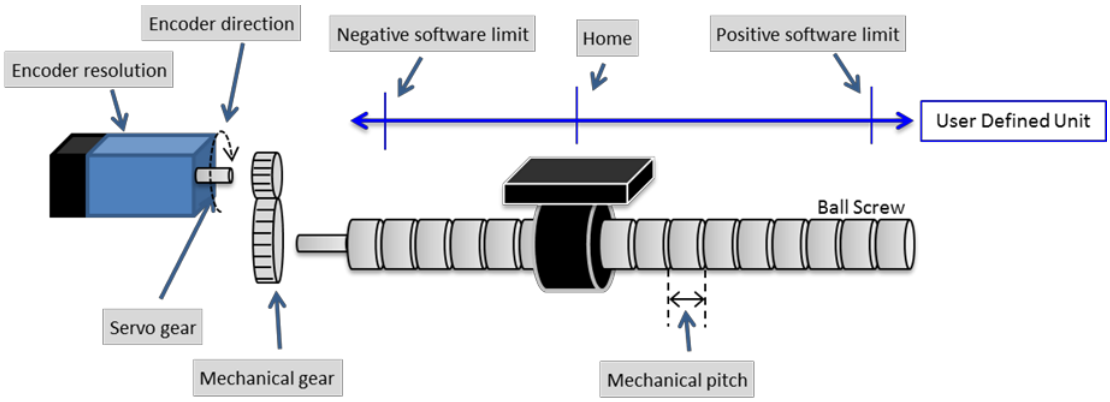
A Rotary Motor, a Velocity Reducer and a Rotary Mechanism

In case the gear ratio of velocity reduction is 1:5 (i.e. the mechanism rotates a revolution while the motor rotates 5 revolutions), the encoder resolution is 1,048,576 pulse/rev, and the user unit is assumed at 0.001 degrees per unit, the parameters are set as follows:

0x00 (Pitch) = 360,000
0x01 (Mechanical revolution) = 1
0x02 (Motor revolution) = 5
0x03 (Encoder resolution) = 1048576

The relationship between the user unit(um) and pulse is:
1 user unit (0.001 degree) = 1048576 x (5/1) / 360000 = 14.5636 pulse

Unit configuration example 2: A screw rod mechanism



A Rotary Motor, a Velocity Reducer and a Linear Screw Rod

In case the gear ratio of velocity reduction is 1:2 (i.e. the mechanism rotates a revolution while the motor rotates 2 revolutions), the encoder resolution is 131,072 pulse/rev, the lead of screw rod is 5 mm/rev, and the user unit is assumed at 1 mm per unit, the parameters are set as follows:

0x00 (Pitch) = 5
0x01 (Mechanical revolution) = 1
0x02 (Motor revolution) = 10
0x03 (Encoder resolution) = 131072

The relationship between the user unit(um) and pulse is:
1 mm = 131072 x (10/1) / 5 = 262144 pulse

Other related parameters are described as followed:

- 0x04: Encoder direction
  The parameter mainly provides the reverse function of axis. The default value is 0. Since in the control unit of axis motion can calculate the position coordinate of axis (user unit) based on the pulse counts returned by the encoder. If the change of the position coordinate corresponding to the axis motion direction is reverse to the user expectation, the parameter can be set to 1 so that the change of the position coordinate corresponding to the axis motion direction can be reversed. This function mainly supports users set the reversal motion with the controller directly.

- 0x05: Encoder type
  This parameter mainly sets the encoder type. If the encoder is incremental, the parameter is set to 0. If it is absolute, the parameter is set to 1.

- 0x06, SubIndex = 0: Enable external encoder
  0x06. SubIndex = 1: External Encoder ratio
  The parameter is mainly used to set another encoder. Also, the parameter Enable (SubIndex 0) can be set to 1 when the command pulse count set to the drive is inconsistent with pulse counts returned by the encoder, so that the position coordinate of axis can be calculated by multiplying the value returned by the encoder by the External encoder ratio. This External encoder ratio indicates the ratio to convert value returned by the encoder to the position coordinate of axis.

- 0x07: Cancel synchronized actual position to command position when servo enable
  To excite the axis, this parameter is mainly used to cancel the synchronization of the command pulse set to the drive and the pulse count returned by the encoder. In general, the command pulse set to the drive is required to synchronize the pulse count returned by the encoder for the servo motor to excite the axis since the servo motor is in the position closed loop control mode. However, the synchronization of the command pulse set to the drive and the pulse returned by the encoder shall be cancelled in the excitation since there is no position closed loop control mode for the step motor. If the synchronization is forced, some errors may occur due to the motor is operating at high velocity at the moment of excitation.

Each group axis is in the axis coordinate system (ACS) for groups. Because the unit corresponding to the coordinate is the user unit, the system shall set the parameters related to the unit with the NexMotion Studio by means of the same method before reading the device configuration file (NCF file).

These parameters are related to the selection and configuration of the mechanism components. Therefore, these parameters shall be configured before NMC_DeviceOpenUp() is called, in order to generate the corresponding device configuration file (NCF file). If NMC_DeviceOpenUp() has been called and the device state is 「OPERATION」, these parameters cannot be changed.

Since the system will get the device configuration file(NCF file) during the process of NMC_DeviceLoadIniConfig(), these parameters cannot be changed if the device is initialized by means of the advanced system initialization and the device state has been 「READY」.

1.3.2. Software Limit Protection

In an actual mechanism, there is often the travel limitation for an axis. Also, there are the position limit with positive/negative direction, the maximum allowable velocity and the maximum allowable acceleration in the axis parameters because of maximum output tongue and velocity limits for a motor. The motion control unit provides the software protection mechanism to ensure the mechanism is operating under conditions of the allowed limits during the axis is in motion. If the software limit protection is enabled, the motion parameters will be checked when an axis motion function is called. If the set limit conditions are not met, the function will return the corresponding error code. When a group is in motion, the device will stop if there is an axis exceeding the set limit protection. These parameters related to the limit protection are described as followed:

1.3.2.1. Position Limit Protection

Param. Num	Sub Index	Description	Remark
0x10	0	Positive software limit (user unit)	(*2)
	1	Enable positive software limit	
	2	Negative software limit (user unit)	(*2)
	3	Enable negative software limit	

(*2) The parameter is effective after the function is enable. The modification in another period cannot be effective immediately.

The parameters are mainly related to the software position limit. To enable the axis software position protection function, the limit position (SubIndex 0 or 2) shall be set first, and then the SubIndex 1 or 3 shall be set to 1 (Enable). To change the enabled limit position, the function shall be closed first by setting the SubIndex 1 or 3 to 0. After the new position limit is set(SubIndex 0 or 2), the function can be enabled by the SubIndex 1 or 3 to 1.

1.3.2.2. Velocity Limit Protection

Param. Num	Sub Index	Description	Remark
0x11	0	Maximum velocity limit (unit/sec)	(*2)
	1	Enable max. velocity limit	

(*2) The parameter is effective after the function is enable. The modification in another period cannot be effective immediately.

The parameters are mainly related to the velocity limit protection. To enable the maximum velocity protection of axis, the limit velocity (SubIndex 0) shall be set first, and then the SubIndex 1 shall be set to 1 (Enable). To change the enabled limit velocity, the function shall be closed first by setting the SubIndex 1 to 0. After the new limit velocity is set, the function can be enabled by the SubIndex 1 to 1.

1.3.2.3. Acceleration Limit Protection

Param. Num	Sub Index	Description	Remark
0x12	0	Maximum acceleration limit (unit/sec^2)	(*2)
	1	Enable max. acceleration limit	

(*2) The parameter is effective after the function is enable. The modification in another period cannot be effective immediately.

The parameters are mainly related to the acceleration limit protection. To enable the maximum acceleration protection of axis, the limit acceleration (SubIndex 0) shall be set first, and then the SubIndex 1 shall be set to 1 (Enable). To change the enabled limit acceleration, the function shall be

closed first by setting the SubIndex 1 to 0. After the new limit acceleration is set, the function can be enabled by the SubIndex 1 to 1. Note: the deceleration used in the function NMC_AxisStop() to enable the emergency stop during the axis is in motion is not limited to the maximum acceleration set by this protection function (SubIndex 1).

1.3.3. Axis Enable and State

Before enabling the axis motion, NMC_AxisEnable() shall be called to transfer the motor to the excitation state. Since this function is a non-blocked, the successful return cannot indicate that the motor has transferred to the excitation state. NMC_AxisGetState() can be called to get the current axis state. If the axis state transfers to 「 NMC_AXIS_STATE_STAND_STILL 」, it indicates that the motor has transferred to the excitation state successfully.

To check if the motor has transferred to the excitation state, not only the axis state but also the axis status read with NMC_AxisGetStatus() can be used. When the bit 6 of Enable (ENA, bit6) becomes 1, the motor has transferred to the excitation state.

Before NMC_AxisEnable() is called, NMC_AxisGetState() can be called to check the motor drive alarm. In case of a motor drive alarm, the axis state will transfer to 「NMC_AXIS_STATE_ERROR」. Also, NMC_AxisGetStatus() can also be called for the purpose. If the ALM (bit 1) and ERR (bit 7) of the axis status become 1, there is the motor drive alarm, and NMC_AxisResetDriveAlm() can be called to reset the drive alarm. Please note that not all drive alarms can be reset with this function. Some drive alarms must be reset by shutting down and opening up the drive. After the drive alarm is reset, NMC_AxisResetState() shall be called to transfer the axis state from 「NMC_AXIS_STATE_ERROR」 to 「NMC_AXIS_STATE_DISABLE」, and then the ALM (bit 1) and ERR (bit 7) of the axis status will become 0. In case there is drive alarm and the NMC_AxisResetState() is called, the function will reset the drive alarm first and transfer the axis state to 「NMC_AXIS_STATE_DISABLE」.

When the axis is excited successfully, the functions related to axis motion can be called. When the motion is completed or there is any accident, NMC_AxisDisable() can be called directly to release the excitation state. After the excitation state is released successfully, the axis state will transfer to 「NMC_AXIS_STATE_DISABLE」, and the ENA (bit 6) will be reset to 0.

1.3.4. Axis Velocity Percentage

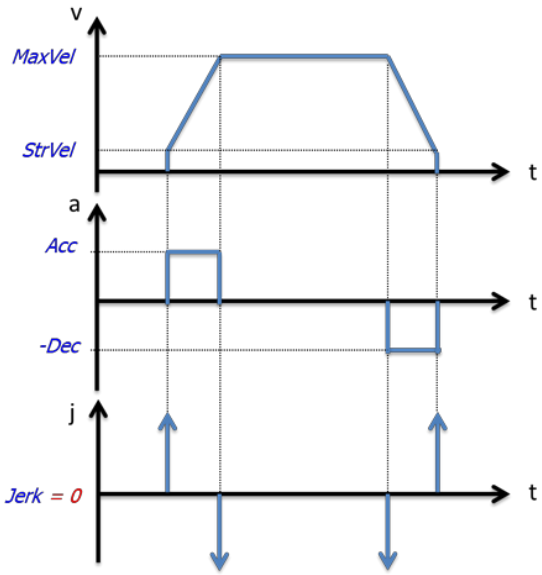
The axis motions, including the point-to-point motion and the JOG motion, will take the configurations in the axis parameter 0x32 as the target velocity. In some application scenarios, usually in the manual operation mode, the target velocity needs to be reduced to confirm the position or the operation process. So, NMC_AxisSetVelRatio() can be used to set the velocity percentage. After the velocity percentage is set successfully, the motion control module will take the target velocity by multiplying the configurations in the axis parameter 0x32(Max. velocity) by such percentage. In addition, NMC_AxisGetVelRatio() can be used to get the current velocity percentage of axis motion.

The function can not only be called before the axis is not in motion but also supports the online calling. That is, when the axis is executing the point-to-point motion or the JOG motion, the function can be called to change the current target velocity in accordance with the set velocity percentage. It will impact the following point-to-point motion or JOG motion commands, and the target velocity will be calculated based on the set velocity percentage.

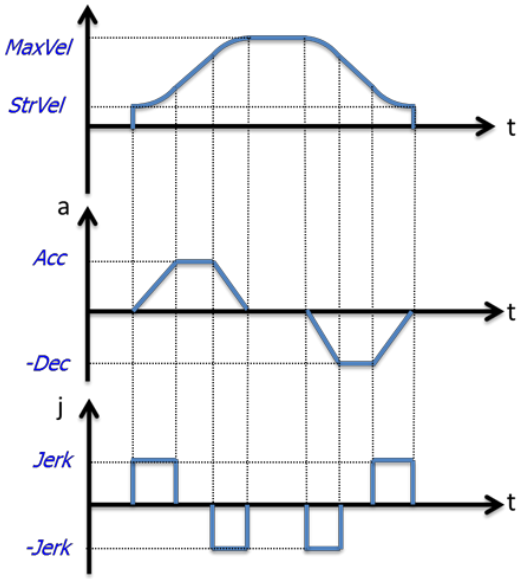
The value of the velocity percentage shall be set within the ranges from 0 (included) to 1 (included). For the value over the range, NMC_AxisSetVelRatio() will return an error code. If the axis is executing the point-to-point or JOG motion and the velocity percentage is set to 0, the axis will decrease the velocity to 0, and the CSTP (bit 9) of the axis status will become 1. Although the axis is still in this case, the point-to-point or JOG motion is still executing, so that the OP (bit 13) of the axis status will be kept 1. If the velocity percentage is set to more than 0 again, the axis will automatically move to the target position (point-to-point motion) or reach to the target speed (jog motion).

1.3.5. Axis Point-to-Point Motion

The axis point-to-point (PTP) motion is one of the motions executed by an axis frequently. NMC_AxisPtp() can be called to set the target position or the relative displacement. Then the motion control module will perform the velocity curve plan in accordance with the axis parameters and move the axis to the specified position as the below figure:



Trapezoidal Velocity Curve



S-shape Velocity Curve

The axis parameters related to the point-to-point motion:

Param. Num	Sub Index	Description	Remark
0x28	0	Base velocity (unit/sec)	
0x31	0	Profile type	
0x32	0	Max. velocity (unit/sec)	
0x33	0	Acceleration (unit/sec^2)	
0x34	0	Mechanical revolution	
0x35	0	Jerk (unit/sec^3)	

These parameters are described as followed:

- 0x28: Base velocity (unit/sec)
  To enable the point-to-point motion of axis from still, the axis will increase the velocity from the initial velocity 0 to the target velocity generally. However, the initial takeoff velocity may often be set for the applications of stepping motor. To fulfill the requirements, the parameter 0x28 can be set to the takeoff velocity. Then the initial velocity will be planned in accordance with the configurations in the parameter 0x28 when enabling the point-to-point or JOG motion. In addition to the initial velocity, the final velocity will also be set in accordance with the configurations in the parameter 0x28.

- 0x30: Absolute or relative programming
  For the point-to-point function NMC_AxisPtp(), the input parameter TargetPos can refer to the absolute position coordinate, or the relative position coordinate (i.e. the relative displacement). If the parameter 0x30 is set to 1, the input parameter TargetPos refers to the relative displacement. If the parameter 0x30 is set to 0, the input parameter TargetPos refers to the absolute position coordinate.

- 0x31: Profile type

When NMC_AxisPtp() is called, the motion control module of device will plan a velocity curve, calculate the command position of axis with each communication period in accordance with the velocity curve, and move the axis to the specified target position. Generally, the velocity curve can be created in two type: the trapezoidal velocity curve and the S-shape velocity curve. If the 0x31 is set to 0, the velocity curve is trapezoidal and the acceleration is discontinuous. If the 0x31 is set to 1, the velocity curve is S-shape and the acceleration is continuous. Under conditions of the same target velocity and acceleration, the motion with the S-shape velocity curve needs more time to reach the target position or the target velocity. However, it may reduce more vibrations since the acceleration is continuous. By contrast, the motion with the trapezoidal velocity curve needs less time to reach the target position or the target velocity. However, it may cause more vibrations since the acceleration is discontinuous.

- 0x32: Max. velocity (unit/sec)
  The parameter refers to the target velocity used in the point-to-point or JOG motion as the MaxVel in the above figures.

- 0x33: Acceleration (unit/sec^2)
  The parameter refers to the acceleration used in the point-to-point or JOG motion as the Acc in the above figures.

- 0x34: Deceleration (unit/sec^2)
  The parameter refers to the deceleration used in the point-to-point motion from the target velocity to the final velocity or the Halt motion as the Dec in the above figures.

- 0x35: Jerk (unit/sec^3)
  The parameter refers to the Jerk in the motion with the S-shape velocity curve as the slope in the acceleration curve figure (a-t).

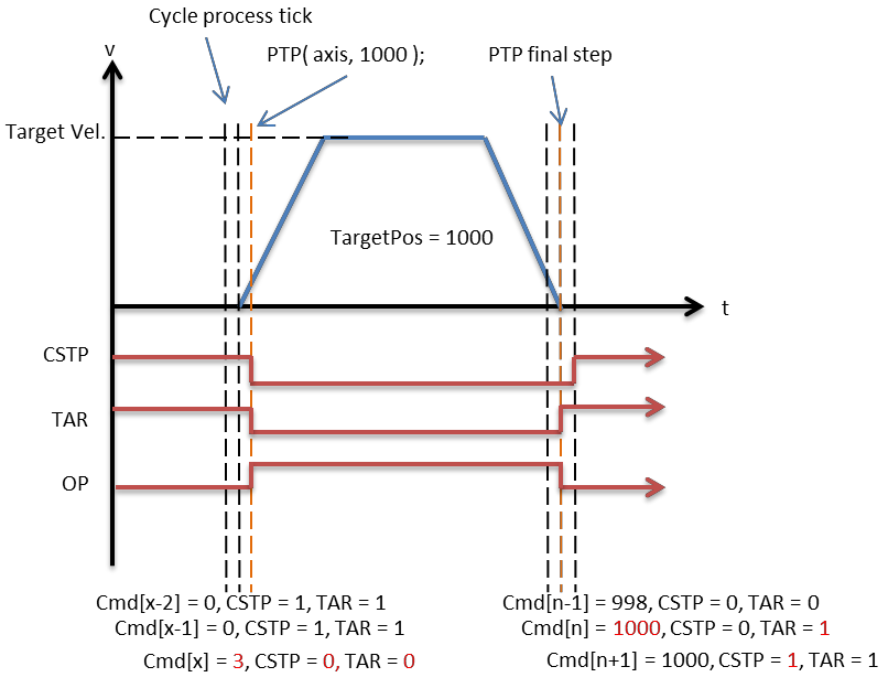
- 0x36: Buffer mode
  In addition to the single axis motion, a series of axis motion commands can also be input and stored in the motion queue. The parameter 0x36 specifies the connection between the new motion commands and the previous one. The operations of the motion queue and the parameter 0x36 will be described in the section of axis motion queue in detail. The following will describe the point-to-point motion separately.

After NMC_AxisPtp() is called successfully, the axis state will transfer to 「NMC_AXIS_STATE_DISCRETE_MOTION」. Please refer to the below table for the state transitions of the axis state before and after the NMC_AxisPtp() is called.

Motion Command / Current State	NMC_AxisPtp()
NMC_AXIS_STATE_DISABLE	Inhibited. The function will return error.
NMC_AXIS_STATE_STAND_STILL	The axis state will transfer to NMC_AXIS_STATE_DISCRETE_MOTION.
NMC_AXIS_STATE_HOMING	Inhibited. The function will return error. The axis in the homing motion will not be impacted.
NMC_AXIS_STATE_DISCRETE_MOTION	Store the input command into the motion queue or execute such input command immediately depended on the parameter 0x36 (buffer mode).
NMC_AXIS_STATE_CONTINUOUS_MOTION	Store the input command into the motion queue or execute such input command immediately depended on

	the parameter 0x36 (buffer mode). If the buffer mode is blending, the command is Inhibited and the function will return error.
NMC_AXIS_STATE_STOPPING	Inhibited. The function will return error.
NMC_AXIS_STATE_STOPPED	Inhibited. The function will return error.
NMC_AXIS_STATE_WAIT_SYNC	Store the input point-to-point command into the motion queue depended on the parameter 0x36 (buffer mode): 1. Abort: clear the motion queue and then store the command. 2. Buffered: store the command 3. Blending: store the command If the buffer mode is blending and the previous motion command stored in the motion queue is JOG motion, the function will return error.
NMC_AXIS_STATE_ERROR	Inhibited. The function will return error.

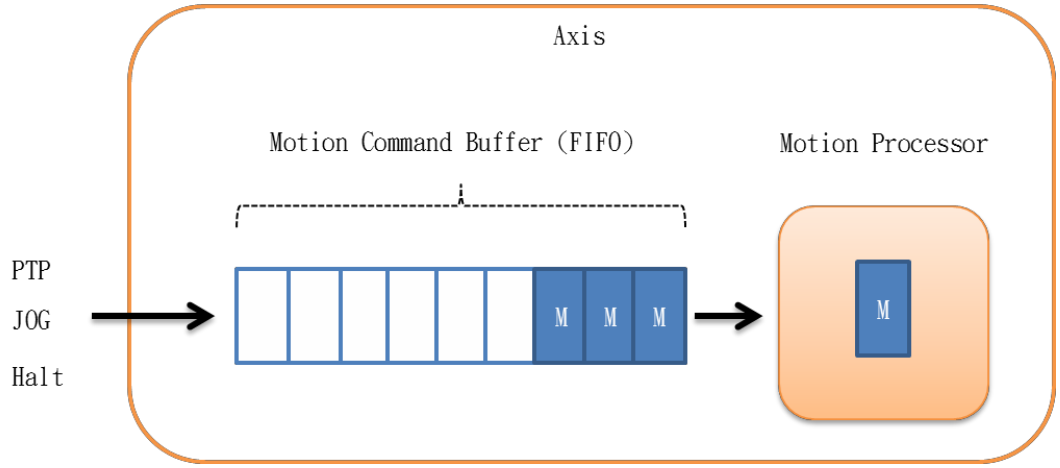
After NMC_AxisPtp() is called and the point-to-point motion has executed, NMC_AxisGetStatus() can be called to get the axis status during the axis is in motion. The bit 8~13 of the axis status are related to the motion state as the below figure:



Motion Status of the Axis in the Point-to-Point Motion

1.3.6. Axis Motion Queue

There is an independence motion queue (motion command buffer) for each axis. When an axis is in the point-to-point or the JOG motion, the new motion command will trigger the axis behavior in accordance with the parameter 0x36 (buffer mode).



Axis Motion Queue

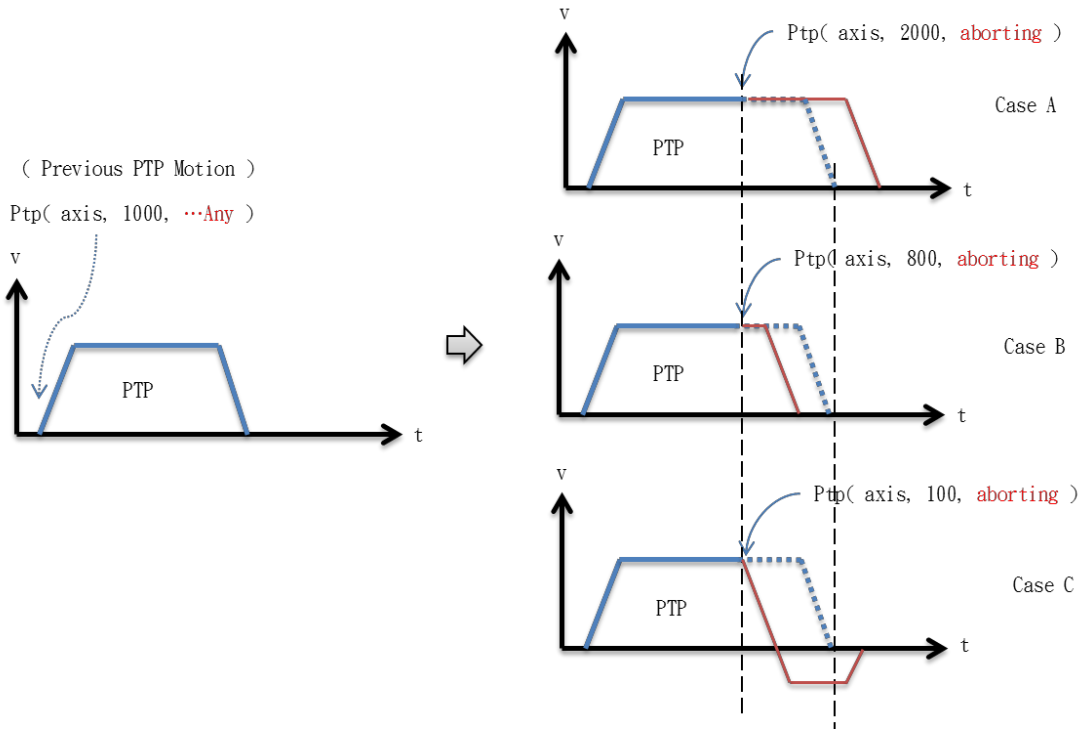
- There is an independent motion queue for each axis. The axis processor will take the motion command from the motion queue and execute such command.
- Buffer size is 32.
- The motions set via NMC_AxisPtp(), NMC_AxisJog() or NMC_AxisHalt() can be stored in the motion queue.
- NMC_AxisGetMotionBuffSpace() can be called to check the residual capacity of the motion queue.
- NMC_AxisHalt() will stop the current motion but not clear the motion queue. The motion commands stored in the motion queue will be taken and executed successively.
- NMC_AxisStop() will stop the current motion and clear the motion queue.
- NMC_AxisDisable() will clear the motion queue.

These usage of the axis parameter 0x36 (buffer mode) is described as followed:

- If the axis state is 「NMC_AXIS_STATE_STAND_STILL」 and an axis motion function is called, the motion will be enabled immediately regardless of the configurations of 0x36 (buffer mode).
- If the axis is in motion and a new motion command is called, the motion command may be connected to the current motion or stored into the motion queue and connected to the previous motion command depended on the configurations in the parameter 0x36. The details are described as follows:

1.3.6.1. Buffer Mode: Aborting (0x36 = 0)

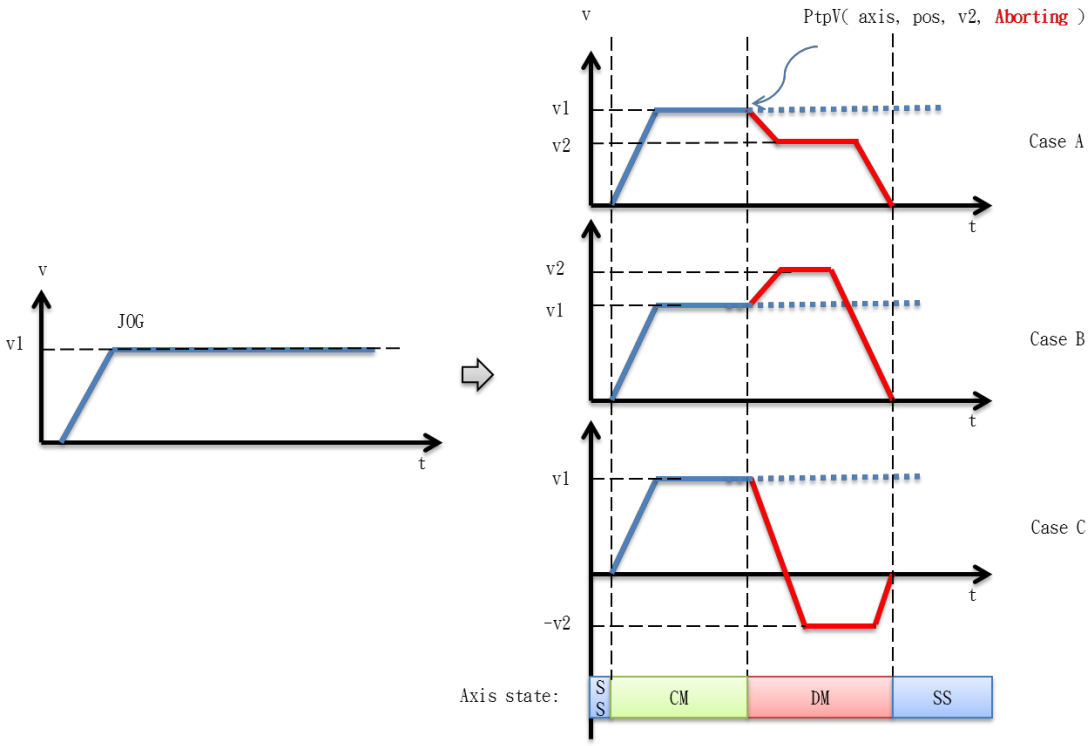
All motion commands stored in the axis motion queue will be cleared, and the motion executing will be aborted. The axis will execute the new motion command. An example is described as follows:



3 Cases for the PTP Motion Aborted by another PTP Motion

In the example in the above figure, the axis enable the point-to-point motion with target position 1000. When the axis is in the point-to-point motion, a new point-to-point motion is commanded and the 0x36 is 0 (Aborting). The axis may perform one of the three cases depended on the new target position. Among these cases, the axis will perform the reverse motion if the new target position coordinate is less than the original one.

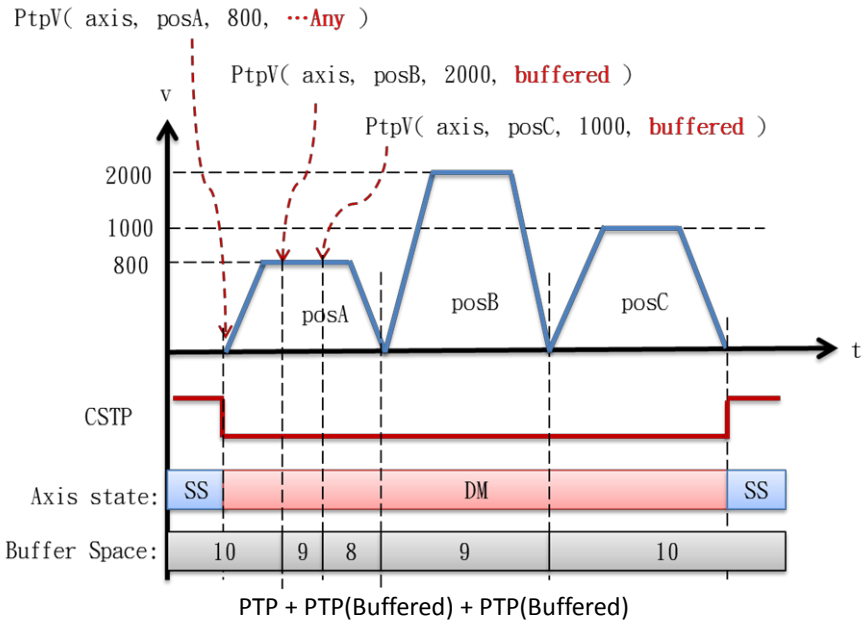
Another example is shown as below figure. The axis is in the JOG motion with the target motion v1. The point-to-point motion with target motion v2 is commanded and the 0x36 is 0 (Aborting). If the target position and the relative direction of the current position are reverse to the current motion direction, the axis will perform the reverse motion as the case C in the figure. If the target position and the relative direction of the current position is the same as the current motion direction, and the relative distance is enough for the velocity acceleration or decrease, the axis will modify the velocity to the new target velocity v2 and move to the target position as the case A and B in the figure.



3 Cases for the JOG Motion Aborted by another PTP Motion

1.3.6.2. Buffer Mode: Buffered (0x36 = 1)

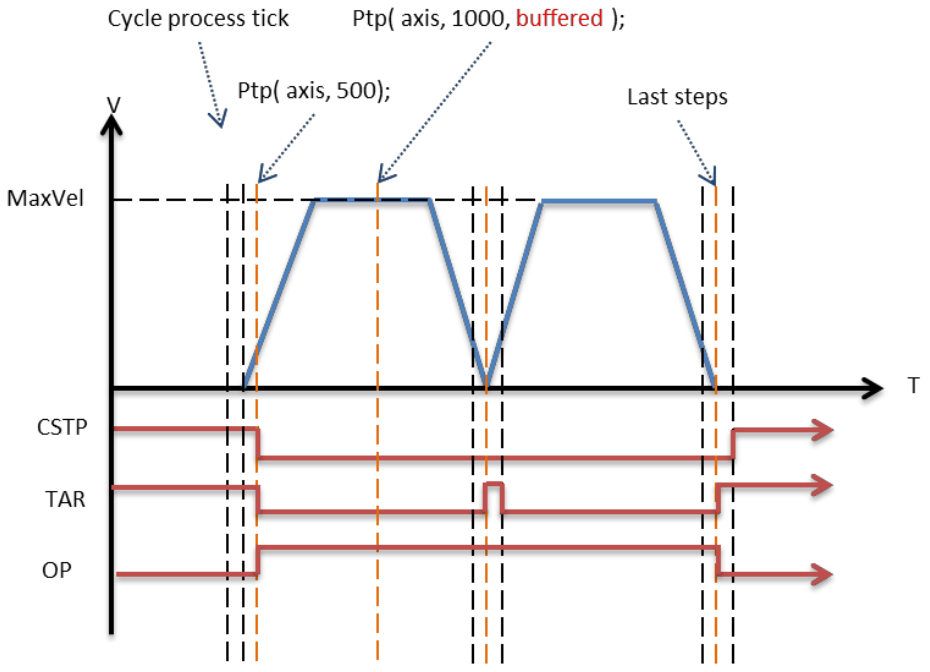
The new motion command will be stored in the motion queue and executed after the previous motion command is completed. The so-called motion command completion means the axis reaches the target position for the point-to-point motion, or the axis reaches the target velocity for the JOG motion.



PTP + PTP(Buffered) + PTP(Buffered)

For example, as the above figure, the axis is in the point-to-point motion and two successive point-to-point motion commands with different target velocities are set. Since the 0x36 is set to buffered, these two point-to-point motion commands will be stored in to the axis motion queue, and they will be automatically executed one-by-one after the axis reaches the target position of the

previous point-to-point motion. Precisely, the criteria to check if the axis reaches the target position is that the TAR (bit 8) of the axis status is 1 as the below figure.

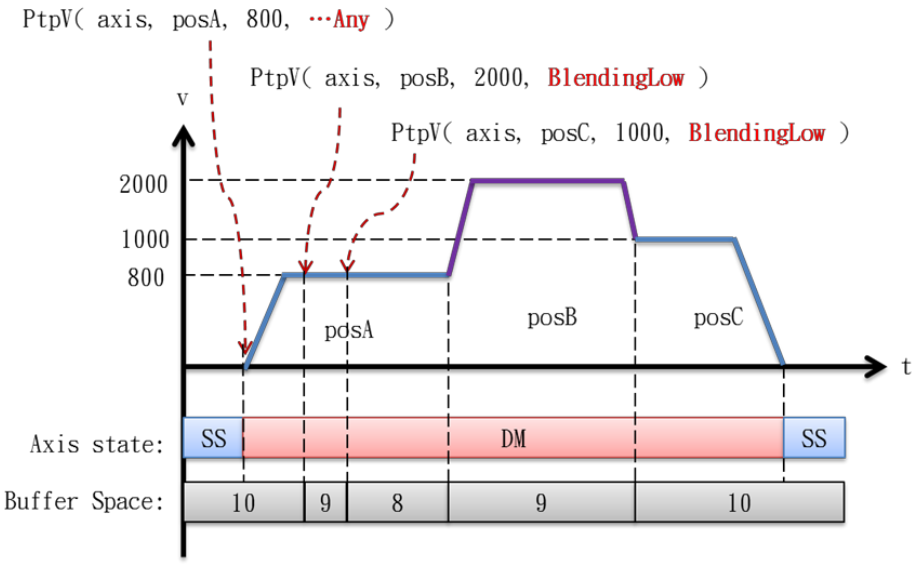


Motion Status for Buffered PTP

In addition to the Abort and Buffered, another behavior is Blending. It aims to specify the velocity to connect the new motion command the previous one. There are four methods to specify the velocity, including the BlendingLow, BlendingHigh, BlendingPrevious and BlendingNext. These methods are described as follows:

1.3.6.3. Buffer Mode: BlendingLow (0x36 = 2)

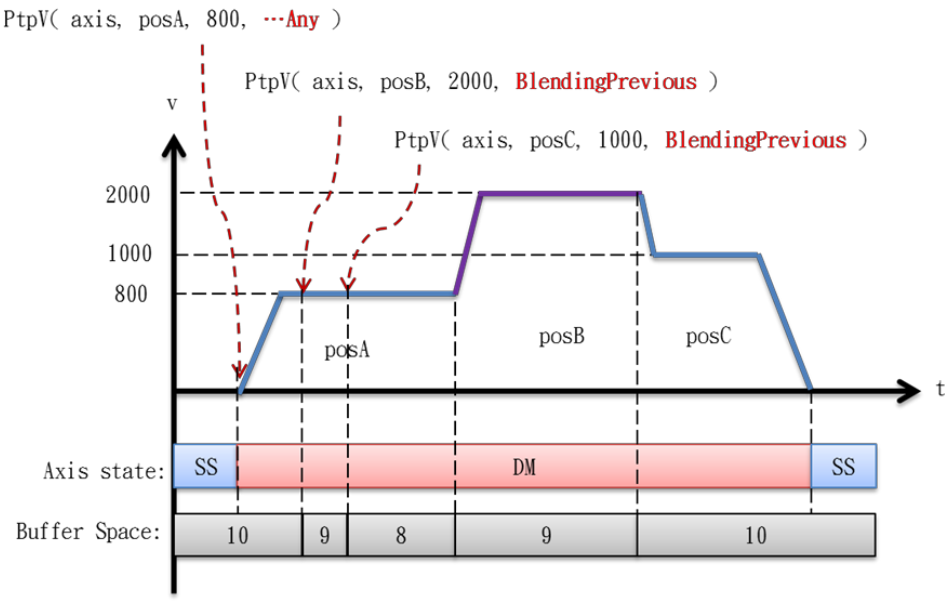
The new motion command will be stored in the motion queue and connected to the previous motion command with a lower target velocity. Based on the below figure, the point-to-point motion with the target velocity 800, target position posA and final velocity 0 is enabled from the still state at the beginning. During the axis is in motion, another point-to-point motion command with the target velocity 2000 is set with the BlendingLow method to connection the previous point-to-point motion. Therefore, the final velocity of the original point-to-point motion is changed from 0 to 800 (2000 > 800). Thus, the velocity curve is generated as the figure 1-3-9. The axis will perform the first point-to-point motion to the target position posA at the target velocity 800 and the final velocity still is 800. After reaching the target position posA, the motion control module will automatically enable the second point-to-point motion with target position posB, and then increase the velocity from the initial velocity 800 (equal to the final velocity of the previous point-to-point motion) to the target velocity 2000. If there is no any successive motion command stored in the motion queue, the axis will reach the target position posB at the final velocity 0. However, there is the third point-to-point motion command with the target velocity 1000 is set with the BlendingLow method to connection the previous point-to-point motion. Therefore, the axis shall reach the target position posB at the final velocity 1000 (2000>1000), and then the motion control module will enable the third point-to-point motion to move the axis to the target position posC.



PTP + PTP(BlendingLow) + PTP(BlendingLow)

1.3.6.4. Buffer Mode: BlendingPrevious (0x36 = 3)

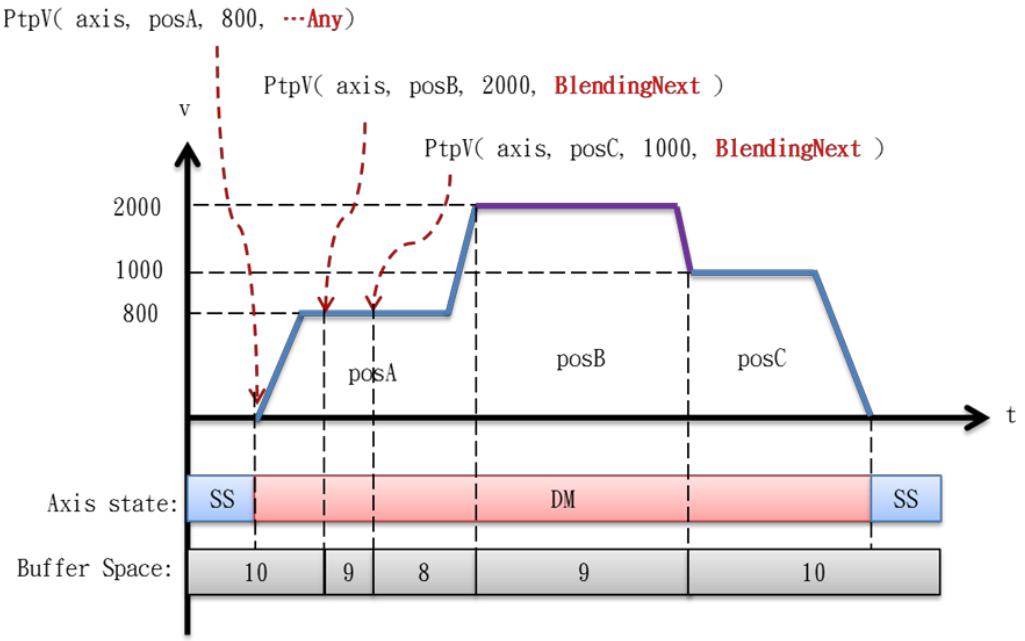
The new motion command will be stored in the motion queue and connected to the previous motion command with the target velocity of the previous motion command. Based on the below figure, each latter point-to-point motion connects to the previous point-to-point with the target velocity of the previous motion command. Therefore, the final velocity is equal to the target velocity of the previous point-to-point motion command when the axis reaches the target position posA and posB.



PTP + PTP(BlendingPrevious) + PTP(BlendingPrevious)

1.3.6.5. Buffer Mode: BlendingNext (0x36 = 4)

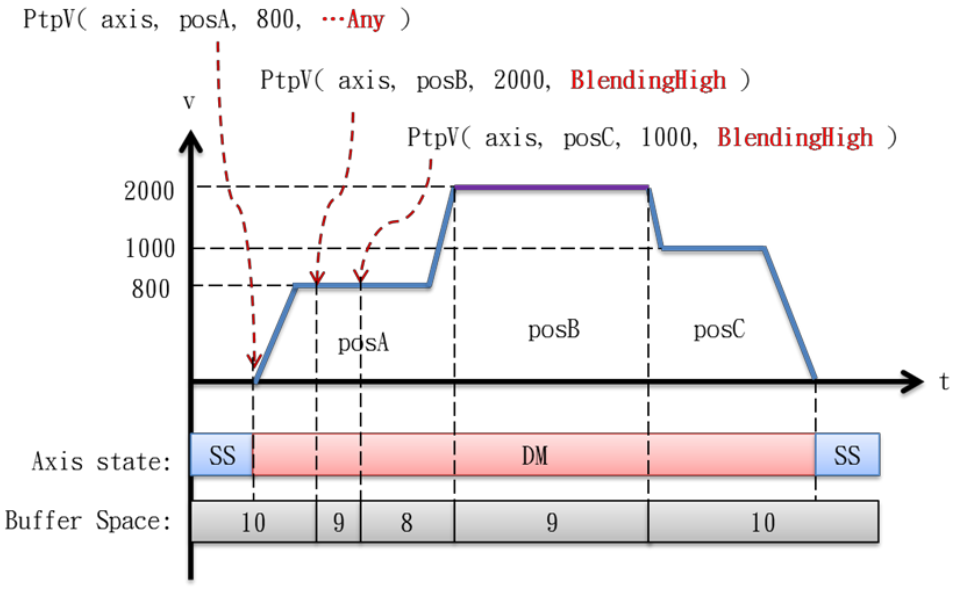
The new motion command will be stored in the motion queue and connected to the previous motion command with the target velocity of the new motion command. Based on the below figure, each latter point-to-point motion connects to the previous point-to-point with the target velocity of the latter motion command. Therefore, the final velocity is equal to the target velocity of the latter point-to-point motion command when the axis reaches the target position posA and posB.



PTP + PTP(BlendingNext) + PTP(BlendingNext)

1.3.6.6. Buffer Mode: BlendingHigh (0x36 = 5)

The new motion command will be stored in the motion queue and connected to the previous motion command with the higher target velocity. Based on the below figure, since the target velocity of the second point-to-point motion is higher than this of the first point-to-point motion, the axis will increase the final velocity to 2000 when it reaches to the target position posA, and then it will move to the target position posB at the initial velocity 2000. Since the target velocity of the third point-to-point motion is also lower than 2000, the axis will reach the target position posB at the final velocity 2000.



PTP + PTP(BlendingHigh) + PTP(BlendingHigh)

1.3.7. Axis JOG Motion

The JOG motion is also one of the motions executed by an axis frequently. After NMC_AxisJog() is called, the motion control module will perform the velocity curve plan in accordance with the related axis parameters, increase/decrease the axis velocity to the target velocity, and then execute the axis motion at the target velocity continuously. Therefore, the axis state is 「NMC_AXIS_STATE_CONTINUOUS_MOTION」 when the axis is in the motion.

The axis parameters related to the JOG motion:

Param. Num	Sub Index	Description	Remark
0x28	0	Base velocity (unit/sec)	
0x31	0	Profile type	
0x32	0	Max. velocity (unit/sec)	
0x33	0	Acceleration (unit/sec^2)	
0x35	0	Jerk (unit/sec^3)	

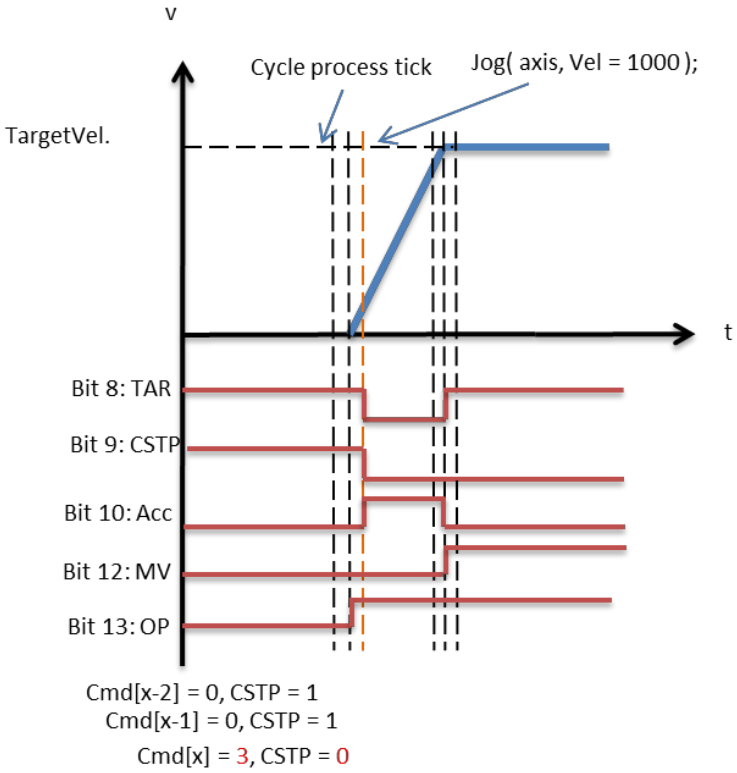
The axis parameters related to the JOG motion include the target velocity (0x32), acceleration(0x33), velocity curve type (0x31) and Jerk(0x35). To set the target velocity different from this set in 0x32, the desired target velocity can also be set with the pointer PMaxVel, and the 0x32 will also be configured accordingly.

After the JOG motion is enabled successfully, the axis state will transfer to 「NMC_AXIS_STATE_CONTINUOUS_MOTION」. Please refer to the below table for the state transitions of the axis state before and after the JOG motion is enabled.

Motion Command / Current State	NMC_AxisJog()
NMC_AXIS_STATE_DISABLE	Inhibited. The function will return error.
NMC_AXIS_STATE_STAND_STILL	The axis state will transfer to NMC_AXIS_STATE_DISCRETE_MOTION.
NMC_AXIS_STATE_HOMING	Inhibited. The function will return error. The axis in the homing motion will not be impacted.
NMC_AXIS_STATE_DISCRETE_MOTION	Store the input command into the motion queue or execute such input command immediately depended on the parameter 0x36 (buffer mode). If the buffer mode is aborting, the axis state will transfer to NMC_AXIS_STATE_CONTINUOUS_MOTION.
NMC_AXIS_STATE_CONTINUOUS_MOTION	Store the input command into the motion queue or execute such input command immediately depended on the parameter 0x36 (buffer mode). If the buffer mode is blending, the command is Inhibited and the function will return error.
NMC_AXIS_STATE_STOPPING	Inhibited. The function will return error.
NMC_AXIS_STATE_STOPPED	Inhibited. The function will return error.
NMC_AXIS_STATE_WAIT_SYNC	Store the input point-to-point command into the motion queue depended on the parameter 0x36 (buffer mode): 1. Abort: clear the motion queue and then store the command. 2. Buffered: store the command 3. Blending: store the command

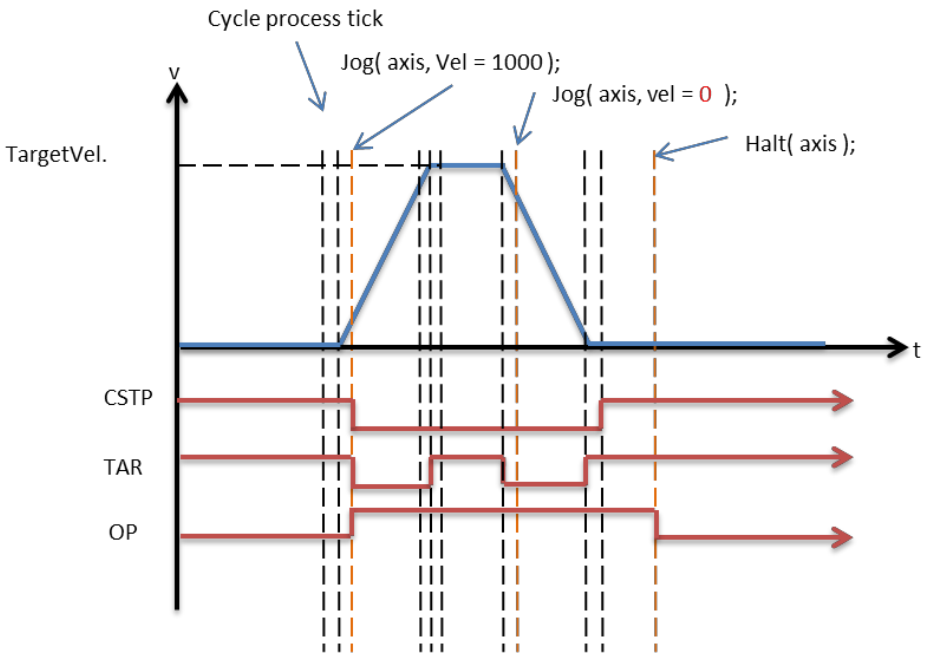
NMC_AXIS_STATE_ERROR	If the buffer mode is blending and the previous motion command stored in the motion queue is JOG motion, the function will return error. Inhibited. The function will return error.

If an axis is in the JOG motion and reaches the target velocity, the TAR (bit 8) of the axis status becomes 1. It indicates that the target velocity is reached. In addition, the MV (bit 12) is kept 1 since the axis is still in motion after reaching the target velocity.



Timing Diagram of Axis Status in JOG Motion

After the JOG motion is enabled successfully, the axis will decrease the velocity to 0 if the JOG velocity is reset to 0 (Aborting) or the velocity percentage is set to 0 with NMC_AxisSetVelRatio(). The related motion status is shown as the below figure:



Motion Status if the Velocity Percentage is set to 0 after the JOG Motion is enabled

If the velocity percentage is set to 0, the target velocity will be reset to 0 so that the axis will decrease the velocity to 0. After the velocity is decreased to 0, the TAR (bit 8) of axis status will become 1 since the target velocity is reached. In addition, the CSTP (bit 9) of axis status will also become 1 since the axis is still. Although the axis velocity is decreased to 0, the axis will start a motion after the velocity percentage is set to a value greater than 0 since the JOG motion command is still effective. Therefore, the OP (bit 13) of axis status is kept 1. If NMC_AxisHalt() is called to stop the motion, the axis will exit the JOG motion and the OP (bit 13) of axis status will become 0. Please refer to the following section for the behaviors related to the motion stop.

1.3.8. Axis Motion Stop

NMC_AxisHalt() or NMC_AxisStop() can be called to stop an axis motion for an axis in motion. The difference between the two functions is mainly the NMC_AxisHalt() is called under the normal cases. The deceleration and velocity curve are based on the configurations in the axis parameters 0x34 (Deceleration) and 0x31 (Profile type). These parameters are the same as those used in the point-to-point and JOG motions. NMC_AxisStop() can be called to stop an axis motion in the emergency. The deceleration and velocity curve used in the deceleration process are based on the other set of parameters. These parameters are described as follows:

Param. Num	Sub Index	Description	Remark
0x20	0	Profile type for AxisStop command	
0x21	0	Deceleration for AxisStop command (unit/sec^2)	
0x22	0	Jerk for AxisStop command (unit/sec^3)	

- 0x20: Profile type for AxisStop command
  The parameter refers to the velocity curve type for the axis deceleration after NMC_AxisStop() is called.

- 0x21: Deceleration for AxisStop command (unit/sec^2)
  The parameter refers to the deceleration for the axis deceleration after NMC_AxisStop() is called. The configuration is not limited to the acceleration limit of the software limit protection (0x12, SubIndex = 0). Generally, the configuration in 0x21 may be greater than this in 0x34 that the axis can decrease the velocity to 0 rapidly in the emergency.

- 0x22: Jerk for AxisStop command (unit/sec^3)
  If the parameter 0x20 is set to the S-shape velocity curve, the parameter can specify the Jerk and is distinguished from the parameter 0x35.

The usages of NMC_AxisHalt() and NMC_AxisStop() are respectively described as follows:

- NMC_AxisHalt():
  When an axis is in the point-to-point motion, JOG motion or homing motion, the function can be called to stop the axis motion normally. After the axis is stopped, the axis state will transfer to 「NMC_AXIS_STATE_STAND_STILL」. This function is regarded as one of the axis motions in normal cases. Please refer to the below table for the state transitions of the axis state before and after the function is called.

Motion Command / Current State	NMC_AxisHalt()
NMC_AXIS_STATE_DISABLE	Inhibited. The function will return error.
NMC_AXIS_STATE_STAND_STILL	No state transition
NMC_AXIS_STATE_HOMING	Transfer to STAND_STILL after the motion completion.
NMC_AXIS_STATE_DISCRETE_MOTION	Store the input command into the motion queue or execute such input command immediately depended on the parameter 0x36 (buffer mode). Transfer to STAND_STILL after the motion completion.
NMC_AXIS_STATE_CONTINUOUS_MOTION	Store the input command into the motion queue or execute such input command immediately depended on the parameter 0x36 (buffer mode). Transfer to STAND_STILL after the motion completion.
NMC_AXIS_STATE_STOPPING	Inhibited. The function will return error.
NMC_AXIS_STATE_STOPPED	Inhibited. The function will return error.
NMC_AXIS_STATE_ERROR	Inhibited. The function will return error.

- NMC_AxisStop()

  When an axis is in the point-to-point motion, JOG motion or homing motion, the function can be called to stop the axis motion emergently. After the function is called, all motion commands stored in the motion queue will be cleared, and the axis state will transfer to 「NMC_AXIS_STATE_STOPPING」. After the axis is stopped, the axis state will transfer to 「NMC_AXIS_STATE_STOPPED」. After the abnormal situation is resolved, NMC_AxisResetState() can be called to reset the axis state. After the function is called successfully, the axis state will transfer to 「NMC_AXIS_STATE_STAND_STILL」 if it is in the excitation state, and it will transfer to 「NMC_AXIS_STATE_DISABLE」 if it is in the nonexcitation state. Please refer to the below table for the state transitions of the axis state before and after the function is called.

Motion Command / Current State	NMC_AxisStop()
NMC_AXIS_STATE_DISABLE	No state transition
NMC_AXIS_STATE_STAND_STILL	Transfer to STAND_STOPPED after the motion completion.
NMC_AXIS_STATE_HOMING	Transfer to STAND_STOPPED after the motion completion.
NMC_AXIS_STATE_DISCRETE_MOTION	Transfer to STAND_STOPPED after the motion completion.
NMC_AXIS_STATE_CONTINUOUS_MOTION	Transfer to STAND_STOPPED after the motion completion.
NMC_AXIS_STATE_STOPPING	No state transition
NMC_AXIS_STATE_STOPPED	No state transition
NMC_AXIS_STATE_ERROR	No state transition

The two functions can be used to stop the motion for an axis only so that the axis index shall be input. NMC_AxisHaltAll() and NMC_AxisStopAll() can be called to stop all axes in the system.

1.3.9. Change Velocity in Motion

When an axis is in motion, the target velocity and acceleration/deceleration can be changed by calling some functions. The section will describe these functions and related limits.

- NMC_AxisVelOverride()

The function can be called to change the target velocity when the axis is in the point-to-point or JOG motion only. Otherwise, the function will return the error code. The input parameter of the function is an absolute value of target velocity which is not impacted by the velocity percentage and will not change the configuration in the axis parameter 0x32 (Max. velocity). For example, if an axis is in the JOG motion and after the function is called successfully, the target velocity will be changed to the input parameter TargetVel regardless of the velocity percentage, but the axis parameter 0x32 (Max. velocity) is kept the original configurations. Since the JOG motion aims to execute an axis motion at the input target velocity, the axis must execute the motion at the target velocity continuously after NMC_AxisVelOverride() is called successfully. If an axis is in the point-to-point motion and it is decreasing the velocity (i.e. the DEC (bit 11) of the axis status is 1), there is no any impact on the point-to-point motion after the function is called. If an axis is in the point-to-point motion and it is accelerating the velocity or in motion at the uniform velocity, it will determine if the specified target velocity can be achieved based on the current velocity, the specified target velocity and the residual distance. If the specified target velocity cannot be achieved, it will change the target velocity reasonably based on the required acceleration/deceleration.

- NMC_AxisAccOverride()

The function can be called to change the target acceleration when the axis is in the point-to-point or JOG motion only. Otherwise, the function will return the error code. The input

parameter of the function is an absolute value of target acceleration, and the configuration in the axis parameter 0x33 (Acceleration) will be changed accordingly. When the axis is accelerating to the target velocity (i.e. the ACC (bit 10) of the axis status is 1), there are impacts on the current motion due to the acceleration change after the function is called. If the axis has reached the target velocity (i.e. the MV (bit 12) of the axis status is 1) or is decelerating (i.e. the DEC (bit 11) of the axis status is 1), there is no any impact on the current motion after the function is called.

- NMC_AxisDecOverride()

  The function can be called to change the target deceleration when the axis is in the point-to-point motion or the motion stop (i.e. NMC_AxisHalt() is called) only. Otherwise, the function will return the error code. The input parameter of the function is an absolute value of target deceleration, and the configuration in the axis parameter 0x34 (Deceleration) will be changed accordingly. When the axis is accelerating to the target velocity (i.e. the ACC (bit 10) of the axis status is 1) or has reached the target velocity, there may have impacts on the current motion due to the deceleration change after the function is called. For example, the new deceleration shortens the required acceleration distance after the function is called, so that an axis which cannot reach the target velocity originally can reach the target velocity. If an axis has decreased the velocity to the final velocity (i.e. the DEC (bit 11) of the axis status is 1), there is no any impact on the current motion after the function is called. If an axis is executing the motion stop after NMC_AxisHalt() is called, the function will change the current deceleration and impact the required time to stop.

1.3.10. Axis Homing Motion

Generally, the axis homing may be applied to two application scenarios. The first is that moves the axis to the origin (or a reference position) by any way and then set the coordinate at such position. The second is that select the proper homing mode, move the axis to the origin through the axis homing process, and set the position as the coordinate datum. Then the following motions can be performed. The section will describe the two homing methods respectively.

1.3.10.1.Set Origin by Manual

NMC_AxisSetHomePos() can be called to set the origin. The function can be called to set the current position to a coordinate when the axis state is 「NMC_AXIS_STATE_DISABLE」 or 「NMC_AXIS_STATE_STAND_STILL」. Otherwise, the function will return the error code. After the function is called successfully, both of the actual position and command position will become the specified coordinate, and the motion control module will calculate the position offset of the servo motor and store the offset is the axis parameter 0x08 (Position offset) automatically. For an axis using the absolute encoder (i.e. the axis parameter 0x05 is set to absolute), the system can calculate the current position coordinate of the axis based on the values returned from the encoder and the position offset stored in 0x08 after opening up.

1.3.10.2.Axis Homing Motion

NMC_AxisHomeDrive() can be called to enable the homing motion supported by the drive. The function can be only called when the axis state is 「NMC_AXIS_STATE_DISABLE」. Otherwise, the function will return the error code.

The axis parameters related to the drive homing motion are described as follows:

Param. Num	Sub Index	Description	Remark
0x80	0	Number of home parameters	(*3)(*4)
	1	EtherCAT CiA HOME method	(*5)
	2	EtherCAT CiA HOME velocity search switch	(*5)
	3	EtherCAT CiA HOME velocity search zero	(*5)
	4	EtherCAT CiA HOME acceleration	(*5)

5	EtherCAT CiA HOME offset	(*5)

(*3): The configurable range is depended on the controller.
(*4): Read only
(*5): The configurable range is depended on the controlled device.

The homing motion is depended on the functions supported by the drive. Therefore, before calling the function to perform the homing motion, developers shall read the drive user manual, confirm the required parameters, and supported homing modes which shall be set to the above axis parameters 0x08: 1~ 5).

If the homing motion may use the origin signal or the positive/negative limit signal, developers shall confirm that the home sensor and the positive/negative limit sensors shall be connected to the I/O connector of the drive correctly by getting the RPEL (bit16), RNEL (bit17) and RHOM (bit18) of the axis status and checking the signal changes.

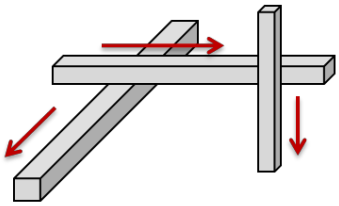
In normal cases, after NMC_AxisHomeDrive() is called successfully, the device will transport the parameters related to the homing motion to the drive and perform the homing process specified automatically. During the motion, the axis state will transfer to 「NMC_AXIS_STATE_HOMING」. When the homing motion is completed, the axis state will transfer to 「NMC_AXIS_STATE_STAND_STILL」, and the RHOM (bit 18) of the axis status will become 1. In case any undesired error occurs during the homing motion, the axis will stop the homing motion, the axis state will transfer to 「NMC_AXIS_STATE_ERROR」, and the axis status will become 1. For example, a developer inputs a homing mode which is not supported by the drive so that the axis state will transfer to 「NMC_AXIS_STATE_ERROR」 after the function is called successfully and the homing motion is executed for a period. In addition, some improper parameters may be configured for the homing motion. For example, too low acceleration may be configured so that the required deceleration distance is too long when the external limit sensor is triggered and then an error occurs. The aixs state will transfer to 「NMC_AXIS_STATE_ERROR」 in case an error occurs during the homing process, NMC_AxisResetState() shall be called to reset the axis state to 「NMC_AXIS_STATE_STAND_STILL」, then other motion commands can be executed.

Since this function is regarded as one of the axis motions, NMC_AxisHalt(), NMC_AxisDisable() or NMC_AxisStop() can be called to stop the homing process during the motion. After calling the function, the axis behaviors are as same as those related functions are called in motion state.

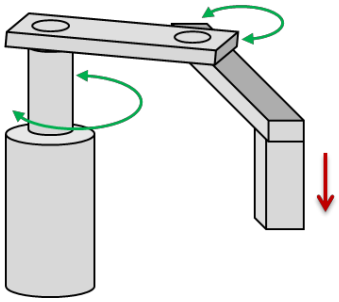
1.4. Group Control

NexMotion can define some axes as a group. A group represents a mechanism or a robot with a specified structural relationship. Now, NexMotion can support these industrial robots as follows:

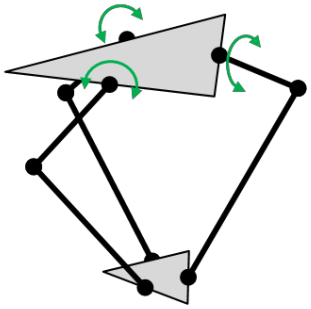
1. Linear Robot
2. SCARA Robot
3. Delta Robot
4. Articulated Robot



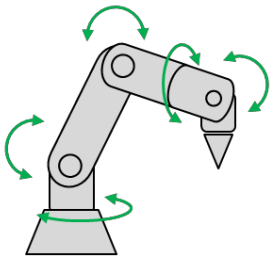
Linear Robot



SCARA Robot



Delta Robot



Articulated Robot

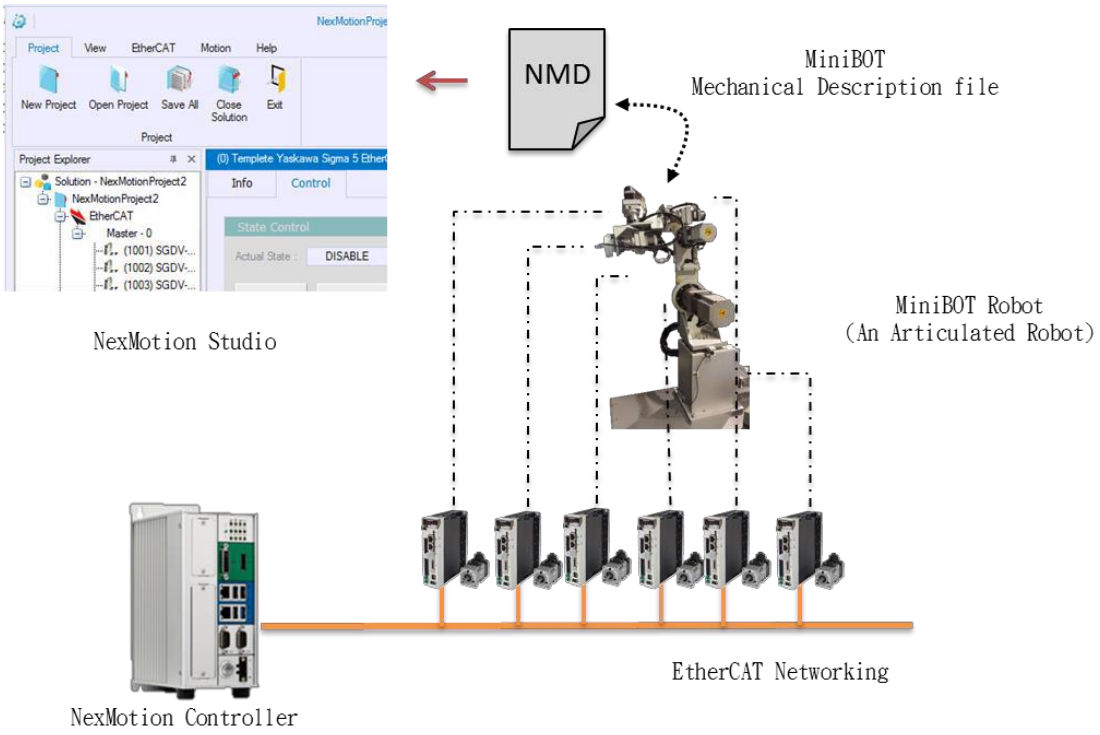
1.4.1. Group Configuration

The mechanism configuration file can be imported with the NexMotion Studio through the steps as follows:

1. Import a Mechanical Description (NMD) file with the NexMotion Studio.
2. Map the servo motor to each axis in the group.
3. Test the group and at the same time the configuration file of the controlled system (NCF file) will be generated.
   Please refer to the user manual of the NexMotion Studio for the detail procedure.

A group can also be defined by manual. First, a similar mechanism configuration profile can be imported with NexMotion Studio as the template. Then the related configurations of the template can be adjust/modified to meet the actual mechanism through the steps as follows:

1. Import a Mechanical Description (NMD) file with the NexMotion Studio as a template.
2. Modify the unit parameters of each axis (Please refer to the unit configuration of axis).
3. Modify the parameters of mechanism kinematics (Please refer to the Mechanism Kinematics Configurations).
4. Map the servo motor to each axis in the group
5. Test the group and generate the configuration file of the controlled system (NCF file).



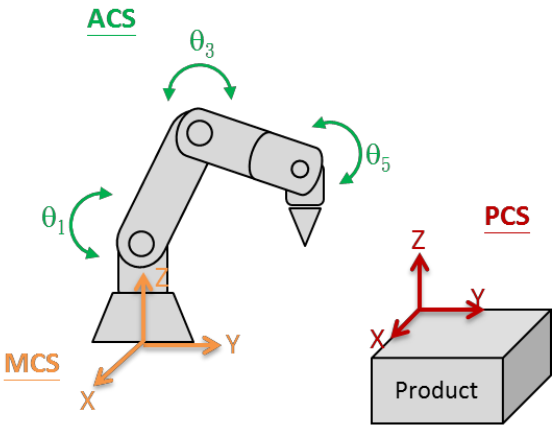
Import a Robot Data with an NMD file

1.4.2. Coordinate System

To control the position of each axis or each tool center point (TCP) in a group (or called a robot), NexMotion supports three coordinate systems to describe the position, including:

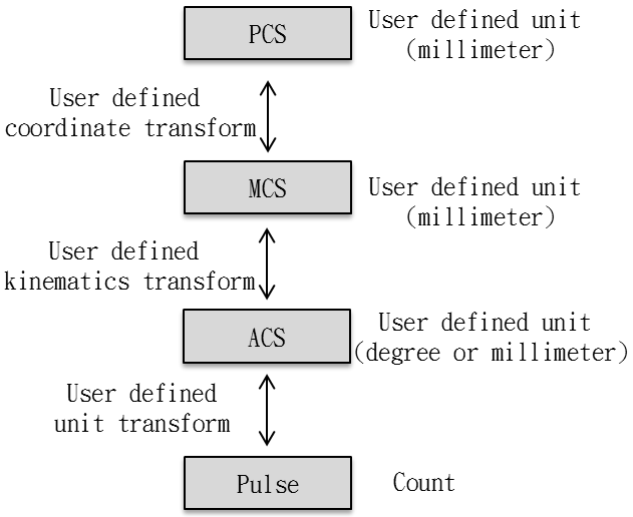
1. Axis Coordinate System (ACS),
2. Machine Coordinate System (MCS) or so-called Geodetic Coordinate System, and
3. Product Coordinate System (PCS).

These coordinate systems are shown as the below figure:



Definitions of ACS, MCS and PCS

NexMotion supports the built-in unit conversion of these coordinate systems. Developers can define the conversions through the parameter configurations. The unit conversions of these coordinate systems are shown as the below figure.



Unit Conversions

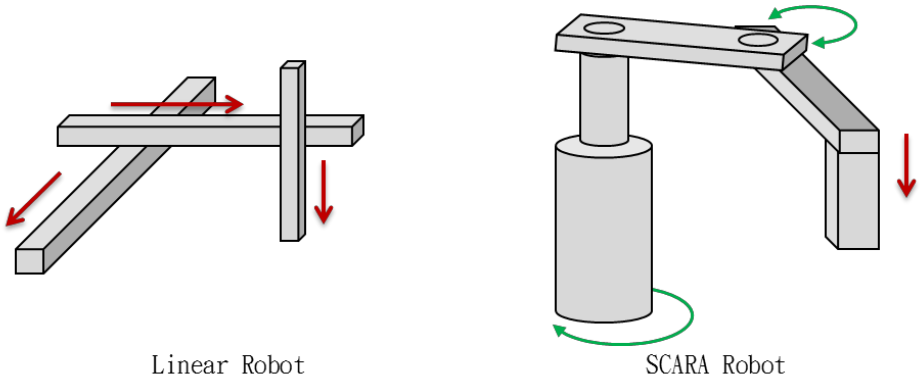
The parameters related to the conversions are described in the following table:

Conversion Type	Parameter Type	Description
Unit conversion	Group axis parameters 0x00~0x06	Set the conversion between the pulse of servo and the physical unit of axis coordinate system (ACS)
Mechanism kinematics conversion	Group parameters 0x00	Set the conversion between the axis coordinate system (ACS) and the machine coordinate system (MCS)
Coordinate system conversion	Group Parameters 0xC0~DF	Set the conversion between the machine coordinate system (MCS) and the product coordinate system (PCS).
TCP coordinate conversion	Group Parameters 0x80~8F	Set the conversion between the tool and the tool center point (TCP).

NexMotion defines the data structure 「Pos_T」 and 「Xyz_T」 to describe the coordinate information. The Pos_T describes the coordinates of ACS, MCS and PCS, and the Xyz_T describes the positions at the x-, y- and z-axis of MCS and PCS.

1.4.2.1. Axis Coordinate System (ACS)

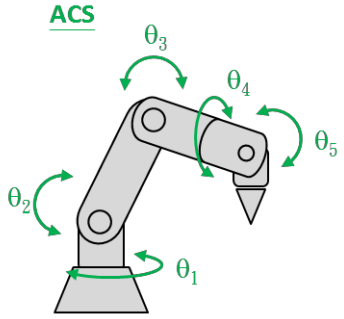
The axis coordinate system (ACS) describes the position of each joint axis in a group mechanism (or called as robot). A robot system may contain various types of joints. Each joint can be regarded as an axis which position can be described through the axis coordinate system (ACS). The common joints can be divided into two types: linear joints and rotary joints. The unit of position can be defined depended on the mechanism type. For example, the physical unit of a linear joint can be defined as the millimeter (mm), and the angle unit of a rotary joint can be defined as the degree.



Robot Joint Types

The device supports the conversion from the pulse count of servo to the physical unit of mechanism. Please refer to the axis unit configuration for the conversion configuration.
A group can contain up to 8 axes. The actual number of axes is determined by the mechanism type.

The ACS description of a 6-axis articulated robot is shown as the below figure.



Name	Unit
θ_1	Degree
θ_2	Degree
θ_3	Degree
θ_4	Degree
θ_5	Degree
θ_6	Degree

Example: The ACS Description of a 6-axis Articulated Robot

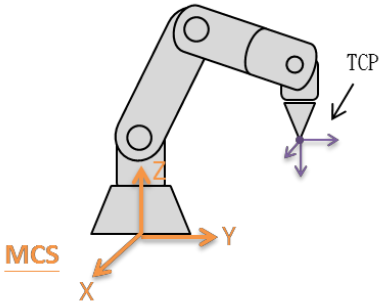
An example shows the axis coordinate system (ACS) described with the data structure 「Pos_T」 as follows:

```
#include "NexMotion.h"
#include "NexMotionError.h"

int main()
{
	Pos_T point;

	// "point" represents as ACS position.
	point.pos[0] = 0.0;  // Degree of joint 1
	point.pos[1] = 90.0;// Degree of Joint 2
	point.pos[2] = 0.0;  // Degree of Joint 3
	point.pos[3] = 0.0;  // Degree of Joint 4
	point.pos[4] = 0.0;  // Degree of Joint 5
	point.pos[5] = 0.0;  // Degree of Joint 6
}
```

The following APIs can be used to get the position coordinate of ACS with the data structure「Pos_T」:
1. NMC_GroupGetCommandPosAcs() : To get the command position of ACS.
2. NMC_GroupGetActualPosAcs() : To get the Actual position of ACS.

1.4.2.2. Machine Coordinate System (MCS)

The machine coordinate system (MCS) describes the position orientations of a tool center point (TCP) relative to the machine origin. The coordinate origin is defined at the installation point of the machine in general. The MCS description of a 6-axis articulated robot is shown as the below figure.



Name	Unit
X	mm
Y	mm
Z	mm
A	Degree
B	Degree
C	Degree

Example: The MCS Description of a 6-axis Articulated Robot

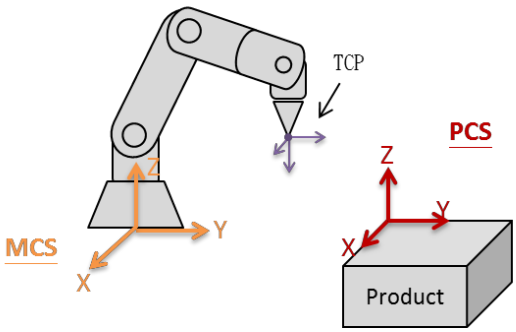
The TCP can be described up to 6 degrees of freedom which representations are X, Y, Z, A, B and C.

- Position: X, Y and Z are the representation of the rectangular coordinate system following the right-hand rule.
- Orientation: A, B and C are the representation of the Euler angle with intrinsic Z-Y-X rotating to Z-, Y- and X-axis, respectively.

The MCS coordinate system is usually used in the homing process. After the homing process, the applications may use the PCS coordinate system directly to represent the position of group in the rectangular coordinate system.

1.4.2.3. Product Coordinate System (PCS)

The product coordinate system (PCS) describes the position orientations of a tool center point (TCP) relative to the workpiece origin. The coordinate origin is defined at the workpiece to be processed. The PCS description of a 6-axis articulated robot is shown as the below figure.



Name	Unit
X	mm
Y	mm
Z	mm
A	Degree
B	Degree
C	Degree

Example: The PCS Description of a 6-axis Articulated Robot

The TCP can be described up to 6 degrees of freedom which representations are X, Y, Z, A, B and C.

- Position: X, Y and Z are the representation of the rectangular coordinate system following the right-hand rule.
- Orientation: A, B and C are the representation of the Euler angle with intrinsic Z-Y-X rotating to Z-, Y- and X-axis, respectively.

Since the difference between the PCS and MCS is the coordinate conversion between them, the initial PCS and MCS coordinates are overlapped. That is, PCS is equal to MCS if the PCS is not defined separately.

An example shows the axis coordinate system (ACS) described with the data structure 「Pos_T」 as follows:

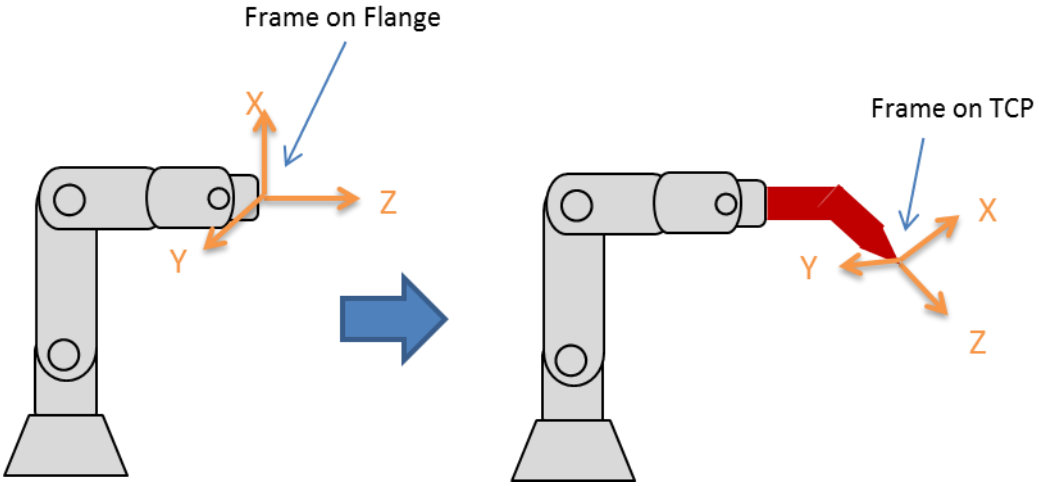
```
#include "NexMotion.h"
#include "NexMotionError.h"

int main()
{
	Pos_T point;

	// "point" represents as PCS position.
	point.pos[0] = 454.5;	// X axis
	point.pos[1] = 0.0;	// Y axis
	point.pos[2] = 755.0;	// Z axis
	point.pos[3] = -90.0;	// A axis
	point.pos[4] = -90.0;	// B axis
	point.pos[5] = -90.0;	// C axis
}
```

The following APIs can be used to get the position coordinate of PCS with the data structure Pos_T:

1. NMC_GroupGetCommandPosPcs(): To get the command position of PCS.
2. NMC_GroupGetActualPosPcs(): To get the actual position of PCS.

1.4.2.4. Tool Configuration

When the user has not set the Tool, the target position in the motion command represents the position and orientation of the flange frame of the robot end point, as shown below. As the Tool parameter is set, the TCP coordinate system will depend on the end point of the Tool according to the conversion relationship.

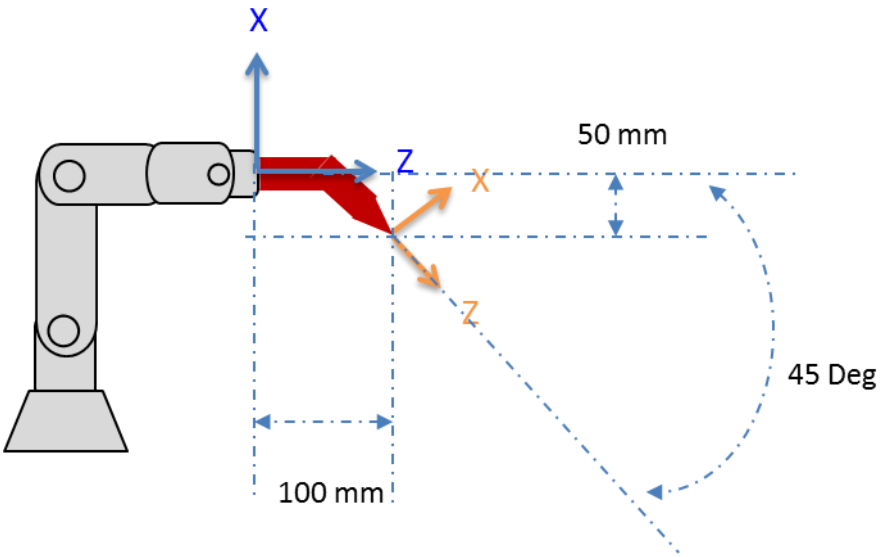


Currently, the system can support up to 16 sets of tool settings. The setting method is to set the group parameter 0x80 to 0x8F. Defined as follows:

Param. Num.	Sub. Index	Data Type	Description
0x80~8F	0	F64_T	Offset along flange x-axis
	1	F64_T	Offset along flange y-axis
	2	F64_T	Offset along flange z-axis
	3	F64_T	Rotation angle about flange z-axis
	4	F64_T	Rotation angle about flange y-axis
	5	F64_T	Rotation angle about flange x-axis

For example:

To set a set of tools Tool (Index = 0) as shown below, the TCP coordinate origin is -50 units relative to the Flange coordinate system x-axis, the y-axis is 0 units, the Z-axis is 100 units, furthermore the coordinate system is rotated - 45 degrees relative to Y-axis of Flange:



Group setting parameters and setting values are as follows:

NUM	Sub	Value	Description
0x80	0	**-50**	Offset along flange x-axis
	1	**0**	Offset along flange y-axis
	2	**100**	Offset along flange z-axis
	3	**0**	Rotation angle about flange z-axis
	4	**-45**	Rotation angle about flange y-axis
	5	**0**	Rotation angle about flange x-axis

If the motion command does not specifically specify the Tool index, the system will refer to the group parameter 0x40:0 setting as the Tool used by the current system.

Param. Num.	Sub. Index	Data Type	Description
0x40	0	I32_T	Tool index selection for motion target

1.4.2.5. Tool Calibration

In addition to directly setting parameters, there are several APIs that use the orientation methods to convert the Tool parameter:

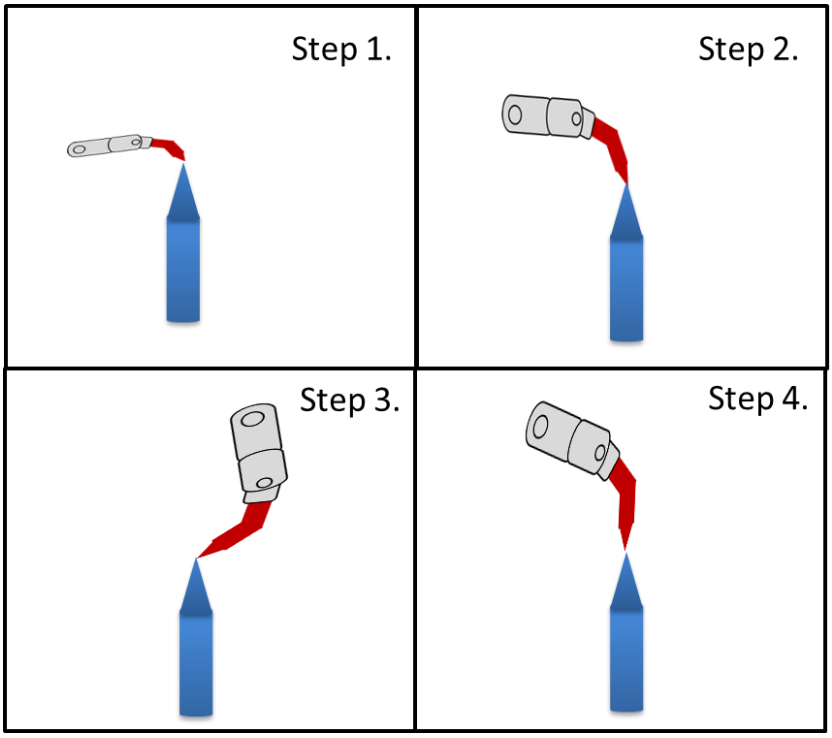
- TCP translation method
- TCP translation with Z-direction setting method
- TCP translation with orientation setting method
- Orientation setting method

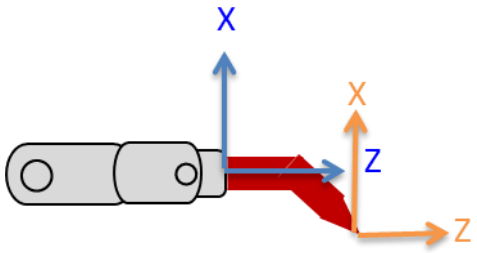
Corresponding APIs setting can refer to 3.4.13 Tool Calibration Functions.Note that during the setting of the Tool, you can choose to teach in the MCS coordinate system or teach in any Base. Never change the Base setting during the process to avoid teaching errors.

Next, we will explain the above four teaching methods.

- TCP translation method

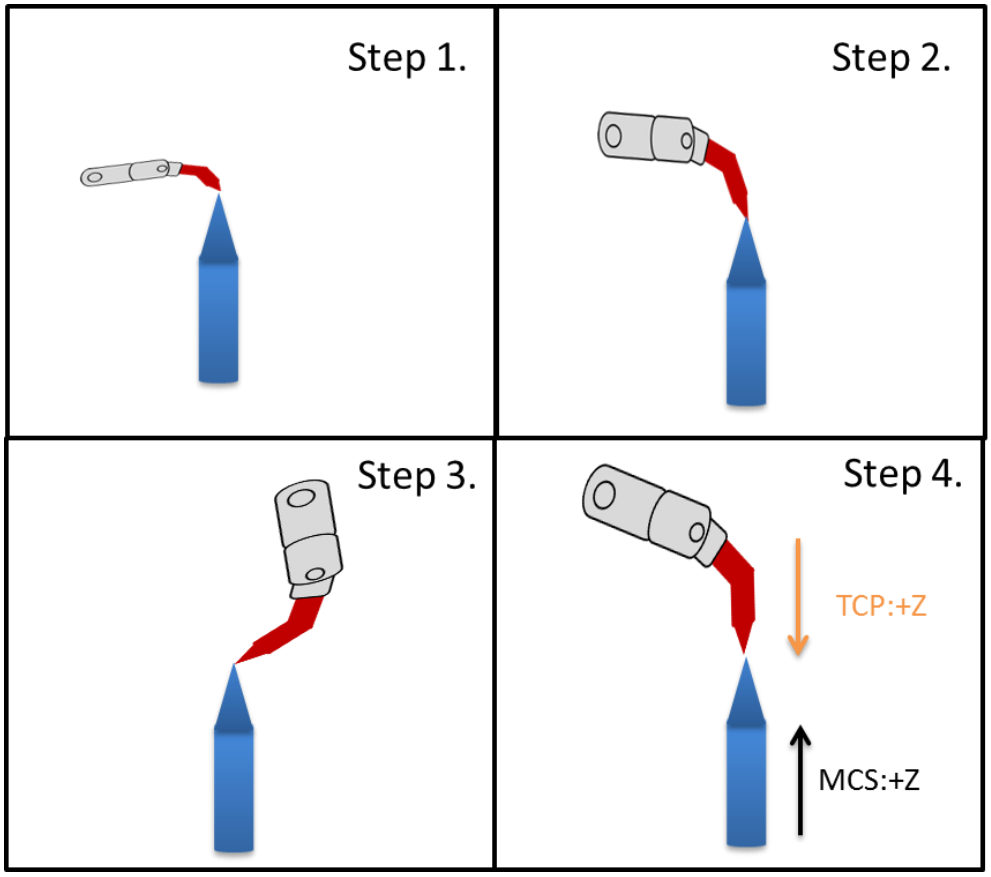
  Through the fixed TCP position, the relationship between the TCP and the flange frame is given by four different orientations. The TCP found by this method has only the translation relationship, and the ABC angle has no change (0).

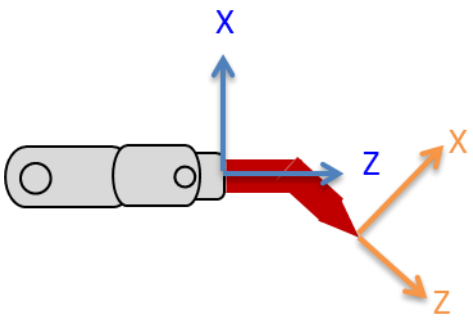




Relationship between TCP coordinate and flange coordinate

- TCP translation with Z-direction setting method
  Through the fixed TCP position, the relationship between the TCP and the flange frame is given by four different orientations. The difference from the translation method is the fourth step, in fourth step, the Z direction of the TCP must point to the negative Z direction of the MCS. The TCP found by this method has a translational relationship and a rotation to the Flange-Y axis.

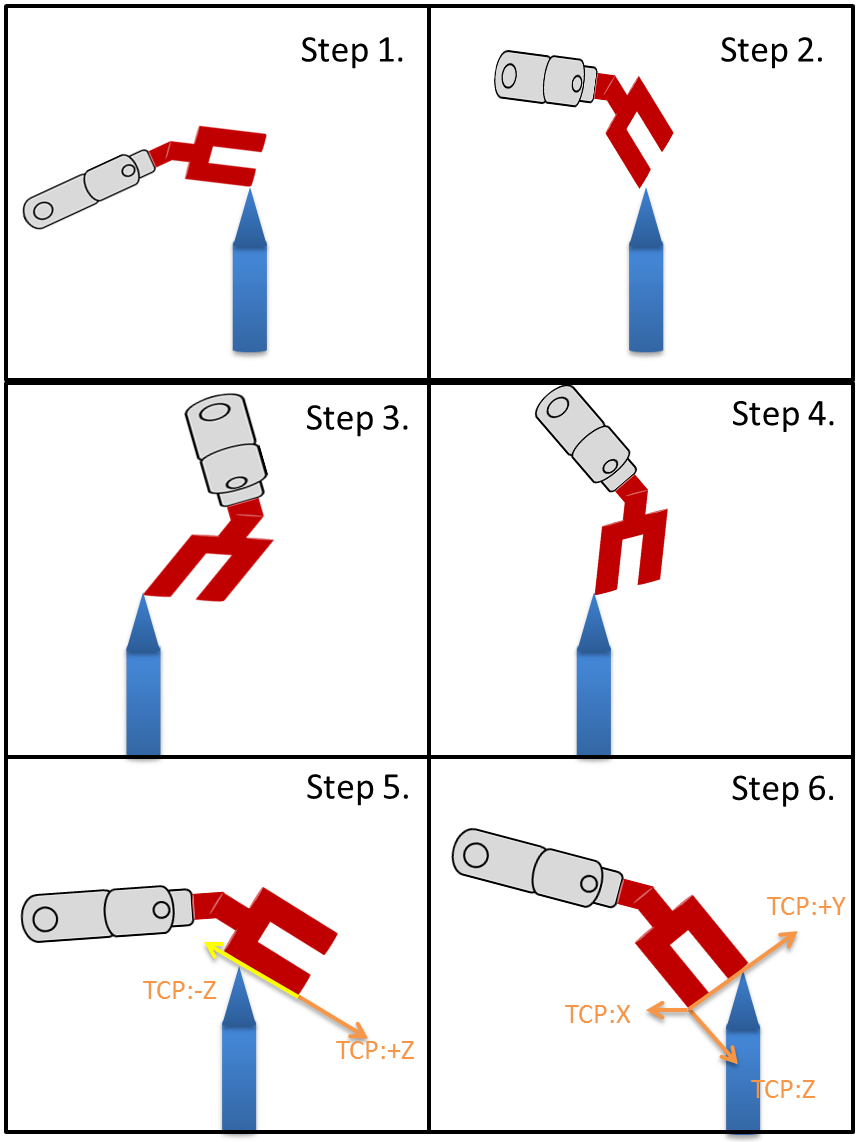


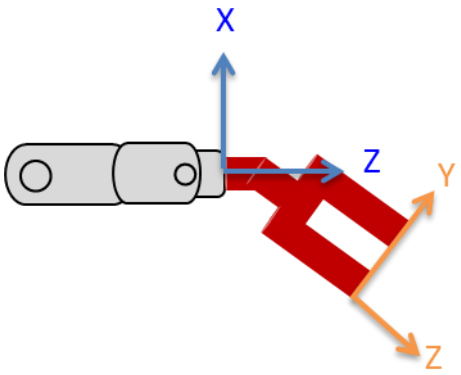


Relationship between TCP coordinate and flange coordinate

- TCP translation with orientation setting method

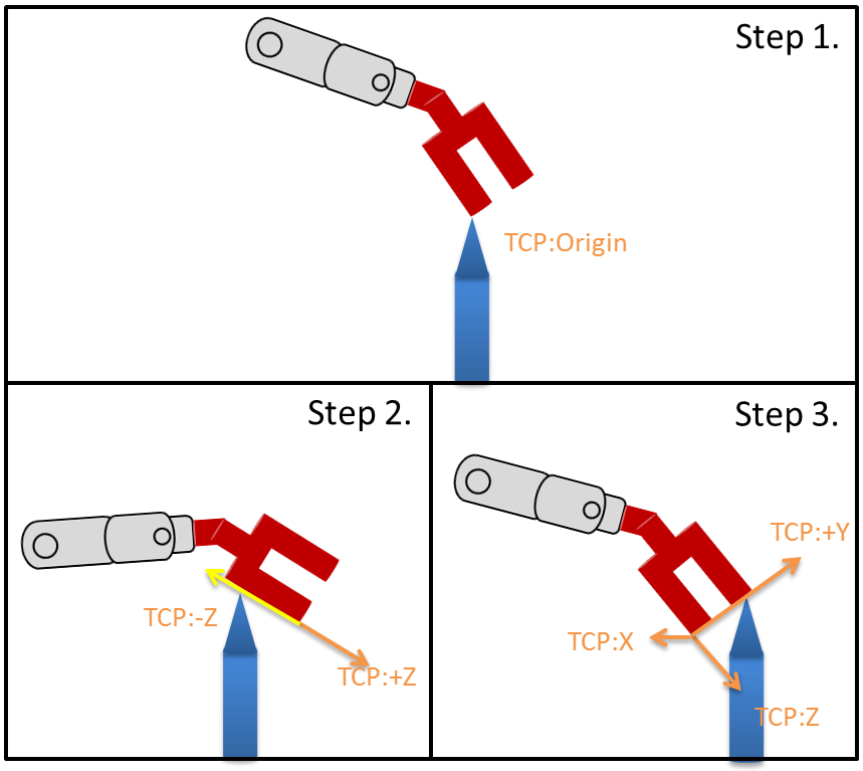
The previous four steps of this method are the same as the "TCP translation method", mainly to find the translation relationship between TCP and Flange coordinates, and steps 5 and 6 are to find the orientation of the TCP coordinates.





Relationship between TCP coordinate and flange coordinate

- Orientation setting method
  This method can use TCP translation teaching method first to find the translation relationship, and then use this method to find the orientation setting. Or you can use this method only if you want to recalibrate your orientation. Generally not used independently.



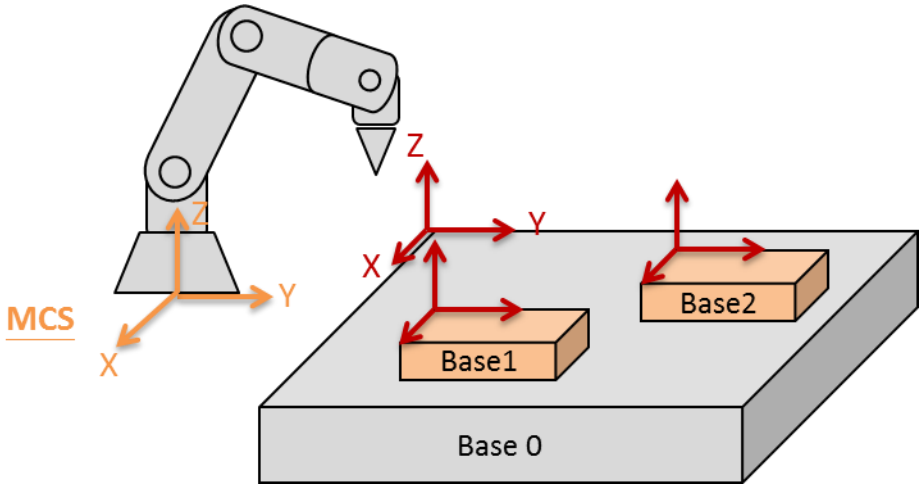
Relationship between TCP coordinate and flange coordinate

1.4.2.6. Base Configuration

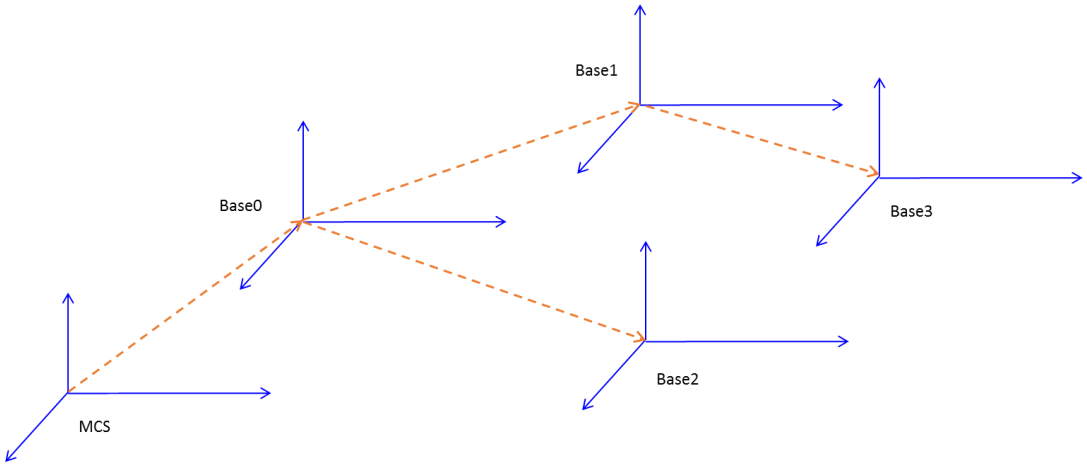
Setting Base is to define the PCS coordinate system. The advantage is that all target positions are relative to the reference coordinate system (Base). When the reference coordinate system change, only the reference coordinate system needs to be reset, then the target position can be used without teach again.

The advantages of coordinate conversion of NexMotion PCS are as follows:

- Support up to 32 set of base
- Have hierarchy setting, up to 3 level



Base having a hierarchical set, as shown above, Base0 is relative to MCS, and Base1 and Base2 are relative to Base0. When Base0 changes, Base1 and Base2 will also change.

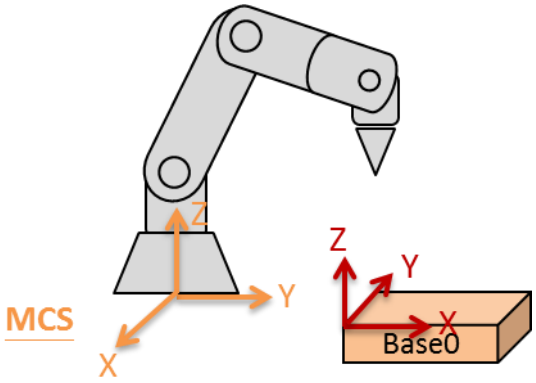


The setting method is to set the group parameter 0xC0 to 0xDF. Defined as follows:

Param. Num.	Sub. Index	Data Type	Description
0xC0~DF	0	F64_T	Offset along reference base x-axis
	1	F64_T	Offset along reference base y-axis
	2	F64_T	Offset along reference base z-axis
	3	F64_T	Rotation angle about reference base z-axis
	4	F64_T	Rotation angle about reference base y-axis
	5	F64_T	Rotation angle about reference PCS x-axis
	6	I32_T	Reference base index

For example:

The coordinate origin of the Base (Index = 0) is 10 units of x-axis, 15 units of y-axis, 5 units of Z-axis with respect to the MCS coordinate system, and the Base0 coordinate system is rotated 90 degrees with respect to the Z-axis of the MCS.



xyz@MCS =(10, 15, 5)

Group setting parameters and setting values are as follows:

NUM	Sub	Value	Description
0xC0	0	**10**	Offset along reference base x-axis
	1	**15**	Offset along reference base y-axis
	2	**5**	Offset along reference base z-axis
	3	**90**	Rotation angle about reference base z-axis
	4	**0**	Rotation angle about reference base y-axis
	5	**0**	Rotation angle about reference PCS x-axis
	6	**-1**	Reference base index

If the motion command does not specifically specify the Base index, the system will refer to the group parameter 0x48:0 setting as the Base used by the current system.

Param. Num.	Sub. Index	資料型態	說明
0x48	0	I32_T	Base index selection for motion target

1.4.2.7. Base Calibration

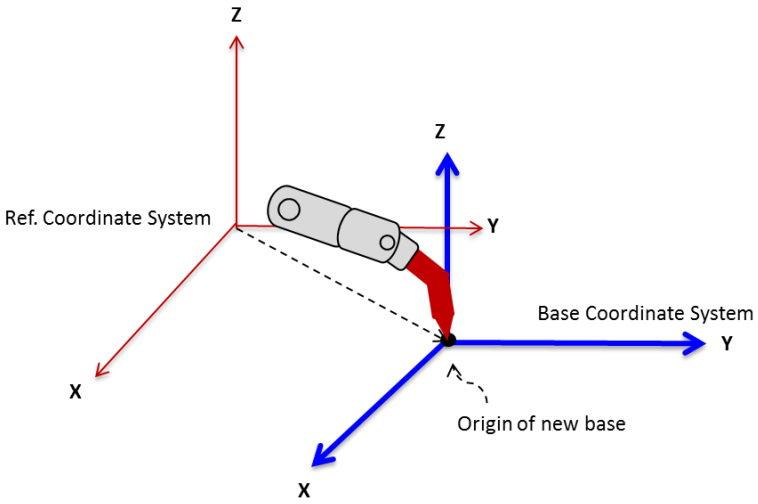
In addition to directly setting parameters, method of automatically calibration the teaching points is provided to set the Base. The automatic calibration methods include:

- Base teaching -1p method
- Base teaching -2p method
- Base teaching -3p method

Corresponding APIs setting can refer to 3.4.14 Base Calibration Functions. As the name implies, 1p method only needs to teach one point. The 2p method only needs to pay two points. The 3p method needs to teach three points. Among them, the difference between the 2p method and the 1p method is that the base taught by the 2p method has more rotation of the Z-axis, and the 3p method can arbitrarily define the X-Y-Z axis.

- Base teaching -1p method steps as follows:

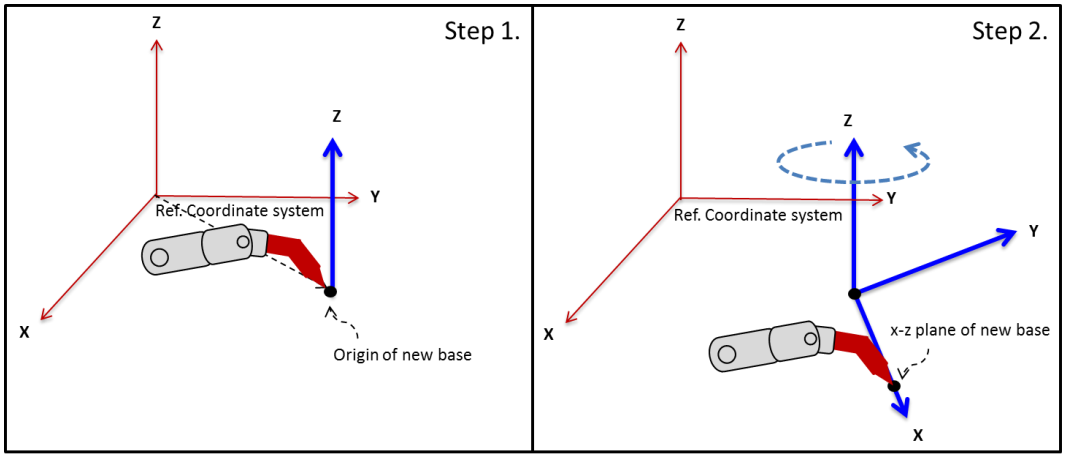
Step1: Define the base coordinate origin



This method is equivalent to move to a new coordinate point in the reference coordinate system。

- Base teaching -2p method steps as follows:

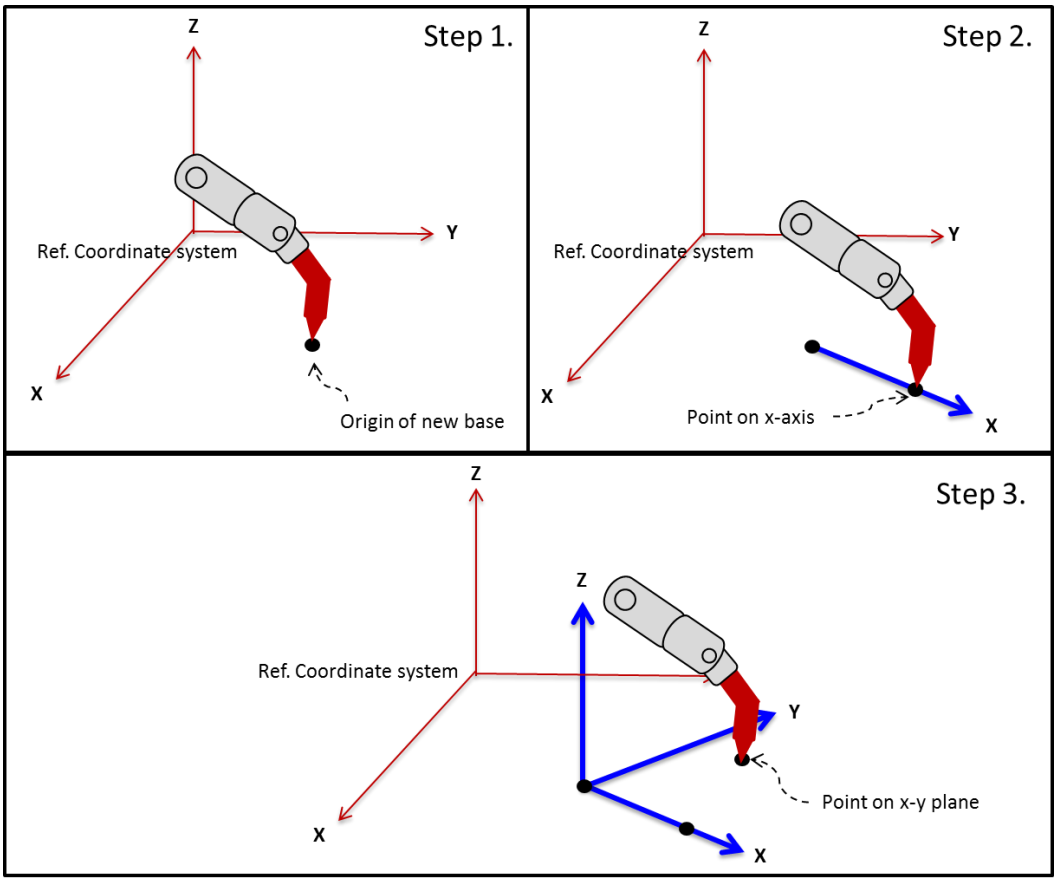
Step1: Define the base coordinate origin
Step2: Define a point on the X-Z plane, the X axis is the positive direction



This method is equivalent to move to a new coordinate point in the reference coordinate system, then rotate with z axis.

- Base teaching -3p method steps as follows:

Step1: Define the base coordinate origin
Step2: Define a point on the X axis, the X axis is the positive direction
Step3: Define a point on the X-Y plane, the Y axis is the positive direction



1.4.3. Mechanism Kinematics Configurations

Now, NexMotion supports four type of industrial robots:

1. Linear Robot (2~8 axes)
2. 6-axis Articulated Robot
3. SCARA Robot
4. Delta Robot

The configuration method is to set the group parameters 0x00:0~48 which definitions are:

1. Parameters 0x00:0 is the mechanism type selection, and
2. Parameters 0x00:1~24 are the mechanism dimension.

The detail parameter definitions are described as the below table:

Sub \ Type	Linear	6-axis Articulated	Delta	SCARA
0	0	1	2	3
1	Axis number (value=2~8	0 (*1)	f(mm)	a1 (mm)

	integer)			
2	N/A	a2 (mm)	e(mm)	a2 (mm)
3	N/A	a3 (mm)	rf(mm)	0 (*1)
4	N/A	a4 (mm)	re(mm)	0 (*1)
5	N/A	0 (*1)	hf(mm)	d1 (mm)
6	N/A	0 (*1)	he(mm)	0 (*1)
7	N/A	d1 (mm)	PEL-J1 (Deg)	d3 (mm)
8	N/A	0 (*1)	PEL-J2 (Deg)	d4 (mm)
9	N/A	0 (*1)	PEL-J3 (Deg)	PEL-J1 (Deg)
10	N/A	d4 (mm)	PEL-J4 (Deg)	PEL-J2 (Deg)
11	N/A	0 (*1)	MEL-J1 (Deg)	PEL-J3 (mm)
12	N/A	d6 (mm)	MEL-J2 (Deg)	PEL-J4 (Deg)
13	N/A	PEL-J1 (Deg)	MEL-J3 (Deg)	MEL-J1 (Deg)
14	N/A	PEL-J2 (Deg)	MEL-J4 (Deg)	MEL-J2 (Deg)
15	N/A	PEL-J3 (Deg)	Theta disable Value = 0(Delta4) Value = 1(Delta3)	MEL-J3 (mm)
16	N/A	PEL-J4 (Deg)	N/A	MEL-J4 (Deg)
17	N/A	PEL-J5 (Deg)	N/A	N/A
18	N/A	PEL-J6 (Deg)	N/A	N/A
19	N/A	MEL-J1 (Deg)	N/A	N/A
20	N/A	MEL-J2 (Deg)	N/A	N/A
21	N/A	MEL-J3 (Deg)	N/A	N/A
22	N/A	MEL-J4 (Deg)	N/A	N/A
23	N/A	MEL-J5 (Deg)	N/A	N/A
24	N/A	MEL-J6 (Deg)	N/A	N/A

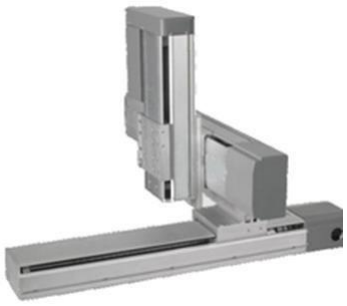
(*1) Reserved and must be set to 0.

- PEL: Positive End Limit
- MEL: Minus End Limit

1.4.3.1. Linear Robot (2~8 axes)

A linear robot sets the motor mechanism with linear axes which number of axes can be 2 to 8. A characteristic of linear robot is the ACS, and the MCS and PCS are overlapped.

The most common linear robot is the robot equipped with an X-Y Table (2-Axis) or an X-Y-Z rectangular coordinate robot (3-Axis) shown as the below figure. A motor can be assembled in each direction of the rectangular coordinate system. The motor motion is mapped to the motions in the directions of Cartesian coordinate system directly.



X-Y-Z Linear Axis

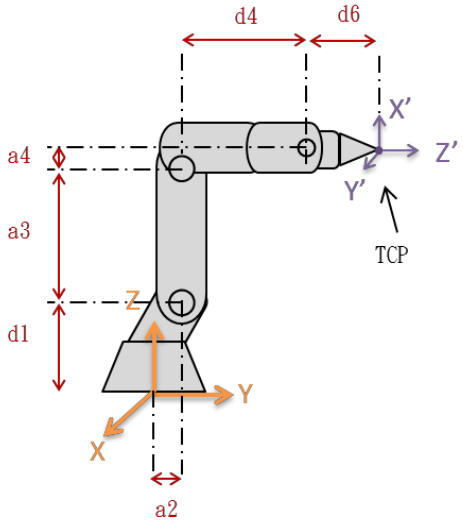
The definition of the mechanism kinematics parameter (group parameter 0x00) is shown as the below table:

Sub \ Type	Linear Parameter Definition
0	0
1	Number of axes (value=2~8 integer)

1.4.3.2. 6-axis Articulated Robot

The motor mechanism configurations and mechanism kinematics parameters of a 6-axis articulated robot are shown as the below figure. The system supports 6 degrees of freedom in the 3D space.

The definition of the mechanism kinematics parameter (group parameter 0x00) is shown as the below table:

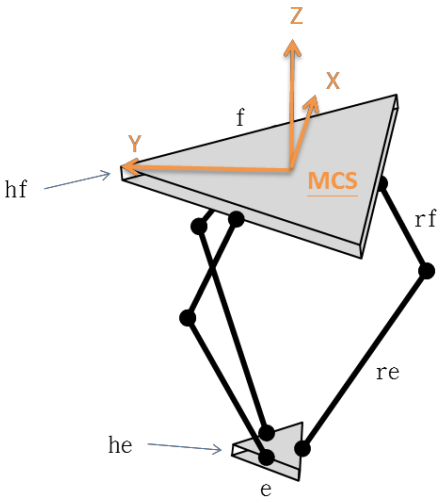
0x00 Sub No.	Parameter Definition	Diagram
1	0 (*1)	
2	a2 (mm)	
3	a3 (mm)	
4	a4 (mm)	
5	0 (*1)	
6	0 (*1)	
7	d1 (mm)	
8	0 (*1)	
9	0 (*1)	
10	d4 (mm)	
11	0 (*1)	
12	d6 (mm)	
13	PEL-J1 (Deg)	Joint 1 Positive End Limit
14	PEL-J2 (Deg)	Joint 2 Positive End Limit
15	PEL-J3 (Deg)	Joint 3 Positive End Limit
16	PEL-J4 (Deg)	Joint 4 Positive End Limit
17	PEL-J5 (Deg)	Joint 5 Positive End Limit
18	PEL-J6 (Deg)	Joint 6 Positive End Limit
19	MEL-J1 (Deg)	Joint 1 Negative End Limit
20	MEL-J2 (Deg)	Joint 2 Negative End Limit
21	MEL-J3 (Deg)	Joint 3 Negative End Limit
22	MEL-J4 (Deg)	Joint 4 Negative End Limit
23	MEL-J5 (Deg)	Joint 5 Negative End Limit
24	MEL-J6 (Deg)	Joint 6 Negative End Limit

(*1) Reserved and must be set to 0.

1.4.3.3. Delta Robot

A delta robot is a parallel robot which motor mechanism configurations and mechanism kinematics parameters are shown as the below figure. The system supports the X-Y-Z translation and 1 rotary in the direction at the Z-axis.

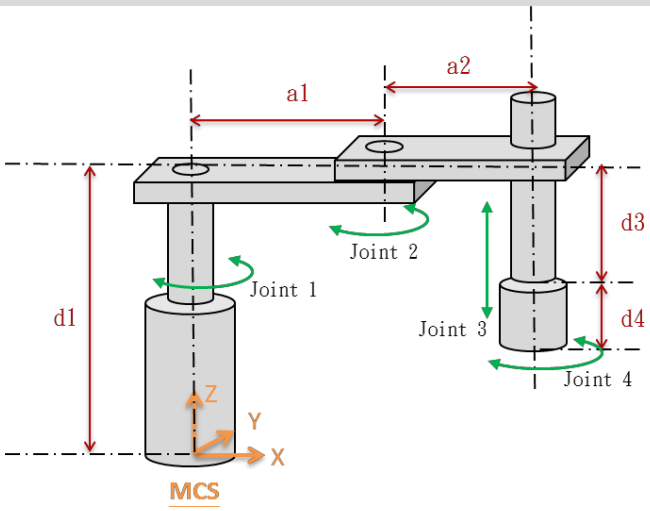
The definition of the mechanism kinematics parameter (group parameter 0x00) is shown as the below table:

0x00 Sub No.	Parameter Definition	Diagram
1	f upper triangular side (mm)	
2	e lower triangular side (mm)	
3	rf upper rod length (mm)	
4	re lower rod length (mm)	
5	hf upper triangular height (mm)	
6	he lower triangular height (mm)	
7	PEL-J1 (Deg)	Joint 1 Positive End Limit
8	PEL-J2 (Deg)	Joint 2 Positive End Limit
9	PEL-J3 (Deg)	Joint 3 Positive End Limit
10	PEL-J4 (Deg)	Joint 4 Positive End Limit
11	MEL-J1 (Deg)	Joint 1 Negative End Limit
12	MEL-J2 (Deg)	Joint 2 Negative End Limit
13	MEL-J3 (Deg)	Joint 3 Negative End Limit
14	MEL-J4 (Deg)	Joint 4 Negative End Limit

1.4.3.4. SCARA Robot

A SCARA robot is a horizontal joint robot which motor mechanism configurations and mechanism kinematics parameters are shown as the below figure. The system supports the X-Y-Z translation and 1 rotary in the direction at the Z-axis.

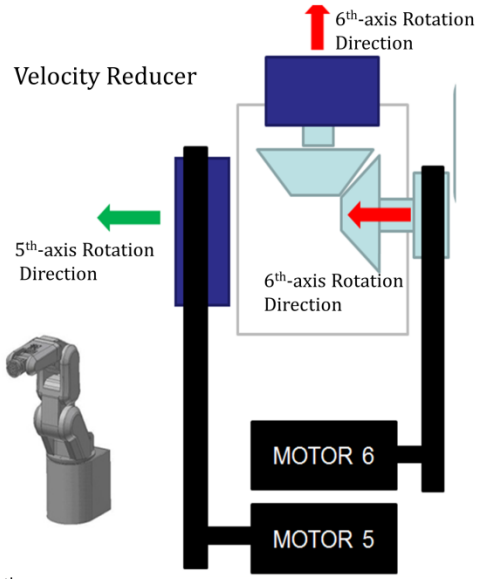
The definition of the mechanism kinematics parameter (group parameter 0x00) is shown as the below table:

0x00 Sub No.	Parameter Definition	Diagram
1	a1 (mm)	
2	a2 (mm)	
3	0 (*1)	
4	0 (*1)	
5	d1 (mm)	
6	0 (*1)	
7	d3 (mm)	
8	d4 (mm)	
9	PEL-J1 (Deg)	Joint 1 Positive End Limit
10	PEL-J2 (Deg)	Joint 2 Positive End Limit
11	PEL-J3 (mm)	Joint 3 Positive End Limit
12	PEL-J4 (Deg)	Joint 4 Positive End Limit
13	MEL-J1 (Deg)	Joint 1 Negative End Limit
14	MEL-J2 (Deg)	Joint 2 Negative End Limit
15	MEL-J3 (mm)	Joint 3 Negative End Limit
16	MEL-J4 (Deg)	Joint 4 Negative End Limit

(*1) Reserved and must be set to 0.

1.4.4. Structural Coupling Compensation

In the mechanism of group, it is common that two axes are coupling due to the mechanism design. For example, the transmission mechanism between the 5th and 6th axes for a 6-axis articulated robot causes the corresponding rotation of the 6-axis external joint structure due to the 5-axis motor rotation as the below figure:



The 5th and 6th Axes Structural Coupling for a 6-axis Articulated Robot

Another example is a 4-axis SCARA robot. If the vertical axis transfers the motor rotation to the vertical motion with a ball screw rod, it will cause the corresponding rotation of the end mechanism. For the compensation of the transmission effect generated by the structural coupling, the coupling information can be set to the motion control module with the group parameter 0x01, in order to calculate the required motion compensation command.

The related group parameter 0x01 is described as follows:

Sub Index	Description
0	Mechanical couple master axis
1	Mechanical couple slave axis
2	Compensation source type
3	Reference master original position
4	Coupling ratio numerator
5	Coupling ratio denominator
6	Enable mechanical couple compensation

- 0x01, SubIndex = 0: Mechanical couple master axis
  The parameter is used to specify the master axis in the structural coupling. For example, the 5th axis of the aforementioned 6-axis articulated robot is the master axis.

- 0x01, SubIndex = 1: Mechanical couple slave axis
  The parameter is used to specify the slave axis in the structural coupling. For example, the 6th axis of the aforementioned 6-axis articulated robot is the slave axis.

- 0x01, SubIndex = 2: Compensation source type
  The parameter is used to specify the position compensation command of the slave axis depended on the actual position or the command position of the master axis.

- 0x01, SubIndex = 3: Reference master original position

The parameter is used to specify the position of the master axis as the calculation basis for the position compensation of the slave axis.

- 0x01, SubIndex = 4: Coupling ratio numerator
- 0x01, SubIndex = 5: Coupling ratio denominator

The above two parameters must be set together, and the coupling ratio required to calculate the position command compensation amount of the slave axis is mainly set in the form of a numerator and a denominator. The numerator cannot be 0, and the numerator with the minus sign represents the reverse compensation. The denominator must be greater than 0. For the 6-axis articulated robot in the above figure, the 5th axis (master axis) mechanism rotating by 90 degrees in the positive direction will cause the 6th axis (slave axis) rotating by 90 degrees in the negative direction. Assume velocity reduction ratio of the 6th axis is 50, the 6th axis mechanism will rotate -1.8 degrees. In order to compensate the rotation caused by the structural coupling, the position command of the 6th axis mechanism needs to add 1.8 degrees (the position compensation amount of slave axis). Thus, the denominator (SubIndex 5) shall be set to 50, the numerator (SubIndex 4) shall be set to 1. The related formula is shown as follows:

The position compensation amount of slave axis = (The position of the master axis mechanism – the position basis of the master axis (SubIndex 2)) * (SubIndex 4) / (SubIndex 5)

- 0x01, SubIndex = 6: Enable mechanical couple compensation
  Enable the structural coupling compensation. After the function is enabled, the device will automatically calculate the position compensation command to the slave axis in accordance with the structural coupling information.

These parameters cannot be modified after the system is in the OPERATION state.

1.4.5. Group Enable and State

NMC_GroupEnable() can be called to execute the servo enable for all axes in a group. Since various brand devices may support different servo enable time, a waiting time can be executed until the device state transfers to 「NMC_AXIS_STATE_DISABLE」. NMC_GroupGetState() can be called to check each group state.

After all axes have been enabled, the group state will transfer to 「NMC_GROUP_STATE_STAND_STILL」. In case one or more axes which axis state is DISABLE, the group state will transfer to 「NMC_GROUP_STATE_DISABLE」. In case one or more axes occurring errors, the group state will transfer to 「NMC_GROUP_STATE_ERROR」. NMC_GroupDisable() can be called to execute the servo disable for all axes in a group.

```
#include "NexMotion.h"
#include "NexMotionError.h"
#include <Windows.h>

int main()
{
	RTN_ERR ret		= 0;
	I32_T devType	= NMC_DEVICE_TYPE_ETHERCAT;
	I32_T devIndex = 0;
	I32_T groupIndex = 0;
	I32_T devID		= 0;
	I32_T state		= 0;

	ret = NMC_DeviceOpenUp( devType, devIndex, &devID );
	if( ret != ERR_NEXMOTION_SUCCESS )
	{
		// Error handling...
	}
	// Device is start up successfully.

	ret = NMC_GroupEnable( devID, groupIndex );
	if( ret != ERR_NEXMOTION_SUCCESS )
	{
		// Error handling...
	}

	// To make sure state is transferring to ENABLE
	Sleep( 3000 );  //3 seconds is a try value. Depends on servo drives.

	ret = NMC_GroupGetState( devID, groupIndex, &state );
	if( ret != ERR_NEXMOTION_SUCCESS )
	{
		// Error handling...
	}

	if( state != NMC_GROUP_STATE_STAND_STILL )
	{
```

```
        // Error handling...
    }

    // The group is enabled successfully.
    // Do something...
    // ...

    ret = NMC_GroupDisable( devID, groupIndex );
    if( ret != ERR_NEXMOTION_SUCCESS )
    {
        // Error handling...
    }

    // To make sure state is transferring to DISABLE
    Sleep( 1000 );  //1 second is a try value. Depends on servo drives.

    NMC_DeviceShutdown( devID );
    return 0;
}
```

In case the device contains multiple axes or groups, NMC_DeviceEnableAll() can be called to execute the servo enable for all axes of all groups. NMC_DeviceGetGroupCount() can be called to check the group quantity controlled by the device. NMC_DeviceDisableAll() can be called to execute the servo disable for all axes of all groups.

```
#include "NexMotion.h"
#include "NexMotionError.h"
#include <Windows.h>

int main()
{
    RTN_ERR ret    = 0;
    I32_T   devType   = NMC_DEVICE_TYPE_ETHERCAT;
    I32_T   devIndex = 0;
    I32_T   groupIndex = 0;
    I32_T   devID      = 0;
    I32_T   groupCount = 0;
    I32_T   state= 0;
    I32_T   i;

    ret = NMC_DeviceOpenUp( devType, devIndex, &devID );
    if( ret != ERR_NEXMOTION_SUCCESS )
    {
        // Error handling...
    }
    // Device is start up successfully.

    ret = NMC_DeviceEnableAll( devID );
```

```
    if( ret != ERR_NEXMOTION_SUCCESS )
    {
        // Error handling...
    }

    // To make sure the state is transferring to ENABLE
    Sleep( 3000 );  //3 seconds is a try value. Depends on servo drives.

    // Get group count in device
    ret = NMC_DeviceGetGroupCount( devID, &groupCount );
    if( ret != ERR_NEXMOTION_SUCCESS )
    {
        // Error handling...
    }

    for( i = 0; i < groupCount; i++ )
    {
        ret = NMC_GroupGetState( devID, groupIndex, &state );
        if( ret != ERR_NEXMOTION_SUCCESS )
        {
            // Error handling...
        }

        if( state != NMC_GROUP_STATE_STAND_STILL )
        {
            // Error handling...
        }
    }

    NMC_DeviceDisableAll( devID );

    // To make sure state is transferring to DISABLE
    Sleep( 1000 );  //1 second is a try value. Depends on servo drives.

    NMC_DeviceShutdown( devID );

    return 0;
}
```

In addition to the group state, NMC_GroupGetStatus() can be called to get more group status. Currently, there are 14 status information, including the external emergency stop signal, alarm, positive hardware limit, negative hardware limit, positive software limit, negative software limit, enable, error state, command stop, acceleration state, deceleration state, state in motion and stop state.

NMC_GroupResetState() shall be called to reset the group state for the following two cases:

1. NMC_GroupStop() can be called to stop a group in motion in each kind of case (or in emergency), and then the group state will transfer to STOPPED. If the group state is STOPPED, the device will reject to execute a new motion command. After the event is resolved and the device can accept a

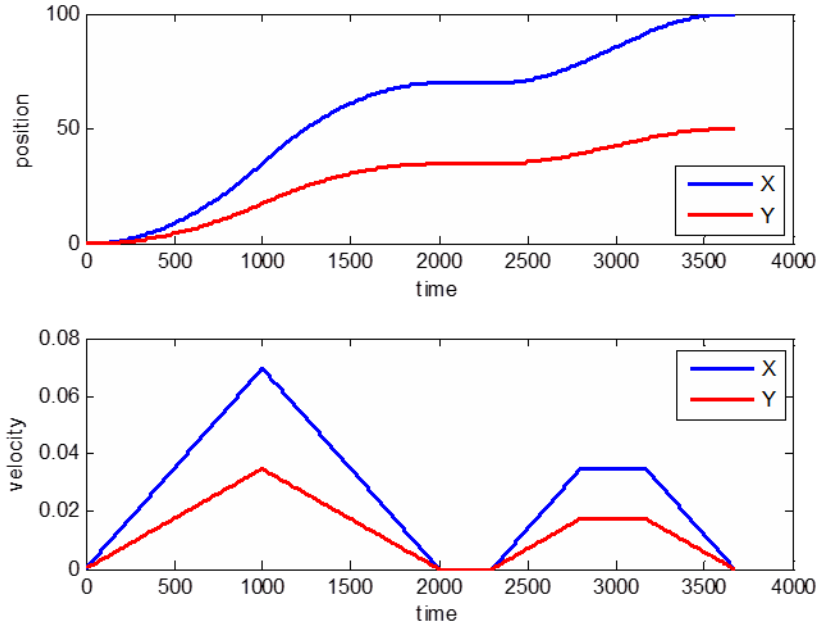
new motion command, NMC_GroupResetState() shall be called to transfer the group state to STAND STILL.

2. If an alarm occurring in the drive of any axis in a group can be reset by the device, it can also be reset by this API. The group state will transfer from ERROR STOP to DISABLE.

If an alarm occurring in the drive of any axis in a group can be reset by the authorized device, NMC_GroupResetDriveAlm() or NMC_GroupResetDriveAlmAll() can be called to reset such alarm.

1.4.6. Group Velocity Percentage Configurations

For each group, there is a velocity percentage configuration which can be set via NMC_GroupSetVelRatio(). The maximum velocity of each kind of group motion refers to such velocity percentage. For example, if the maximum velocity is 100 and the velocity percentage is 70%, the maximum motion velocity shall be 100 * 70% = 70 and the velocity plan will be developed accordingly. The below figure shows a group executing the point-to-point motion at the X- and Y-axis and the velocity percentage is set to 0 at the time 1000. After the configuration, the motion decreases the velocity to 0. The velocity percentage is set to 50 at the time 2300, and then the motion increases the velocity until it reaches the target position.




```
#include "NexMotion.h"
#include "NexMotionError.h"

int main()
{

}
```

NMC_GroupGetVelRatio() can be called to get the current velocity percentage of the group.

1.4.7. Group Point-to-Point Motion

The APIs related to the group point-to-point motion are listed as follows:

- Axis coordinate system:
  NMC_GroupPtpAcs()
  NMC_GroupPtpAcsAll()
- Cartesian coordinate system:
  NMC_GroupPtpCart()
  NMC_GroupPtpCartAll()

The difference between these APIs is the number of controllable axes and the coordinate system of target position but there motion behaviors are the same. That is, to execute the point-to-point motion in each one of the two coordinate systems, the device will develop the shortest path plan for each axis (axis coordinate system), and then automatically moves all axes to the target position at the

same time by setting the longest required time of an axis as the basis and reducing the required velocity of other axes accordingly. The motion velocity of each axis is calculated in accordance with the axis parameters of each group axis. The velocity parameters related to the point-to-point motion are described as follows:

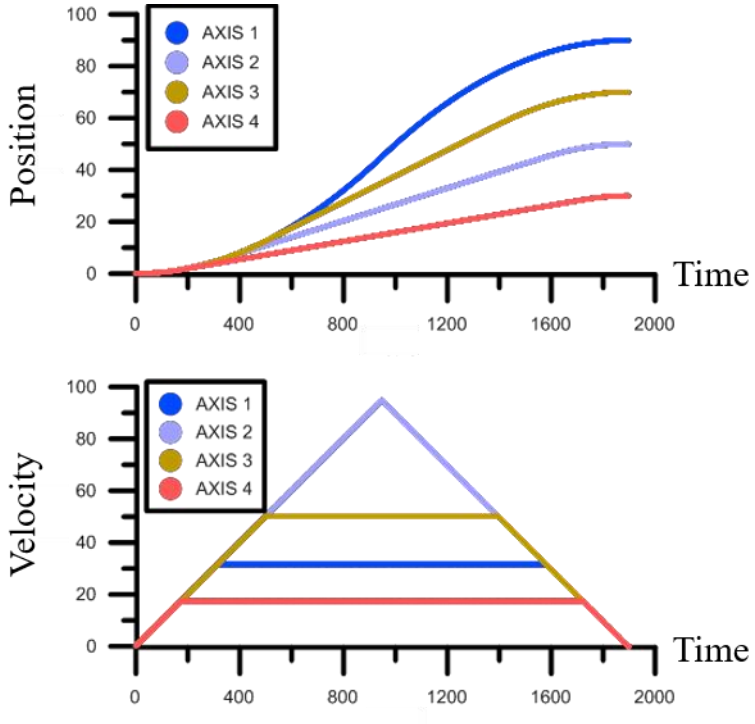
Param. Num.	Sub. Index	Data Type	Description
0x30	0	I32_T	Absolute or relative programming
0x31	0	I32_T	Profile type
0x32	0	F64_T	Max. velocity (unit/sec)
0x33	0	F64_T	Acceleration (unit/sec^2)
0x34	0	F64_T	Deceleration (unit/sec^2)
0x35	0	F64_T	Jerk (unit/sec^3)

The APIs related to get/set group axis parameters are listed as follows:
NMC_GroupAxSetParamI32()
NMC_GroupAxGetParamI32()
NMC_GroupAxSetParamF64()
NMC_GroupAxGetParamF64()

In the axis coordinate system, NMC_GroupPtpAcs() can be called to move an axis in the group to a specified position, and NMC_GroupPtpAcsAll() can be called to move more than one axes to the specified position. The axis mask and the position in the axis coordinate system shall be specified.

In the space coordinate system, NMC_GroupPtpCart() can be called to move an axis in the Cartesian coordinate axis to a specified position (by operating in MCS or PCS), and NMC_GroupPtpCartAll() can be called to move more than one Cartesian coordinate axes to the specified position. The mask and the position of Cartesian coordinate axis shall be specified.

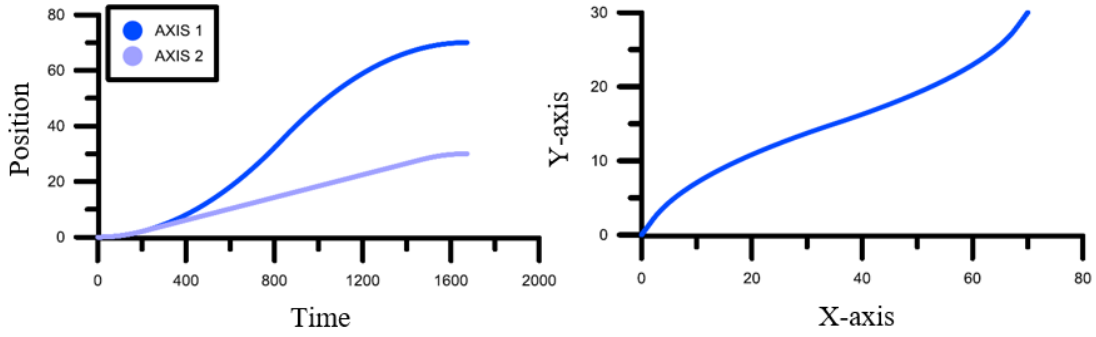
The below figure and the sample program show the usage of NMC_GroupPtpAcsAll(). It moves the 1st axis to the position 90, the 2nd axis to the position 50, the 3rd axis to the position 70, and the 4th axis to the position 30. After the API is called, the group performs the interpolation for the axes to be moved.




```
#include "NexMotion.h"
```

```
#include "NexMotionError.h"

int main()
{
	RTN_ERR ret    = 0;
	I32_T	groupIndex     = 0;
	I32_T	groupAxesIdxMask = 15; // Move the 1st, 2nd, 3rd and 4th axes
	I32_T	devID	= 0;
	Pos_T acsPos   = { 0 };

	acsPos[0] = 90; // Target position of the 1st axis
	acsPos[1] = 50; // Target position of the 2nd axis
	acsPos[2] = 70; // Target position of the 3rd axis
	acsPos[3] = 30; // Target position of the 4th axis

	ret = NMC_GroupPtpAcsAll( devID, groupIndex, groupAxesIdxMask, &acsPos );
	if( ret != ERR_NEXMOTION_SUCCESS )
	{
		// Error handling...
	}

	return 0;
}
```

The below figure and the sample program show the usage of NMC_GroupPtpCartAll(). It moves the group to position 70 along the X direction and the position 30 along the Y direction. After the API is called, the group performs the interpolation for the axes to be moved.




```
#include "NexMotion.h"
#include "NexMotionError.h"

int main()
{
	RTN_ERR ret	= 0;
	I32_T	groupIndex     = 0;
	I32_T	cartAxesMask = 5; // Move the X- and Z-axis
	I32_T	devID	= 0;
	Pos_T targetPos	= { 0 };
```

```
    targetPos[0] = 70; // Target position of the X-axis
    targetPos[2] = 30; // Target position of the Y-axis

    ret = NMC_GroupPtpCartAll( devID, groupIndex, cartAxesMask, &targetPos );
    if( ret != ERR_NEXMOTION_SUCCESS )
    {
        // Error handling...
    }

    return 0;
}
```

1.4.8. Group JOG Motion

NMC_GroupJogAcs() can be called to move an axis in the group at the maximum velocity (by operating in the ACS). After the function is called successfully, the group axis will operate at the specified maximum velocity. NMC_GroupHalt() can be called to stop the group axis motion, or NMC_GroupHaltAll() can be called to stop the group. The velocity plan of JOG motion is based on the axis parameters as follows:

Param. Num.	Sub. Index	Data Type	Description
0x31	0	I32_T	Profile type
0x32	0	F64_T	Max. velocity (unit/sec)
0x33	0	F64_T	Acceleration (unit/sec^2)
0x34	0	F64_T	Deceleration (unit/sec^2)
0x35	0	F64_T	Jerk (unit/sec^3)

The APIs related to get/set group axis parameters are listed as follows:
NMC_GroupAxSetParamI32()
NMC_GroupAxGetParamI32()
NMC_GroupAxSetParamF64()
NMC_GroupAxGetParamF64()

If the axis meets the hardware or the software limit during the motion, it will stop automatically, and the group state will transfer from MOVING to STOPPED via STOPPING. NMC_GroupResetState() can be called to transfer the group state to STAND STILL.

1.4.9. Group Motion Stop

All motion commands can be stopped with the commands as follows:

API	Description
NMC_GroupHalt() NMC_GroupStop()	Stop a group
NMC_GroupHaltAll() NMC_GroupStopAll()	Stop all groups
NMC_DeviceHaltAll() NMC_DeviceStopAll()	Stop all axes and groups

Comparison between Halt and Stop:

	Halt	Stop
Behavior	Stop the motion in operation. If there is no any motion command in the motion queue (motion buffer), the group state will transfer to NMC_GROUP_STATE_STAND_STILL. Otherwise, if there is some motion commands in the motion queue (motion buffer), these commands will be executed successively.	Stop the motion in operation and clear all commands in the motion queue (motion buffer). The group state will transfer to NMC_GROUP_STATE_STAND_STOPPED. No motion command can be executed until NMC_GroupResetState() is called.
Parameter	0x31: Halt profile type 0x34: Halt deceleration (unit/sec^2) 0x35: Halt jerk (unit/sec^3) 0x36-0: Buffer mode	0x20 : Stop profile type 0x21 :Stop deceleration (unit/sec^2) 0x22 :Stop jerk (unit/sec^3)
Stored in queue	The command can be stored into the motion queue (motion buffer) for the usage based on the group parameter 0x36 sub 0 (buffer mode). If the 0x36 is 0 (Aborting), the command will stop the motion in operation immediately. If the 0x36 is 1 (Buffered), the command will be stored in the motion queue.	No.

1.4.10. Group Line Interpolation

The following APIs can be called to execute the line motions in the Cartesian coordinate system:

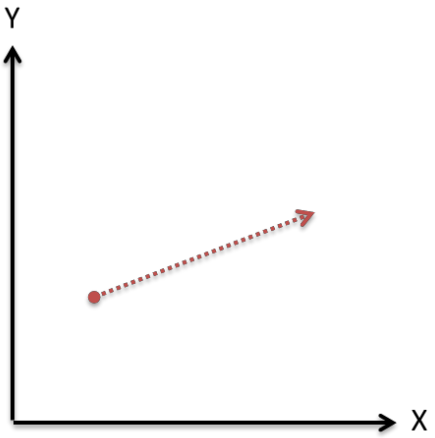
API	Description
NMC_GroupLineXY()	2D line interpolation (X- and Y-axis)
NMC_GroupLine()	3D line interpolation and 3D orientation control (up to 6 degrees of freedom)

The line interpolation motion functions can be divided into the following cases:

1. Line interpolation at the linear axis, and
2. Line interpolation at the linear axis and the orientation control at the orientation axis (A-, B- and C-axis).

1.4.10.1.Multi-dimensional Line interpolation

Generally, NMC_GroupLineXY() can be called to execute the 2D line interpolation motion for a 2D mechanism (such as a XY Table) or a multi-dimensional mechanism as the below figure:

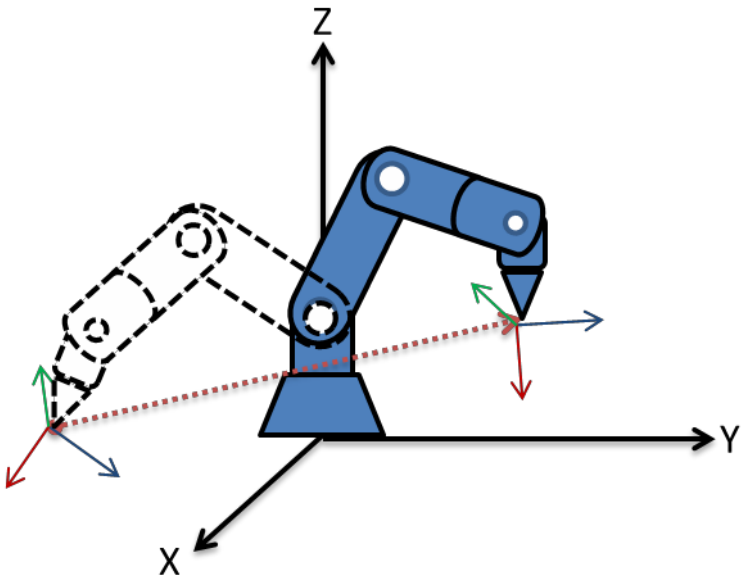


NMC_GroupLine() can be called to execute the interpolation motion for 2D or multi-dimensional linear mechanism. The group line interpolation develops the plan in accordance with the group parameters as follows:

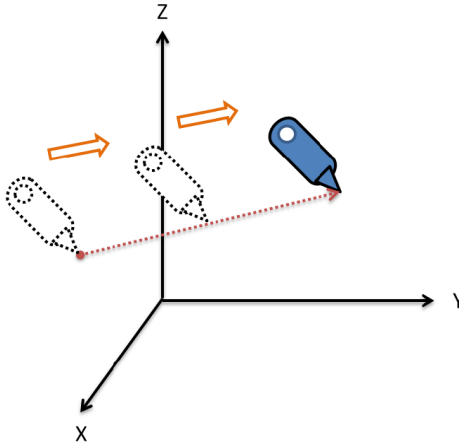
Definition	Value	Description
Target position type	0x30	0: Absolute position, 1: Incremental position
Velocity plan method	0x31	0: T curve,1: S-shape curve
Maximum velocity	0x32	Maximum velocity of the velocity plan (unit/sec)
Acceleration	0x33	Acceleration of the velocity plan (unit/sec^2)
Deceleration	0x34	Deceleration of the velocity plan (unit/sec^2)
Jerk	0x35	Jerk of the velocity plan (unit/sec^3) (Effective for S-shape curve only)

1.4.10.2.3D Line interpolation and Orientation Control

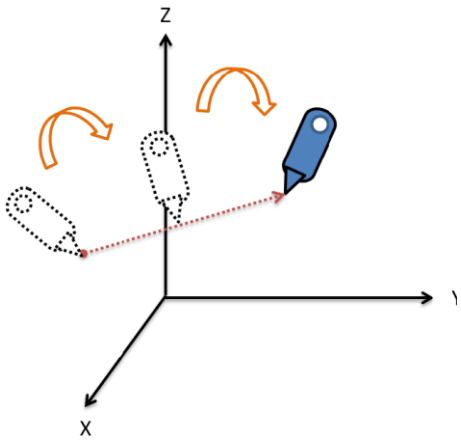
If the group mechanism type is set to a robot, and the A, B and C values are defined to the orientation angles, the below figure shows the line interpolation and orientation control of a 6-axis articulated robot:



The below figures describe the motion with/without the orientation control:



Line interpolation (without Orientation Control)



Line interpolation + Orientation Control

- The group line interpolation and orientation control develops the plan in accordance with the group parameters as follows:

Definition	Value	Description
Target position type	0x30	0: Absolute position, 1: Incremental position
Velocity plan method	0x31	0: T curve,1: S-shape curve
Maximum velocity	0x32	Maximum velocity of the velocity plan (unit/sec)
Acceleration	0x33	Acceleration of the velocity plan (unit/sec^2)
Deceleration	0x34	Deceleration of the velocity plan (unit/sec^2)
Jerk	0x35	Jerk of the velocity plan (unit/sec^3) (Effective for S-shape curve only)

- Velocity parameters of orientation control:

Definition	Value	Description

Independent interpolation of orientation angle	0x3A	0: Independent interpolation, 1: Dependent interpolation
Maximum velocity of orientation angle	0x3B	Maximum velocity of orientation angle (unit/sec)
Acceleration of orientation angle	0x3C	Acceleration of orientation angle (unit/sec^2)
Deceleration of orientation angle	0x3D	Deceleration of orientation angle (unit/sec^2)
Jerk of orientation angle	0x3E	Jerk of orientation angle (unit/sec^3) (Effective for S-shape curve only)

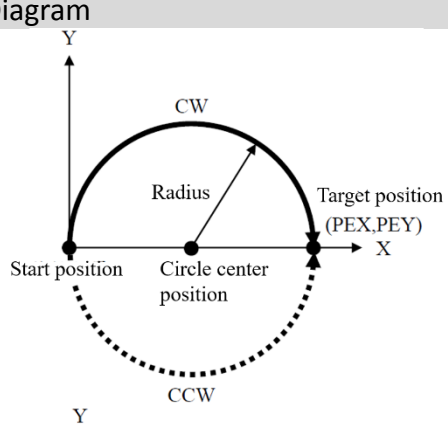
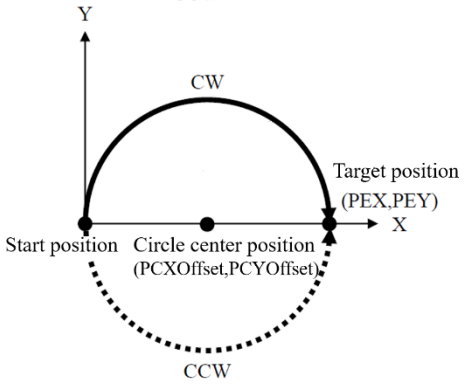
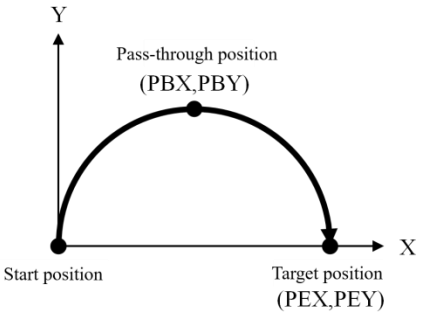
The parameter 0x3A determines the orientation angle motion uses the independent interpolation computation:

- If the 0x3A is 0 (independent interpolation), the orientation control adopts the independent interpolation for the velocity plan based on the parameters 0x3B ~ 0x3E.
- If the 0x3A is 1 (dependent interpolation), the variation of orientation control operates based on the linear motion proportionally.

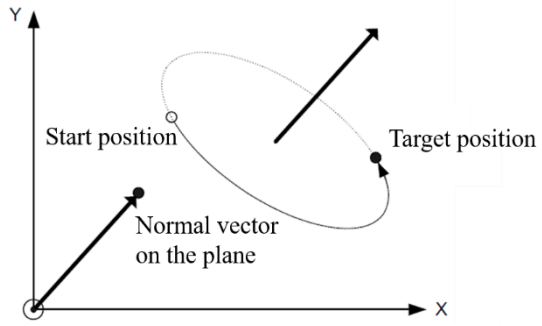
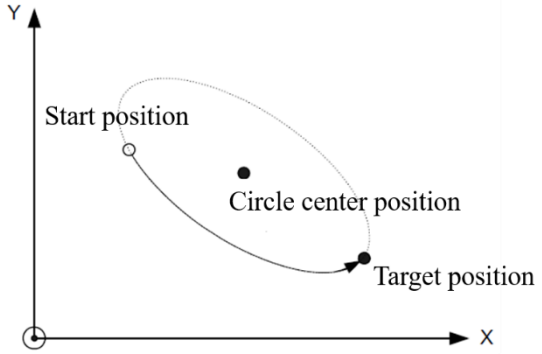
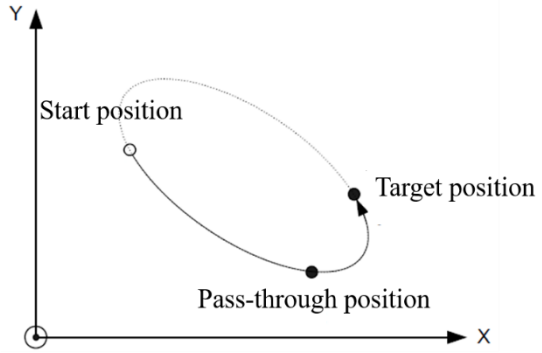
1.4.11. Group Arc Interpolation

The device supports the following methods to define the arc motion in the Cartesian coordinate system:

- 2D arc

Definition Method	API	Diagram
● End position ● Radius ● Direction	NMC_GroupCirc2R()	
● End position ● Circle center position ● Direction	NMC_GroupCirc2C()	
● End position ● Pass-through position	NMC_GroupCirc2B()	

- 3D arc

Definition Method	API	Diagram
• End position • Orientation • Radius • Direction • Normal vector on the plane	NMC_GroupCircR()	
• End position • Orientation • Circle center position • Direction	NMC_GroupCircC()	
• End position • Orientation • Pass-through position	NMC_GroupCircB()	

The parameters related to the velocity plan are described as follows:

- The arc tangent velocity parameters:

Definition	Value	Description
Target position type	0x30	0: Absolute position, 1: Incremental position
Velocity plan method	0x31	0: T curve,1: S-shape curve
Maximum velocity	0x32	Maximum velocity of the velocity plan (unit/sec)
Acceleration	0x33	Acceleration of the velocity plan (unit/sec^2)
Deceleration	0x34	Deceleration of the velocity plan (unit/sec^2)
Jerk	0x35	Jerk of the velocity plan (unit/sec^3) (Effective for S-shape curve only)

- Velocity parameters of orientation control:

Definition	Value	Description
Independent interpolation of orientation angle	0x3A	0: Independent interpolation, 1: Dependent interpolation

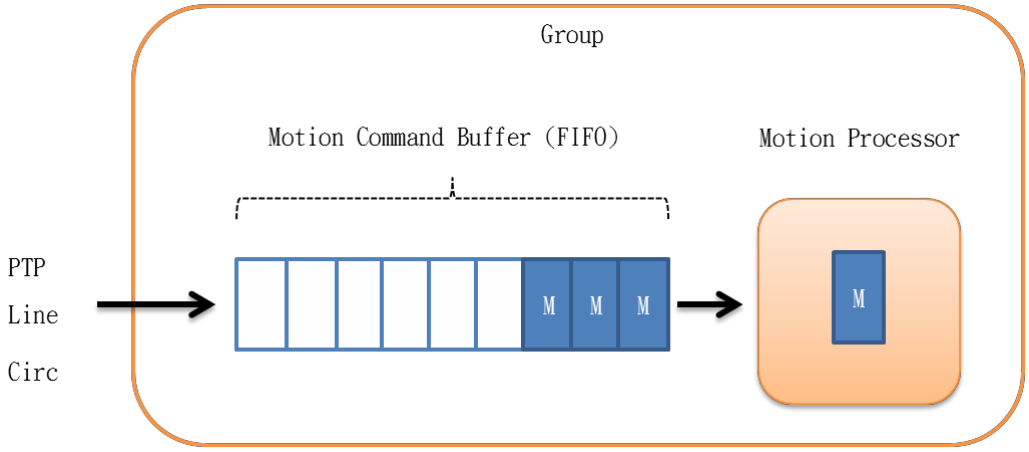
Maximum velocity of orientation angle	0x3B	Maximum velocity of orientation angle (unit/sec)
Acceleration of orientation angle	0x3C	Acceleration of orientation angle (unit/sec^2)
Deceleration of orientation angle	0x3D	Deceleration of orientation angle (unit/sec^2)
Jerk of orientation angle	0x3E	Jerk of orientation angle (unit/sec^3) (Effective for S-shape curve only)

The parameter 0x3A determines the orientation angle motion uses the independent interpolation computation:

- If the 0x3A is 0 (independent interpolation), the orientation control adopts the independent interpolation for the velocity plan based on the parameters 0x3B ~ 0x3E.
- If the 0x3A is 1 (dependent interpolation), the variation of orientation control operates based on the linear motion proportionally.

1.4.12. Connection Motion

There is a motion queue in each group, and the motion commands can be stored in the motion queue successively. The motion queue supports first-in first out (FIFO), and the device executes the motion commands stored in the motion queue in order. NMC_GroupGetMotionBuffSpace() can be called to check the residual buffer space of the motion queue, and the default buffer space of the motion queue is 32.



The motion commands which can be stored in the motion queue are listed as follows:

1. Point-to-point motion(PTP)
2. Line interpolation motion(Line)
3. Arc interpolation motion(Circ)

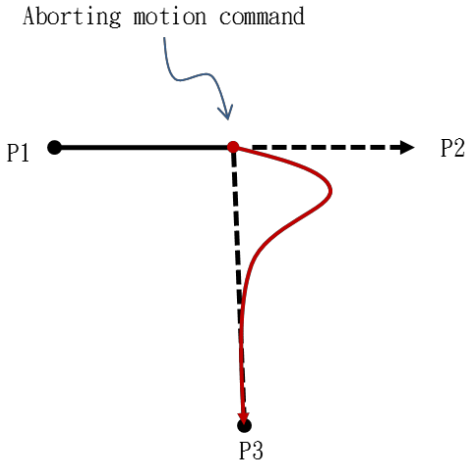
After one of the above motion commands are called, the device will determine the connection between such motion command and the current (or previous) motion command. The parameters of buffer mode are described as follows:

Group parameter 0x36:0 (Buffer mode)

Value	Buffer Mode	Description
0	Aborting	Stop the current motion and then execute the new motion command immediately.
1	Buffered	Store the new motion command in the motion queue and execute such command after the previous motion is completed.
2	Blending	Connect the new and the previous motion commands with smooth trajectory and velocity. The smooth level is determined based on the configurations in the group parameters 0x36:1~3.

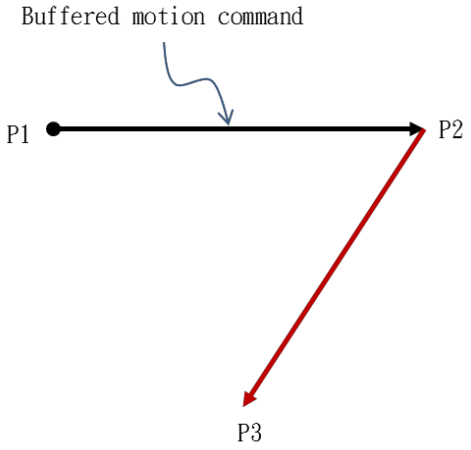
1.4.12.1.Buffer Mode: Aborting

The default value of the parameter is the Aborting mode. That is, after a new motion command is set, the current motion will be stopped, and the new motion commands will be executed immediately. Moreover, all the motion commands stored in the motion queue will also be cleared. The below figure shows the Aborting mode:



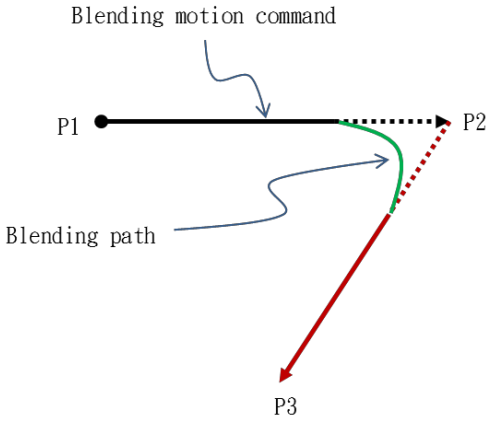
1.4.12.2.Buffer Mode: Buffered

If the buffer mode is set to buffered, the new motion command will be stored in the motion queue. The motion command will be executed after the previous motion command is completed, and the previous motion command velocity will decrease to 0. The mode is usually used to ensure the trajectory will pass through the target point. The below figure shows the Buffered mode:



1.4.12.3.Buffer Mode: Blending

If the path connection is set to the Blending mode, the trajectory and velocity of two successive motions will be connected. The mode can enhance the smoothness of processing trajectory, raise the motion velocity and reduce the processing time. However, the connection path may not pass the target point. The below figure shows the Blending mode:



The configurations of the buffer mode include BlendingLow, BlendingPrevious, BlendingNext and BlendingHigh. The connection path can be achieved by configuring the transition mode (group

parameter 0x36 Sub 1) for the connection method and the transition parameter (group parameter 0x36 Sub 2) to fine tune the behavior in connection. These parameters are described as the below table:

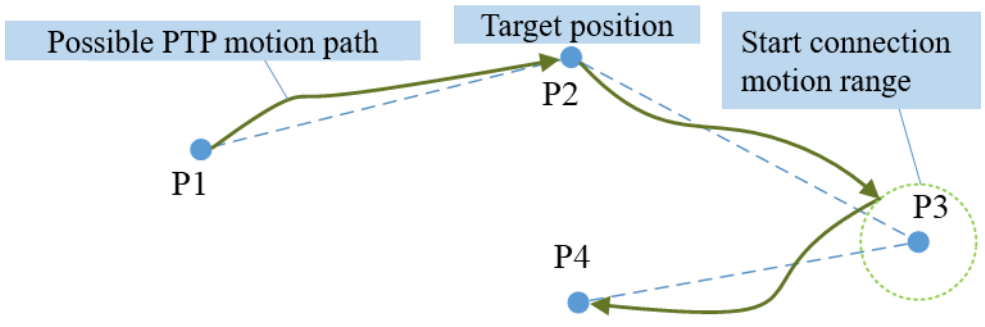
Transition Mode (group parameter 0x36 Sub 1):

Value	Transition Mode	Description
0	None	The velocity of the connection motion is determined by the buffer mode, including BlendingLow, BlendingPrevious, BlendingNext, BlendingHigh and Blending.
1	Start Velocity	The velocity of the connection motion is determined by the velocity percentage of the current motion.
2	Constant Velocity	The velocity of the connection motion is determined by the velocity percentage of the next motion.
3	Corner Distance	The velocity of the connection motion is determined by the distance of corner.
4	Corner Deviation	The velocity of the connection motion is determined by the deviation of corner.

Transition parameter (group parameter 0x36 Sub 2):
The parameter can be set to the velocity percentage, corner distance or corner deviation in accordance with the configurations of the buffer mode and transition mode.

Example: The connection motion between point-to-point (PTP) motions.
The below table shows a connection motion command:

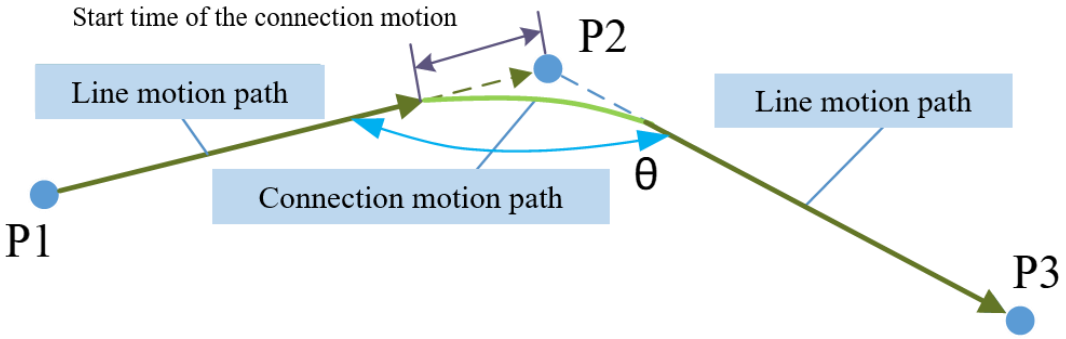
Order	Motion Command	Buffer Mode	Blending Mode	Blending Parameter
1	PTP(P2)	Ignored (No current motion)	Ignored (No current motion)	Ignored (No current motion)
2	PTP(P3)	Buffered	Ignored	Ignored
3	PTP(P4)	Buffered	Corner Distance	Distance Value



If the PTP(P3) does not enable the connection motion, the previous motion will reach the position P2 precisely. If the PTP(P4) enables the connection motion and the interpolation point is near the position P3 (the specified range of corner distance), the command position (P4) will be set immediately and the final velocity will not be decreased to 0.

Example: The connection motion between two-line interpolation motions:

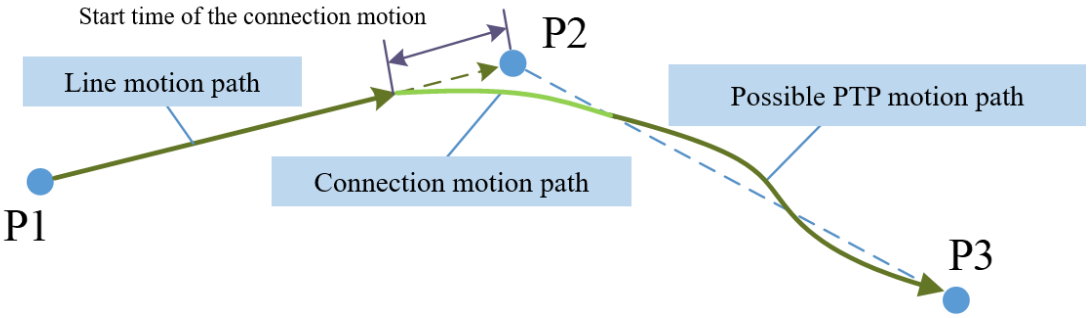
Order	Motion Command	Buffer Mode	Blending Mode	Blending Parameter
1	Line(P2)	Ignored (No current motion)	Ignored (No current motion)	Ignored (No current motion)
2	Line (P3)	Blending	Corner Distance	Distance Value



If the Line(P3) enables the path connection mode (Buffer mode = Blending) and the interpolation point reaches the specified corner distance, the device will smooth the velocity and path of Line(P2) and Line(P3). The final velocity will not be decreased to 0, and the path plan will not pass through the point P2.

Example: The connection motion between a line interpolation (Line) motion and a point-to-point (PTP) motion:

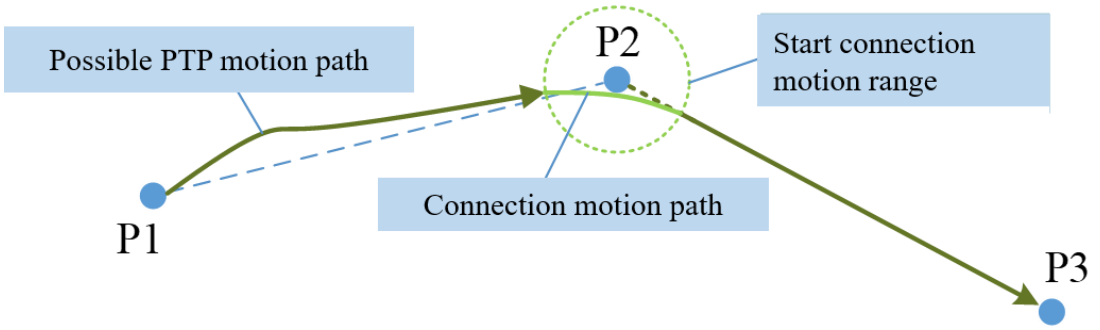
Order	Motion Command	Buffer Mode	Blending Mode	Blending Parameter
1	Line(P2)	Ignored (No current motion)	Ignored (No current motion)	Ignored (No current motion)
2	PTP(P3)	Blending	Corner Distance	Distance Value



If the PTP(P3) enables the path connection mode (Buffer mode = Blending) and the interpolation point reaches the specified corner distance, the device will smooth the velocity and path of Line(P2) and PTP(P3). The final velocity will not be decreased to 0, and the path plan will not pass through the point P2.

Example: The connection motion between a point-to-point (PTP) motion and a line interpolation (Line) motion:

Order	Motion Command	Buffer Mode	Blending Mode	Blending Parameter
1	PTP(P2)	Ignored (No current motion)	Ignored (No current motion)	Ignored (No current motion)
2	Line(P3)	Blending	Corner Distance	Distance Value



If the Line(P3) enables the path connection mode (Buffer mode = Blending) and the interpolation point reaches the specified corner distance, the device will smooth the velocity and path of PTP(P2) and Line(P3). The final velocity will not be decreased to 0, and the path plan will not pass through the point P2.

The connection of various motion commands shall follow the corresponding rules. Please refer to the below table for these rules (○ indicates supported, and ✕ indicates unsupported).

Line interpolation - line interpolation (or line interpolation - arc interpolation / arc interpolation - line interpolation / arc interpolation - arc interpolation)

Buffer Mode / Transition Mode	Aborting	Buffered	Blending Low	Blending Previous	Blending Next	Blending High	Blending
None	○	○	○	○	○	○	○
Start Velocity	○	✕	○	○	✕	○	✕
Constant Velocity	○	✕	○	✕	○	○	✕
Corner Distance	✕	✕	✕	✕	✕	✕	○
Corner Deviation	✕	✕	✕	✕	✕	✕	○

Line interpolation - point-to-point motion (or arc interpolation - point-to-point motion)

Buffer Mode / Transition Mode	Aborting	Buffered	Blending Low	Blending Previous	Blending Next	Blending High	Blending
None	○	○	○	○	○	○	○
Start Velocity	✕	✕	✕	✕	○	✕	✕
Constant Velocity	✕	✕	✕	✕	✕	✕	✕
Corner Distance	✕	✕	○	✕	✕	✕	✕
Corner Deviation	✕	✕	✕	✕	✕	✕	✕

Point-to-point motion – line interpolation (or point-to-point motion - arc interpolation)

Buffer Mode / Transition Mode	Aborting	Buffered	Blending Low	Blending Previous	Blending Next	Blending High	Blending
None	○	○	○	×	×	×	×
Start Velocity	×	×	×	×	×	×	×
Constant Velocity	×	×	×	×	×	×	×
Corner Distance	×	×	○	×	×	×	×
Corner Deviation	×	×	×	×	×	×	×

Point-to-point motion – point-to-point motion

Buffer Mode / Transition Mode	Aborting	Buffered	Blending Low	Blending Previous	Blending Next	Blending High	Blending
None	○	○	○	×	×	×	×
Start Velocity	×	×	×	×	×	×	×
Constant Velocity	×	×	×	×	×	×	×
Corner Distance	×	×	○	×	×	×	×
Corner Deviation	×	×	×	×	×	×	×

Sample version (All)

Buffer Mode / Transition Mode	Aborting	Buffered	Blending Low	Blending Previous	Blending Next	Blending High	Blending
None	○	○	○	○	○	○	○
Start Velocity	○	×	○	○	×	○	×
Constant Velocity	○	×	○	×	○	○	×
Corner Distance	×	×	×	×	×	×	○
Corner Deviation	×	×	×	×	×	×	○

1.4.13. Group Homing Motion

The functions of the group homing motion and the axis homing motion are the same. Please refer to the section, Axis Homing Motion, for the related usage and timing. The functions related to the group homing motion are described as follows:

- NMC_GroupSetHomePos():
  The (ACS) coordinate of each axis shall be set for the position of such axis.

- NMC_GroupAxesHomeDrive():
  The group axis to execute the axis homing motion can be set. After the function is called successfully, the specified group axis will execute the corresponding axis homing process in accordance with the related parameters. The group state is HOMING in case an axis is still in homing. During the homing process, the group state will transfer to ERROR in case an error occurs at an axis.

2. Device Parameters

2.1. System Parameters

Param. Num.	Sub. Index	Data Type	Description	Range	Remark
0x00	0	I32_T	Motion cycle time (us).	1000~4000 microsecond	(*1)(*3)
0x01	0	I32_T	Number of activated axis	0~64	(*1)
0x02	0	I32_T	Number of activated group	0~64	(*1)

(*1): The parameter is effective after the system is started. During the system starting, the parameter cannot be modified.

(*2): The parameter is effective after the function is enable. The modification in another period cannot be effective immediately.

(*3): The configurable range is depended on the controller.

(*4): Read only.

2.2. Axis Parameters

Param. Num.	Sub. Index	Data Type	Description	Range	Remark
0x00	0	I32_T	Mechanical pitch (unit/rev)	1 ~ 2147483647	(*1)
0x01	0	I32_T	Mechanical revolution	1 ~ 2147483647	(*1)
0x02	0	I32_T	Motor revolution	1 ~ 2147483647	(*1)
0x03	0	I32_T	Encoder resolution (pulse/rev)	1 ~ 2147483647	(*1)
0x04	0	I32_T	Encoder direction	0:Not inverse, 1:Inverse	(*1)
0x05	0	I32_T	Encoder type	0:Incremental, 1:Absolute	(*1)
0x06	0	I32_T	Enable the encoder	0:Disable, 1:Enable	(*1)
0x06	1	F64_T	External Encoder ratio	(0+) ~ 2147483647	(*1)
0x07	0	I32_T	Cancel synchronized actual position to command position when servo enable	0:Not Cancel, 1:Cancel	(*1)
0x08	0	F64_T	Position offset	-2147483648 ~ 2147483647	(*1)
0x0A	0	I32_T	Motor ID assignment	0~63, -1: No assignment	(*1)
0x10	0	F64_T	Positive software limit (user unit)	-2147483648 to 2147483647	(*2)
	1	I32_T	Enable positive software limit	0:Disable, 1:Enable	
	2	F64_T	Negative software limit (user unit)	-2147483648 ~ 2147483647	(*2)
	3	I32_T	Enable negative software limit	0:Disable, 1:Enable	
0x11	0	F64_T	Maximum velocity limit (unit/sec)	1 ~ 2147483647	(*2)
	1	I32_T	Enable max. velocity limit	0:Disable, 1:Enable	
0x12	0	F64_T	Maximum acceleration limit (unit/sec^2)	1 ~ 2147483647	(*2)
	1	I32_T	Enable max. acceleration limit	0:Disable, 1:Enable	
0x20	0	I32_T	Profile type for AxisStop command	0: T-Curve, 1:S-Curve	
0x21	0	F64_T	Deceleration for AxisStop command (unit/sec^2)	1 ~ 2147483647	
0x22	0	F64_T	Jerk for AxisStop command (unit/sec^3)	1 ~ 2147483647	
0x28	0	F64_T	Base velocity (unit/sec)	0 ~ 2147483647	
0x30	0	I32_T	Absolute or relative programming	0: Absolute, 1:Relative	
0x31	0	I32_T	Profile type	0: T-Curve, 1:S-Curve	
0x32	0	F64_T	Max. velocity (unit/sec)	1 ~ 2147483647	
0x33	0	F64_T	Acceleration (unit/sec^2)	1 ~ 2147483647	
0x34	0	F64_T	Deceleration (unit/sec^2)	1 ~ 2147483647	
0x35	0	F64_T	Jerk (unit/sec^3)	1 ~ 2147483647	
0x36	0	I32_T	Buffer mode	0:Aborting 1:Buffered 2:BlendingLow 3:BlendingPrevious 4:BlendingNext 5:BlendingHigh	
0x80	0	I32_T	Number of home parameters	5	(*3)(*4)
	1	I32_T	EtherCAT CiA HOME method	Refer to drive user manual	(*5)
	2	F64_T	EtherCAT CiA HOME velocity search switch	Refer to drive user manual	(*5)
	3	F64_T	EtherCAT CiA HOME velocity search zero	Refer to drive user manual	(*5)
	4	F64_T	EtherCAT CiA HOME acceleration	Refer to drive user manual	(*5)
	5	F64_T	EtherCAT CiA HOME offset	Refer to drive user manual	(*5)

(*1): The parameter is effective after the system is started. During the system starting, the parameter cannot be modified and will return error.

(*2): The parameter is effective after the function is enable. The modification in another period cannot be effective immediately.

(*3): The configurable range is depended on the controller.

(*4): Read only.

(*5): The configurable range is depended on the controlled device.

2.3. Group Parameters

Num.	Sub	Type	Description	Value Description	Common
0x00	0	I32_T	Kinematics type	0:Linear (2~8 Axes) 1:Articulated Robot (AR6) 2:Delta 3:SCARA	(*1)
	1	F64_T	Kinematics parameter 1~48	Refer to kinematics chapter	(*1)
0x01	0	I32_T	Mechanical couple master axis	0~7 group axis	(*1)
	1	I32_T	Mechanical couple slave axis	0~7 group axis	(*1)
	2	I32_T	Compensation source type	0:Actual position, 1:Command position	(*1)
	3	F64_T	Reference master original position	unit, Value -2147483648 to 2147483647	(*1)
	4	F64_T	Coupling ratio numerator	Value != 0 (Negative value represents the opposite direction compensation)	(*1)
	5	F64_T	Coupling ratio denominator	Value > 0	(*1)
	6	I32_T	Enable mechanical couple compensation	0:Disable, 1:Enable	(*1)
0x15	0	F64_T	Max. velocity reference. (PCS,TCP)	unit/sec, value = 1 ~ 2147483647	
0x16	0	F64_T	Safety velocity limit. (PCS,TCP)	unit/sec, value = 1 ~ 2147483647	(*1)
0x20	0	I32_T	STOP profile type	0: T-Curve, 1:S-Curve	
0x21	0	F64_T	STOP deceleration	unit/sec^2,value = 1 ~ 2147483647	
0x22	0	F64_T	STOP jerk	unit/sec^3,value = 1 ~ 2147483647	
0x23	0	I32_T	Halt buffer mode	0:Aborting, 1:Buffered	
0x30	0	I32_T	Cart. relative positioning	0: Absolute positioning, 1:Relative Positioning	
0x31	0	I32_T	Cart. profile type	0: T-Curve, 1:S-Curve	
0x32	0	F64_T	Cart. max. velocity	unit/sec, value = 1 ~ 2147483647	
0x33	0	F64_T	Cart. acceleration	unit/sec^2,value = 1 ~ 2147483647	
0x34	0	F64_T	Cart. deceleration	unit/sec^2,value = 1 ~ 2147483647	
0x35	0	F64_T	Cart. jerk	unit/sec^3,value = 1 ~ 2147483647	
0x36	0	I32_T	Buffer mode	0:Aborting, 1:Buffered, 2:Blending	
	1	I32_T	Blending mode	0: Corner distance 1: MaxCorner deviation	
	2	F64_T	Blending corner distance	pos unit, Value 0 to 2147483647	
	3	F64_T	Blending corner deviation	pos unit, Value 0 to 2147483647	
0x3A	0	I32_T	Orientation motion independent	0: Not independent 1:Independent	
0x3B	0	F64_T	Max. orientation velocity	unit/sec, value = 1 ~ 2147483647	
0x3C	0	F64_T	Orientation acceleration	unit/sec^2,value = 1 ~ 2147483647	
0x3D	0	F64_T	Orientation deceleration	unit/sec^2,value = 1 ~ 2147483647	
0x3E	0	F64_T	Orientation jerk	unit/sec^3,value = 1 ~ 2147483647	

0x40	0	I32_T	Tool index selection for motion target	value = -1 :No tool assignment (Flange) value = 0~15: tool 0~15	
	1	I32_T	Tool index selection for read position	value = -2 : Parameter disable value = -1 :No tool assignment (Flange) value = 0~15: tool 0~15	
0x48	0	I32_T	Base index selection for motion target	value = -1 :No base assignment (MCS) value = 0~31: base 0~31	
	1		Base index selection for read position	value = -2 : Parameter disable value = -1 :No base assignment value = 0~31: base 0~31	
0x80~8F	0	F64_T	Offset along flange x-axis	unit, Value -2147483648 to 2147483647	
	1	F64_T	Offset along flange y-axis	unit, Value -2147483648 to 2147483647	
	2	F64_T	Offset along flange z-axis	unit, Value -2147483648 to 2147483647	
	3	F64_T	Rotation angle about flange z-axis	degree	
	4	F64_T	Rotation angle about flange y-axis	degree	
	5	F64_T	Rotation angle about flange x-axis	degree	
0xC0~DF	0	F64_T	Offset along reference base x-axis	unit, Value -2147483648 to 2147483647	
	1	F64_T	Offset along reference base y-axis	unit, Value -2147483648 to 2147483647	
	2	F64_T	Offset along reference base z-axis	unit, Value -2147483648 to 2147483647	
	3	F64_T	Rotation angle about reference base z-axis	degree	
	4	F64_T	Rotation angle about reference base y-axis	degree	
	5	F64_T	Rotation angle about reference PCS x-axis	degree	
	6	I32_T	Reference base index	value = -1: reference to MCS value = 0~31:reference to base 0~31	

(*1): The parameter is effective after the system is started. During the system starting, the parameter cannot be modified and will return error.

(*2): The parameter is effective after the function is enable. The modification in another period cannot be effective immediately.

3. C/C++ Library

3.1. API Overview

The APIs of NexMotion Library are listed in the following tables, and defined in the header file, NexMotion.h.

Version and error information functions

Name	Description
NMC_GetLibVersion	Return the version number of the Library
NMC_GetLibVersionString	Return the version number of the Library in the format of a C string
NMC_GetErrorDescription	Return the error code description in the format of a C string

Device open up and shut down

Name	Description
NMC_DeviceOpenUp	Open up the device (Blocking call)
NMC_DeviceShutdown	Shut down the device (Blocking call)
NMC_DeviceOpenUpRequest	Request the device open up (Non-blocking call)
NMC_DeviceWaitOpenUpRequest	Wait for device open up (Blocking call)
NMC_DeviceShutdownRequest	Request the device shut down (Non-blocking call)
NMC_DeviceWaitShutdownRequest	Wait for device shut down (Blocking call)

Advanced device open up and shut down

Name	Description
NMC_DeviceCreate	Get the device ID
NMC_DeviceDelete	Delete the device ID
NMC_DeviceLoadIniConfig	Load the device configurations
NMC_DeviceResetConfig	Reset the device configurations
NMC_DeviceStart	Start the device (Blocking call)
NMC_DeviceStop	Stop the device (Blocking call)
NMC_DeviceStartRequest	Request the device start (Non-blocking call)
NMC_DeviceStopRequest	Request the device stop (Non-blocking call)
NMC_DeviceGetState	Get the device state

Watch dog functions

Name	Description
NMC_DeviceWatchdogTimerEnable	Enable the watch dog timer
NMC_DeviceWatchdogTimerDisable	Disable the watch dog timer

NMC_DeviceWatchdogTimerReset	Reset the watch dog timer

System configuration functions

Name	Description
NMC_DeviceSetParam	Set device parameters
NMC_DeviceGetParam	Get device parameters
NMC_SetIniPath	Set the path of ini file

I/O control functions

Name	Description
NMC_GetInputMemorySize	Get the size of mapped input memory
NMC_GetOutputMemorySize	Get the size of mapped output memory
NMC_ReadInputMemory	Read the mapped input memory
NMC_ReadOutputMemory	Read the mapped output memory
NMC_WriteOutputMemory	Write the mapped output memory
NMC_ReadInputBit	Read the digital input (Input) map memory bit value
NMC_ReadInputI8	Read the digital input (Input) map memory byte value
NMC_ReadInputI16	Read the digital input (Input) mapping memory word value
NMC_ReadInputI32	Read the digital input (Input) mapping memory double-word value
NMC_ReadOutputBit	Read the digital output (Output) map memory bit value
NMC_ReadOutputI8	Read the digital output (Output) map memory byte value
NMC_ReadOutputI16	Read the digital output (Output) mapping memory word value
NMC_ReadOutputI32	Read the digital output (Output) mapping memory double-word value
NMC_WriteOutputBit	Write digital output (Output) map memory bit value
NMC_WriteOutputI8	Write digital output (Output) map memory byte value
NMC_WriteOutputI16	Write digital output (Output) map memory word value
NMC_WriteOutputI32	Write the digital output (Output) mapping memory double-word value

Axis or group quantity

Name	Description
NMC_DeviceGetAxisCount	Get the axis quantity
NMC_DeviceGetGroupCount	Get the group quantity
NMC_DeviceGetGroupAxisCount	Get the quantity of group axis

Read axis/group description

Name	Description
NMC_AxisGetDescription	Read the name description information of the specified axis
NMC_GroupGetDescription	Read the name description information of the specified group

Enable and disable functions for all axes and groups

Name	Description
NMC_DeviceEnableAll	Enable all axes and groups in the system
NMC_DeviceDisableAll	Disable all axes and groups in the system

Halt and stop functions for all axes and groups

Name	Description
NMC_DeviceHaltAll	Halt all axes and groups in the system (Stand still state)
NMC_DeviceStopAll	Stop all axes and groups in the system (Stopped state)

System information

Name	Description
NMC_MessagePopFirst	Read system message queue
NMC_MessageOutputEnable	Transfer a copy of message to MS Windows system message

Function trace functions

Name	Description
NMC_DebugSetTraceMode	Set API trace mode
NMC_DebugSetHookData	Set the data structure index for the hook function
NMC_DebugSetHookFunction	Set the hook function
NMC_DebugGetApiAddress	Read the API address

Axis configuration functions

Name	Description
NMC_AxisSetParamI32	Set the axis parameters (I32_T, data type)
NMC_AxisGetParamI32	Get the axis parameters (I32_T, data type)
NMC_AxisSetParamF64	Set the axis parameters (F64_T, data type)
NMC_AxisGetParamF64	Get the axis parameters (F64_T, data type)

Axis state control functions

Name	Description
NMC_AxisEnable	Enable an axis

NMC_AxisDisable	Disable an axis
NMC_AxisGetStatus	Get the axis status
NMC_AxisGetState	Get the axis state
NMC_AxisResetState	Reset the axis state
NMC_AxisResetDriveAlm	Reset the axis servo alarm

Axis motion status functions

Name	Description
NMC_AxisGetCommandPos	Get the command position of axis
NMC_AxisGetActualPos	Get the actual position of axis
NMC_AxisGetCommandVel	Get the command velocity of axis
NMC_AxisGetActualVel	Get the actual velocity of axis
NMC_AxisGetMotionBuffSpace	Get the size of buffer space of axis motion command

Axis motion control functions

Name	Description
NMC_AxisPtp	Enable the point-to-point motion of axis
NMC_AxisJog	Enable the JOG motion of axis

Axis returns to home functions

Name	Description
NMC_AxisSetHomePos	Set the origin of an axis
NMC_AxisHomeDrive	Drive an axis to move to the origin (by drive).

Axis motion status functions

Name	Description
NMC_AxisHalt	Halt an axis (Stand still state)
NMC_AxisStop	Stop an axis (Stopped state)
NMC_AxisHaltAll	Halt all axes (Stand still state)
NMC_AxisStopAll	Stop all axes (Stopped state)

Axis motion change functions

Name	Description
NMC_AxisVelOverride	Change the velocity for an axis in motion
NMC_AxisAccOverride	Change the acceleration for an axis in motion
NMC_AxisDecOverride	Change the deceleration for an axis in motion

Gross axis velocity ratio configuration functions

Name	Description
NMC_AxisSetVelRatio	Set the velocity ratio of an axis
NMC_AxisGetVelRatio	Get the velocity ratio of an axis

Group configuration functions

Name	Description
NMC_GroupSetParamI32	Set the group parameters (I32_T, data type)
NMC_GroupGetParamI32	Get the group parameters (I32_T, data type)
NMC_GroupSetParamF64	Set the group parameters (F64_T, data type)
NMC_GroupGetParamF64	Get the group parameters (F64_T, data type)
NMC_GroupAxSetParamI32	Set the parameters of group axis (I32_T, data type)
NMC_GroupAxGtParamI32	Get the parameters of group axis (I32_T, data type)
NMC_GroupAxSetParamF64	Set the parameters of group axis (F64_T, data type)
NMC_GroupAxGetParamF64	Get the parameters of group axis (F64_T, data type)

Group state control functions

Name	Description
NMC_GroupEnable	Enable a group
NMC_GroupDisable	Disable a group
NMC_GroupGetStatus	Get the group status
NMC_GroupGetState	Get the group state
NMC_GroupResetState	Reset the group state
NMC_GroupResetDriveAlm	Reset the group servo alarm
NMC_GroupResetDriveAlmAll	Reset all group servo alarms

Gross group velocity ratio configuration functions

Name	Description
NMC_GroupSetVelRatio	Set the velocity percentage of a group
NMC_GroupGetVelRatio	Get the velocity percentage of a group

Group point-to-point motion functions (axis coordinate system)

Name	Description
NMC_GroupPtpAcs	Enable the point-to-point motion of group axis
NMC_GroupPtpAcsAll	Enable the point-to-point motion of multiple group axes

Group axis JOG motion functions (axis coordinate system)

Name	Description
NMC_GroupJogAcs	Enable the JOG motion of group axis

Group JOG motion functions (Cartesian coordinate system)

Name	Description
NMC_GroupJogTcpFrame	Start a group jog motion along the TCP coordinates
NMC_GroupJogPcsFrame	Start a group jog motion along the PCS coordinates

Group point-to-point motion functions (Cartesian coordinate system)

Name	Description
NMC_GroupPtpCart	Enable the point-to-point motion of group axis
NMC_GroupPtpCartAll	Enable the point-to-point motion of multiple group axes

Group halt and stop functions

Name	Description
NMC_GroupHalt	Halt a group (Stand still state)
NMC_GroupStop	Stop a group (Stopped state)
NMC_GroupHaltAll	Halt all groups (Stand still state)
NMC_GroupStopAll	Stop all groups (Stopped state)

Group motion status functions

Name	Description
NMC_GroupGetCommandPosAcs	Get the command position of a group in axis coordinate system (ACS)
NMC_GroupGetActualPosAcs	Get the actual position of a group in axis coordinate system (ACS)
NMC_GroupGetCommandPosPcs	Get the command position of a group in programming coordinate system (PCS)
NMC_GroupGetActualPosPcs	Get the actual position of a group in programming coordinate system (PCS)
NMC_GroupGetMotionBuffSpace	Get the free space of motion command buffer of a group

Group returns to home functions

Name	Description
NMC_GroupSetHomePos	Set the origin of a group axis in the axis coordinate system
NMC_GroupAxesHomeDrive	Drive a group to move to the origin.

Group 2D line or arc interpolation motion functions

Name	Description
NMC_GroupLineXY	Group 2D linear interpolation
NMC_GroupCirc2R	Group 2D arc interpolation with end point and radius
NMC_GroupCirc2C	Group 2D arc interpolation with end point and center
NMC_GroupCirc2B	Group 2D arc interpolation with end point and border point

Group 3D line or arc interpolation motion functions

Name	Description
NMC_GroupLine	Group 3D linear interpolation
NMC_GroupCircR	Group 3D arc interpolation with end point and radius
NMC_GroupCircC	Group 3D arc interpolation with end point and center
NMC_GroupCircB	Group 3D arc interpolation with end point and border point

Tool teaches related functions

Name	Description
NMC_ToolCalib_4p	Tool teaching - TCP translation
NMC_ToolCalib_4pWithZ	Tool teaching - TCP translation and Z direction setting
NMC_ToolCalib_4pWithOri	Tool teaching - TCP translation and orientation setting
NMC_ToolCalib_Ori	Tool teaching - TCP orientation setting

Base teaches related functions

Name	Description
NMC_BaseCalib_1p	Base teaching -1p method
NMC_BaseCalib_2p	Base teaching -2p method
NMC_BaseCalib_3p	Base teaching -3p method

3D simulation functions

Name	Description
NMC_Group3DShow	Create a 3D simulation dialog or show the dialog
NMC_Group3DHide	Hide the 3D simulation dialog
NMC_Group3DAlwaysTop	Set the 3D simulation window to always be at the top
NMC_Group3DDrawPath	Enable/Disable the function draw TCP path in 3D dialog
NMC_Group3DClearPath	Clear all TCP path in 3D simulation dialog

3.2. System APIs

3.2.1. Version and Error Information Functions

3.2.1.1. NMC_GetLibVersion

Return the version number of the Library: Major.Minor.Stage.Build

- C/C++ Syntax Definition:

I32_T NMC_GetLibVersion(I32_T *PRetMajor, I32_T *PRetMinor, I32_T *PRetStage, I32_T *PRetBuild);

- Parameters:

I32_T *PRetMajor: [Input] A pointer variable, [Output] The major version number. Input NULL(0) can ignore the parameter.
I32_T *PRetMinor: [Input] A pointer variable, [Output] The minor version number. Input NULL(0) can ignore the parameter.
I32_T *PRetStage: [Input] A pointer variable, [Output] The stage of version number. Input NULL(0) can ignore the parameter. The return value 1 ~ 4 indicate the trial versions, and the 5 indicates the official release version.
I32_T *PRetBuild: [Input] A pointer variable, [Output] The build of version number. Input NULL(0) can ignore the parameter.

- Return values:

The version number will be returned with the data type of I32_T. The return value means:
Version = (Major×10,000,000) + (Minor×100,000) + (Stage×10,000) + Build

- Usage:

- Examples:

```
U32_T     version;
I32_T             major, minor, stage, build;

version = NMC_GetLibVersion( &major, &minor, &stage, &build );
printf( "Library version = %d, (%d,%d,%d,%d) \n", version, major, minor, stage, build
);
```

- Reference:

None.

3.2.1.2. NMC_GetLibVersionString

Return the version number of the Library in the format of a C string: Major.Minor.Stage.Build

- C/C++ Syntax Definition:

void NMC_GetLibVersionString(char *PRetVersionString, U32_T StringSize);

- Parameters:

char *PRetVersionString: A C string which is 32 characters at least
U32_T StringSize: The size of the input C string

- Return values:

None.

- Usage:

- Examples:

```
char        versionString[32];

NMC_GetLibVersionString( versionString, 32 );
printf( "Library version = (%s) \n", versionString );
```

- Reference:

None.

3.2.1.3. NMC_GetErrorDescription

Return the error code description in the format of a C string

- C/C++ Syntax Definition:

const char* NMC_GetErrorDescription(RTN_ERR ErrorCode, _opt_null_ char *PRetErrorDesc, U32_T StringSize);

- Parameters:

RTN_ERR ErrorCode: The error code to be inquired.
_opt_null_ char *PRetErrorDesc: A char array which is 512 characters at least
U32_T StringSize: The size of the input char array

- Return values:

Const char* : The error code description. It will be NULL if the ErrorCode is undefined.

- Usage:

After the function is called successfully, save the returned error code description.

- Examples:

- Reference:

None.

3.2.2. Device Open up and Shut Down

3.2.2.1. NMC_DeviceOpenUp

Open up the device (Blocking call).

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceOpenUp(I32_T DevType, I32_T DevIndex, I32_T *PRetDevID);

- Parameters:

I32_T DevType: The specified device type. 0: Simulator, 1: EtherCAT
I32_T DevIndex: The specified index of device which is set to 0.
I32_T *PRetDevID: [Output] The device ID (DevID) after the function is called successfully.

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

NMC_DeviceOpenUp() can be called to control a device before the other control APIs are called. The function is a blocking call. It costs a short time to process the initialization, including the loading of configuration file, and returns after the device state is in the operation state.

- Examples:

- Reference:

NMC_DeviceShutdown()

3.2.2.2. NMC_DeviceShutdown

Shut down the device (Blocking call).

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceShutdown(I32_T DevID);

- Parameters:

I32_T DevID: Device ID (DevID)

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

After the application is closed, the function shall be called to shut down the device and release system resources. After called, the function will block the application until the application is closed.

- Examples:

- Reference:

NMC_DeviceOpenUp()

3.2.2.3. NMC_DeviceOpenUpRequest

Request the device open up (Non-blocking call).

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceOpenUpRequest(I32_T DevType, I32_T DevIndex);

- Parameters:

I32_T DevType: The specified device type.
I32_T DevIndex: The specified index of device which is set to 0.

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

The function will return immediately after called. NMC_DeviceWaitOpenUpRequest() or NMC_DeviceGetState() shall be used to read the device state, in order to check if the device is in the operation state.

- Examples:

- Reference:

NMC_DeviceWaitOpenUpRequest()

3.2.2.4. NMC_DeviceWaitOpenUpRequest

Wait for device open up (Blocking call).

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceWaitOpenUpRequest(U32_T WaitMs, I32_T *PRetDevID);

- Parameters:

U32_T WaitMs: Waiting time for device open up (unit: ms). It can be set to the NMC_WAIT_TIME_INFINITE (0xFFFFFFFF) to wait for the completion of open up.
I32_T *PRetDevID: [Output] The device ID (DevID) after the function is called successfully.

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

Generally, the function is used to check if the device open up is successful and called after NMC_DeviceOpenUpRequest(). After called, the function will block the application until the device is in the operation or error state.

- Examples:

- Reference:

NMC_DeviceOpenUpRequest()

3.2.2.5. NMC_DeviceShutdownRequest

Request the device shut down (Non-blocking call).

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceShutdownRequest(I32_T DevID);

- Parameters:

I32_T DevID: Device ID (DevID)

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

After called, the function will send the request for device shut down, and the function will be returned immediately. NMC_DeviceWaitShutdownRequest() or NMC_DeviceGetState() shall be used to read the device state, in order to check if the device is not in the operation state.

- Examples:

- Reference:

3.2.2.6. NMC_DeviceWaitShutdownRequet

Wait for device shut down (Blocking call).

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceWaitShutdownRequest(I32_T DevID, U32_T WaitMs);

- Parameters:

I32_T DevID: Device ID (DevID)
U32_T WaitMs: Waiting time for device open up (unit: ms). It can be set to the NMC_WAIT_TIME_INFINITE (0xFFFFFFFF) to wait for the completion of open up.

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

Generally, the function is used to check if the device shut down is successful and called after NMC_DeviceShutdownRequest(). After called, the function will block the application until the device is in the operation or error state.

- Examples:

- Reference:

NMC_DeviceShutdownRequest()

3.2.3. Advanced Device Open up and Shut down

3.2.3.1. NMC_DeviceCreate

Get the device ID

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceCreate(I32_T DevType, I32_T DevIndex, I32_T *PRetDevID);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T DevIndex: The specified index of device which is set to 0.
I32_T *PRetDevID: [Output] The device ID (DevID) after the function is called successfully.

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

Please refer to the section, Advanced System Initialization.

- Examples:

- Reference:

3.2.3.2. NMC_DeviceDelete

Delete the device ID

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceDelete(I32_T DevID);

- Parameters:

I32_T DevID: Device ID (DevID)

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

Please refer to the section, Advanced System Initialization.

- Examples:

- Reference:

3.2.3.3. NMC_DeviceLoadIniConfig

Load the device configurations

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceLoadIniConfig(I32_T DevID);

- Parameters:

I32_T DevID: Device ID (DevID)

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

The function will load the device configurations in accordance with the file, NexMotionLibConfig.ini. The file, NexMotionLibConfig.ini, is saved in the default path, C:\Nexcom. Users must not create the file, nor modify the file name or content in order to avoid the loading error.
NMC_DeviceLoadIniConfig() will search the file, NexMotionLibConfig.ini, in the following path in order:

1. NexMotion.dll folder
2. C:\Nexcom
3. C:\Windows\System32

To specify a new path for the file, NexMotionLibConfig.ini, NMC_SetIniPath() can be called.
After NMC_DeviceLoadIniConfig() is called successfully, the device state is in the 「READY」 state and ready to start.

- Examples:

- Reference:

NMC_SetIniPath()

3.2.3.4. NMC_DeviceResetConfig

Reset the device configurations.

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceResetConfig(I32_T DevID);

- Parameters:

I32_T DevID: Device ID (DevID)

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

NMC_DeviceResetConfig() is used to reset the device configurations. After the function is called successfully, the device will return to the 「INIT」 state. If the device is in the 「OPERATION」state, the function shall not be called.

- Examples:

- Reference:

3.2.3.5. NMC_DeviceStart

Start the device (Blocking call).

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceStart(I32_T DevID);

- Parameters:

I32_T DevID: Device ID (DevID)

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

If the device state is in the 「READY」 state, the function can be called to start the device. After the function is called and returned successfully, the device is in 「OPERATION」 state.
Please refer to the section, Advanced System Initialization.

- Examples:

- Reference:

3.2.3.6. NMC_DeviceStop

Stop the device (Blocking call).

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceStop(I32_T DevID);

- Parameters:

I32_T DevID: Device ID (DevID)

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

If the device state is in the 「OPERATION」state, the function can be called to stop the device. After the function is called and returned successfully, the device is in 「READY」state.

- Examples:

- Reference:

3.2.3.7. NMC_DeviceStartRequest

Request the device start (Non-blocking call).

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceStartRequest(I32_T DevID);

- Parameters:

I32_T DevID: Device ID (DevID)

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

If the device is in the ready state, the function can be called to send the request for device start, and the function will be returned immediately. NMC_DeviceGetState(), can be used to read the device state, in order to check if the device is not in the 「OPERATION」state.

- Examples:

- Reference:

3.2.3.8. NMC_DeviceStopRequest

Request the device stop (Non-blocking call).

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceStopRequest(I32_T DevID);

- Parameters:

I32_T DevID: Device ID (DevID)

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

If the device state is in the 「OPERATION」 state, the function can be called to send the request for device stop, and the function will be returned immediately. NMC_DeviceGetState(), can be used to read the device state, in order to check if the device is not in the 「READY」 state.

- Examples:

- Reference:

3.2.3.9. NMC_DeviceGetState

Get the device state.

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceGetState(I32_T DevID, I32_T *PRetDeviceState);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T *PRetDeviceState: Return the device state after called successfully.

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

- Examples:

- Reference:

3.2.4. Watch Dog Functions

3.2.4.1. NMC_DeviceWatchdogTimerEnable

Enable the watch dog timer.

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceWatchdogTimerEnable(I32_T DevID, I32_T TimeoutMs, I32_T WDTMode);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T TimeoutMs: Timeout value for the watch dog timer. Unit: milliseconds. Range: 20~200000 ms.
I32_T WDTMode: Timeout mode. 0: Device will transfer to ready state if the device in the operation state.

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

Please refer to the section, Watch Dog Timer.

- Examples:

- Reference:

3.2.4.2. NMC_DeviceWatchdogTimerDisable

Disable the watch dog timer.

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceWatchdogTimerDisable(I32_T DevID);

- Parameters:

I32_T DevID: Device ID (DevID)

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

Please refer to the section, Watch Dog Timer.

- Examples:

- Reference:

3.2.4.3. NMC_DeviceWatchdogTimerReset

Reset the watch dog timer.

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceWatchdogTimerReset(I32_T DevID);

- Parameters:

I32_T DevID: Device ID (DevID)

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

Please refer to the section, Watch Dog Timer.

- Examples:

- Reference:

3.2.5. System Configuration Functions

3.2.5.1. NMC_DeviceSetParam

3.2.5.2. NMC_DeviceGetParam

Set/Get device parameters.

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceSetParam(I32_T DevID, I32_T ParamNum, I32_T SubIndex, I32_T ParaValue);
RTN_ERR NMC_DeviceGetParam(I32_T DevID, I32_T ParamNum, I32_T SubIndex, I32_T *PRetParaValue);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T ParamNum: Parameter number
I32_T SubIndex: Parameter sub-index
I32_T ParaValue: Value to be set
I32_T *PRetParaValue: [Output] The value to be returned after the function is called successfully.

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

Please refer to the section, System Parameters.

- Examples:

- Reference:

Please refer to the section, System Parameters.

3.2.5.3. NMC_SetIniPath

Set the path of the ini file.

- C/C++ Syntax Definition:

RTN_ERR NMC_SetIniPath(_opt_null_ const char *PIniPath);

- Parameters:

I32_T DevID: Device ID (DevID)
const char *PIniPath: The ini file path to be set. It shall be a C string, and can be set to NULL(0) to reset the path to the default value.

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

During the starting, the system will load the configurations in accordance with the file, NexMotionLibConfig.ini. Also, the function can be called to specify the ini file path. The default path is C:\NEXCOM\.

- Examples:

- Reference:

NMC_DeviceOpenUp() and NMC_DeviceOpenUpRequest()

3.2.6. I/O Control Functions

3.2.6.1. NMC_GetInputMemorySize

Get the size of mapped input(Input) memory.

- C/C++ Syntax Definition:

RTN_ERR NMC_GetInputMemorySize (I32_T DevID, U32_T *PRetSizeByte);

- Parameters:

I32_T DevID: Device ID (DevID)
U32_T *PRetSizeByte: The size of I/O output memory will be returned after the function is called successfully. Unit: byte.

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

NMC_GetInputMemorySize() is used to read the accessible range of the I/O memory in the controller (0 to the return size, in bytes). Different controllers (different models) may have different I/O memory sizes that can be controlled, so this API can be used to confirm the actual reachable range.

The function can be called after the device is started.

- Examples:

- Reference:

3.2.6.2. NMC_GetOutputMemorySize

Get the size of mapped output (Output) memory.

- C/C++ Syntax Definition:

RTN_ERR NMC_GetOutputMemorySize(I32_T DevID, U32_T *PRetSizeByte);

- Parameters:

I32_T DevID: Device ID (DevID)
U32_T *PRetSizeByte: The size of I/O output memory will be returned after the function is called successfully. Unit: byte.

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

NMC_GetOutputMemorySize () is used to read the accessible range of the I/O memory in the controller (0 to the return size, in bytes). Different controllers (different models) may have different I/O memory sizes that can be controlled, so this API can be used to confirm the actual reachable range.

The function can be called after the device is started.

- Examples:

- Reference:

3.2.6.3. NMC_ReadInputMemory

Read the mapped input (Input) memory.

- C/C++ Syntax Definition:

RTN_ERR NMC_ReadInputMemory(I32_T DevID, U32_T OffsetByte, U32_T SizeByte, void *PRetValues);

- Parameters:

I32_T DevID: Device ID (DevID)
U32_T OffsetByte: Memory offset (byte) from 0.
U32_T SizeByte: Size of the memory to be read.
void *PRetValues: The contents within the specified memory segment will be saved into the variable after the function is called successfully.

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

Please refer to the I/O Control chapter for information on how to map I/O to the controller's internal I/O memory.
This function can be called after the controller is started.

Suppose an I/O device sets its DI-0 to DI-15 to the first Byte and the second Byte of the I/O Input memory. I want to read the value of the fourth DI input (DI-3). Please refer to the following example:

- Examples:

```
RTN_ERR ret;
U32_T offsetByte = 0;
U32_T sizeByte   = 2;
U16_T diValue    = 0;
U16_T di_03;

ret = NMC_ReadInputMemory( devId, offsetByte, sizeByte, &diValue );
di_03 = ( diValue >> 3 ) & 1;
```

- Reference:

3.2.6.4. NMC_ReadOutputMemory

Read the mapped output (Output) memory.

- C/C++ Syntax Definition:

RTN_ERR NMC_ReadOutputMemory(I32_T DevID, U32_T OffsetByte, U32_T SizeByte, void *PRetValues);

- Parameters:

I32_T DevID: Device ID (DevID)
U32_T OffsetByte: Memory offset (byte) from 0.
U32_T SizeByte: Size of the memory to be read.
void *PRetValues: The contents within the specified memory segment will be saved into the variable after the function is called successfully.

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

Please refer to the I/O Control chapter for information on how to map I/O to the controller's internal I/O memory.
This function can be called after the controller is started.

Suppose an I/O device sets its DO-0 to DO-15 to the first Byte and the second Byte of the I/O output memory. I want to read the value of the fourth DO output (DO-3). Please refer to the following example:

- Examples:

```
RTN_ERR ret;
U32_T offsetByte = 0;
U32_T sizeByte   = 2;
U16_T doValue    = 0;
U16_T do_03;

ret = NMC_ReadOutputMemory( devId, offsetByte, sizeByte, &doValue );
do_03 = ( doValue >> 3 ) & 1;
```

- Reference:

3.2.6.5. NMC_WriteOutputMemory

Write the mapped output (Output) memory.

- C/C++ Syntax Definition:

RTN_ERR NMC_WriteOutputMemory(I32_T DevID, U32_T OffsetByte, U32_T SizeByte, const void *PValues);

- Parameters:

I32_T DevID: Device ID (DevID)
U32_T OffsetByte: Memory offset (byte) from 0.
U32_T SizeByte: Size of the memory to be read.
void *PRetValues: The contents within the specified memory segment will be saved into the variable after the function is called successfully.

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

Please refer to the I/O Control chapter for information on how to map I/O to the controller's internal I/O memory.
This function can be called after the controller is started.

Suppose an I/O device has its DO-0 to DO-15 set to the first Byte and the second Byte of the I/O output memory. I want to set the value of the fourth output point (DO-3). Refer to the following example:

- Examples:

```
RTN_ERR ret;
U32_T offsetByte = 0;
U32_T sizeByte   = 2;
U16_T doValue    = 0;
U16_T do_03;

NMC_ReadOutputMemory( devId, offsetByte, sizeByte, &doValue );
doValue = doValue | ( ( 1 << 3 ) ); // Set bit 3 ON
//doValue = doValue & (~( 1 << 3 ) ); // Set bit 3 OFF
NMC_WriteOutputMemory( devId, offsetByte, sizeByte, &doValue );
```

- Reference:

3.2.6.6. NMC_ReadInputBit
3.2.6.7. NMC_ReadInputI8
3.2.6.8. NMC_ReadInputI16
3.2.6.9. NMC_ReadInputI32

Read the mapped input memory by bit, word or dword.

- C/C++ Syntax Definition:

RTN_ERR NMC_ReadInputBit(I32_T DevID, U32_T OffsetByte, U32_T BitIndex, BOOL_T *PRetBitValue);
RTN_ERR NMC_ReadInputI8(I32_T DevID, U32_T OffsetByte, I8_T *PRetI8Value);
RTN_ERR NMC_ReadInputI16(I32_T DevID, U32_T OffsetByte, I16_T *PRetI16Value);
RTN_ERR NMC_ReadInputI32(I32_T DevID, U32_T OffsetByte, I32_T *PRetI32Value);

- Parameters:

I32_T DevID: Device ID (DevID)
U32_T OffsetByte: Memory offset (byte) from 0
U32_T BitIndex: Bit index in a byte, value must be 0~7
BOOL_T *PRetBitValue: Return bit value (0 or 1)
I8_T *PRetI8Value: Return byte (I8_T) value
I16_T *PRetI16Value: Return word (I16_T) value
I32_T *PRetI32Value: Return double-word(I32_T) value

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

The function can be called after the device is started.

- Examples:

- Reference:

3.2.6.10.NMC_ReadOutputBit
3.2.6.11.NMC_ReadOutputI8
3.2.6.12.NMC_ReadOutputI16
3.2.6.13.NMC_ReadOutputI32

Read the mapped output memory by bit, word or dword.

- C/C++ Syntax Definition:

RTN_ERR NMC_ReadOutputBit(I32_T DevID, U32_T OffsetByte, U32_T BitIndex, BOOL_T *PRetBitValue);
RTN_ERR NMC_ReadOutputI8(I32_T DevID, U32_T OffsetByte, I8_T *PRetI8Value);
RTN_ERR NMC_ReadOutputI16(I32_T DevID, U32_T OffsetByte, I16_T *PRetI16Value);
RTN_ERR NMC_ReadOutputI32(I32_T DevID, U32_T OffsetByte, I32_T *PRetI32Value);

- Parameters:

I32_T DevID: Device ID (DevID)
U32_T OffsetByte: Memory offset (byte) from 0
U32_T BitIndex: Bit index in a byte, value must be 0~7
BOOL_T *PRetBitValue: Return bit value (0 or 1)
I8_T *PRetI8Value: Return byte (I8_T) value
I16_T *PRetI16Value: Return word (I16_T) value
I32_T *PRetI32Value: Return double-word(I32_T) value

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

The function can be called after the device is started.

- Examples:

- Reference:

3.2.6.14.NMC_WriteOutputBit
3.2.6.15.NMC_WriteOutputI8
3.2.6.16.NMC_WriteOutputI16
3.2.6.17.NMC_WriteOutputI32

Write the mapped output memory by bit, word or dword.

- C/C++ Syntax Definition:

RTN_ERR NMC_WriteOutputBit(I32_T DevID, U32_T OffsetByte, U32_T BitIndex, BOOL_T BitValue);
RTN_ERR NMC_WriteOutputI8(I32_T DevID, U32_T OffsetByte, I8_T I8Value);
RTN_ERR NMC_WriteOutputI16(I32_T DevID, U32_T OffsetByte, I16_T I16Value);
RTN_ERR NMC_WriteOutputI32(I32_T DevID, U32_T OffsetByte, I32_T I32Value);

- Parameters:

I32_T DevID: Device ID (DevID)
U32_T OffsetByte: Memory offset (byte) from 0
U32_T BitIndex: Bit index in a byte, value must be 0~7
BOOL_T BitValue: Bit value (0 or 1) to be set
I8_T I8Value: Return byte (I8_T) value to be set
I16_T I16Value: Return word (I16_T) value to be set
I32_T I32Value: Return double-word(I32_T) value to be set

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

The function can be called after the device is started.

- Examples:

- Reference:

3.2.7. Axis or Group Quantity

3.2.7.1. NMC_DeviceGetAxisCount

Get the axis quantity.

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceGetAxisCount(I32_T DevID, I32_T *PRetAxisCount);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T *PRetAxisCount: The quantity of axis started will be returned after the function is called successfully.

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

After the configuration file is loaded, the function can be called to confirm the quantity of axis started. After NMC_DeviceResetConfig() is called to reset the device configurations, the quantity of axis is 0.

- Examples:

- Reference:

3.2.7.2. NMC_DeviceGetGroupCount

Get the group quantity.

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceGetGroupCount(I32_T DevID, I32_T *PRetGroupCount);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T *PRetGroupCount: The quantity of group started will be returned after the function is called successfully.

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

After the configuration file is loaded, the function can be called to confirm the quantity of group started. After NMC_DeviceResetConfig() is called to reset the device configurations, the quantity of group is 0.

- Examples:

- Reference:

3.2.7.3. NMC_DeviceGetGroupAxisCount

Get the quantity of group axis.

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceGetGroupAxisCount(I32_T DevID, I32_T GroupIndex, I32_T *PRetGroupAxisCount);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T *PRetGroupAxisCount: The quantity of group axis will be returned after the function is called successfully.

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

After the configuration file is loaded, the function can be called to confirm the actual quantity of group axis.

- Examples:

- Reference:

3.2.8. Read axis/group description

3.2.8.1. NMC_AxisGetDescription

Read the name description information of the specified axis

- C/C++ Syntax Definition:

RTN_ERR NMC_AxisGetDescription(I32_T DevID, I32_T AxisIndex, U32_T DescStrSize, char *PRetAxisDescription);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T AxisIndex: The specified axis index to be read from
U32_T DescStrSize: C-style string buffer size, in byte
char *PRetAxisDescription:Given C-style string buffer and return description

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

This function is used to read the name description of the specified axis. The information source is based on the NCF file loaded by the system, so it must be loaded before it can be read normally. In other words, the system state must be in READY state.

- Examples:

- Reference:

3.2.8.2. NMC_GroupGetDescription

Read the name description information of the specified group

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupGetDescription(I32_T DevID, I32_T GroupIndex, U32_T DescStrSize, char *PRetGroupDescription);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: The specified group index to be read from
U32_T DescStrSize: C-style string buffer size, in byte
char * PRetGroupDescription:Given C-style string buffer and return description

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

This function is used to read the name description of the specified group. The information source is based on the NCF file loaded by the system, so it must be loaded before it can be read normally. In other words, the system status must be in READY state.

- Examples:

- Reference:

3.2.9. Enable and Disable Functions for All Axes and Groups

3.2.9.1. NMC_DeviceEnableAll

Enable all axes and groups in the system.

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceEnableAll(I32_T DevID);

- Parameters:

I32_T DevID: Device ID (DevID)

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

The function can be called after the device is started.
After the function is called, all axes and groups will be enabled (servo on). The start procedure will enable all axes and then all groups in order. The procedure will stop for any failure and return the error immediately.

- Examples:

- Reference:

3.2.9.2. NMC_DeviceDisableAll

Disable all axes and groups in the system.

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceDisableAll(I32_T DevID);

- Parameters:

I32_T DevID: Device ID (DevID)

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

The function can be called after the device is started.
After the function is called, all axes and groups will be disabled (servo off). The stop procedure will disable all axes and then all groups in order. The procedure will stop for any failure and return the error immediately.

- Examples:

- Reference:

3.2.10. Halt and Stop Functions for All Axes and Groups

3.2.10.1.NMC_DeviceHaltAll

Halt all axes and groups in the system (Stand still state).

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceHaltAll(I32_T DevID);

- Parameters:

I32_T DevID: Device ID (DevID)

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

The function can be called after the device is started.
After the function is called, all axes and groups will be halted. The halt procedure will halt all axes and then all groups in order. The procedure will stop for any failure and return the error immediately.

- Examples:

- Reference:

3.2.10.2.NMC_DeviceStopAll

Stop all axes and groups in the system (Stopped state).

- C/C++ Syntax Definition:

RTN_ERR NMC_DeviceStopAll(I32_T DevID);

- Parameters:

I32_T DevID: Device ID (DevID)

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

The function can be called after the device is started.
After the function is called, all axes and groups will be stopped. The stop procedure will stop all axes and then all groups in order. The procedure will stop for any failure and return the error immediately.

- Examples:

- Reference:

3.2.11. System Information

3.2.11.1.NMC_MessagePopFirst

Read system message queue.

- C/C++ Syntax Definition:

RTN_ERR NMC_MessagePopFirst(_opt_null_ NmcMsg_T *PRetMsg);

- Parameters:

_opt_null_ NmcMsg_T *PRetMsg: Data structure of system information which can be set to NULL to remove the message from the message queue. Note that if the parameter is not NULL, the "sizeOfStruct" member variable in the structure must be initialized before calling NMC_MessagePopFirst(). See the sample program below.

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.
If there is no system message, the function will return ERR_NEXMOTION_QUEUE_EMPTY.

- Usage:

NMC_MessagePopFirst() will read and remove the first (oldest) message from the message queue in order.

- Examples:

```
RTN_ERR  ret;
NmcMsg_T msg;

msg.sizeOfStruct = sizeof(msg);
do{
    ret = NMC_MessagePopFirst( &msg );
    if( ret == ERR_NEXMOTION_SUCCESS )
    {
        printf( " Msg:Type=%d, code=%d, src=%s, text=%s\n", msg.type,
msg.code, msg.source, msg.text );
    }
}while( ret == ERR_NEXMOTION_SUCCESS );
```

- Reference:

NMC_MessageOutputEnable()

3.2.11.2.NMC_MessageOutputEnable

Transfer a copy of message to MS Windows system message.

- C/C++ Syntax Definition:

void NMC_MessageOutputEnable(BOOL_T Enable);

- Parameters:

BOOL_T Enable: False (0): not to transfer, True (1): transfer

- Return values:

None.

- Usage:

After NMC_MessageOutputEnable(True) is called, the system message will be transferred to MS Windows system message.

- Examples:

- Reference:

NMC_MessagePopFirst()

3.2.12. Function Trace Functions

3.2.12.1.NMC_DebugSetTraceMode

Set API trace mode.

- C/C++ Syntax Definition:

void NMC_DebugSetTraceMode(I32_T TraceMode);

- Parameters:

I32_T TraceMode:

0: Turn off the trace function and do not output anything
1: Output the APIs with errors only
2: Output all APIs

- Return values:

None.

- Usage:

Please refer to the section, Function Trace.

3.2.12.2.NMC_DebugSetHookData

Set the data structure index for the hook function.

- C/C++ Syntax Definition:

void NMC_DebugSetHookData(void *PHookUserData);

- Parameters:

void *PHookUserData: [Input] A pointer variable to set a variable or a data structure specified by user

- Return values:

None.

- Usage:

Please refer to the section, Function Trace.

3.2.12.3.NMC_DebugSetHookFunction

Set the hook function.

- C/C++ Syntax Definition:

void NMC_DebugSetHookFunction(PF_NmcHookAPI PFHookFuncPtr);

- Parameters:

PF_NmcHookAPI PFHookFuncPtr: A function pointer to set the hook function

- Return values:

None.

- Usage:

Please refer to the section, Function Trace.

3.2.12.4.NMC_DebugGetApiAddress

Read the API address.

- C/C++ Syntax Definition:

const void* NMC_DebugGetApiAddress(const char *PApiName);

- Parameters:

const char *PApiName: [Input] The function name of a NexMotion API

- Return values:

const void*: Return the function pointer to set the NexMotion API.

- Usage:

Please refer to the section, Function Trace.

3.3. Axis APIs

3.3.1. Axis Configuration Functions

3.3.1.1. NMC_AxisSetParamI32

3.3.1.2. NMC_AxisGetParamI32

3.3.1.3. NMC_AxisSetParamF64

3.3.1.4. NMC_AxisGetParamF64

Set/Get the axis parameters (I32_T/F64_T, data type)

- C/C++ Syntax Definition:

```
RTN_ERR NMC_AxisSetParamI32( I32_T DevID, I32_T AxisIndex, I32_T ParamNum, I32_T SubIndex,
        I32_T ParaValueI32 );
RTN_ERR NMC_AxisGetParamI32( I32_T DevID, I32_T AxisIndex, I32_T ParamNum, I32_T SubIndex,
        I32_T *PRetParaValueI32 );
RTN_ERR NMC_AxisSetParamF64( I32_T DevID, I32_T AxisIndex, I32_T ParamNum, I32_T SubIndex,
        F64_T ParaValueF64 );
RTN_ERR NMC_AxisGetParamF64( I32_T DevID, I32_T AxisIndex, I32_T ParamNum, I32_T SubIndex,
        F64_T *PRetParaValueF64 );
```

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T AxisIndex: Axis index
I32_T ParamNum: Parameter number
I32_T SubIndex: Parameter sub-index
I32_T ParaValueI32: [Input] Value to be set (signed integer)
F64_T ParaValueF64: [Input] Value to be set (double precision float)
I32_T *PRetParaValueI32: [Output] Value to be returned (signed integer)
F64_T *PRetParaValueF64: [Output] Value to be returned (double precision float)

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

Please refer to the section: Axis Parameters.
The device will load the configurations in according with the configuration file during starting. The function can be used to set/get some axis parameters. A suitable function shall be selected depended on the data type of the parameters (I32_T or F64_T).

- Examples:

- Reference:

Please refer to the section: Axis Parameters.

3.3.2. Axis State Control Functions

3.3.2.1. NMC_AxisEnable

Enable an axis.

- C/C++ Syntax Definition:

RTN_ERR NMC_AxisEnable(I32_T DevID, I32_T AxisIndex);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T AxisIndex: Axis index

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

1. Before the axis motion, the function shall be called to drive the axis into the excitation.
2. After the axis is excitation, the function will return successfully.
3. If the axis is in the error state, the function will return the error code. After the error is resolved and the NMC_AxisResetState() is called, the function can be called to drive the axis into the excitation again.

- Examples:

```
RTN_ERR ret = 0;
ret = NMC_AxisEnable( 0, 0 );
```

- Reference:

NMC_AxisDisable(), NMC_AxisGetState() and NMC_AxisResetState()

3.3.2.2. NMC_AxisDisable

Disable an axis.

- C/C++ Syntax Definition:

RTN_ERR NMC_AxisDisable(I32_T DevID, I32_T AxisIndex);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T AxisIndex: Axis index

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

1. If the axis is excitation, the function can be called to disable the excitation.
2. If the axis is in motion, the function can also be called to stop the motion and release the excitation. All the motions stored in the motion queue will be removed.

- Examples:

```
RTN_ERR ret = 0;
ret = NMC_AxisDisable( 0, 0 );
```

- Reference:

NMC_AxisEnable() and NMC_AxisGetState()

3.3.2.3. NMC_AxisGetStatus

Get the axis status.

- C/C++ Syntax Definition:

RTN_ERR NMC_AxisGetStatus(I32_T DevID, I32_T AxisIndex, I32_T *PRetAxisStatus);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T AxisIndex: Axis index
I32_T *PRetAxisStatus: Return the axis status.

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

- Examples:

```
RTN_ERR ret = 0; I32_T   status = 0;
ret = NMC_AxisGetStatus( 0, 0, &status );
```

- Reference:

NMC_AxisGetState()

3.3.2.4. NMC_AxisGetState

Get the axis state.

- C/C++ Syntax Definition:

RTN_ERR NMC_AxisGetState(I32_T DevID, I32_T AxisIndex, I32_T *PRetAxisState);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T AxisIndex: Axis index
I32_T *PRetAxisState: Return the axis state.

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

1. The function can be called in a periodic procedure to confirm the current axis state.
2. Before being in motion or executing the Homing procedure, the axis shall be excitation. After NMC_AxisEnable() is called, NMC_AxisGetState() can be called to confirm the axis is excitation.
3. If the axis is in the error state or not stopped normally, and NMC_AxisResetState() is called, NMC_AxisGetState() can be called to confirm the axis error state is resolved.

- Examples:

```
RTN_ERR ret = 0;
I32_T       state = 0;
ret = NMC_AxisGetState( 0, 0, &state );
```

- Reference:

1. NMC_AxisGetStatus()
2. NMC_AxisEnable()
3. NMC_AxisResetState()

3.3.2.5. NMC_AxisResetState

Reset the axis state.

- C/C++ Syntax Definition:

RTN_ERR NMC_AxisResetState(I32_T DevID, I32_T AxisIndex);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T AxisIndex: Axis index

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

1. If an axis is in the error state, or an axis is requested with a forced stop command and has stopped, the function can be called to reset the axis to the nonexcitation state or the normal excitation state. NMC_AxisGetState() can be called to confirm the axis is normal.
2. If the axis drive is in the alarm state and the function is called, the device will reset the alarm automatically. After the error is resolved, the axis state will transfer to the nonexcitation state from the error state (AXIS_STATE_ERROR).
3. If an axis is requested with a forced stop command and has stopped, the function can be called to reset the axis to nonexcitation state (AXIS_STATE_STAND_STILL).

- Examples:

```
RTN_ERR ret = 0;
ret = NMC_AxisResetState( 0, 0 );
```

- Reference:

1. NMC_AxisGetState()
2. NMC_AxisResetDriveAlm()

3.3.2.6. NMC_AxisResetDriveAlm

Reset the axis servo alarm.

- C/C++ Syntax Definition:

RTN_ERR NMC_AxisResetDriveAlm(I32_T DevID, I32_T AxisIndex);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T AxisIndex: Axis index

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

1. When an alarm of axis drive is issued, the drive will automatically release the excitation in general. After the error is resolved, the function can be called to reset the alarm, and NMC_AxisGetStatus() can be called to get the axis status for the confirmation of alarm reset.
2. Unless all drive alarms can be reset with the function, some drive alarms shall be reset by means of re energization.
3. After the function is called to reset the drive alarms successfully, the axis state cannot be reset from the error state, unless NMC_AxisResetState() is called to reset the axis to the nonexcitation (AXIS_STATE_DISABLE).

- Examples:

```
RTN_ERR ret = 0;
ret = NMC_AxisResetDriveAlm( 0, 0 );
```

- Reference:

1. NMC_AxisGetStatus()
2. NMC_AxisResetState().

3.3.2.7. NMC_AxisGetDriveAlmCode

3.3.3. Axis Motion Status Functions

3.3.3.1. NMC_AxisGetCommandPos

3.3.3.2. NMC_AxisGetActualPos

- C/C++ Syntax Definition:

RTN_ERR NMC_AxisGetCommandPos(I32_T DevID, I32_T AxisIndex, F64_T *PRetCmdPos);
RTN_ERR NMC_AxisGetActualPos(I32_T DevID, I32_T AxisIndex, F64_T *PRetActPos);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T AxisIndex: Axis index

- Return values:

1. Return an error code. If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.
2. F64_T *PRetCmdPos: Return the command position. Unit: user unit.
3. F64_T *PRetActPos: Return the encoder feedback position. Unit: user unit.

- Usage:

The function can be called to get the command position and the encoder feedback position for an axis.

- Examples:

```
RTN_ERR ret = 0;
F64_T      cmdPos = 0.0;
F64_T      actPos = 0.0;
ret = NMC_AxisGetCommandPos( 0, 0, &cmdPos );
ret = NMC_AxisGetActualPos( 0, 0, &actPos );
```

- Reference:

3.3.3.3. NMC_AxisGetCommandVel

3.3.3.4. NMC_AxisGetActualVel

- C/C++ Syntax Definition:

RTN_ERR NMC_AxisGetCommandVel(I32_T DevID, I32_T AxisIndex, F64_T *PRetCmdVel);
RTN_ERR NMC_AxisGetActualVel(I32_T DevID, I32_T AxisIndex, F64_T *PRetActVel);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T AxisIndex: Axis index

- Return values:

1. Return an error code. If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.
2. F64_T *PRetCmdVel: Return the command velocity. Unit: user unit/sec.
3. F64_T *PRetActVel: Return the encoder feedback velocity. Unit: user unit/sec.

- Usage:

The functions can be called to get the command velocity and the encoder feedback velocity for an axis.

- Examples:

```
RTN_ERR ret = 0;
F64_T      cmdVel = 0.0;
F64_T      actVel = 0.0;
ret = NMC_AxisGetCommandVel( 0, 0, &cmdVel );
ret = NMC_AxisGetActualVel( 0, 0, &actVel );
```

- Reference:

3.3.3.5. NMC_AxisGetMotionBuffSpace

- C/C++ Syntax Definition:

RTN_ERR NMC_AxisGetMotionBuffSpace(I32_T DevID, I32_T AxisIndex, I32_T *PRetFreeSpace);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T AxisIndex: Axis index

- Return values:

1. Return an error code. If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.
2. I32_T *PRetFreeSpace: Return the quantity of motions which can still be stored into the axis motion queue.

- Usage:

Because the axis supports the motion queue, the function can be called to confirm the quantity of motions which can still be stored into the axis motion queue.

- Examples:

```
RTN_ERR ret = 0;
I32_T           space = 0;
ret = NMC_AxisGetMotionBuffSpace( 0, 0, &space );
```

- Reference:

1. NMC_AxisPtp()
2. NMC_AxisJog()
3. NMC_AxisHalt()

3.3.4. Axis motion control functions

3.3.4.1. NMC_AxisHomeDrive

- C/C++ Syntax Definition:

```
RTN_ERR NMC_AxisHomeDrive( I32_T DevID, I32_T AxisIndex );
```

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T AxisIndex: Axis index

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

1. Before executing an axis motion, the user usually calls this function to start the axis Homing program, returning the axis to the defined origin position.
2. The parameters related to Homing are defined in the axis parameter table, including: Homing method, acceleration, velocity and offset. Among them, the method of Homing program must be determined by the method provided by the driver.
3. When an error occurs during the Homing program on an axis, Bit 7 of the axis status changes to 1, and the axis state switches to 「AXIS_STATE_ERROR」. At this time, when the axis is completely stopped (check whether the axis status Bit 9 is 1), you can call NMC_AxisResetDriveAlm() or NMC_AxisResetState() to return to the normal state.
4. After the function is called and the motion homing successfully, the bit 18 of axis status will become 1, and the axis state will transfer to the normal excitation (AXIS_STATE_STAND_STILL)

- Examples:

```
RTN_ERR    ret = 0;
ret = NMC_AxisHomeDrive( 0, 0 );
```

- Reference:

3.3.4.2. NMC_AxisSetHomePos

3.3.4.3. NMC_AxisPtp

- C/C++ Syntax Definition:

RTN_ERR NMC_AxisPtp(I32_T DevID, I32_T AxisIndex, F64_T TargetPos, _opt_null_ const F64_T *PMaxVel);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T AxisIndex: Axis index
F64_T TargetPos: Target position (Unit: user unit). The value will be interpreted to absolute or relative distance based on the axis parameter 0x30 (absolute or relative programming).
F64_T *PMaxVel: Input the target velocity with a pointer variable. Moreover, input 0 directly for no target velocity, and the drive can plan the motion based on the velocity configuration of the axis parameter AXP_VM.

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

1. The function can be called for point-to-point motion if the axis is excitation.
2. If the axis is homing, the function will return the error code.
3. If the axis is in the axis state AXIS_STATE_STOPPING, AXIS_STATE_STOPPED or AXIS_STATE_ERROR, the function will return the error code. After NMC_AxisResetState() is called to reset the axis to normal excitation (AXIS_STATE_STAND_STILL), the function can be called for point-to-point motion successfully.
4. If the axis is executing other motions, it will execute the corresponding behavior depended on the axis parameter AXP_BUFF_PARAM after the function is called.
5. The function can be called to enable the point-to-point motion. After the motion completely, the axis will move to the input target position. If the axis parameter 0x30 (Absolute or relative programming) is set to 1, the target position and the relative distance from the current position shall be input in the function. If the relative distance is set to 0, the target position will be set as an absolute position.
6. If the axis is executing the point-to-point motion, the axis state will transfer to AXIS_STATE_DISCRETE_MOTION. After the axis moves to the target position and there is no successive motion, the bit 9 of axis status will become to 1, and the axis will transfer to the normal excitation (AXIS_STATE_STAND_STILL).
7. The drive will plan the velocity curve depended on the axis parameters, AXP_PROF_TYPE, AXP_ACC, AXP_DEC and AXP_JERK.
8. The maximum velocity can be input with the pointer variable, PMaxVel. Then the corresponding axis parameter AXP_VM will be modified to the input value, and the velocity plan will be performed accordingly.
9. If the pointer variable, PMaxVel, is set to 0, the drive will perform the velocity plan based on the axis parameter AXP_VM as the target velocity.
10. If the axis state is AXIS_STATE_STAND_STILL, the function will enable the point-to-point motion immediately after called whether the content of axis parameter AXP_BUFF_PARAM.
11. After the function is called and if the axis state is AXIS_STATE_WAIT_SYNC and the axis parameter AXP_BUFF_PARAM is aborting, the motions stored in the motion queue will be removed. Then the point-to-point motion will be stored into the motion queue and wait for trigger signal.
12. If the axis has not reached the target position during the point-to-point motion, NMC_AxisHalt() can be called to stop the motion.

- Examples:

```
RTN_ERR ret = 0;
```

```
ret = NMC_AxisPtp( 0, 0, 100, 0 ) // Plan the motion based on the velocity configuration of the axis parameter AXP_VM
```

- Reference:

1. NMC_AxisSetParamI32(), NMC_AxisSetParamF64()
2. NMC_AxisResetState()
3. NMC_AxisGetStatus()
4. NMC_AxisHalt()

3.3.4.4. NMC_AxisJog

- C/C++ Syntax Definition:

RTN_ERR NMC_AxisJog(I32_T DevID, I32_T AxisIndex, I32_T Dir, _opt_null_ const F64_T *PMaxVel);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T AxisIndex: Axis index
I32_T Dir: Direction. 1: Forward, -1: Reverse
F64_T *PMaxVel: Input the target velocity with a pointer variable. Moreover, input 0 directly for no target velocity, and the drive will get the configuration in axis parameter AXP_VM as the target velocity.

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

1. The function can be called for JOG motion if the axis is excitation.
2. If the axis is homing, the function will return the error code.
3. If the axis is in the axis state AXIS_STATE_STOPPING, AXIS_STATE_STOPPED or AXIS_STATE_ERROR, the function will return the error code. After NMC_AxisResetState() is called to reset the axis to normal excitation (AXIS_STATE_STAND_STILL), the function can be called for JOG motion successfully.
4. After the function is called, the drive will increase/decrease the velocity to the target velocity in accordance with the configuration in the axis parameter AXP_ACC, and the axis state will transfer to AXIS_STATE_CONTINUOUS_MOTION.
5. After the target velocity is input with the pointer variable PMaxVel, the axis parameter AXP_VM will be modified accordingly. The input target velocity can be 0.
6. After the function is called and the target velocity is reached, the bit 8 of the axis status becomes to 0, and the motion is continued at the target velocity.
7. If the axis is executing other motions, it will execute the corresponding behavior depended on the axis parameter AXP_BUFF_PARAM after the function is called.
8. After the function is called, NMC_AxisHalt() can be called to stop the motion.

- Examples:

```
RTN_ERR ret = 0;
F64_T      maxVel = 100;
ret = NMC_AxisJog( 0, 0, -1, &maxVel ); // Set the axis move reversely. Modify the axis parameter AXP_VM to 100 as the target velocity for the velocity plan.
```

- Reference:

1. NMC_AxisSetParamI32(), NMC_AxisSetParamF64()
2. NMC_AxisResetState()
3. NMC_AxisGetStatus()
4. NMC_AxisHalt()

3.3.5. Axis Motion Status Functions

3.3.5.1. NMC_AxisHalt

- C/C++ Syntax Definition:

RTN_ERR NMC_AxisHalt(I32_T DevID, I32_T AxisIndex);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T AxisIndex: Axis index

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

1. When an axis is executing the point-to-point motion or JOG motion, the function can be called to stop the axis motion. The drive will decrease the velocity from the configuration in the axis parameter AXP_DEC to this in the AXP_V_BASE.
2. If the axis is homing, the function will return the error code.
3. If the axis is in the axis state AXIS _STATE_STOPPING, AXIS_STATE_STOPPED or AXIS_STATE_ERROR, the function will return the error code.
4. If the axis is in the axis state AXIS _STATE_DISABLE or AXIS_STATE_STAND_STILL, the function will not return any error code.
5. If the axis parameter AXP_BUFF_PARAM is set to aborting, the function can be called to stop the axis motion normally. If the axis parameter AXP_BUFF_PARAM is set to buffered, the function can be called to wait the bit 8 of axis status to 1 (the completion of the previous motion) and then to stop the axis motion normally.
6. After the function is called and the motion is stopped successfully, the axis state will transfer to AXIS_STATE_DISCRETE_MOTION. After the axis is stopped, the bit 8 and 9 of axis status will become 1, and the axis state will transfer to normal excitation (AXIS_STATE_STAND_STILL).

- Examples:

```
RTN_ERR ret = 0;
ret = NMC_AxisHalt( 0, 0 );
```

- Reference:

3.3.5.2. NMC_AxisStop

- C/C++ Syntax Definition:

RTN_ERR NMC_AxisStop(I32_T DevID, I32_T AxisIndex);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T AxisIndex: Axis index

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

1. Whether an axis is executing each type of motion and such axis motion shall be stopped for some undesired cases, the function can be called to perform the forced stop procedure. The drive will decrease the velocity from the configuration in the axis parameter AXP_STOP_PROF_DEC to this in the AXP_V_BASE.
2. If the axis is homing, the function will return the error code.
3. After the function is called to perform the forced stop procedure and the axis has not stopped, the axis state will transfer to AXIS_STATE_STOPPING. After the axis is stopped, the bit 9 of axis status will become 1, and the axis state will transfer to AXIS_STATE_STOPPED.
4. If the axis state is AXIS_STATE_STAND_STILL, it will transfer to AXIS_STATE_STOPPED after the function is called.
5. After the function is called to perform the forced stop procedure and the axis has stopped, the axis is inhibited to execute any axis motion, until NMC_AxisResetState() is called to reset the axis to normal excitation (AXIS_STATE_STAND_STILL).

- Examples:

```
RTN_ERR ret = 0;
ret = NMC_AxisStop( 0, 0 );
```

- Reference:

NMC_AxisResetState()

3.3.5.3. NMC_AxisHaltAll

- C/C++ Syntax Definition:

RTN_ERR NMC_AxisHaltAll(I32_T DevID);

- Parameters:

I32_T DevID: Device ID (DevID)

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

The usage is similar to this of NMC_AxisHalt(). The function can be called to stop the motions of all axes in a specified device normally.

- Examples:

```
RTN_ERR ret = 0;
ret = NMC_AxisHaltAll( 0 );
```

- Reference:

NMC_AxisHalt()

3.3.5.4. NMC_AxisStopAll

- C/C++ Syntax Definition:

RTN_ERR NMC_AxisStopAll(I32_T DevID);

- Parameters:

I32_T DevID: Device ID (DevID)

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

The usage is similar to this of NMC_AxisStop(). The function can be called to stop the motions of all axes in a specified device forcedly.

- Examples:

```
RTN_ERR ret = 0;
ret = NMC_AxisStopAll( 0 );
```

- Reference:

NMC_AxisStop()

3.3.6. Axis Motion Change Functions

3.3.6.1. NMC_AxisVelOverride

- C/C++ Syntax Definition:

RTN_ERR NMC_AxisVelOverride(I32_T DevID, I32_T AxisIndex, F64_T TargetVel);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T AxisIndex: Axis index
F64_T TargetVel: Target velocity (Unit: user unit/sec)

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

1. The function can only be called to change the target velocity after the axis point-to-point motion (NMC_AxisPtp) or the axis JOG motion (NMC_AxisJog) is enabled.
2. The target velocity input through the function will be effective to the current motion only, not to modify the axis parameter AXP_VM.
3. If the JOG motion is enabled, the input target velocity can be reached, and the bit 8 of axis status will become to 1.
4. If the point-to-point motion is enabled, the input target velocity may not be reached. To meet the specified target position and acceleration/deceleration, the drive will plan the practical velocity curve based on the input target velocity.

- Examples:

```
RTN_ERR ret = 0;
F64_T      maxVel = 100.0;
ret = NMC_AxisJog( 0, 0, 1, &maxVel );  // Enable the axis JOG motion at target velocity 100.
Sleep(100);
ret = NMC_AxisVelOverride( 0, 0, 50.0 );       // Decrease the target velocity to 50.
```

- Reference:

1. NMC_AxisPtp()
2. NMC_AxisJog()

3.3.6.2. NMC_AxisAccOverride

- C/C++ Syntax Definition:

RTN_ERR NMC_AxisAccOverride(I32_T DevID, I32_T AxisIndex, F64_T TargetAcc);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T AxisIndex: Axis index
F64_T TargetAcc: Target acceleration

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

1. The function can only be called to change the target velocity after the axis point-to-point motion (NMC_AxisPtp) or the axis JOG motion (NMC_AxisJog) is enabled.
2. The input acceleration will be set to the axis parameter AXP_ACC.
3. If the axis has reached the target velocity (bit 12 of axis status is 1), there is no any impact of the input acceleration on the current motion.

- Examples:

```
RTN_ERR ret = 0;
I32_T           status = 0;
F64_T      maxVel = 50.0;
ret = NMC_AxisJog( 0, 0, 1, &maxVel );  // Enable the axis JOG motion at target velocity 50.
Sleep(100);
ret = NMC_AxisGetStatus( 0, 0, &status );
if( !( status & 0x1000 ) )
{
      ret = NMC_AxisAccOverride ( 0, 0, 100.0 );    // // Change the acceleration to 100.
}
```

- Reference:

3.3.6.3. NMC_AxisDecOverride

- C/C++ Syntax Definition:

RTN_ERR NMC_AxisDecOverride(I32_T DevID, I32_T AxisIndex, F64_T TargetDec);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T AxisIndex: Axis index
F64_T TargetDec: Target deceleration

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

1. The function can only be called to change the target deceleration after the axis point-to-point motion (NMC_AxisPtp) or the normal stop (NMC_AxisHalt) is enabled.
2. The input deceleration will be set to the axis parameter AXP_DEC.
3. If the axis point-to-point is enabled and the axis velocity is changed from the target velocity to the configured end velocity gradually (bit 11 of axis status is 1), there is no any impact of the input deceleration on the current motion.
4. If the normal stop of axis (NMC_AxisHalt) is enabled and the axis has not stopped, the input deceleration will impact on the current motion.

- Examples:

```
RTN_ERR ret = 0;
I32_T           status = 0;
F64_T     maxVel = 50.0;
ret = NMC_AxisPtp( 0, 0, 100, 0 );  // Enable the axis point-to-point motion at the target velocity based on the AXP_VM.
ret = NMC_AxisDecOverride ( 0, 0, 100.0 );   // Change the deceleration to 100.0
```

- Reference:

3.3.7. Gross Axis Velocity Ratio Configuration Functions

3.3.7.1. NMC_AxisSetVelRatio

- C/C++ Syntax Definition:

RTN_ERR NMC_AxisSetVelRatio(I32_T DevID, I32_T AxisIndex, F64_T Percentage);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T AxisIndex: Axis index
F64_T Percentage: [Input] Velocity percentage

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

1. Whether the axis is in motion, the function can be called to set the velocity percentage. The default value of the velocity percentage is 1.0.
2. The input velocity percentage shall be more than or equal to 0, and less than 1.
3. If the axis is in the axis state AXIS _STATE_STOPPING, AXIS_STATE_STOPPED or AXIS_STATE_ERROR, the function will return the error code.
4. If the axis is homing, the function will return the error code.
5. After the function is called to set the velocity percentage, the target velocity of the later axis motion shall be the input maximum velocity multiplied by such velocity percentage.
6. If the axis is executing the point-to-point motion or the JOG motion and the function is called, the input velocity percentage will change the target velocity of the current motion.

- Examples:

```
RTN_ERR ret = 0;
F64_T      maxVel = 50.0;
ret = NMC_AxisJog( 0, 0, 1, &maxVel );  // Enable axis JOG motion at target velocity 50.
ret = NMC_AxisSetVelRatio( 0, 0, 0.5 );      // Change velocity percentage to target velocity 25.
```

- Reference:

3.3.7.2. NMC_AxisGetVelRatio

- C/C++ Syntax Definition:

RTN_ERR NMC_AxisGetVelRatio(I32_T DevID, I32_T AxisIndex, F64_T *PPercentage);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T AxisIndex: Axis index

- Return values:

1. Return an error code. If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.
2. F64_T *PPercentage: Return the velocity percentage.

- Usage:

The function is called to get the current velocity percentage.

- Examples:

```
RTN_ERR ret = 0;
F64_T      velRatio = 0.0;
ret = NMC_AxisGetVelRatio( 0, 0, &velRatio );
```

- Reference:

3.4. Group APIs

3.4.1. Group Configuration Functions

3.4.1.1. NMC_GroupSetParamI32

Set the group parameters (I32_T, data type).

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupSetParamI32(I32_T DevID, I32_T GroupIndex, I32_T ParamNum, I32_T SubIndex, I32_T ParaValueI32);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T ParamNum: Parameter number
I32_T SubIndex: Parameter sub-index
I32_T ParaValueI32: [Input] Value to be set (signed integer)

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T      devID      = 0;
I32_T      groupIndex     = 0;
I32_T      paramNum      = 0x30; // The set command is an absolute or incremental parameter
I32_T      subIndex = 0;
I32_T      paraValueI32 = 1;    // The command is incremental
RTN_ERR ret    = 0;

ret = NMC_GroupSetParamI32( devID, groupIndex, paramNum, subIndex, paraValueI32 );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.1.2. NMC_GroupGetParamI32

Get the group parameters (I32_T, data type).

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupGetParamI32(I32_T DevID, I32_T GroupIndex, I32_T ParamNum, I32_T SubIndex, I32_T *PRetParaValueI32);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T ParamNum: Parameter number
I32_T SubIndex: Parameter sub-index
I32_T *PRetParaValueI32: [Input] A pointer variable, [Output] Value of the parameter

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T      devID      = 0;
I32_T      groupIndex     = 0;
I32_T      paramNum     = 0x30; // The get command is an absolute or incremental parameter
I32_T      subIndex = 0;
I32_T      paraValueI32 = 0;
RTN_ERR ret    = 0;

ret = NMC_GroupGetParamI32( devID, groupIndex, paramNum, subIndex, &paraValueI32 );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.1.3. NMC_GroupSetParamF64

Set the group parameters (F64_T, data type).

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupSetParamF64(I32_T DevID, I32_T GroupIndex, I32_T ParamNum, I32_T SubIndex, F64_T ParaValueF64);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T ParamNum: Parameter number
I32_T SubIndex: Parameter sub-index
F64_T ParaValueF64: [Input] Value to be set (double precision float)

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T      devID      = 0;
I32_T      groupIndex     = 0;
I32_T      paramNum      = 0x21; // Set the deceleration to stop motion.
I32_T      subIndex = 0;
F64_T paraValueF64 = 80.5; // Value of the deceleration to stop motion.
RTN_ERR ret    = 0;

ret = NMC_GroupSetParamF64( devID, groupIndex, paramNum, subIndex,
paraValueF64 );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.1.4. NMC_GroupGetParamF64

Get the group parameters (F64_T, data type).

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupGetParamF64(I32_T DevID, I32_T GroupIndex, I32_T ParamNum, I32_T SubIndex, F64_T *PRetParaValueF64);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T ParamNum: Parameter number
I32_T SubIndex: Parameter sub-index
F64_T *PRetParaValueF64: [Output] Value to be returned (double precision float)

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T      devID      = 0;
I32_T      groupIndex     = 0;
I32_T      paramNum      = 0x21; // Set the deceleration to stop motion.
I32_T      subIndex = 0;
F64_T paraValueF64 = 0;
RTN_ERR ret    = 0;

ret = NMC_GroupGetParamF64( devID, groupIndex, paramNum, subIndex,
&paraValueF64 );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.1.5. NMC_GroupAxSetParamI32

Set the parameters of group axis (I32_T, data type).

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupAxSetParamI32(I32_T DevID, I32_T GroupIndex, I32_T GroupAxisIndex, I32_T ParamNum, I32_T SubIndex, I32_T ParaValueI32);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T GroupAxisIndex: Group axis index
I32_T ParamNum: Parameter number
I32_T SubIndex: Parameter sub-index
I32_T ParaValueI32: [Input] Value to be set (signed integer)

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T devID          = 0;
I32_T groupIndex = 0;
I32_T groupAxisIndex = 2;
I32_T      paramNum     = 0x30; // The set command is an absolute or incremental parameter
I32_T      subIndex = 0;
I32_T      paraValueI32 = 1;    // The command is incremental
RTN_ERR ret    = 0;

ret = NMC_GroupAxSetParamI32( devID, groupIndex, groupAxisIndex, paramNum, subIndex, paraValueI32 );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.1.6. NMC_GroupAxGetParamI32

Get the parameters of group axis (I32_T, data type).

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupAxGetParamI32(I32_T DevID, I32_T GroupIndex, I32_T GroupAxisIndex, I32_T ParamNum, I32_T SubIndex, I32_T *PRetParaValueI32);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T GroupAxisIndex: Group axis index
I32_T ParamNum: Parameter number
I32_T SubIndex: Parameter sub-index
I32_T *PRetParaValueI32: [Output] Value to be returned (signed integer)

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T devID          = 0;
I32_T groupIndex = 0;
I32_T groupAxisIndex = 2
I32_T      paramNum     = 0x30; // The get command is an absolute or incremental parameter
I32_T      subIndex = 0;
I32_T      paraValueI32   = 0;
RTN_ERR ret    = 0;

ret = NMC_GroupAxGetParamI32( devID, groupIndex, groupAxisIndex, paramNum, subIndex, &paraValueI32 );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.1.7. NMC_GroupAxSetParamF64

Set the parameters of group axis (F64_T, data type).

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupAxSetParamF64(I32_T DevID, I32_T GroupIndex, I32_T GroupAxisIndex, I32_T ParamNum, I32_T SubIndex, F64_T ParaValueF64);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T GroupAxisIndex: Group axis index
I32_T ParamNum: Parameter number
I32_T SubIndex: Parameter sub-index
F64_T ParaValueF64: [Input] Value to be set (double precision float)

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T devID          = 0;
I32_T groupIndex = 0;
I32_T groupAxisIndex = 2;
I32_T      paramNum     = 0x21; // Set the deceleration to stop motion.
I32_T      subIndex = 0;
F64_T paraValueF64      = 80.5; // Value of the deceleration to stop motion.
RTN_ERR ret    = 0;

ret = NMC_GroupAxSetParamF64( devID, groupIndex, groupAxisIndex, paramNum, subIndex, paraValueF64 );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.1.8. NMC_GroupAxGetParamF64

Get the parameters of group axis (F64_T, data type).

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupAxGetParamF64(I32_T DevID, I32_T GroupIndex, I32_T GroupAxisIndex, I32_T ParamNum, I32_T SubIndex, F64_T *PRetParaValueF64);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T GroupAxisIndex: Group axis index
I32_T ParamNum: Parameter number
I32_T SubIndex: Parameter sub-index
F64_T *PRetParaValueF64: [Output] Value to be returned (double precision float)

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T devID          = 0;
I32_T groupIndex = 0;
I32_T groupAxisIndex = 2;
I32_T      paramNum     = 0x21; // Set the deceleration to stop motion.
I32_T      subIndex = 0;
F64_T paraValueF64      = 0;  // Value of the deceleration to stop motion.
RTN_ERR ret    = 0;

ret = NMC_GroupAxGetParamF64( devID, groupIndex, groupAxisIndex, paramNum, subIndex, &paraValueF64 );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.2. Group State Control Functions

3.4.2.1. NMC_GroupEnable

Enable all group axes (Servo On). If all group axes are enabled successfully, the group state will transfer from GROUP_DISABLE to GROUP_ENABLE.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupEnable(I32_T DevID, I32_T GroupIndex);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T devID    = 0;
I32_T groupIndex = 0;
RTN_ERR ret       = 0;

ret = NMC_GroupEnable( devID, groupIndex );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.2.2. NMC_GroupDisable

Disable all group axes (Servo Off). If all group axes are disabled successfully, the group state will transfer to GROUP_DISABLE.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupDisable(I32_T DevID, I32_T GroupIndex);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T devID      = 0;
I32_T groupIndex = 0;
RTN_ERR ret              = 0;

ret = NMC_GroupDisable( devID, groupIndex );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.2.3. NMC_GroupGetStatus

Get the group status by bit. The meanings of the bits are described as follows. If the bit value is 1, the event is triggered.

bit	Description
0	Trigger the external emergency switch
1	Drive alarm
2	Over the positive limit of hardware
3	Over the negative limit of hardware
4	Over the positive limit of software
5	Over the negative limit of software
6	All groups enabled (i.e. all group state are GROUP_STAND_STILL)
7	A group axis error (i.e. a group axis state is GROUP_ERROR_STOP)
9	No position change for all group axes
10	The (line or arc) motion in the Cartesian coordinate system is accelerated (or decelerated) to the maximum velocity. The bit is 0 in case of the PTP or JOG motion.
11	The (line or arc) motion in the Cartesian coordinate system is decelerated to the target velocity or 0. The bit is 0 in case of the PTP or JOG motion.
12	The (line or arc) motion in the Cartesian coordinate system is executed at the maximum velocity. The bit is 0 in case of the PTP or JOG motion.
13	The group is in motion (i.e. the group state is GROUP_MOVING, GROUP_HOMING or GROUP_STOPPING).
14	The group is stopped (i.e. the group state is GROUP_STOPPED).

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupGetStatus(I32_T DevID, I32_T GroupIndex, I32_T *PRetStatusInBit);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T *PRetStatusInBit: [Input] A pointer variable, [Output] Status by bit.

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T devID      = 0;
I32_T groupIndex = 0;
I32_T statusInBit = 0;
RTN_ERR ret              = 0;

ret = NMC_GroupGetStatus( devID, groupIndex, &statusInBit ); if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.2.4. NMC_GroupGetState

Get the state ofgroup. Please refer to the below table for details.

Value	State	Description
0	GROUP_DISABLE	A group axis is disabled.
1	GROUP_STAND_STILL	All group axes are enabled.
2	GROUP_STOPPED	After NMC_GroupStop() is called, the group is stopped.
3	GROUP_STOPPING	After NMC_GroupStop() is called, the group is stopping.
4	GROUP_MOVING	The group is moving.
5	GROUP_HOMING	The group is homing.
6	GROUP_ERROR_STOP	An error is occurred in a group axis.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupGetState(I32_T DevID, I32_T GroupIndex, I32_T *PRetState);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T *PRetState: [Input] A pointer variable, [Output] Group state

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T    devID      = 0;
I32_T groupIndex = 0;
I32_T    state         = 0;
RTN_ERR ret    = 0;

ret = NMC_GroupGetState( devID, groupIndex, &state );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.2.5. NMC_GroupResetState

Reset the group state. It can reset the group state from GROUP_STOPPED to GROUP_STAND_STILL. If the group state is GROUP_ERROR_STOP and the API is called, all drive alarms will be reset automatically. Then the group state will transfer to GROUP_STAND_STILL after all drive alarms are reset.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupResetState(I32_T DevID, I32_T GroupIndex);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T devID      = 0;
I32_T groupIndex = 0;
RTN_ERR ret              = 0;

ret = NMC_GroupResetState( devID, groupIndex );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.2.6. NMC_GroupResetDriveAlm

Reset the group servo alarm.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupResetDriveAlm(I32_T DevID, I32_T GroupIndex, I32_T GroupAxisIndex);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T GroupAxisIndex: The index of the group axis which alarm to be reset.

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T devID          = 0;
I32_T groupIndex = 0;
I32_T groupAxisIndex = 0;
RTN_ERR ret                  = 0;

ret = NMC_GroupResetDriveAlm( devID, groupIndex, groupAxisIndex );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.2.7. NMC_GroupResetDriveAlmAll

Reset all group servo alarms.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupResetDriveAlmAll(I32_T DevID, I32_T GroupIndex);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T devID      = 0;
I32_T groupIndex = 0;
RTN_ERR ret              = 0;

ret = NMC_GroupResetDriveAlmAll( devID, groupIndex );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.2.8. NMC_GroupGetDriveAlmCode

3.4.3. Gross Group Velocity Ratio Configuration Functions

3.4.3.1. NMC_GroupSetVelRatio

Set the velocity percentage of a group from 0.0 ~ 1000.0%.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupSetVelRatio(I32_T DevID, I32_T GroupIndex, F64_T Percentage);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
F64_T Percentage: Velocity percentage to be set (0% ~ 100.0%)

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T      devID      = 0;
I32_T      groupIndex = 0;
F64_T percentage = 100.0;
RTN_ERR ret    = 0;

ret = NMC_GroupSetVelRatio( devID, groupIndex, percentage );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.3.2. NMC_GroupGetVelRatio

Get the velocity percentage of a group from 0.0 ~ 1000.0%.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupGetVelRatio(I32_T DevID, I32_T GroupIndex, F64_T *PRetPercentage);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
F64_T *PRetPercentage: [Input] A pointer variable, [Output] Velocity percentage.

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T      devID      = 0;
I32_T      groupIndex = 0;
F64_T percentage = 0.0;
RTN_ERR ret    = 0;

ret = NMC_GroupGetVelRatio( devID, groupIndex, &percentage );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.4. Group Point-to-Point Motion Functions (Axis Coordinate System)

3.4.4.1. NMC_GroupPtpAcs

Enable the point-to-point motion for a group axis in the axis coordinate system (ACS). The default maximum velocity will refer to the value in the input parameters. If required, the maximum velocity can be set in the API, and the specified value will be stored in the corresponding parameters automatically.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupPtpAcs(I32_T DevID, I32_T GroupIndex, I32_T GroupAxisIndex, F64_T AcsPos, _opt_null_ const F64_T *PAcsMaxVel);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T GroupAxisIndex: The group axis index to be driven a motion
F64_T AcsPos: The point position to be reached
_opt_null_ const F64_T *PAcsMaxVel: A pointer variable which can specify the maximum velocity.
Input NULL (0) to igore the parameter.

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T      devID      = 0;
I32_T      groupIndex     = 0;
I32_T      groupAxisIndex = 2;
F64_T acsPos   = 35.5;
F64_T acsMaxVel    = 80.0;
RTN_ERR ret    = 0;

ret = NMC_GroupPtpAcs( devID, groupIndex, groupAxisIndex, acsPos,
&acsMaxVel );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.4.2. NMC_GroupPtpAcsAll

Enable the point-to-point motion for multiple group axes in the axis coordinate system (ACS). The specified group axes will start the motion and reach the target position at the same time.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupPtpAcsAll(I32_T DevID, I32_T GroupIndex, I32_T GroupAxesIdxMask, const Pos_T *PAcsPos);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T GroupAxesIdxMask: The group axis index mask is specified to execute the motion. Please refer to the below table.
Pos_T *PAcsPos: A pointer variable to set the target position.

Group axis	8	7	6	5	4	3	2	1
Bit index	7	6	5	4	3	2	1	0
The power is the index of bit	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

The usage of the group axis index mask (GroupAxesIdxMask) is described as follows:
If the group axes to be moved are the 1st, 3rd and 8th axes, the GroupAxesIdxMask is $2^0 + 2^2 + 2^7 = 133$.

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T      devID      = 0;
I32_T      groupIndex     = 0;
I32_T      groupAxesIdxMask = 133; // Move the 1st, 3rd and 8th axes.
Pos_T acsPos   = { 10, 20, 30, 40, 50, 60, 70 };
RTN_ERR ret    = 0;

ret = NMC_GroupPtpAcsAll( devID, groupIndex, groupAxesIdxMask, &acsPos );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.5. Group Axis JOG Motion Functions (Axis Coordinate System)

3.4.5.1. NMC_GroupJogAcs

Enable the JOG motion for a group axis in the axis coordinate system (ACS). The default maximum velocity will refer to the value in the input parameters. If required, the maximum velocity can be set in the API, and the specified value will be stored in the corresponding parameters automatically.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupJogAcs(I32_T DevID, I32_T GroupIndex, I32_T GroupAxisIndex, I32_T Dir, _opt_null_ const F64_T *PAcsMaxVel);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T GroupAxisIndex: Group axis index
I32_T Dir: Rotation direction
_opt_null_ const F64_T *PAcsMaxVel: A pointer variable which can specify the maximum velocity.
Input NULL (0) to igore the parameter.

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T devID          = 0;
I32_T groupIndex = 0;
I32_T groupAxisIndex = 2;
I32_T    dir                 = 1;
RTN_ERR ret    = 0;

ret = NMC_GroupJogAcs( devID, groupIndex, groupAxisIndex, dir, NULL );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.6. Group Point-to-Point Motion Functions (Cartesian coordinate system)

3.4.6.1. NMC_GroupPtpCart

Enable the point-to-point motion of group axis in the Cartesian coordinate system.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupPtpCart(I32_T DevID, I32_T GroupIndex, I32_T CartAxis, F64_T CartPos);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T CartAxis: The coordinate axis index for the motion in the Cartesian coordinate system. Please refer to the below table.
F64_T CartPos: The point position to be reached

Cartesian Coordinate Axis	V	U	C	B	A	Z	Y	X
Value	7	6	5	4	3	2	1	0

The usage of the coordinate axis index in the Cartesian coordinate system (CartAxis) is described as follows:
If the motion is executed on the z-axis in the Cartesian coordinate system, the CartAxis is 2.

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T     devID     = 0;
I32_T     groupIndex = 0;
I32_T     cartAxis   = 2;
F64_T     cartPos    = 55.5;
RTN_ERR ret    = 0;

ret = NMC_GroupPtpCart( devID, groupIndex, cartAxis, cartPos );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.6.2. NMC_GroupPtpCartAll

Enable the point-to-point motion of multiple group axes on the points in the Cartesian coordinate system.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupPtpCartAll(I32_T DevID, I32_T GroupIndex, I32_T CartAxesMask, const Pos_T *PTargetPos);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T CartAxesMask: The group axis index mask is specified to execute the point-to-point motion in the Cartesian coordinate system. Please refer to the below table.
const Pos_T *PTargetPos: A pointer variable to set the target position.

Cartesian Coordinate Axis	V	U	C	B	A	Z	Y	X
Bit index	7	6	5	4	3	2	1	0
The power is the index of bit	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

The usage of the group axis index mask in the Cartesian coordinate system (CartAxesMask) is described as follows:
If the group axes to be moved are the X-, Z- and A-axis, the GroupAxesIdxMask is $2^0 + 2^2 + 2^3 = 13$.

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T      devID      = 0;
I32_T      groupIndex     = 0;
I32_T      cartAxesMask = 13; // Move the X-, Z- and A-axis
Pos_T targetPos      = { 10, 20, 30, 40, 50, 60, 70 };
RTN_ERR ret    = 0;

ret = NMC_GroupPtpCartAll( devID, groupIndex, cartAxesMask, &targetPos );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.7. Group JOG motion functions (Cartesian coordinate system)

3.4.7.1. NMC_GroupJogCartFrame

Enable the JOG motion of group axis in the Cartesian coordinate system.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupJogCartFrame(I32_T DevID, I32_T GroupIndex, I32_T CartAxis, I32_T Dir, _opt_null_ const F64_T *PMaxVel);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T CartAxis: The coordinate axis index for the motion in the Cartesian coordinate system. Please refer to the below table.

Cartesian Coordinate Axis	V	U	C	B	A	Z	Y	X
Bit index	7	6	5	4	3	2	1	0
The power is the index of bit	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

The usage of the group axis index in the Cartesian coordinate system (CartAxis) is described as follows: If the group axis to be moved is the Z-axis, the CartAxis is 2.

I32_T Dir: Direction. 0: Forward, 1: Reverse
_opt_null_ const F64_T *PMaxVel: A pointer variable to set the maximum. Input NULL (0) to ignore the parameter.

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

- Reference:

None.

3.4.7.2. NMC_GroupJogTCPFrame

3.4.7.3. NMC_GroupJogPcsFrame

3.4.8. Group Halt and Stop Functions

3.4.8.1. NMC_GroupHalt

Halt a group (Stand still state). The API will decrease the velocity for each type of motion, and the group state will transfer from GROUP_MOVING to GROUP_STOPPING during the velocity decreasing, and to the GROUP_STAND_STILL when the velocity is 0.
The API shall specify the motion stop in accompanying the buffer mode. If the buffer mode is aborting, the current motion will decrease the velocity immediately. If the buffer mode is buffered, the API will be added into the motion buffer and will not be effective immediately.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupHalt(I32_T DevID, I32_T GroupIndex);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T devID      = 0;
I32_T groupIndex = 0;
RTN_ERR ret              = 0;

ret = NMC_GroupHalt( devID, groupIndex );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.8.2. NMC_GroupStop

Stop a group (Stop state). The API will decrease the velocity for each type of motion, and the group state will transfer from GROUP_MOVING to GROUP_STOPPING during the velocity decreasing, and to the GROUP_STAND_STOPPED when the velocity is 0.
To execute a new motion, NMC_GroupResetState() shall be called to reset the group state GROUP_STOPPED.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupStop(I32_T DevID, I32_T GroupIndex);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T    devID      = 0;
I32_T groupIndex = 0;
RTN_ERR ret              = 0;

ret = NMC_GroupStop( devID, groupIndex );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.8.3. NMC_GroupHaltAll

Halt all groups (Stand still state). The API will decrease the velocity for each type of motion, and the group state will transfer from GROUP_MOVING to GROUP_STOPPING during the velocity decreasing, and to the GROUP_STAND_STILL when the velocity is 0.
The API shall specify the motion stop in accompanying the buffer mode. If the buffer mode is aborting, the current motion will decrease the velocity immediately. If the buffer mode is buffered, the API will be added into the motion buffer and will not be effective immediately.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupHaltAll(I32_T DevID);

- Parameters:

I32_T DevID: Device ID (DevID)

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T       devID = 0;
RTN_ERR ret    = 0;

ret = NMC_GroupHaltAll( devID );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.8.4. NMC_GroupStopAll

Stop all groups (Stop state). The API will decrease the velocity for each type of motion, and the group state will transfer from GROUP_MOVING to GROUP_STOPPING during the velocity decreasing, and to the GROUP_STAND_STOPPED when the velocity is 0.
To execute a new motion, NMC_GroupResetState() shall be called to reset the group state GROUP_STOPPED.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupStopAll(I32_T DevID);

- Parameters:

I32_T DevID: Device ID (DevID)

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T      devID = 0;
RTN_ERR ret    = 0;

ret = NMC_GroupStopAll( devID );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.9. Group Motion Status Functions

3.4.9.1. NMC_GroupGetCommandPosAcs

Get the command position of a group axis in the axis coordinate system (ACS).

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupGetCommandPosAcs(I32_T DevID, I32_T GroupIndex, Pos_T *PRetCmdPosAcs);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
Pos_T *PRetCmdPosAcs: [Input] A pointer variable, [Output] The command position of the specified group axis.

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T      devID      = 0;
I32_T      groupIndex = 0;
Pos_T cmdPosAcs   = { 0 };
RTN_ERR ret    = 0;

ret = NMC_GroupGetCommandPosAcs( devID, groupIndex, &cmdPosAcs );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.9.2. NMC_GroupGetActualPosAcs

Get the actual position of a group axis in the axis coordinate system (ACS). The actual value is converted from the count value of the encoder based on the gear ratio or the lead screw pitch in the axis coordinate system (ACS).

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupGetActualPosAcs(I32_T DevID, I32_T GroupIndex, Pos_T *PRetActPosAcs);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
Pos_T *PRetActPosAcs: [Input] A pointer variable, [Output] The actual position of each group axis

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T      devID      = 0;
I32_T      groupIndex = 0;
Pos_T actPosAcs     = { 0 };
RTN_ERR ret    = 0;

ret = NMC_GroupGetActualPosAcs( devID, groupIndex, &actPosAcs );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.9.3. NMC_GroupGetCommandPosPcs

Get the command position of a group axis in the Cartesian coordinate system (PCS).

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupGetCommandPosPcs(I32_T DevID, I32_T GroupIndex, Pos_T *PRetCmdPosPcs);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
Pos_T *PRetCmdPosPcs: [Input] A pointer variable, [Output] The command position of the specified group axis.

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T      devID      = 0;
I32_T      groupIndex = 0;
Pos_T cmdPosPcs    = { 0 };
RTN_ERR ret    = 0;

ret = NMC_GroupGetCommandPosPcs( devID, groupIndex, &cmdPosPcs );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.9.4. NMC_GroupGetActualPosPcs

Get the actual position of a group axis in the Cartesian coordinate system (PCS). The actual value is converted from the count value of the encoder based on the gear ratio or the lead screw pitch in the Cartesian coordinate system (PCS).

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupGetActualPosPcs(I32_T DevID, I32_T GroupIndex, Pos_T *PRetActPosPcs);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
Pos_T *PRetActPosPcs: [Input] A pointer variable, [Output] The actual position of each group axis

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T      devID      = 0;
I32_T      groupIndex = 0;
Pos_T actPosPcs     = { 0 };
RTN_ERR ret    = 0;

ret = NMC_GroupGetCommandPosPcs( devID, groupIndex, &actPosPcs );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.9.5. NMC_GroupGetCommandPos

Get the command position of a group. The returned value is specified depended on the coordinate system.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupGetCommandPos(I32_T DevID, I32_T GroupIndex, I32_T CoordSys, Pos_T *PRetCmdPos);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T CoordSys: Specified the coordinate system (0:MCS, 1:PCS, 2:ACS)
Pos_T *PRetCmdPosPcs: [Input] A pointer variable, [Output] The command position depended on the specified coordinate system

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T      devID      = 0;
I32_T      groupIndex = 0;
I32_T      coordSys  = 2; // Get the command position in the axis coordinate system
Pos_T cmdPos = { 0 };
RTN_ERR ret    = 0;

ret = NMC_GroupGetCommandPos( devID, groupIndex, coordSys, &cmdPos );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.9.6. NMC_GroupGetActualPos

Get the actual position of a group. The returned value is specified depended on the coordinate system.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupGetActualPos(I32_T DevID, I32_T GroupIndex, I32_T CoordSys, Pos_T *PRetActPos);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T CoordSys: Specified the coordinate system (0:MCS, 1:PCS, 2:ACS)
Pos_T *PRetActPos: [Input] A pointer variable, [Output] The command position depended on the specified coordinate system

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T      devID      = 0;
I32_T      groupIndex = 0;
I32_T      coordSys  = 2; // Get the actual position in the axis coordinate system
Pos_T actPos   = { 0 };
RTN_ERR ret    = 0;

ret = NMC_GroupGetActualPos( devID, groupIndex, coordSys, &actPos );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.9.7. NMC_GroupGetMotionBuffSpace

Get the size of buffer space of group motion command.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupGetMotionBuffSpace(I32_T DevID, I32_T GroupIndex, I32_T *PRetFreeSpace);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
F64_T *PRetFreeSpace: [Input] A pointer variable, [Output] The free size of the motion buffer

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T devID      = 0;
I32_T groupIndex = 0;
I32_T freeSpace = 0;
RTN_ERR ret              = 0;

ret = NMC_GroupGetMotionBuffSpace( devID, groupIndex, &freeSpace );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.10. Group Returns to Home Functions

3.4.10.1.NMC_GroupSetHomePos

Set the origin of a group axis in the axis coordinate system.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupSetHomePos(I32_T DevID, I32_T GroupIndex, I32_T GroupAxesIdxMask, const Pos_T *PHomePosAcs);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T GroupAxesIdxMask: The group axis index mask is specified to execute the motion. Please refer to the below table.
const Pos_T *PHomePosAcs: A pointer variable to set the origin in the axis coordinate system (ACS).

Group axis	8	7	6	5	4	3	2	1
Bit index	7	6	5	4	3	2	1	0
The power is the index of bit	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

The usage of the group axis index mask (GroupAxesIdxMask) is described as follows:
If the group axes to be moved are the 1st, 3rd and 8th axes, the GroupAxesIdxMask is $2^0 + 2^2 + 2^7 = 133$.

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

- Reference:

None.

3.4.10.2.NMC_GroupAxesHomeDrive

Drive a group to move to the origin.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupAxesHomeDrive(I32_T DevID, I32_T GroupIndex, I32_T GroupAxesIdxMask);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T GroupAxesIdxMask: The group axis index mask is specified to execute the motion. Please refer to the below table.

Group axis	8	7	6	5	4	3	2	1
Bit index	7	6	5	4	3	2	1	0
The power is the index of bit	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

The usage of the group axis index mask (GroupAxesIdxMask) is described as follows:
If the group axes to be moved are the 1st, 3rd and 8th axes, the GroupAxesIdxMask is $2^0 + 2^2 + 2^7 = 133$.

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

- Reference:

None.

3.4.11. Group 2D Line or Arc Interpolation Motion Functions

3.4.11.1.NMC_GroupLineXY

Enable the group line interpolation motion on the XY plane from the current position to the target position in the Cartesian coordinate system. The group state will transfer from GROUP_STAND_STILL to GROUP_MOVING. If the group reaches the target position (i.e. the velocity is decreased to 0), the group state will transfer to GROUP_STAND_STILL. The default maximum velocity will refer to the value in the input parameters. If required, the maximum velocity can be set in the API, and the specified value will be stored in the corresponding parameters automatically.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupLineXY(I32_T DevID, I32_T GroupIndex, _opt_null_ const F64_T *PX, _opt_null_ const F64_T *PY, _opt_null_ const F64_T *PMaxVel);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
_opt_null_ const F64_T *PX: A pointer variable to set the target position at X-axis. Input NULL (0) to ignore the parameter (not to move).
_opt_null_ const F64_T *PY: A pointer variable to set the target position at Y-axis. Input NULL (0) to ignore the parameter (not to move).
_opt_null_ const F64_T *PMaxVel: A pointer variable to set the maximum velocity. Input NULL (0) to ignore the parameter.

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T      devID      = 0;
I32_T      groupIndex = 0;
F64_T      posX= 10.0;
F64_T posY      = 20.0;
RTN_ERR ret    = 0;

ret = NMC_GroupLineXY( devID, groupIndex, &posX, &posY, NULL );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.11.2.NMC_GroupCirc2R

Enable the group arc interpolation motion on the XY plane from the current position to the target position in the Cartesian coordinate system (radius method). The group state will transfer from GROUP_STAND_STILL to GROUP_MOVING. If the group reaches the target position (i.e. the velocity is decreased to 0), the group state will transfer to GROUP_STAND_STILL. The default maximum velocity will refer to the value in the input parameters. If required, the maximum velocity can be set in the API, and the specified value will be stored in the corresponding parameters automatically.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupCirc2R(I32_T DevID, I32_T GroupIndex, _opt_null_ const F64_T *PEX, _opt_null_ const F64_T *PEY, F64_T Radius, I32_T CW_CCW, _opt_null_ const F64_T *PMaxVel);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
_opt_null_ const F64_T *PEX: A pointer variable to set the target position at X-axis. Input NULL (0) to ignore the parameter (not to move).
_opt_null_ const F64_T *PEY: A pointer variable to set the target position at Y-axis. Input NULL (0) to ignore the parameter (not to move).
F64_T Radius: Radius of arc (negative value indicates the larger radian path).
I32_T CW_CCW: Rotation direction of arc (0=CW; 1=CCW)
_opt_null_ const F64_T *PMaxVel: A pointer variable to set the maximum velocity. Input NULL (0) to ignore the parameter.

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T      devID      = 0;
I32_T      groupIndex = 0;
F64_T      posX= 50.0;
F64_T posY     = 50.0;
F64_T radius   = 25.0;
I32_T      cwCcw      = 1;
RTN_ERR ret    = 0;

ret = NMC_GroupCirc2R( devID, groupIndex, &posX, &posX, radius, cwCcw, NULL );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.11.3.NMC_GroupCirc2C

Enable the group arc interpolation motion on the XY plane from the current position to the target position in the Cartesian coordinate system (circle center method). The group state will transfer from GROUP_STAND_STILL to GROUP_MOVING. If the group reaches the target position (i.e. the velocity is decreased to 0), the group state will transfer to GROUP_STAND_STILL. The default maximum velocity will refer to the value in the input parameters. If required, the maximum velocity can be set in the API, and the specified value will be stored in the corresponding parameters automatically.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupCirc2C(I32_T DevID, I32_T GroupIndex, _opt_null_ const F64_T *PEX, _opt_null_ const F64_T *PEY, _opt_null_ const F64_T *PCXOffset, _opt_null_ const F64_T *PCYOffset, I32_T CW_CCW, _opt_null_ const F64_T *PMaxVel);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
_opt_null_ const F64_T *PEX: A pointer variable to set the target position at X-axis. Input NULL (0) to ignore the parameter (not to move).
_opt_null_ const F64_T *PEY: A pointer variable to set the target position at Y-axis. Input NULL (0) to ignore the parameter (not to move).
_opt_null_ const F64_T *PCXOffset: A pointer variable to set the position of the circle center at X-axis. Input NULL (0) to ignore the parameter (not to move).
_opt_null_ const F64_T *PCYOffset: A pointer variable to set the position of the circle center at Y-axis. Input NULL (0) to ignore the parameter (not to move).
I32_T CW_CCW: Rotation direction of arc (0=CW; 1=CCW)
_opt_null_ const F64_T *PMaxVel: A pointer variable to set the maximum velocity. Input NULL (0) to ignore the parameter.

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T      devID      = 0;
I32_T      groupIndex = 0;
F64_T posTarX = 20.0;
F64_T posTarY =      0.0;
F64_T posCenX      = 10.0;
F64_T posCenY      =      0.0;
I32_T      cwCcw     = 1;
RTN_ERR ret    = 0;

ret = NMC_GroupCirc2C( devID, groupIndex, &posTarX, NULL, &posCenX, NULL, cwCcw, NULL );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.11.4.NMC_GroupCirc2B

Enable the group arc interpolation motion on the XY plane from the current position to the target position in the Cartesian coordinate system (pass-through method). The group state will transfer from GROUP_STAND_STILL to GROUP_MOVING. If the group reaches the target position (i.e. the velocity is decreased to 0), the group state will transfer to GROUP_STAND_STILL. The default maximum velocity will refer to the value in the input parameters. If required, the maximum velocity can be set in the API, and the specified value will be stored in the corresponding parameters automatically.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupCirc2B(I32_T DevID, I32_T GroupIndex, _opt_null_ const F64_T *PEX, _opt_null_ const F64_T *PEY, _opt_null_ const F64_T *PBX, _opt_null_ const F64_T *PBY, _opt_null_ const F64_T *PMaxVel);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
_opt_null_ const F64_T *PEX: A pointer variable to set the target position at X-axis. Input NULL (0) to ignore the parameter (not to move).
_opt_null_ const F64_T *PEY: A pointer variable to set the target position at Y-axis. Input NULL (0) to ignore the parameter (not to move).
_opt_null_ const F64_T *PBX: A pointer variable to set the pass-through point at X-axis. Input NULL (0) to ignore the parameter (not to move).
_opt_null_ const F64_T *PBY: A pointer variable to set the pass-through point at Y-axis. Input NULL (0) to ignore the parameter (not to move).
_opt_null_ const F64_T *PMaxVel: A pointer variable to set the maximum velocity. Input NULL (0) to ignore the parameter.

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T      devID      = 0;
I32_T      groupIndex = 0;
F64_T posTarX = 20.0;
F64_T posTarY = 20.0;
F64_T posBorX      = 0.0;
F64_T posBorY      = 20.0;
RTN_ERR ret    = 0;

ret = NMC_GroupCirc2B( devID, groupIndex, &posTarX, &posTarY, &posBorX,
&posBorY, NULL );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.11.5.NMC_GroupCirc2BEx

3.4.12. Group 3D Line or Arc Interpolation Motion Functions

3.4.12.1.NMC_GroupLine

Enable the group line interpolation motion from the current position to the target position in the Cartesian space. The group state will transfer from GROUP_STAND_STILL to GROUP_MOVING. If the group reaches the target position (i.e. the velocity is decreased to 0), the group state will transfer to GROUP_STAND_STILL.

- C/C++ Syntax Definition:

RTN_ERR NMC_GroupLine(I32_T DevID, I32_T GroupIndex, I32_T CartAxisMask, const Pos_T *PCartPos, _opt_null_ const F64_T *PMaxVel);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T CartAxisMask: The mask is specified to execute the motion in the Cartesian space. Please refer to the below table.
const Pos_T *PCartPos: A pointer variable to set the target position
_opt_null_ const F64_T *PMaxVel: A pointer variable to set the maximum velocity. Input NULL (0) to ignore the parameter.

Cartesian space coordinate Axis	V	U	C	B	A	Z	Y	X
Bit index	7	6	5	4	3	2	1	0
The power is the index of bit	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

The usage of the mask in the Cartesian space coordinate system (CartAxesMask) is described as follows:
If the group axes to be moved are the X-, Z- and A-axis, the GroupAxesIdxMask is $2^0 + 2^2 + 2^3 = 13$.

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T      devID      = 0;
I32_T      groupIndex     = 0;
I32_T      cartAxesMask = 13; // Move the X-, Z- and A-axis
Pos_T targetPos      = { 10, 20, 30, 40, 50, 60, 70 };
RTN_ERR ret    = 0;

ret = NMC_GroupLine( devID, groupIndex, cartAxesMask, &targetPos, NULL );
if( ret != 0 ) return ret;
```

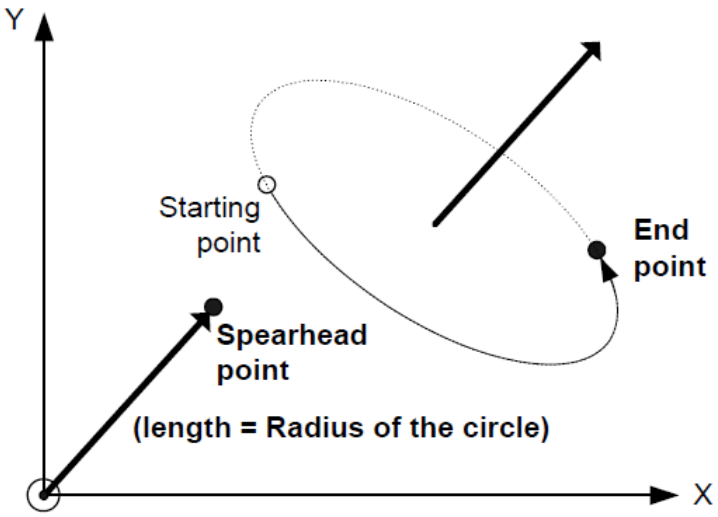
- Reference:

None.

3.4.12.2.NMC_GroupCircR

Enable the group arc interpolation motion from the current position to the target position in the Cartesian space (radius method). The group state will transfer from GROUP_STAND_STILL to GROUP_MOVING. If the group reaches the target position (i.e. the velocity is decreased to 0), the group state will transfer to GROUP_STAND_STILL.
The path algorithm of the arc motion (radius method) is developed based on the PLCopen specification. Please pay attention to the following items:

- The starting point of normal vector of the circle plane is the origin in the Cartesian space coordinate system. The unit vector cannot be input.
- The vector formed by the starting point and the end point must be perpendicular to the normal vector of the circle plane. Otherwise, an error code will be returned.
- The path developed based on the radius method can be divided to the larger radian path and the small radian path, and one of the paths shall be selected. They shall be determined by the sign of the radius. That is, the positive sign indicates the small radian path, and the negative sign indicates the larger radian path.
- If the arc is a 2D path, the normal vector of the circle plane is dispensable to set, but the rotation direction (CW_CCW) shall be set because the right-hand rule will be used in the XY-, YZ- or ZX-plane.
- If the arc is a 3D path, the rotation direction (CW_CCW) shall not be referred because the 3D path cannot be determined based on the right-hand rule.



- C/C++ Syntax Definition:

RTN_ERR NMC_GroupCircR(I32_T DevID, I32_T GroupIndex, I32_T CartAxisMask, const Pos_T *PCartPos, const Xyz_T *PNormalVector, F64_T Radius, I32_T CW_CCW, _opt_null_ const F64_T *PMaxVel);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T CartAxisMask: The mask is specified to execute the motion in the Cartesian space. Please refer to the below table.
const Pos_T *PCartPos: A pointer variable to set the target position
const Xyz_T *PNormalVector: A pointer variable to set the normal vector of the circle plane
F64_T Radius: Radius of arc (negative value indicates the larger radian path).
I32_T CW_CCW: Rotation direction of arc (0=CW; 1=CCW)
_opt_null_ const F64_T *PMaxVel: A pointer variable to set the maximum velocity. Input NULL (0) to ignore the parameter.

Cartesian space	V	U	C	B	A	Z	Y	X

coordinate Axis								
Bit index	7	6	5	4	3	2	1	0
The power is the index of bit	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

The usage of the mask in the Cartesian space coordinate system (CartAxesMask) is described as follows:
If the group axes to be moved are the X-, Z- and A-axis, the GroupAxesIdxMask is $2^0 + 2^2 + 2^3 = 13$.

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T      devID      = 0;
I32_T      groupIndex     = 0;
I32_T      cartAxesMask = 3; // Move the X- and Y-axis
Pos_T targetPos      = { 50, 50, 0, 0, 0, 0, 0 };
Xyz_T normalVec    = { 0, 0, 50 };
F64_T radius    = 50.0;
I32_T      cwCcw     = 1;
RTN_ERR ret    = 0;

ret = NMC_GroupCircR( devID, groupIndex, cartAxesMask, &targetPos,
&normalVec, radius, cwCcw, NULL );
if( ret != 0 ) return ret;
```

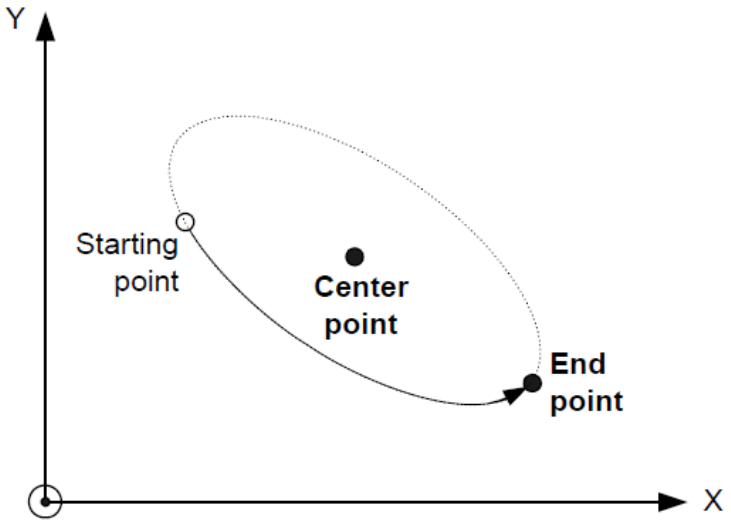
- Reference:

None.

3.4.12.3.NMC_GroupCircC

Enable the group arc interpolation motion from the current position to the target position in the Cartesian space (circle center method). The group state will transfer from GROUP_STAND_STILL to GROUP_MOVING. If the group reaches the target position (i.e. the velocity is decreased to 0), the group state will transfer to GROUP_STAND_STILL.
The path algorithm of the arc motion (circle center method) is developed based on the PLCopen specification. Please pay attention to the following items:

- The path developed based on the circle center method can be divided to the larger radian path and the small radian path, and one of the paths shall be selected. They shall be determined by the rotation direction (CW_CCW). That is, the CW indicates the small radian path, and the CCW indicates the larger radian path.
- If the arc is a 2D path, the path can be determined based on the rotation direction (CW_CCW) because the right-hand rule will be used in the XY-, YZ- or ZX-plane.



- C/C++ Syntax Definition:

RTN_ERR NMC_GroupCircC(I32_T DevID, I32_T GroupIndex, I32_T CartAxisMask, const Pos_T *PCartPos, I32_T CenOfsMask, const Xyz_T *PCenOfs, I32_T CW_CCW, _opt_null_ const F64_T *PMaxVel);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T CartAxisMask: The mask is specified to execute the motion in the Cartesian space. Please refer to the below table.
const Pos_T *PCartPos: A pointer variable to set the target position
I32_T CenOfsMask: The mask of the circle center in the Cartesian space. Please refer to the below table.
const Xyz_T *PCenOfs: A pointer variable to set the position of the circle center
I32_T CW_CCW: Rotation direction of arc (0=CW; 1=CCW)
_opt_null_ const F64_T *PMaxVel: A pointer variable to set the maximum. Input NULL (0) to ignore the parameter.

Cartesian space coordinate Axis	V	U	C	B	A	Z	Y	X
Bit index	7	6	5	4	3	2	1	0
The power is the index of bit	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

The usage of the mask in the Cartesian space coordinate system (CartAxesMask) is described as follows:
If the group axes to be moved are the X-, Z- and A-axis, the GroupAxesIdxMask is $2^0 + 2^2 + 2^3 = 13$.

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

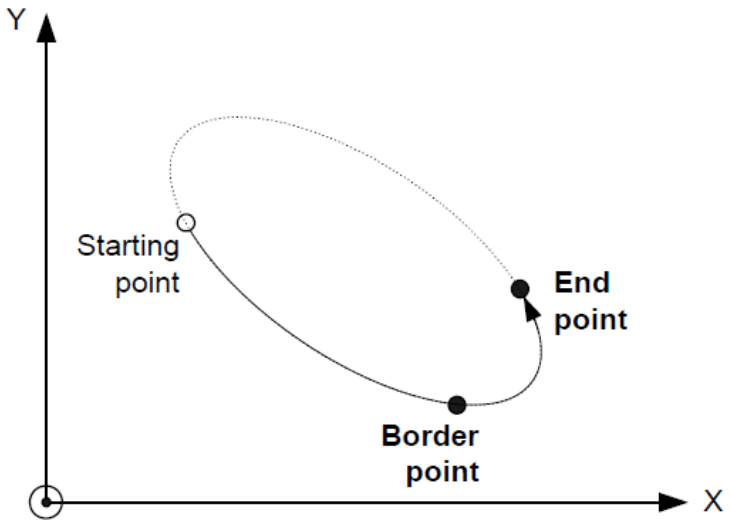
```
I32_T      devID      = 0;
I32_T      groupIndex     = 0;
I32_T      cartAxesMask = 7; // Move the X-, Y- and Z-axis
Pos_T targetPos      = { 50, 50, 0, 0, 0, 0, 0 };
I32_T      cenOfsMask    = 7; // Move the X-, Y- and Z-axis
Xyz_T      cenOfs     = { 50, 0, 50 };
I32_T      cwCcw     = 1;
RTN_ERR ret    = 0;

ret = NMC_GroupCircC( devID, groupIndex, cartAxesMask, &targetPos,
cenOfsMask, &cenOfs, cwCcw, NULL );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.12.4.NMC_GroupCircB

Enable the group arc interpolation motion from the current position to the target position in the Cartesian space (pass-through method). The group state will transfer from GROUP_STAND_STILL to GROUP_MOVING. If the group reaches the target position (i.e. the velocity is decreased to 0), the group state will transfer to GROUP_STAND_STILL.
The path algorithm of the arc motion (pass-through method) is developed based on the PLCopen specification. Please refer to the below figure for the coordination.



- C/C++ Syntax Definition:

RTN_ERR NMC_GroupCircB(I32_T DevID, I32_T GroupIndex, I32_T CartAxisMask, const Pos_T *PCartPos, I32_T BorPosMask, const Xyz_T *PBorPoint, _opt_null_ const F64_T *PMaxVel);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
I32_T CartAxisMask: The mask is specified to execute the motion in the Cartesian space. Please refer to the below table.
const Pos_T *PCartPos: A pointer variable to set the target position
I32_T BorPosMask: The mask of the pass-through point in the Cartesian space. Please refer to the below table.
const Xyz_T *PBorPoint: A pointer variable to set the pass-through point
_opt_null_ const F64_T *PMaxVel: A pointer variable to set the maximum. Input NULL (0) to ignore the parameter.

Cartesian space coordinate Axis	V	U	C	B	A	Z	Y	X
Bit index	7	6	5	4	3	2	1	0
The power is the index of bit	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

The usage of the mask in the Cartesian space coordinate system (CartAxesMask) is described as follows:
If the group axes to be moved are the X-, Z- and A-axis, the GroupAxesIdxMask is $2^0 + 2^2 + 2^3 = 13$.

- Return values:

Return an error code with the data type of RTN_ERR.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Examples:

```
I32_T      devID      = 0;
I32_T      groupIndex     = 0;
I32_T      cartAxesMask = 7; // Move the X-, Y- and Z-axis
Pos_T targetPos      = { 50, 50, 0, 0, 0, 0, 0 };
I32_T      borPosMask    = 7; // Move the X-, Y- and Z-axis
Xyz_T      borPoint  = { 50, 0, 50 };
RTN_ERR ret    = 0;

ret = NMC_GroupCircB( devID, groupIndex, cartAxesMask, targetPos, borPosMask,
borPoint, NULL );
if( ret != 0 ) return ret;
```

- Reference:

None.

3.4.12.5.NMC_GroupCircBEx

3.4.13. Tool Calibration Functions

3.4.13.1.NMC_ToolCalib_4p

Tool calibration - TCP translation method

- C/C++ Syntax Definition:

RTN_ERR NMC_ToolCalib_4p(const Pos_T *PMcsKinP1, const Pos_T *PMcsKinP2, const Pos_T *PMcsKinP3, const Pos_T *PMcsKinP4 , CoordTrans_T *PRetToolCoordTrans, F64_T *PRetTolerance);

- Parameters:

const Pos_T *PMcsKinP1: The first step position, TCP must (as far as possible) fall in the reference position
const Pos_T *PMcsKinP2: The second step position, TCP must (as far as possible) fall in the reference position
const Pos_T *PMcsKinP3: The third step position, TCP must (as far as possible) fall in the reference position
const Pos_T *PMcsKinP4: The fourth step position, TCP must (as far as possible) fall in the reference position
CoordTrans_T *PRetToolCoordTrans: Return setting of Tool coordinate convertion
F64_T *PRetTolerance: Return tolerance

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

- Examples:

- Reference:

None

3.4.13.2.NMC_ToolCalib_4pWithZ

Tool calibration - TCP translation with Z-direction setting method

- C/C++ SYNTAX DEFINITION:

RTN_ERR NMC_ToolCalib_4pWithZ(const Pos_T *PMcsKinP1, const Pos_T *PMcsKinP2, const Pos_T *PMcsKinP3, const Pos_T *PMcsKinP4ZDir, CoordTrans_T *PRetToolCoordTrans, F64_T *PRetTolerance)

- Parameters:

const Pos_T *PMcsKinP1: The first step position, TCP must (as far as possible) fall in the reference position
const Pos_T *PMcsKinP2: The second step position, TCP must (as far as possible) fall in the reference position
const Pos_T *PMcsKinP3: The third step position, TCP must (as far as possible) fall in the reference position
const Pos_T * PMcsKinP4ZDir: The fourth step position, the positive direction of Z axis of TCP needs to indicate negative direction of Z axis of MCS
CoordTrans_T *PRetToolCoordTrans: Return setting of Tool coordinate convertion
F64_T *PRetTolerance: Return tolerance

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

- Examples:

- Reference:

None

3.4.13.3.NMC_ToolCalib_4pWithOri

Tool calibration - TCP translation with orientation setting method

- C/C++ SYNTAX DEFINITION:

RTN_ERR NMC_ToolCalib_4pWithOri(const Pos_T *PMcsKinP1, const Pos_T *PMcsKinP2, const Pos_T *PMcsKinP3, const Pos_T *PMcsKinP4, const Pos_T *PMcsKinMinusZAxisPt, const Pos_T *PMcsKinYZPlanPt, CoordTrans_T *PRetToolCoordTrans, F64_T *PRetTolerance)

- Parameters:

const Pos_T *PMcsKinP1: The first step position, TCP must (as far as possible) fall in the reference position
const Pos_T *PMcsKinP2: The second step position, TCP must (as far as possible) fall in the reference position
const Pos_T *PMcsKinP3: The third step position, TCP must (as far as possible) fall in the reference position
const Pos_T *PMcsKinP4: The fourth step position, TCP must (as far as possible) fall in the reference position
const Pos_T *PMcsKinMinusZAxisPt: The fifth step position, reference position must fall in negative direction of Z axis of TCP
const Pos_T *PMcsKinYZPlanPt: The sixth step position, reference position must fall in positive direction of Y axis of TCP
CoordTrans_T *PRetToolCoordTrans: Return setting of Tool coordinate convertion
F64_T *PRetTolerance: Return tolerance

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

- Examples:

- Reference:

None

3.4.13.4.NMC_ToolCalib_Ori

Tool calibration - Orientation setting method

- C/C++ SYNTAX DEFINITION:

RTN_ERR NMC_ToolCalib_Ori(const Pos_T *PMcsKinOrg, const Pos_T *PMcsKinMinusZAxisPt, const Pos_T *PMcsKinYZPt, CoordTrans_T *PRetToolCoordTrans);

- Parameters:

const Pos_T * PMcsKinOrg: The first step position, TCP must (as far as possible) fall in the reference position

const Pos_T *PMcsKinMinusZAxisPt: The second step position, reference position must fall in negative direction of Z axis of TCP

const Pos_T *PMcsKinYZPlanPt: The third step position, reference position must fall in positive direction of Y axis of TCP

CoordTrans_T *PRetToolCoordTrans: Return setting of Tool coordinate convertion(Only modified the orientation transformation paramter)

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

- Examples:

- Reference:

None

3.4.14. Base Calibration Functions

3.4.14.1.NMC_BaseCalib_1p

Base teaching -1p method

- C/C++ SYNTAX DEFINITION:

RTN_ERR NMC_BaseCalib_1p(const Pos_T *PRefBaseP1, CoordTrans_T *PRetBaseCoordTrans)

- Parameters:

const Pos_T *PRefBaseP1: The first step position

CoordTrans_T *PRetBaseCoordTrans: Return the relationship respected to reference coordinate convertion

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

- Examples:

- Reference:

None

3.4.14.2.NMC_BaseCalib_2p

Base teaching -2p method

- Base teaching -2p methodC/C++ SYNTAX DEFINITION:

Base teaching -3p methodRTN_ERR NMC_BaseCalib_2p(const Pos_T *PRefBaseP1, const Pos_T *PRefBaseP2, CoordTrans_T *PRetBaseCoordTrans)

- Parameters:

const Pos_T *PRefBaseP1: The first step position

const Pos_T *PRefBaseP2: The second step position

CoordTrans_T *PRetBaseCoordTrans: Return the relationship respected to reference coordinate convertion

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

- Examples:

- Reference:

None

3.4.14.3.NMC_BaseCalib_3p

Base teaching -3p method

- Base teaching -2p methodC/C++ SYNTAX DEFINITION:

Base teaching -3p methodRTN_ERR NMC_BaseCalib_3p(const Pos_T *PRefBaseP1, const Pos_T *PRefBaseP2, const Pos_T *PRefBaseP3, CoordTrans_T *PRetBaseCoordTrans);

- Parameters:

const Pos_T *PRefBaseP1: The first step position

const Pos_T *PRefBaseP2: The second step position

const Pos_T *PRefBaseP3: The third step position

CoordTrans_T *PRetBaseCoordTrans: Return the relationship respected to reference coordinate convertion

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

- Examples:

- Reference:

None

3.4.15. 3D Simulation Functions

3.4.15.1.NMC_Group3DShow

Create or display a 3D simulation window

- C/C++ SYNTAX DEFINITION:

RTN_ERR NMC_Group3DShow(I32_T DevID, I32_T GroupIndex)

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

  After excuting system, NMC_Group3DShow() could be called to create a 3D view and load the 3D model, it may take several seconds to several minutes to complete 3D window creation based on different performance hardware platforms. Use NMC_Group3DHide() to hide the window, then use NMC_Group3DShow() can display again.

  NMC_Group3DShow() will return error code if there is no build-in group 3D model.

- Examples:

- Reference:

None

3.4.15.2.NMC_Group3DHide

Hide a 3D simulation window

- C/C++ SYNTAX DEFINITION:

RTN_ERR NMC_Group3DHide(I32_T DevID, I32_T GroupIndex);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

After excuting system, NMC_Group3DShow() could be called to create a 3D view and load the 3D model, can use NMC_Group3DHide() to hide the window, then use NMC_Group3DShow() can display again.

- Examples:

- Reference:

None

3.4.15.3.NMC_Group3DAlwaysTop

Set a 3D window properties are always on top

- C/C++ SYNTAX DEFINITION:

RTN_ERR NMC_Group3DAlwaysTop(I32_T DevID, I32_T GroupIndex, BOOL_T Enable);

- Parameters:

I32_T DevID: Device ID (DevID)
I32_T GroupIndex: Group index
BOOL_T Enable: FALSE (0):cancel the property; TRUE(1): Set the property

- Return values:

Return an error code.
If the function is called successfully, the return value is ERR_NEXMOTION_SUCCESS (0). Otherwise, the return value is an error code. All error codes are defined in the header file, NexMotionError.h.

- Usage:

  After excuting system, NMC_Group3DShow() could be called to create a 3D view and load the 3D model, then can use NMC_Group3DAlwaysTop() to set window always on top.

- Examples:

- Reference:

None

4. Definitions in Library

4.1. Data Type

4.1.1. Basic Data Type

All C/C++ data types used in APIs are defined in the header file, nex_type.h, and described as the below table:

Type	C/C++ Type	Description	Byte	Range
BOOL_T	int	Boolean	4	0:False, 1:True
U8_T	unsigned char	Unsigned integer	1	0 ~ 255
U16_T	unsigned short	Unsigned integer	2	0 ~ 65535
U32_T	unsigned int	Unsigned integer	4	0 ~ 4294967295
U64_T	unsigned __int64	Unsigned integer	8	0 ~ 18446744073709551615
I8_T	char	Signed integer	1	-128 ~ 127
I16_T	short	Signed integer	2	-32768 ~ 32767
I32_T	int	Signed integer	4	-2147483648 ~ 2147483647
I64_T	__int64	Signed integer	8	-9223372036854775808 ~ 9223372036854775807
F32_T	float	Float	4	IEEE-754, 7th decimal place
F64_T	double	Double precision float	8	IEEE-754, 15th decimal place
RTN_ERR	int	Error code	4	-2147483648 ~ 2147483647

4.1.2. Data Types related to Motion

The data types related to motion are defined in the header file, NexMotionDef.h.

4.1.2.1. Structure: Pos_T

It is used to describe the coordinate position of group and can be ACS coordinates or MCS/PCS coordinates depended on the definitions of API.

- C/C++ Syntax Definition:

```
typedef struct
{
    F64_T pos[MAX_POS_SIZE];
} Pos_T;
```

- Members:
  F64_T pos[MAX_POS_SIZE]: A position array of size 8.
  The pos[0~7] represents the coordinate axis 0~7 in the group for the ACS coordinate system, and it represents the X-axis, Y-axis, Z-axis, A-axis, B-axis, C-axis, U-axis, and V-axis for the Cartesian coordinate system (MCS/PCS).

4.1.2.2. Structure: Xyz_T

It is used to describe the X-axis, Y-axis and Z-axis of the Cartesian coordinate system.

- C/C++ Syntax Definition:

```
typedef struct
{
    F64_T pos[MAX_XYZ_SIZE];
} Xyz_T;
```

- Members:
  F64_T pos[MAX_XYZ_SIZE]: A position array of size 3.
  The pos[0], pos[1] and pos[2] represent the X-axis, Y-axis, and Z-axis of the Cartesian coordinate system, respectively.

4.1.2.3. Structure: CoordTrans_T

描述兩個標系之間之座標轉換關係

- **C/C++語法定義:**

```
typedef struct
{
    F64_T pose[NMC_MAX_POSE_DATA_SIZE];
} CoordTrans_T;
```

- **成員:**

F64_T pose[NMC_MAX_POSE_DATA_SIZE]: 大小為 6 的 F64_T 陣列。

pose[0]、pose[1]和 pose[2]分別代表卡式坐標系統 X 軸、Y 軸和 Z 軸之相對位置(平移)

pose[3]、pose[4]和 pose[5]分別代表卡式坐標系統 Z 軸、Y 軸和 X 軸之旋轉角度(degree)

4.1.3. Other Data Types

4.1.3.1. Hook Function Type: PF_NmcHookAPI

- C/C++ Syntax Definition:

```
typedef void(*PF_NmcHookAPI)( const void *FPFuncAddress , const char
*PFuncName, RTN_ERR ReturnCode, void *PUserData );
```

- Members:
  const void *FPFuncAddress: The index of the called function.
  const char *PFuncName: The name of the called function.
  RTN_ERR ReturnCode: The return value of the called function.
  void*PUserData: The index of user data set by NMC_DebugSetHookData().

4.1.3.2. Structure: NmcTime_T

A data structure is used to describe the system time.

- C/C++ Syntax Definition:

```
typedef struct
{
    U32_T year;          //  1601 through 30827
    U32_T month;             //  1 through 12.
    U32_T day;           //  1 through 31.
    U32_T hour;          //  0 through 23.
    U32_T minute;            //  0 through 59.
    U32_T second;            //  0 through 59
    U32_T milliseconds;      //  0 through 999
} NmcTime_T;
```

- Members:
  U32_T year: Year (C.E.)
  U32_T month: Month (1 ~ 12).
  U32_T day: Day (1 ~ 31).
  U32_T hour: Hour (1~ 23).
  U32_T minute: Minute (0 ~ 59).
  U32_T second: Second (0 ~ 59).
  U32_T milliseconds: Millisecond (0 ~ 999).

4.1.3.3. Structure: NmcMsg_T

A data structure is used to describe the system message.

- C/C++ Syntax Definition:

```
typedef struct
{
    U32_T       sizeOfStruct;
    NmcTime_T   localTime; U32_T   index;
    I32_T           type;
    char        source[NMC_MAX_MSG_SOURCE_SIZE];
    I32_T           id;
    I32_T           code;
    char        text[NMC_MAX_MSG_TEXT_SIZE];
} NmcMsg_T;
```

- Members:

U32_T	sizeOfStruct: The size of NmcMsg_T, and equivalent to the sizeof(NmcMsg_T).
NmcTime_T	localTime: The system time recorded when the message is generated.
U32_T	index: The index of message.
I32_T	type: The type of message. 0: Normal, 1: Warning, 2: Error
char	source[NMC_MAX_MSG_SOURCE_SIZE]: The source of message (reserved).
I32_T	id: The identification of message (reserved).
I32_T	code: The code of message.
char	text[NMC_MAX_MSG_TEXT_SIZE]: The content of message.

4.2. Constant Definitions

The constants related to NexMotion are defined in the header file, NexMotionDef.h.

4.2.1. Device Type Selection

Name	Value	Description
NMC_DEVICE_TYPE_SIMULATOR	0	The device type of simulator.
NMC_DEVICE_TYPE_ETHERCAT	1	The device type of EtherCAT.

4.2.2. Timeout Configuration of Wait Function

Name	Value	Description
NMC_WAIT_TIME_INFINITE	0xFFFFFFFF	Waiting for the completion of the function execution.

4.2.3. Device State

Name	Value	Description
NMC_DEVICE_STATE_INIT	1	Init: The device parameters are not configured or have been removed.
NMC_DEVICE_STATE_READY	2	Ready: The device parameters have been configured.
NMC_DEVICE_STATE_ERROR	3	Error: The device enters a critical error state, such as the communication failure.
NMC_DEVICE_STATE_OPERATION	4	Operation

4.2.4. Coordinate System Selection

Name	Value	Description
NMC_COORD_MCS	0	Mechanical coordinate system
NMC_COORD_PCS	1	Programming coordinate system
NMC_COORD_ACS	2	Axis coordinate system

4.2.5. Axis state

The axis state is read via NMC_AxisGetState(). The variable represents the meanings as follows:

Name	Value	Description
NMC_AXIS_STATE_DISABLE	0	Disable: The servo is OFF.
NMC_AXIS_STATE_STAND_STILL	1	Enable: The servo is stand still.
NMC_AXIS_STATE_HOMING	2	Executing the Homing motion.
NMC_AXIS_STATE_DISCRETE_MOTION	3	Executing the point-to-point motion.
NMC_AXIS_STATE_CONTINUOUS_MOTION	4	Executing the continuous motion.
NMC_AXIS_STATE_STOPPING	5	Receiving the Stop command and slowing down to stop.
NMC_AXIS_STATE_STOPPED	6	Receiving the Stop command and stopped
NMC_AXIS_STATE_WAIT_SYNC	7	Receiving the Wait command and waiting for SYNC signal.
NMC_AXIS_STATE_ERROR	10	Stop for error.

4.2.6. Bit Code of Axis status

The axis status is read via NMC_AxisGetStatus(). Each bit code of the variable represents the meanings as follows:

Name	Value	Description
NMC_AXIS_STATUS_EMG	0	A latched signal is issued at the EMG signal (*1).
NMC_AXIS_STATUS_ALM	1	A latched signal is issued at the servo alarm (*1)
NMC_AXIS_STATUS_PEL	2	A latched signal is issued at the positive limit signal (*1)
NMC_AXIS_STATUS_NEL	3	A latched signal is issued at the negative limit signal (*1)
NMC_AXIS_STATUS_PSEL	4	A latched signal is issued at the software positive limit signal (*1)
NMC_AXIS_STATUS_NSEL	5	A latched signal is issued at the software negative limit signal (*1)
NMC_AXIS_STATUS_ENA	6	The axis is Enable or Disable.
NMC_AXIS_STATUS_ERR	7	Axis error
NMC_AXIS_STATUS_TAR	8	The axis has reached the target position.
NMC_AXIS_STATUS_CSTP	9	The axis commands to Stop.
NMC_AXIS_STATUS_ACC	10	The axis is operating in acceleration.
NMC_AXIS_STATUS_DEC	11	The axis is operating in deceleration.
NMC_AXIS_STATUS_MV	12	The axis is operating at the maximum velocity.
NMC_AXIS_STATUS_OP	13	The axis is operating.
NMC_AXIS_STATUS_STOP	14	The axis is STOP.
NMC_AXIS_STATUS_RPEL	16	Positive limit signal: 1: Triggered, 0: Not Triggered
NMC_AXIS_STATUS_RNEL	17	Negative limit signal: 1: Triggered, 0: Not Triggered
NMC_AXIS_STATUS_RHOM	18	Home signal: 1: High level, 0: Low level

(*1) After the error state is resolved, the Latched signal will be cleared to 0 after the function NMC_AxisResetState() is called.

4.2.7. Bit Mask of Motion Status

The motion status is read via NMC_AxisGetStatus(). Each bit mask of the variable represents the meanings

as follows:

Name	Value	Description
NMC_AXIS_STATUS_MASK_EMG	0x00000001	The mask for a latched signal issued at the EMG signal (*1).
NMC_AXIS_STATUS_MASK_ALM	0x00000002	The mask for a latched signal issued at the servo Alarm (*1)
NMC_AXIS_STATUS_MASK_PEL	0x00000004	The mask for a latched signal issued at the positive limit signal (*1)
NMC_AXIS_STATUS_MASK_NEL	0x00000008	The mask for a latched signal issued at the negative limit signal (*1)
NMC_AXIS_STATUS_MASK_PSEL	0x00000010	The mask for a latched signal issued at the software positive limit signal (*1)
NMC_AXIS_STATUS_MASK_NSEL	0x00000020	The mask for a latched signal issued at the software negative limit signal (*1)
NMC_AXIS_STATUS_MASK_ENA	0x00000040	The mask to indicate the axis is Enable or Disable.
NMC_AXIS_STATUS_MASK_ERR	0x00000080	The mask for axis error
NMC_AXIS_STATUS_MASK_TAR	0x00000100	The mask to indicate the axis has reached the target position.
NMC_AXIS_STATUS_MASK_CSTP	0x00000200	The mask to indicate the axis commands to Stop.
NMC_AXIS_STATUS_MASK_ACC	0x00000400	The mask to indicate the axis is operating in acceleration.
NMC_AXIS_STATUS_MASK_DEC	0x00000800	The mask to indicate the axis is operating in deceleration.
NMC_AXIS_STATUS_MASK_MV	0x00001000	The mask to indicate the axis is operating at the maximum velocity.
NMC_AXIS_STATUS_MASK_OP	0x00002000	The mask to indicate the axis is operating.
NMC_AXIS_STATUS_MASK_STOP	0x00004000	The mask to indicate the axis is STOP.
NMC_AXIS_STATUS_MASK_RPEL	0x00010000	The mask for the positive limit signal: 1: Triggered, 0: Not Triggered
NMC_AXIS_STATUS_MASK_RNEL	0x00020000	The mask for the negative limit signal: 1: Triggered, 0: Not Triggered
NMC_AXIS_STATUS_MASK_RHOM	0x00040000	The mask for the home signal: 1: High level, 0: Low level

(*1) After the error state is resolved, the Latched signal will be cleared to 0 after the function NMC_AxisResetState() is called.

4.2.8. Group Coordinate Number

Name	Value	Description
GROUP_AXIS_X	0	X axis of the coordinate system
GROUP_AXIS_Y	1	Y axis of the coordinate system
GROUP_AXIS_Z	2	Z axis of the coordinate system
GROUP_AXIS_A	3	An axis of the coordinate system
GROUP_AXIS_B	4	B axis of the coordinate system
GROUP_AXIS_C	5	C axis of the coordinate system
GROUP_AXIS_U	6	U axis of the coordinate system
GROUP_AXIS_V	7	V axis of the coordinate system

4.2.9. Group Coordinate Number Mask

Name	Value	Description
GROUP_AXIS_MASK_X	0x00000001	The mask for X axis of the coordinate system
GROUP_AXIS_MASK_Y	0x00000002	The mask for Y axis of the coordinate system
GROUP_AXIS_MASK_Z	0x00000004	The mask for Z axis of the coordinate system
GROUP_AXIS_MASK_A	0x00000008	The mask for A axis of the coordinate system
GROUP_AXIS_MASK_B	0x00000010	The mask for B axis of the coordinate system
GROUP_AXIS_MASK_C	0x00000020	The mask for C axis of the coordinate system
GROUP_AXIS_MASK_U	0x00000040	The mask for U axis of the coordinate system
GROUP_AXIS_MASK_V	0x00000080	The mask for V axis of the coordinate system

4.2.10. Group State

The group state is read via NMC_GroupGetState(). The variable represents the meanings as follows:

Name	Value	Description
NMC_GROUP_STATE_DISABLE	0	Disable: The group servo is OFF.
NMC_GROUP_STATE_STAND_STILL	1	Enable: The group servo is stand still.
NMC_GROUP_STATE_STOPPED	2	Receiving the Stop command and stopped
NMC_GROUP_STATE_STOPPING	3	Receiving the Stop command and slowing down to stop.
NMC_GROUP_STATE_MOVING	4	Executing the move command
NMC_GROUP_STATE_HOMING	5	Executing the Homing motion.
NMC_GROUP_STATE_ERROR	6	Stop for error.

4.2.11. Bit Code of Group State

The group status is read via NMC_GroupGetStatus(). Each bit code of the variable represents the meanings as follows:

Name	Value	Description
NMC_GROUP_STATUS_EMG	0	A latched signal is issued at the external EMG signal (*1).
NMC_GROUP_STATUS_ALM	1	A latched signal is issued at a group axis servo Alarm (*1)
NMC_GROUP_STATUS_PEL	2	A latched signal is issued at the positive limit signal of a group axis (*1)
NMC_GROUP_STATUS_NEL	3	A latched signal is issued at the negative limit signal of a group axis (*1)
NMC_GROUP_STATUS_PSEL	4	A latched signal is issued at the software positive limit signal a group axis (*1)
NMC_GROUP_STATUS_NSEL	5	A latched signal is issued at the software negative limit signal a group axis (*1)
NMC_GROUP_STATUS_ENA	6	The group is Enable or Disable.
NMC_GROUP_STATUS_ERR	7	Group (a group axis) error
NMC_GROUP_STATUS_CSTP	9	No displacement of all group axes (i.e. the displacement is 0).
NMC_GROUP_STATUS_ACC	10	Moving in the Cartesian coordinate system (along a line or an arc) and accelerating to the maximum velocity or decelerating. The value is 0 for PTP or JOG motion.
NMC_GROUP_STATUS_DEC	11	Moving in the Cartesian coordinate system (along a line or an arc) and decelerating to the target position or STOP. The value is 0 for PTP or JOG motion.
NMC_GROUP_STATUS_MV	12	Moving in the Cartesian coordinate system (along a line or an arc) at the maximum velocity. The value is 0 for PTP or JOG motion.
NMC_GROUP_STATUS_OP	13	The group is moving. That is the state is GROUP_MOVING, GROUP_HOMING or GROUP_STOPPING.
NMC_GROUP_STATUS_STOP	14	The group is STOP. That is the state is GROUP_STOPPED.

(*1) After the error state is resolved, the Latched signal will be cleared to 0 after the function NMC_GroupResetState() is called.

4.3. Error Code

The error codes are defined in the header file, NexMotionError.h, and described as the below table:

Define	value	Description
ERR_NEXMOTION_SUCCESS	0	The operation completed successfully
ERR_NEXMOTION_EXTERNAL_LIBRARY_NOT_FOUND	-1	The system cannot find the externel library
ERR_NEXMOTION_API_NOT_FOUND	-2	The system cannot find an API address when specified library is loading
ERR_NEXMOTION_LOAD_EXTERNAL_LIBRARY_FAILED	-3	The system cannot load the externel library
ERR_NEXMOTION_LOAD_RUNTIME_FAILED	-4	The system cannot load the runtime library
ERR_NEXMOTION_FILE_NOT_FOUND	-5	The system cannot find the specified file
ERR_NEXMOTION_FILE_OPEN_FAILED	-6	The system cannot open the specified file
ERR_NEXMOTION_FILE_LOAD_FAILED	-7	The system cannot load the specified file
ERR_NEXMOTION_FILE_BAD_FORMAT	-8	The format of the file is bad
ERR_NEXMOTION_FILE_READ_PROHIBIT	-9	The file cannot be read
ERR_NEXMOTION_FILE_WRITE_PROHIBIT	-10	The file cannot be write
ERR_NEXMOTION_FILE_VERSION_INCOMPTIBLE	-11	The version of the file is incompatible
ERR_NEXMOTION_OPENUP_RUNTIME_FAILED	-12	The system cannot openup the runtime
ERR_NEXMOTION_OUT_OF_SYSTEM_RESOURCES	-15	The system resources is inefficient
ERR_NEXMOTION_EXTERNAL_CALL_FAILED	-16	An external call or system call has made an error
ERR_NEXMOTION_SYSTEM_NOT_INITIALIZATION	-21	The system cannot be accessed before initialization process
ERR_NEXMOTION_SYSTEM_CLOSED_DENIED	-22	The system cannot be closed before clean up process
ERR_NEXMOTION_OPERATION_DENIED	-23	In current state, the system cannot accept the operation
ERR_NEXMOTION_PERMISSION_DENIED	-24	The user does not have permission
ERR_NEXMOTION_UNEXPECTED_EXCEPTION	-25	The operation occurred unexpected exception
ERR_NEXMOTION_SYSTEM_NOT_READY	-26	The system is not ready
ERR_NEXMOTION_OPERATION_BUSY	-27	The system is busy and cannot accept the operation
ERR_NEXMOTION_WAIT_FAILED	-28	The wait function has failed. No wait condition
ERR_NEXMOTION_PROCESS_TIMEOUT	-31	System perform current process is timeout
ERR_NEXMOTION_RUNTIME_RESPONSE_TIMEOUT	-32	Runtime module does not respond in a certain time
ERR_NEXMOTION_OBJECT_ID_INVALID	-41	The system cannot find the object through specified "ID", "Index", or "Handle"
ERR_NEXMOTION_PARAMETER_NUMBER_INVALID	-42	The parameter number is invalid
ERR_NEXMOTION_PARAMETER_VALUE_INVALID	-43	The parameter value is invalid
ERR_NEXMOTION_PARAMETER_READ_ONLY	-44	The parameter is read only
ERR_NEXMOTION_ACCESS_AREA_INVALID	-45	The operation attempted to access invalid area

ERR_NEXMOTION_POINTER_NULL	-46	The input pointer variable is null
ERR_NEXMOTION_QUEUE_EMPTY	-47	The queue is empty.
ERR_NEXMOTION_STRUCT_SIZE_INCOMPTIBLE	-48	The size of structure is incomptible
ERR_NEXMOTION_INITIAL_AXIS_POSITION_INVALID	-100	The initial position of an axis is out of limit
ERR_NEXMOTION_INVERSE_KINEMATICS_FAILED	-101	Inverse kinematics process is failed
ERR_NEXMOTION_INVERSE_KINEMATICS_OVER_AXIS_LIMIT	-102	The solution of inverse kinematics is out of limit
ERR_NEXMOTION_INVERSE_KINEMATICS_SINGULAR	-103	The solution of inverse kinematics is singular
ERR_NEXMOTION_KINEMATICS_TYPE_INVALID	-104	The type of kinematics is invalid
ERR_NEXMOTION_AXIS_COUNT_INVALID	-105	The count of axis is invalid
ERR_NEXMOTION_GROUP_COUNT_INVALID	-106	The count of group is invalid
ERR_NEXMOTION_AXIS_MAPPING_INVALID	-107	The map setting of the axis is invalid
ERR_NEXMOTION_KINEMATICS_PARAMETER_INVALID	-108	The kinematics parameter of a group is invalid
ERR_NEXMOTION_EMERGEERR_NEXMOTION_EMERGENCY_STOP_ACTIVENCY_STOP_ACTIVE	-109	Emergency stop signal is detected.
ERR_NEXMOTION_ENABLE_SWITCH_FULL_PRESSED	-110	Enabling switch full pressed signal is detected.
ERR_NEXMOTION_SAFE_GUARD_ACTIVE	-111	Safe-guard signal is detected.
ERR_NEXMOTION_SAFETY_ERROR	-112	Safety error is detected.