创博股份有限公司

IoT 智动化解决方案

NexMotion 函式库

使用者手册

版本：1.4

日期：2018-7-12

版权与免责声明

本文件内的所有数据属创博股份有限公司(以下简称本公司)专属财产,均受知识产权相关法规（包括但不限于著作权法 ）所保障。任何未经本公司授权的使用均属侵权行为。若未经本公司事先书面同意，本文件资料之全部或部份均不可被复印、销售、散布、修改、发表、储存或以其他方式作不当利用。

为力求文件之正确性及完整性，本公司保留在任何时间、不另行通知之情况下，变更或修改本文件之权利。

运行中的机械或设备具有一定的危险性，使用者在做任何操作前，应特别注意并应做好安全防护措施，本公司不承担因不当使用本文件所述设备所造成的直接或间接损失。

文件版本纪录

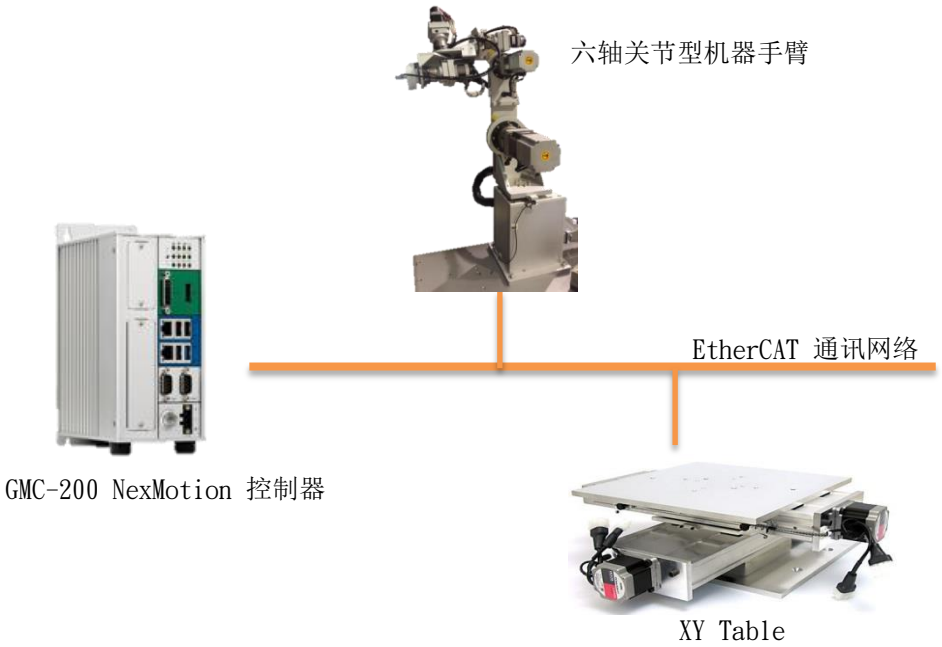
版本	修改纪录
1.0	First released.
1.1	Add Programming theory
1.2	1. Insert a chapter 3.2.8 get description APIs. 2. Add an axis parameter. Chapter 2.2
1.3	Modify description of NMC_GroupGetMotionBuffSpace()
1.4	1.Add Ch1.4.2.4~7 for tool/base setting and teaching 2.Add Ch2.3 Tool/Base parameters for group 3.Add Ch3.4.13~14 tool/base teaching APIs

目录

1. 编程原理

NexMotion 应用程序的基本开发流程如下:

1. 控制系统规划
2. 利用 NexMotion Studio 对受控系统进行设定与测试
3. 利用 NexMotion Studio 产生受控系统配置档(NCF 档案)
4. 控制程序开发
5. 编译与除错
6. 测试与微调控制程序



控制系统范例（控制器 + 受控系统）

前 3 个步骤主要目的为确认:

1.受控系统的安装与配线是否妥善? 比如伺服马达是否已正确安装? I/O 装置是可正确运作?等

2.轴控参数是否正确，是否可以透过 NexMotion studio 进行设备操作，比如说受控系统为一只六轴关节式机器手臂(Articulated robot)和一组两轴之 XY 平台系统(XY Table)，可以透过 NexMotion studio 进行机构参数之设定,单位设定以及加减速参数等设定，并使用 NexMotion studio 进行归原点操作，点对点运动操作等确认运行的方向，移动的距离单位等是否符合系统规划。NexMotion Studio 的使用方法可参考”NexMotion studio 使用者手册”。

待前述步骤完成后，会自动在系统默认路径中(C:\Nexcom\)产生组态档案，使用者应避免更动该路径底下之档案，避免不预期之系统错误产生。

第 4 步骤:控制程序开发，意即透过 NexMotion 所提供之函式库进行控制程序之开发，透过呼叫 API 对控制器下达命令以完成各式各样之应用。在接下来的章节中将分别函式库的功能作一系统性的说明包含

1. 系统操作
2. I/O 控制
3. 单轴控制
4. 群组控制

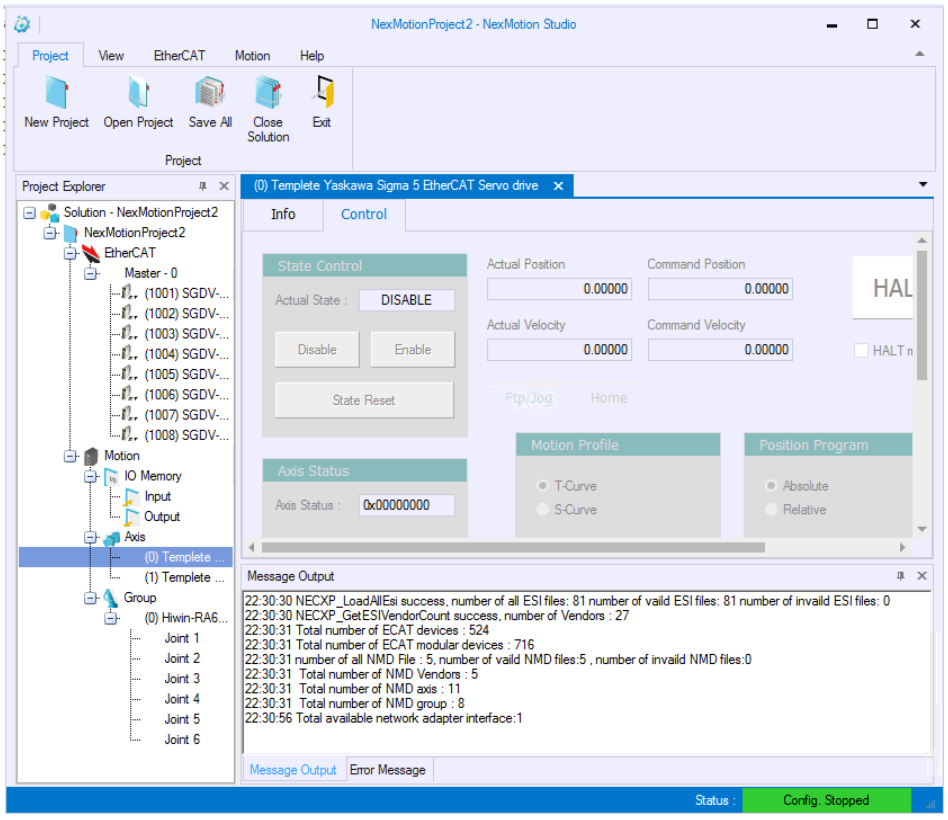
可以透过 API 命名来了解其分类，比如说

以 NMC_Device…开头的 API 为「系统操作」类的 API，

以 NMC_Axis…开头的 API 为「单轴控制」相关 API，

以 NMC_Group…开头的 API 为「群组控制」相关 API，以此类推

亦可搭配参考产品安装目录下所提供之范例程序作为应用开发之基础进行延伸变化。



NexMotion Studio 开发设定工具画面

1.1. 系统操作

系统相关操作章节描述包括:

1. 系统初始化
2. 看门狗
3. 除错功能

1.1.1. 系统初始化

开始使用函式库之前必须先进行初始化的流程，最简单的方式就是呼叫 NMC_DeviceOpenUp() 进行启动，若成功启动会回传控制器的标识符(ID)，尔后可使用此标识符来对控制器下达命令。

下面为一个系统启动和系统关闭的 C 程序范例:

```
#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...
    }

    // Controller is start up successfully.
    // Do something...
    // 可撰写控制相关程序…

    NMC_DeviceShutdown( devID );

    return 0;
}
```

NMC_DeviceOpenUp() 是一个阻塞式(Blocked) 函式，该函式会进行所有系统初始化的工作包括

1. 建立控制器，产生控制器标识符
2. 加载控制器组态档（NCF 档案）
3. 启动控制器通讯

上述这些工作将会花费数秒钟，若函式成功返回代表控制器完成启动，接下来的程序便可开始对单轴或群组等下达运动控制命令。另外，若不希望使用阻塞式函式，本函式库也另提供非阻塞式

(Non-blocked)的呼叫方式 NMC_DeviceOpenUpRequest()，此函数调用后立即返回，控制器收到命令后同样开始进行上述启动工作，待过一段时间后控制器完成启动工作其装置状态(Device state)会切换至「OPERATION」，可使用 NMC_DeviceGetState()进行轮询(Polling)状态或利用 NMC_DeviceWaitOpenUpRequest() 等待启动完成。

1.1.2. 关闭控制器

关闭控制器只需要呼叫 NMC_DeviceShutdown()，当使用者呼叫此函式后，控制器会立即关闭通讯，因此用户必须注意应用程序的关闭程序，例如，确认相关设备之运动状态皆已停妥后再呼叫 NMC_DeviceShutdown()将控制器彻底关闭。

1.1.3. 进阶系统初始化

一般初始化，可采用前述初始化流程较为简便快速，但某些应用情况下希望启动前先进行部分参数设定，可采用下述分段的方式进行控制器初始化:

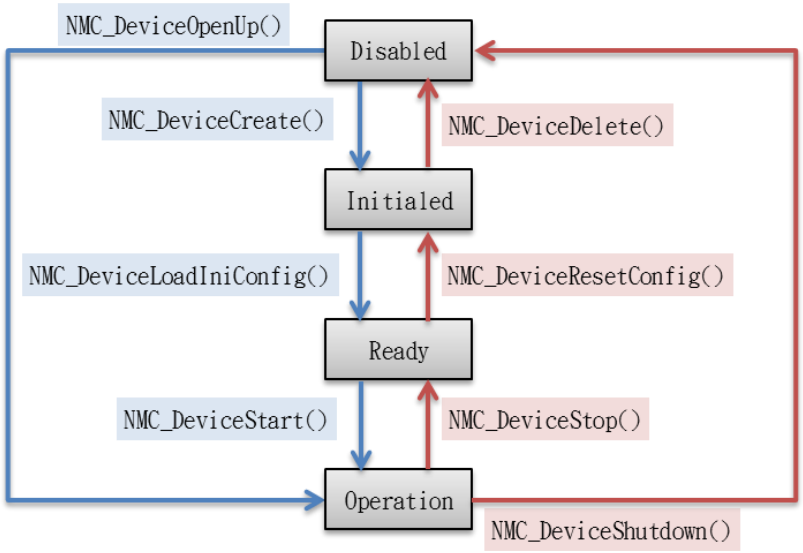
```
#include "NexMotion.h"
#include "NexMotionError.h"


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 someting here...
    // Parameter setting...

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

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

```
    // ...

    NMC_DeviceShutdown( devID );

    return 0;
}
```

若采用此种分段的启动放式，下列 3 个 API 必须依序被呼叫使用

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

目的是将控制器之控制器状态(Device state)会切换至「OPERATION」，下图为相关函式的呼叫与控制器状态变化的关系图:



控制器状态变化

上述的控制器状态可使用 NMC_DeviceGetState()读取之。

1.1.4. 看门狗定时器(Watch Dog Timer)

看门狗定时器(Watch dog timer)为控制器的一个专用定时器，当定时器被起启动后且在规定的时间内应用程序必须清除定时器的计数值，否则当定时器到达设定的时间后控制器会自动进行相对应的关闭动作。

此功能主要是为防范应用程序当机或者系统其他程序造成系统当机情况下的一种保护措施，当系统当机时因控制器的计数器没有办法再被应用程序清除时而发生过时(Timeout)而启动关闭程序。

在开发除错阶段若开启此功能需特别注意，常发生除错过程因中断程序(断点)，导致定时器过时系统关闭而发生不预期状况，因此建议开发阶段可以不启用此功能，待开发完毕后再加上此功能做额外强化的安全保护。

看门狗定时器(Watch dog timer)之使用的方式一般为启动一个系统 timer 中断或者建立一个独立的执行序(Thread)，透过 NMC_DeviceWatchdogTimerEnable()启动看门狗定时器，并使用 NMC_DeviceWatchdogTimerReset()来定时清除定时器，清除的频率通常比设定的时间要快上一倍甚至更多。若要停止看门狗定时器使用 NMC_DeviceWatchdogTimerDisable()

下面为一个例子:

```
#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;
```

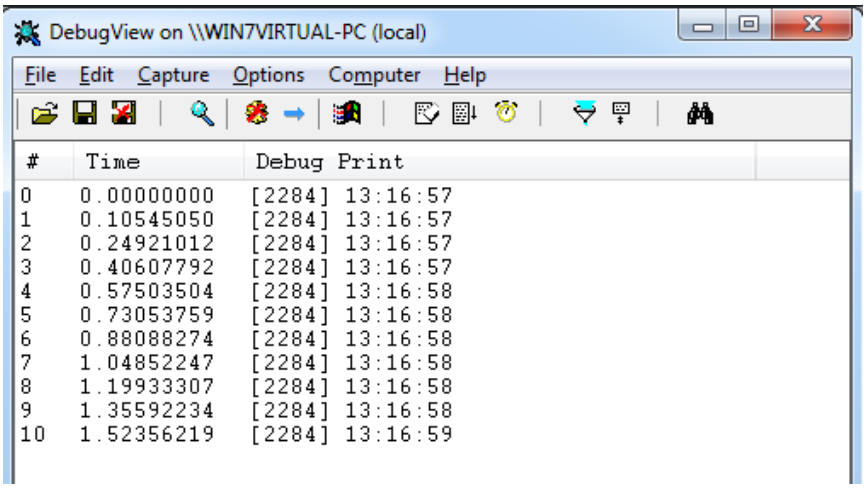
```
    HANDLE  threadHandle;

    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...
    }

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

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

    NMC_DeviceShutdown( devID );

    return 0;
}
```

1.1.5. 除错相关

协助用户除错相关功能有两类，第一类是系统讯息，第二类是API trace。

1.1.5.1. 系统讯息

系统讯息是指控制器在运行过程中会发出讯息存入到控制器的消息队列(Message queue)，可透过NMC_MessagePopFirst() 将队列中之讯息读出，当讯息从队列中读出后讯息既从队列中删除。为方便除错，可呼叫NMC_MessageOutputEnable()将系统讯息复制转发一份至MS Windows系统讯息(OutputDebugString)之中，使用者可下载DebugView小工具来拦截Windows之系统讯息。DebugView 工具可至 https://docs.microsoft.com/zh-tw/sysinternals/downloads/debugview 下载



DebugView 画面

1.1.5.2. 函式追踪(API Trace)

函式追踪(API Trace)主要是追踪用户之应用程序呼叫NexMotion 函式库的情况。透过使用NMC_DebugSetTraceMode()可开启API Trace的功能，根据设定的Trace模式当NexMotion 所提供之API被呼叫时可将呼叫的函式名称跟输入的参数值和错误回传值输出到MS Windows系统讯息(OutputDebugString)之中，用户可利用DebugView窗口软件拦截观察NexMotion函数调用之情况。

另外，更进一步可使用嵌入函式(Hooked function或称Callback function)，将自定义的嵌入函式嵌入到NexMotion 的函数之中。将用户自定义义的函数可透过NMC_DebugSetHookFunction() 注册到系统之中，当NexMotion 函式库中的API被呼叫执行后要返回前会自动呼叫使用者所嵌入的函式(Hooked function)。

注册到系统中的嵌入函数是全局的，亦即一旦注册成功，所有被呼叫的NexMotion API都会呼叫同一个嵌入函式，可透过NMC_DebugGetApiAddress()查询呼叫的API名称。

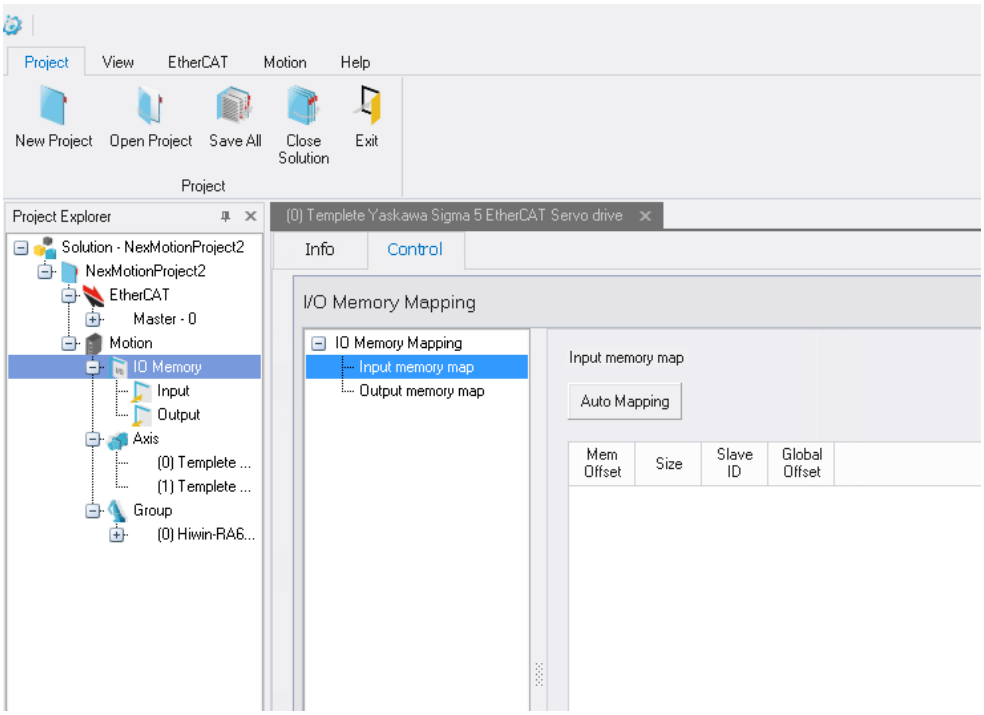
下面为一个使用范例:

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

void MyHookFunction(
      const void   *PFuncAddress  // [i] Function name call this hook funciton.
    , 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...
    }

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

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

    NMC_DeviceShutdown( devID );

    return 0;
}
```

1.2. I/O 控制

NexMotion 提供弹性且高速的 I/O 控制，存取 I/O 如同存取内存一般的。使用上，在存取 I/O 前须先使用 NexMotion studio 来设定 I/O 装置的内存映像位置，将 I/O 映像到虚拟内存后，存取该位置内存时既可控制或读取 I/O，其设定方式请参考 NexMotion Studio 使用手册。



NexMotion Studio 设定画面

设定后，即可使用下列函式来存取虚拟内存(控制或读取 I/O 装置)

- NMC_ReadInputMemory()
- NMC_ReadOutputMemory()
- NMC_WriteOutputMemory()

须注意，I/O 内存的更新率规格是每 10 ms 更新一次(100 Hz)，因此若呼叫 Digital output 的控制频率高于 100Hz 将无法及时响应。

1.3. 单轴(Axis)控制

有别于机构中协同作动的轴(group axis)，在本章节所介绍的单轴控制功能，其适用范围对应到机械结构端乃为独立作动(Independent)的轴(single axis)。就功能面来看，本章节的内容主要区分为三大部分，第一部分为单轴的设定，包含：单位设定、软件极限保护设定；第二部分则为运动控制的功能，包含:激磁、点对点运动、JOG 运动、停止运动与 Change on fly 的功能；第三部分主要着重在读取单轴运动相关信息的功能，包含:读取单轴目前的状态、运动信息。

1.3.1. 单轴单位设定

由于单轴运动控制 API 所需输入的参数，譬如：点对点运动的位置，或是 JOG 运动的速度，都是运行在用户定义的单位(user unit)。为了建立用户定义的单位与实际设定到驱动器的命令脉波数(Command count)之间的关系，用户必须先设定单位相关参数。下表列出与单位设定有关之单轴参数:

Param. Num.	Sub. Index	说明	备注
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) 启动后无法修改参数

- 0x00: Mechanical pitch (user unit/rev)

  此参数主要描述机构端转一圈等于多少个 user unit

- 0x01: Mechanical revolution

  0x02: Motor revolution

  上述两个参数必须搭配设定，主要用来描述马达与机构端之间的齿轮比关系。以旋转机构来说，马达与机构之间通常存在减速机，假设减速机的齿轮比等于 80，代表马达端转动 80 圈，机构端会相对应转 1 圈，则 0x01 设定值为 1，0x02 设定值为 80。以直线机构使用导螺杆来说，若马达端与机构端之间只存在联轴器，中间没有减速机构，则 0x01 与 0x02 的设定值皆为 1。但若马达端与机构端之间存在减速机构，则必须依据减速机构的齿轮比设定 0x01 与 0x02。

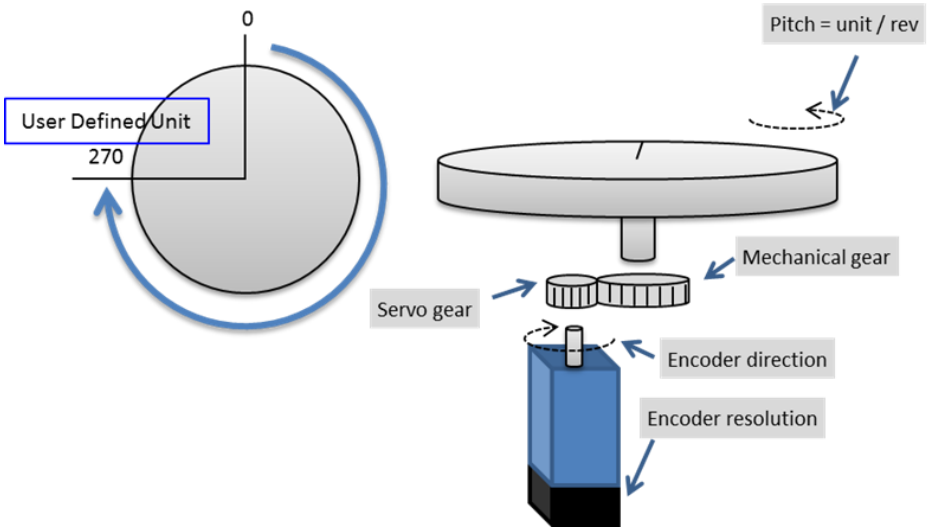
- 0x03: Encoder resolution (pulse/rev)

  此参数主要设定编码器的分辨率，代表马达转一圈，编码器会回传多少个脉波数(pulse count)。以 20 bit 分辨率的编码器来说，2 的 20 次方等于 1048576，则此参数的设定值为 1048576。

单位转换公式如下:

$$1\ \text{User unit} = \ \text{EncoderResolution} \times \frac{Motor\ revolution}{Mechanical\ revolution} \div Pitch\ \ (\text{Pulse})$$

单位设定范例一：圆盘机构



旋转型电机 + 减速机 + 旋转机构

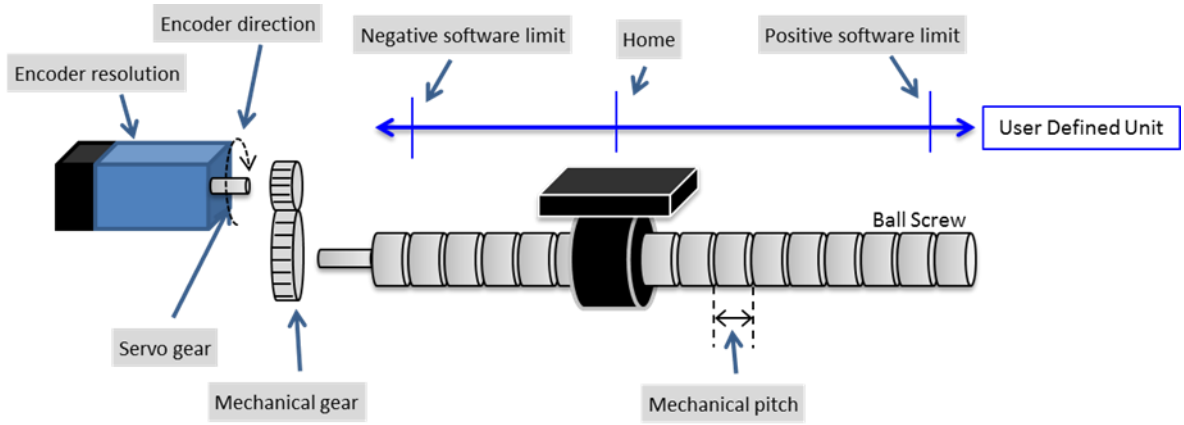
当减速机减速比 为 1:5（马达转 5 圈时机构转 1 圈），Encoder 分辨率为 1,048,576 pulse/rev
吾假设用户单位(user unit)订为每单位 0.001 degree ，参数设定如下:

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

User unit(um) 和 pulse 之间的关系为:

1 user unit (0.001 degree) = 1048576 x (5/1) / 360000 = 14.5636 pulse

单位设定范例二：螺杆机构



旋转型电机 + 减速机 + 线性螺杆

当减速机减速比为 1:2（马达转 5 圈时机构转 1 圈），Encoder 分辨率为 131,072 pulse/rev，螺杆导程为 5 mm/rev，吾假设用户单位(user unit)订为每单位 1 mm ，参数设定如下:

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

User unit(mm) 和 pulse 之间的关系为:
1 mm = 131072 x (10/1) / 5 = 262144 pulse

其他相关参数说明:

- 0x04: Encoder direction

此参数主要提供单轴反向的功能，default 值为 0。由于在单轴运动控制单元中，会依据编码器回传的脉波数计算单轴的位置坐标(user unit)，若单轴运动方向对应到位置坐标值的改变，与使用者预期的相反，可以将此参数设定为 1，则单轴运动方向对应到位置坐标值的改变将会反向。此功能主要是让用户可以在控制器端直接可以进行运动反向的设定。

- 0x05: Encoder type

此参数主要设定编码器的类型，若编码器为增量型，则设定值为 0；若为绝对型，则设定值为 1。

- 0x06，SubIndex = 0: Enable external encoder
  0x06，SubIndex = 1: External Encoder ratio

此参数主要用来设定另一个编码器，或是当输入到驱动器的命令脉波数与编码器回传的脉波数不一致时，可以将 Enable 的参数(SubIndex 0)设定为 1，则单轴的位置坐标值会利用编码器回传的读值乘上 External encoder ratio 进行计算，此 ratio 代表的是编码器读值转换为单轴位置坐标必须乘以的比例。

- 0x07: Cancel synchronized actual position to command position when servo enable

此参数主要用来设定在激磁单轴时，设定到驱动器的命令脉波数是否解除与编码器回传的脉波数同步。一般来说，对于伺服马达，在激磁单轴时，由于处在位置闭回路控制模式，因此，设定到驱动器的命令脉波数需要与编码器的脉波数同步；但对于步进马达来说，由于没有位置闭回路控制，因此，在激磁时，设定到驱动器的命令脉波数必须解除与编码器的脉波数同步，若强制两者同步，甚至可能发生在激磁的当下，马达随即高速运转，造成错误发生。

除了单轴(Axis)外，对于群组(Group)来说，各群组轴在 ACS 坐标系下，由于坐标值对应的单位为user unit因此,在系统读取控制器组态档 (NCF 档案)之前,也必须利用 NexMotion Studio 设定与单位相关的参数，设定方式与单轴设定相同。

上述这些参数因为与机构组件的选用与配置有关,因此,在呼叫 NMC_DeviceOpenUp()之前，必须先利用 NexMotion Studio 设定相关的参数，以产生相对应的控制器组态档 (NCF 档案)。若使用者已经成功呼叫 NMC_DeviceOpenUp()，若控制器状态处于「OPERATION」，则无法再更改与单位设定相关的参数。

若用户使用进阶系统初始化的方式初始化控制器，因为在 NMC_DeviceLoadIniConfig()的阶段，系统会读取控制器组态档 (NCF 档案)，所以，一旦控制器的状态处于「READY」，则无法再更改与单位设定相关的参数。

1.3.2. 软件极限保护

在实际的机构中,单轴往往有行程上的限制,且因为马达有最大可输出的力矩与速度限制，因此，在单轴运动的参数中，有相对应的正负方向之位置极限、最大可允许速度与最大可允许加速度。在运动控制单元中，有提供相对应的软件保护机制，确保单轴在运动的过程中，机构在允许的限制条件下作动。对于单轴运动来说,当启动软件极限保护功能,在呼叫单轴运动的函式时，会检查相关的运动参数，若不满足设定的极限条件，则函式会回传相对应的错误码。对于群组的单轴来说，当群组正在进行轨迹运动，若各别的单轴有超出设定的极限保护值，控制器会进行停止运动。以下，分别说明与极限保护相关的参数，

1.3.2.1. 位置极限保护

Param. Num.	Sub. Index	说明	备注

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): Enable 该功能时参数生效, 其他时间修改不会立刻变更

上述的参数主要与软件位置极限有关，若要启动单轴软件位置保护功能，必须先设定极限位置(SubIndex 0 或 2)，设定完位置后，再设定 SubIndex 1 或 3 为 1(Enable)，以启动软件保护功能。启动后，若欲更改极限位置，必须先关闭此功能(将 SubIndex 1 或 3 设定为 0)，待设定好新的极限位置后(SubIndex 0 或 2)，再启用(将 SubIndex 1 或 3 设定为 1)方可运作。

1.3.2.2. 速度极限保护

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

(*2): Enable 该功能时参数生效, 其他时间修改不会立刻变更

上述参数主要与单轴的速度限制保护有关，若要启动单轴最大速度保护功能，必须先设定极限速度(SubIndex 0)，设定完速度后，再设定 SubIndex 1 为 1(Enable)，以启动单轴最大速度限制保护功能。启动后，若欲更改极限速度，必须先关闭此功能(将 SubIndex 1 设定为 0)，待设定好新的极限速度后(SubIndex 0)，再启用(将 SubIndex 1 设定为 1)方可运作。

1.3.2.3. 加速度限制保护

Param. Num.	Sub. Index	说明	备注
0x12	0	Maximum acceleration limit (unit/sec2)	(*2)
	1	Enable max. acceleration limit	

(*2): Enable 该功能时参数生效, 其他时间修改不会立刻变更

上述参数主要与单轴的最大加速度限制保护有关，若要启动单轴最大加速度保护功能，必须先设定极限加速度(SubIndex 0)，设定完加速度后，再设定 SubIndex 1 为 1(Enable)，以启动单轴最大加速度限制保护功能。启动后，若欲更改极限加速度，必须先关闭此功能(将 SubIndex 1 设定为 0)，待设定好新的极限加速度后(SubIndex 0)，再启用(将 SubIndex 1 设定为 1)方可运作。请注意，单轴在运动过程中，启动紧急停止运动 NMC_AxisStop()所使用的减加速度并不受限于此保护功能所制定的最大加速度(SubIndex 1)。

1.3.3. 单轴启动与状态

在启动单轴进行运动之前，必须先呼叫 NMC_AxisEnable()，让马达进入激磁状态。由于此函式为非阻塞式呼叫，当呼叫此函式并成功回传，并不代表马达已经进入激磁状态，使用者可以呼叫 NMC_AxisGetState() 取得单轴目前的状态，当单轴状态切换到「NMC_AXIS_STATE_STAND_STILL」，代表马达已经成功进入激磁状态。

除了透过单轴状态(State of axis)确认马达是否已经进入激磁状态，使用者也可以呼叫 NMC_AxisGetStatus()读取单轴运动信息(Status of axis)，当 ENABLE (ENA,bit 6)转变为 1，代表马达已经成功进入激磁状态。

在呼叫 NMC_AxisEnable()之前，可以先呼叫 NMC_AxisGetState()，确认马达驱动器是否有报警(Alarm)，当马达驱动器有报警，单轴状态会切换到「NMC_AXIS_STATE_ERROR」。使用者也可以呼叫 NMC_AxisGetStatus()，当单轴运动信息(status of axis)的 ALM(bit 1)与 ERR(bit 7)转变为 1，代表马达驱动器有报警的情况，此时，必须呼叫 NMC_AxisResetDriveAlm()，以清除驱动器的报警。请注意，并不是所有的驱动器报警都可以透过此函式清除，有些驱动器报警必须藉由断电再上电的方式重置。当清除驱动器报警后，必须再呼叫 NMC_AxisResetState()将单轴的状态从「NMC_AXIS_STATE_ERROR」切换到「NMC_AXIS_STATE_DISABLE」，此时，单轴运动信息的 ALM(bit 1)与 ERR(bit 7) 会清除为 0。在呼叫 NMC_AxisResetState()时，若驱动器有报警，此函式会先清除驱动器报警后再将单轴状态回复到「NMC_AXIS_STATE_DISABLE」。

当单轴成功进入激磁状态后，即可以呼叫单轴运动相关函式进行运动。当运动结束，或是在运动中发生突发状况，使用者可以直接呼叫 NMC_AxisDisable()，将马达解除激磁状态。当成功解除激磁状态，单轴状态会切换到「NMC_AXIS_STATE_DISABLE」，单轴运动信息的 ENA(bit 6)会清除为 0。

1.3.4. 单轴速度百分比设定

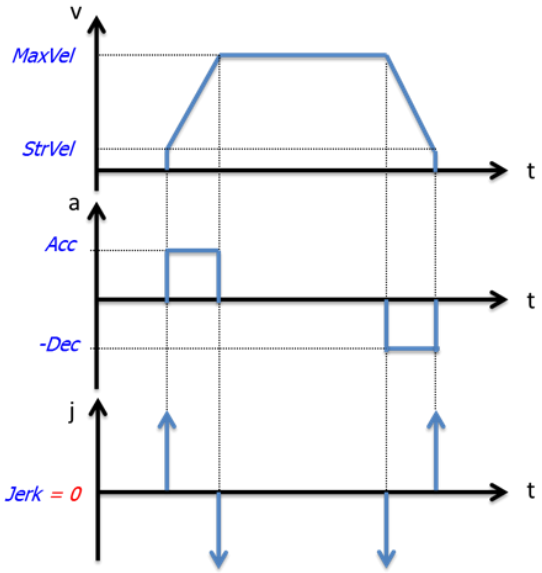
在单轴的运动中，包括点对点运动与 JOG 运动，都会依据单轴参数:0x32 的设定值作为目标速度。在某些应用情境中，通常是在手动操作模式下，为了确认点位或运作程序是否满足需求，会希望能够降低运动的目标速度，此时，用户可以呼叫 NMC_AxisSetVelRatio()设定速度百分比，成功设定后，运动控制模块将会依据单轴参数:0x32 (Max. velocity)乘上设定的速度百分比做为单轴运动的目标速度。此外，呼叫 NMC_AxisGetVelRatio()可以取得目前单轴运动的速度百分比。

此函式除了可以在单轴尚未进行任何运动前呼叫，也支持在线呼叫，也就是当单轴正在进行点对点运动或是 JOG 运动时，呼叫此函式除了会依据设定的百分比自动改变目前运动的目标速度，对于接下来输入的单轴点对点运动与 JOG 运动命令，其目标速度也都会依据此设定的百分比进行规划。

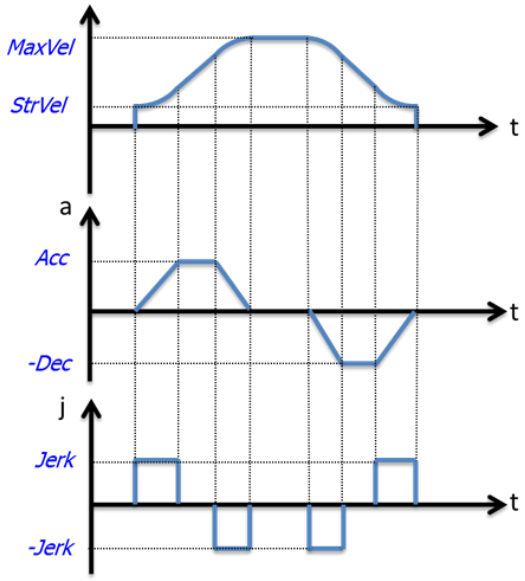
有关速度百分比设定的数值，范围从 0(包含) ~ 1(包含)，设定超出此范围的数值，在呼叫 NMC_AxisSetVelRatio()时将会回传错误码。若单轴在进行点对点或 JOG 运动时，将速度百分比设为 0，则单轴将降速到 0，且单轴运动信息 (Status of axis)的 CSTP(bit 9)会转变为 1。虽然在此状况下，单轴处于静止状态，但原先执行的点对点运动或 JOG 运动仍然在运作，因此单轴运动信息的 OP(bit 13)会保持为 1，当速度百分比再被设定为大于 0 的值，单轴将会自动运动到目标位置(点对点运动)或目标速度(JOG 运动)。

1.3.5. 单轴点对点(PTP)运动

在单轴运动中，点对点(PTP)运动是一个很常使用到的运动形式，使用者可以呼叫 NMC_AxisPtp()，给定目标位置或相对位移，运动控制模块将依据相关的单轴参数进行速度曲线规划，将单轴移动到用户指定的位置，如下图所示。



梯形速度曲线



S 形速度曲线

点对点运动相关之相关单轴参数:

Param. Num.	Sub. Index	说明	备注
0x28	0	Base velocity (unit/sec)	
0x31	0	Profile type	
0x32	0	Max. velocity (unit/sec)	
0x33	0	Acceleration (unit/sec2)	
0x34	0	Deceleration (unit/sec2)	
0x35	0	Jerk (unit/sec3)	

以下分别说明相关参数的意义与使用方式：

- 0x28： Base velocity (unit/sec)

当单轴在静止状态下启动点对点运动，一般来说，速度曲线会从初速为 0 开始加速到目标速度，但在步进马达的应用中，常需要设定初始的起跳速度，在此需求下，使用者可以将参数 0x28 设定为起跳速度，则在启动点对点运动与 JOG 运动时，初始速度会依据参数 0x28 的设定值进行规划。除了初始速度，当单轴在进行点对点运动或是停止运动时，末速度也会设定为参数 0x28 的设定值。

- 0x30： Absolute or relative programming

在点对点运动 NMC_AxisPtp()中，使用者输入的参数 TargetPos，可以是绝对位置坐标，也可以是相对位移。当 0x30 参数设定为 1，则点对点运动输入的参数 TargetPos 为相对位移；当 0x30 参数设定为 0，则 TargetPos 为绝对位置坐标。

- 0x31： Profile type

当使用者呼叫 NMC_AxisPtp()，控制器的运控控制模块会先规划一个速度曲线，并依据此速度曲线在每个通讯周期时间计算出单轴的命令位置，最后将单轴移动到指定的目标位置。通常速度曲线有两种形式，包含梯形速度曲线与 S 形速度曲线。当 0x31 设定为 0，速度曲线为梯形曲线，加速度不连续；当 0x31 设定为 1，速度为 S 形曲线，加速度连续。在相同的目标速度与加速度的情况下，采用 S 形速度曲线需要较多的时间到达目标位置或是目标速度，但因为加速度为连续，较可避免振动的发生。若采用梯形速度曲线，到达目标位置或目标速度所需要的时间较少，但因为加速度不连续，容易引起振动的产生。

- 0x32： Max. velocity (unit/sec)

此参数设定值为单轴点对点运动与 JOG 运动所采用的目标速度，如上图的 MaxVel。

- 0x33： Acceleration (unit/sec2)

此参数设定值为单轴点对点运动、JOG 运动从初始速度到目标速度所采用的加速度，如上图的 Acc。。

- 0x34： Deceleration (unit/sec2)

此参数设定值为单轴点对点运动从目标速度到末速度或是 Halt 运动所采用的减加速度，如上图的 Dec。

- 0x35： Jerk (unit/sec3)

此参数设定值为 S 形速度曲线的 Jerk，乃加速度曲线图(a-t)的斜率。

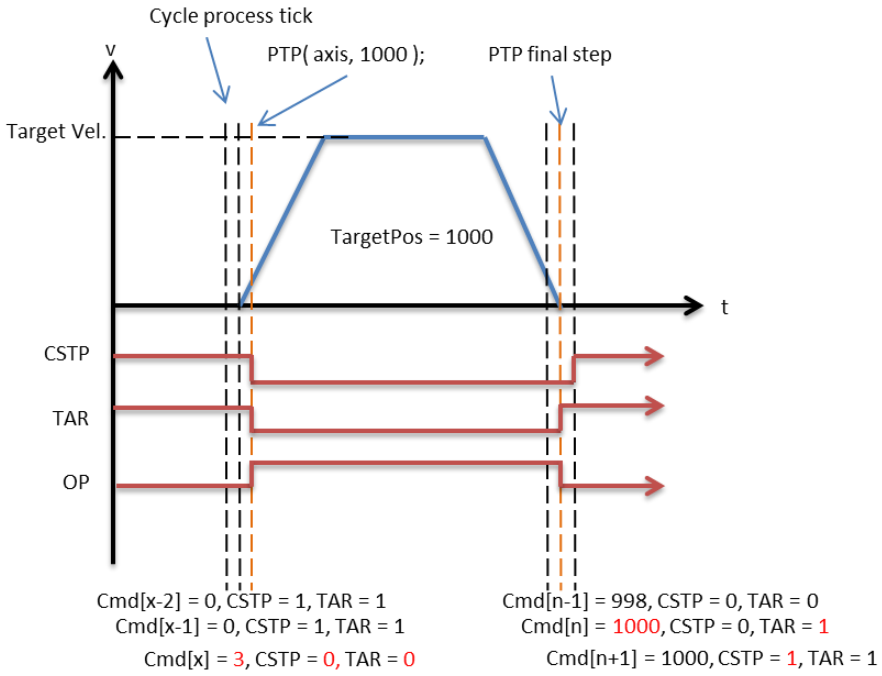
- 0x36： Buffer mode

单轴除了可以进行单节运动，也可以输入一连串的运动命令储存到单轴的运动队列(Motion Queue)中，透过参数 0x36 的设定，定义新输入的运动命令与前一个运动命令之间的连接方式。有关单轴运动队列的运作行为，以及参数 0x36 的说明，将在单轴运动队列章节进行说明，以下，先针对单一个点对点运动的相关内容进行说明。

当成功呼叫NMC_AxisPtp()后，单轴轴状态会切换到「NMC_AXIS_STATE_DISCRETE_MOTION」。有关呼叫 NMC_AxisPtp()之前与呼叫后的单轴状态改变之行为，请参考下列表格：

运动命令 / 目前状态	NMC_AxisPtp()
NMC_AXIS_STATE_DISABLE	禁止，回传错误
NMC_AXIS_STATE_STAND_STILL	状态改变至 AXIS_STATE_DISCRETE_MOTION
NMC_AXIS_STATE_HOMING	禁止，回传错误，正在进行的 homing 不会被影响
NMC_AXIS_STATE_DISCRETE_MOTION	依照单轴参数:0x36(Buffer mode)设定值储存到运动队列或立即执行
NMC_AXIS_STATE_CONTINUOUS_MOTION	依照单轴参数:0x36(Buffer mode)设定值储存到运动队列或立即执行 当 Buffer mode 为 Blending，禁止，回传错误
NMC_AXIS_STATE_STOPPING	禁止，回传错误
NMC_AXIS_STATE_STOPPED	禁止，回传错误
NMC_AXIS_STATE_WAIT_SYNC	依照单轴参数:0x36(Buffer mode)决定 PTP 命令被储存到运动队列 1. Abort: 清除运动队列后存入 2. Buffered: 存入运动队列 3. Blending: 存入运动队列 若 Buffer mode 为 Blending，而储存在运动队列的上一个运动命令为 JOG，则回传错误。
NMC_AXIS_STATE_ERROR	禁止，回传错误

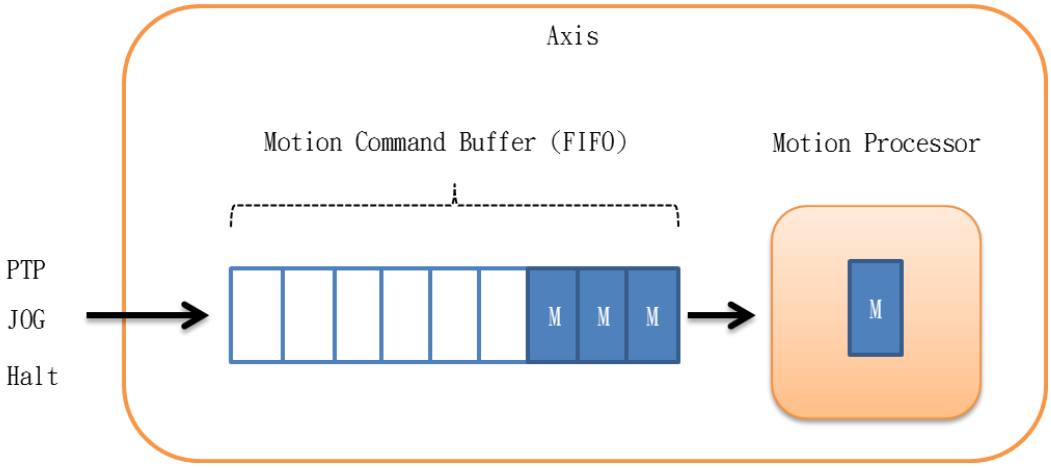
成功呼叫 NMC_AxisPtp()且开始进行 PTP 运动后，过程中的运动状态可以透过呼叫 NMC_AxisGetStatus()取得单轴运动信息，其中 bit 8 ~ bit 13 与运动状态相关，以下图表示：



PTP 运动过程中的运动状态

1.3.6. 单轴运动队列

每个单轴皆具有独立之运动命令队列(Motion command buffer)，因此在进行 PTP 或 JOG 运动过程中，若再输入新的运动命令，则依据单轴参数:0x36 (Buffer mode)的设定值，会有不同的行为模式:



单轴运动队列

- 每个单轴各有独立之运动队列(Queue)，单轴的处理单元会至运动队列中取出运动命令执行。
- Buffer size = 32
- NMC_AxisPtp()、NMC_AxisJog()和 NMC_AxisHalt()运动可被填入运动队列之中
- 可透过 NMC_AxisGetMotionBuffSpace()查看目前队列的容量

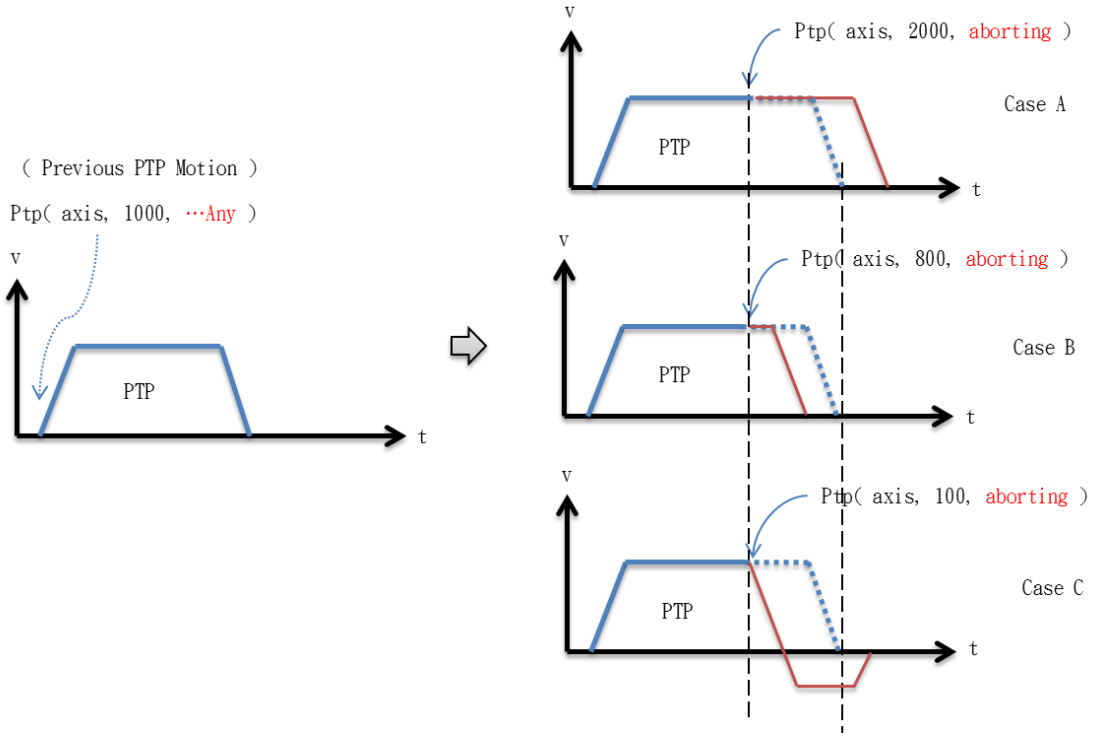
- NMC_AxisHalt()会停止目前的运动但不会清空队列，如果队列中有其他运动命令会接续执行
- NMC_AxisStop()会停止目前的运动并且清空队列
- NMC_AxisDisable()会连带清空队列

以下说明单轴参数:0x36 (Buffer mode)的使用行为：

- 当单轴状态在「NMC_AXIS_STATE_STAND_STILL」，呼叫单轴运动函式时，不管 0x36(Buffer mode)的参数设定值为何，会立即启动。
- 当单轴正在进行运动的过程中，使用者呼叫新的运动命令，则新的运动命令以何种方式与目前的运动或前一次运动命令(储存在单轴运动队列中的运动命令)连结起来，乃依据单轴参数:0x36 的设定值，说明如下：

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

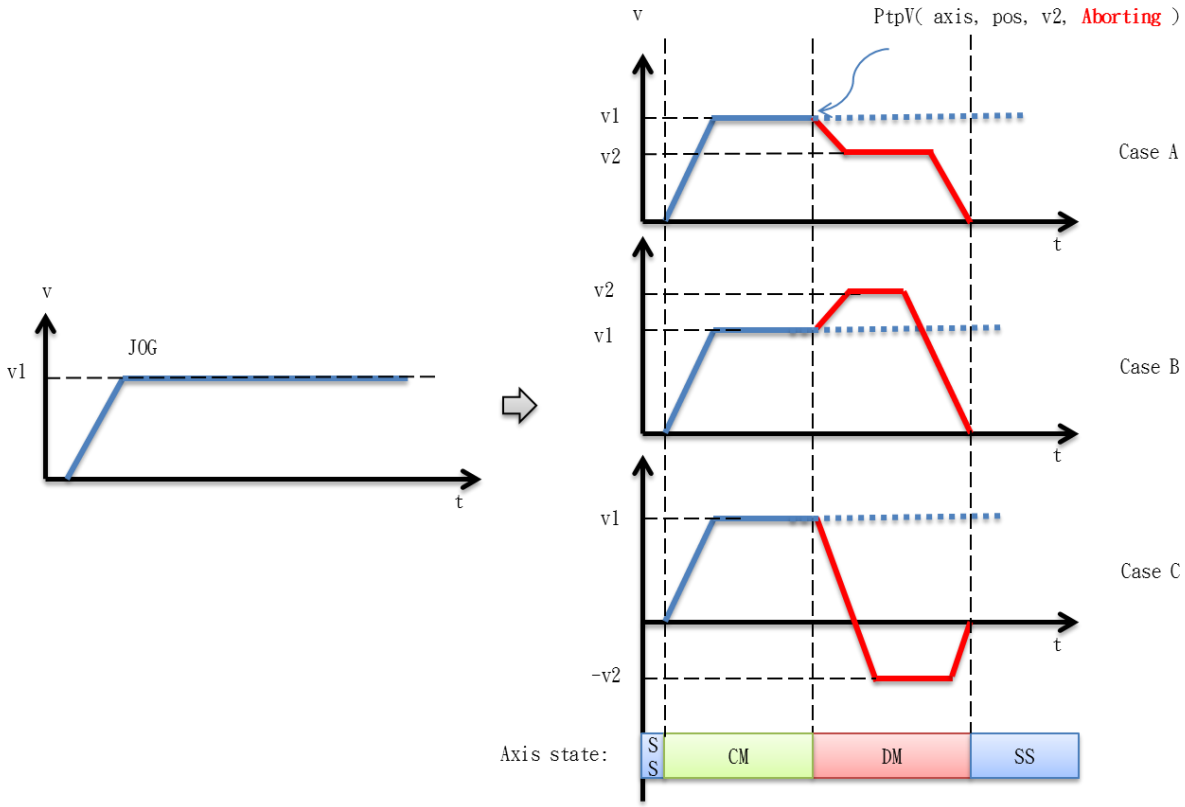
储存在单轴运动队列中的运动命令将会被清除，而原本正在进行的单轴运动将会被舍弃，由新的运动命令接管单轴的运动。以下面范例进行说明：



PTP 运动中由另一 PTP 运动中断(Aborting)之三种情况

在上图的范例中，原先单轴启动了点对点运动，目标位置为 1000，当正在运动的过程中，又呼叫了新的点对点运动，且 0x36 设定值为 0 (Aborting)，则依据新的目标位置会有图中所示之三种可能情况，其中，当单轴往正方向运动，而新的目标位置小于目前的位置时，会产生反转运动。

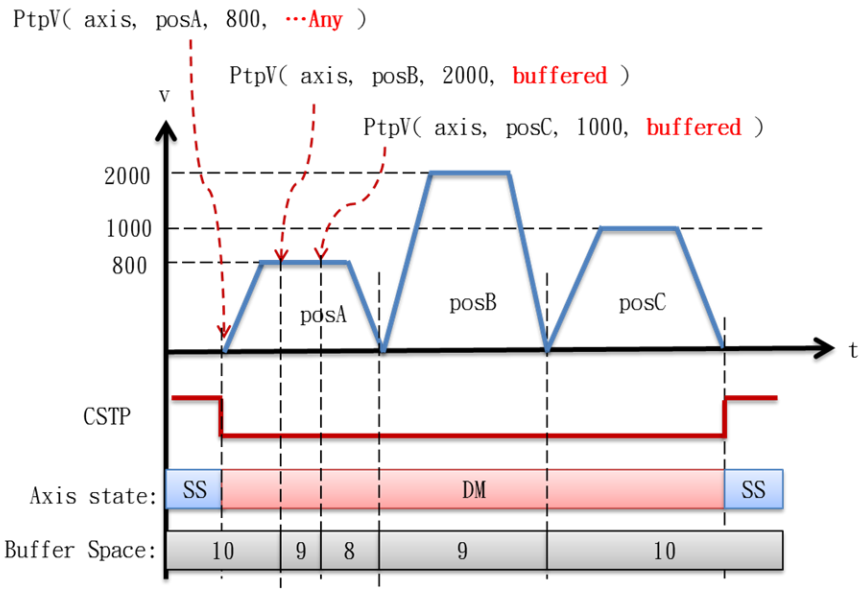
另一个范例如下图所示，原先单轴正在进行 JOG 运动，目标速度设定为 v1，又呼叫了点对点运动，目标速度为 v2，且 0x36 设定值为 0 (Aborting)，当目标位置与目前位置的相对方向与目前运动方向相反时，会产生反转运动，如图中 Case C 所示。当目标位置与目前位置的相对方向与目前运动方向相同时，且相对距离足够进行加减速，则目标速度会改为 v2，并运动到目标位置，如图中 Case A 与 B 所示。



原为 JOG 运动，由另一 PTP 运动中断(Aborting)之三种情况

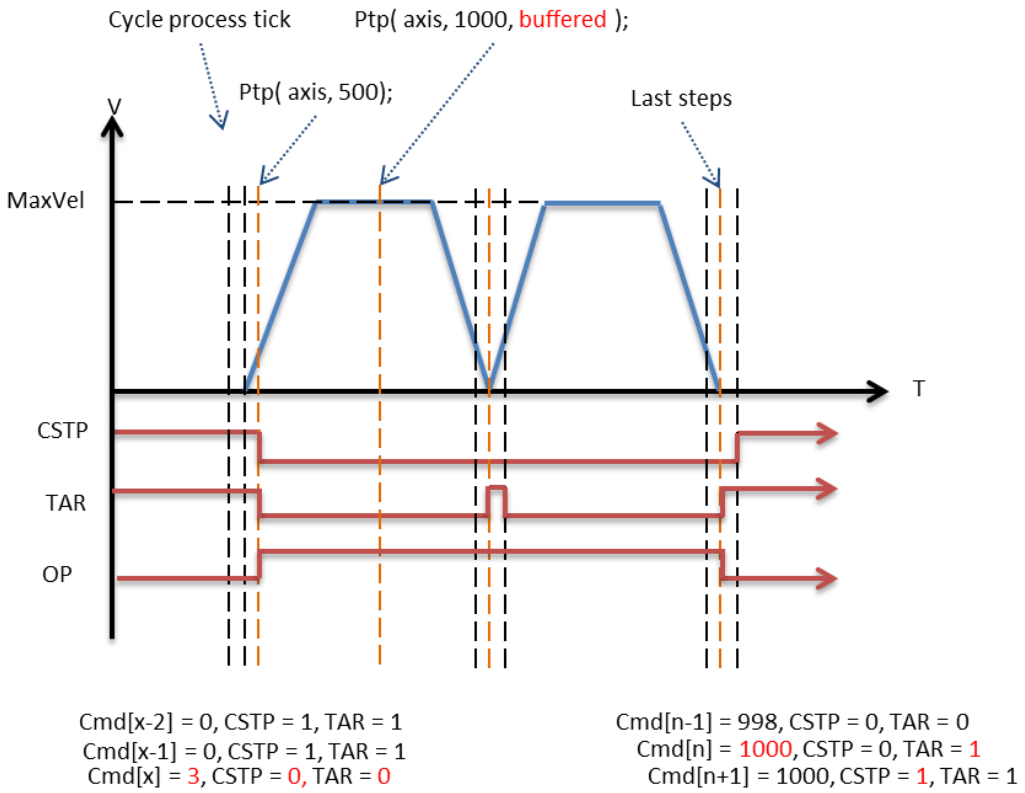
1.3.6.2. Buffer Mode: Buffered (0x36 = 1)

新输入的运动命令将被储存在运动队列中，待前一个输入的运动命令完成后才会被启动。所谓的”运动命令完成”，若运动命令为点对点运动，代表到达目标位置；若运动命令为 JOG 运动，代表达到目标速度。



PTP + PTP(Buffered) + PTP(Buffered)

以上图为例，在进行第一个 PTP 运动过程中，又连续下了两个 PTP 运动且各自有不同的目标速度，由于 0x36 参数设定为 Buffered，这两个 PTP 运动会被储存到单轴运动队列中，等到单轴到达目前运动的目标位置后，储存在队列中的运动命令将会被自动执行。确切的说，所谓到达目标位置，主要以单轴运动信息(Status of axis)中，TAR(bit 8)为 1 作为判断基准，如下图所示。



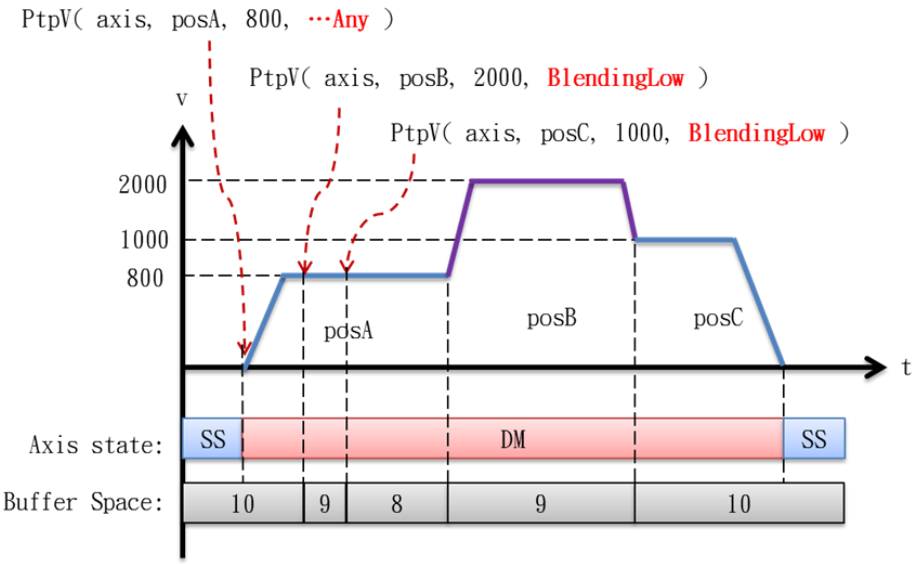
Buffered PTP 之运动状态图

除了前述的 Abort 与 Buffered，有另一种行为称为 Blending，主要是新的运动命令与前一个运动命令会以指定的速度连接起来，有四种指定速度的方式，包含：BlendingLow、BlendingHigh、BlendingPrevious 与 BlendingNext，以下分别说明：

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

新的运动命令将被储存在运动队列中，并与前一个输入的运动命令以较低的目标速度链接起来。以下图来看，一开始从静止状态下启动点对点运动，目标速度为 800，到达目标点位 posA 时，末速度为 0。然而由于在运动过程中，新输入另一个点对点运动，目标速度为 2000，且以 BlendingLow 的方式与前一个点对点运动连接，因此，原先的点对点运动末速度由 0 变为 800(2000 > 800)，而产生出如图 1-3-9 中所示的速度曲线图，从目标速度 800 一直运动到目标点位 posA，末速度仍为 800。到达目标点位 posA 后，运动控制模块会自动启动第二个点对点运动，目标点位为 posB，因此，速度曲线图由初速度 800(等于上一个点对点运动的末速度)加速到目标速度 2000。若接下来没有其他的运动储存在运动队列中，则移动到目标点位 posB 时，末速度为 0。然而由于有另一个点对点运动(目标速度 1000)也以 BlendingLow 的方式与目前的点对点

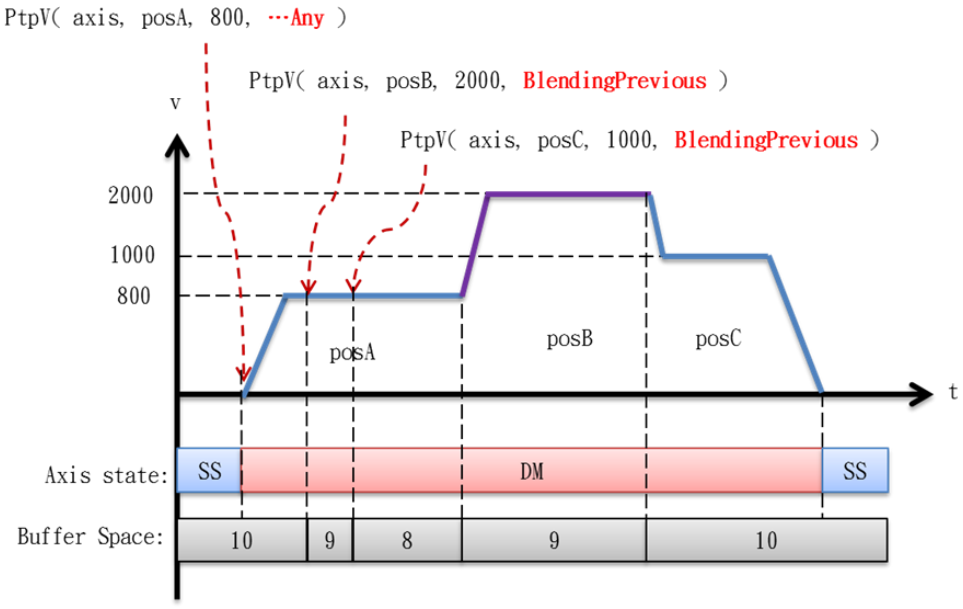
运动相连，因此，到达目标点位 posB 时，末速度为 1000(2000 > 1000)，此时，运动控制模块会再启动第三个点对点运动，将单轴移动到目标点位 posC。



PTP + PTP(BlendingLow) + PTP(BlendingLow)

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

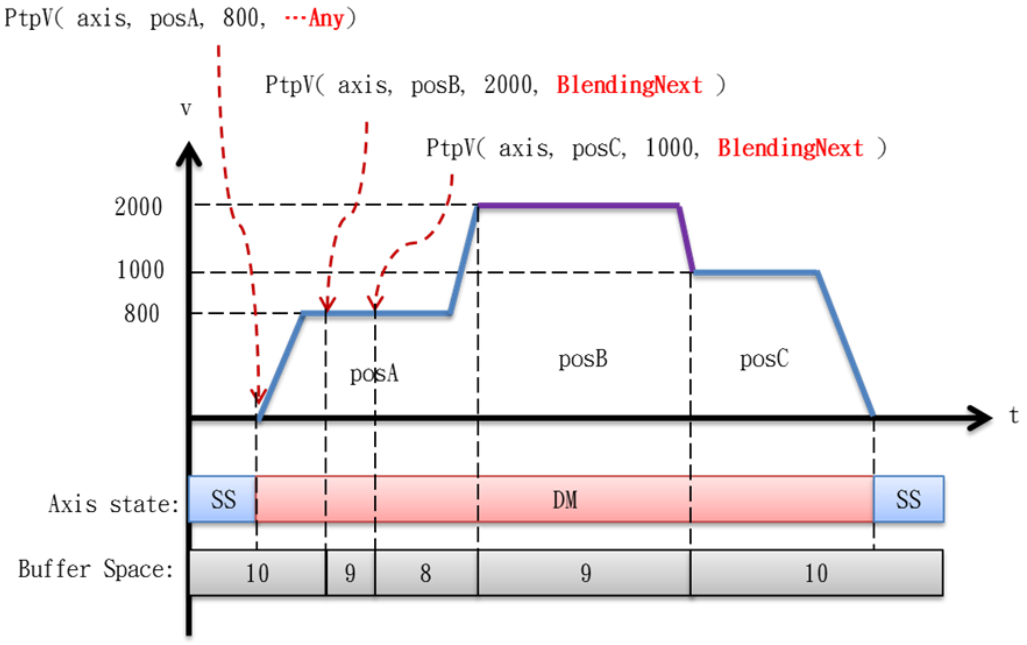
新的运动命令将被储存在运动队列中，并与前一个输入的运动命令以前一个运动命令的目标速度链接起来。以下图来看，新输入的点对点运动与前一个运动的连接，皆以前一个运动的目标速度相连，因此，在移动到目标点位 posA 与 posB 时，点对点运动的末速度皆等于目标速度。



PTP + PTP(BlendingPrevious) + PTP(BlendingPrevious)

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

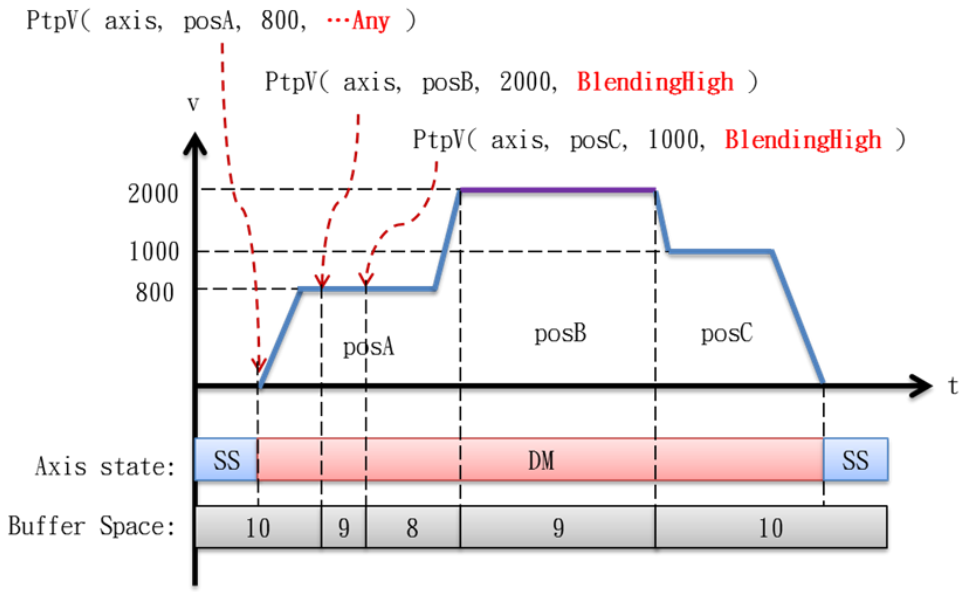
新的运动命令将被储存在运动队列中，并与前一个输入的运动命令以新的运动命令之目标速度链接起来。以下图来看，新输入的点对点运动与前一个运动的连接，皆以新的运动之目标速度相连，因此，在移动到目标点位 posB 与 posC 时，点对点运动的初速度皆已经到达目标速度。



PTP + PTP(BlendingNext) + PTP(BlendingNext)

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

新的运动命令将被储存在运动队列中，并与前一个输入的运动命令以较高的目标速度链接起来。以下图来看，由于第二个点对点运动的目标速度大于第一个点对点运动，因此，到达目标点位 posA 时，末速度会加速到 2000，并以 2000 为初始速度移动到目标点位 posB，由于第三个点对点运动的目标速度也小于 2000，因此，到达目标点位 posB 时，末速度为 2000。



PTP + PTP(BlendingHigh) + PTP(BlendingHigh)

1.3.7. 单轴 JOG 运动

在单轴运动中，JOG 运动也是一个很常使用到的运动形式，使用者可以呼叫 NMC_AxisJog()，运动控制模块将依据相关单轴参数进行速度曲线规划，将单轴加减速到目标速度，并以此目标速度持续运动下去。因此，当单轴在此运动模式下，其状态称为连续运动(NMC_AXIS_STATE_CONTINUOUS_MOTION)。

JOG 运动相关之相关单轴参数:

Param. Num.	Sub. Index	说明	备注
0x28	0	Base velocity (unit/sec)	
0x31	0	Profile type	
0x32	0	Max. velocity (unit/sec)	
0x33	0	Acceleration (unit/sec2)	
0x35	0	Jerk (unit/sec3)	

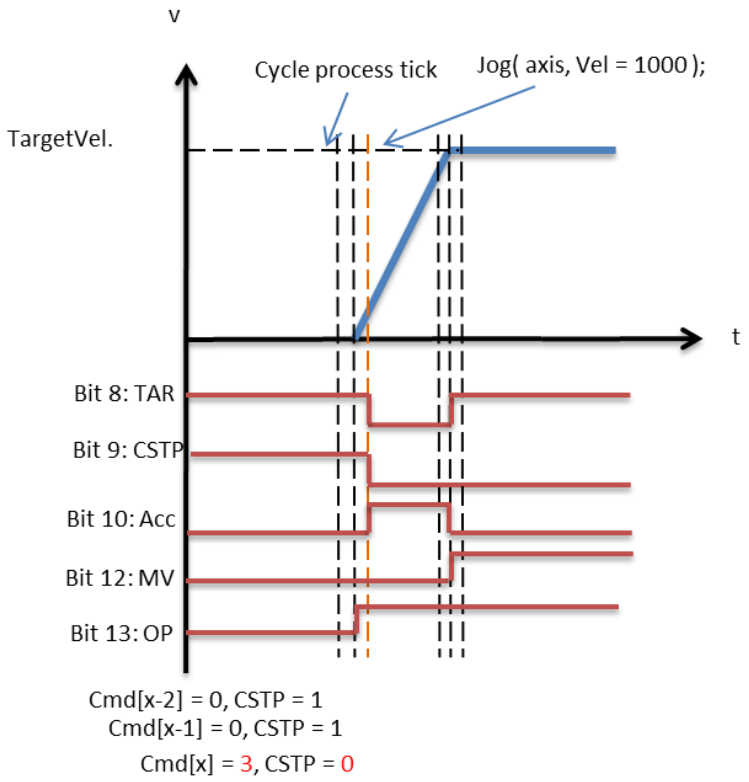
与 JOG 运动相关的单轴参数包括：目标速度(0x32)、加速度(0x33)、速度曲线形式(0x31)、Jerk(0x35)。若使用者想要指定不同于 0x32 设定的目标速度，也可以透过指针 PMaxVel 将所欲的目标速度传入，参数 0x32 的设定值也会连动改变。

成功启动 JOG 运动后，单轴的状态会切换到 NMC_AXIS_STATE_CONTINUOUS_MOTION，有关启动 JOG 运动前与启动后的单轴状态变化，请参阅下列表格：

运动命令 / 目前状态	NMC_AxisJog()

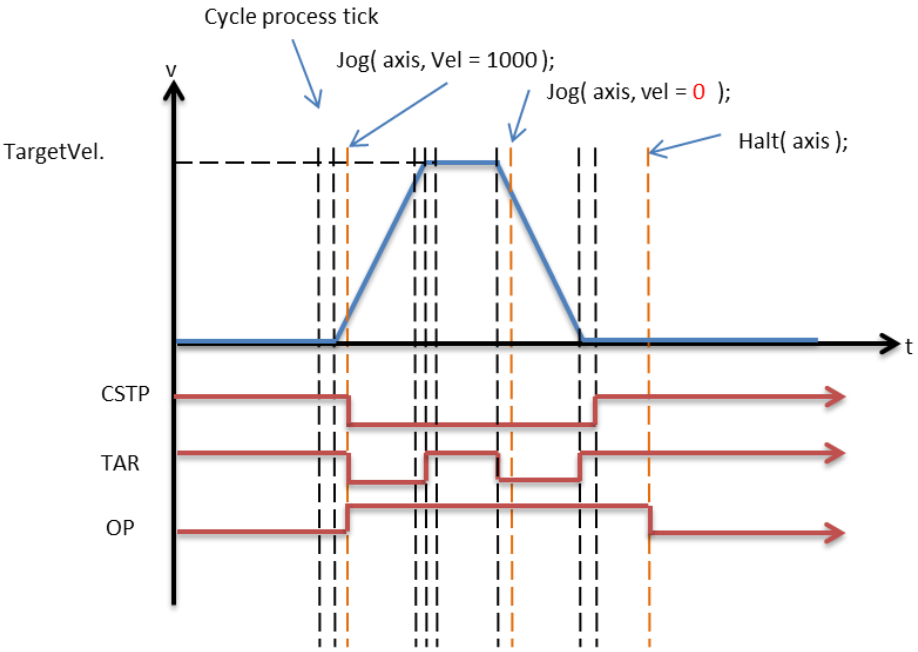
NMC_AXIS_STATE_DISABLE	禁止，回传错误
NMC_AXIS_STATE_STAND_STILL	状态改变至 NMC_AXIS_STATE_CONTINUOUS_MOTION
NMC_AXIS_STATE_HOMING	禁止，回传错误，正在进行的 homing 不会被影响
NMC_AXIS_STATE_DISCRETE_MOTION	依照单轴参数:0x36(Buffer mode)设定储存到运动队列或立即执行。若为 Aborting mdoe 状态改变为 NMC_AXIS_STATE_CONTINUOUS_MOTION
NMC_AXIS_STATE_CONTINUOUS_MOTION	依照单轴参数:0x36(Buffer mode)设定值储存到运动队列或立即执行。当 Buffer mode 为 Blending，禁止，回传错误
NMC_AXIS_STATE_STOPPING	禁止，回传错误
NMC_AXIS_STATE_STOPPED	禁止，回传错误
NMC_AXIS_STATE_WAIT_SYNC	依照单轴参数:0x36(Buffer mode)决定 JOG 命令被储存到运动队列 1. Abort: 清除运动队列后存入 2. Buffered: 存入运动队列 3. Blending: 存入运动队列 若 Buffer mode 为 Blending，而储存在运动队列的上一个运动命令为 JOG，则回传错误。
NMC_AXIS_STATE_ERROR	禁止，回传错误

对于 JOG 运动来说，当单轴到达目标速度，其运动信息(Axis Status)中的 TAR(bit 8)转变为 1，代表目标速度已经到达。此外，由于到达目标速度后，单轴仍然以此速度继续运动，所以运动信息(Axis Status)中的 MV(bit 12)会保持为 1。



JOG 运动信息(Axis Status)时序图

当成功启动 JOG 运动后,若将 JOG 速度重设为 0 (Aborting)或利用 NMC_AxisSetVelRatio() 将速度百分比设定为 0，则单轴将降速到 0，相关的运动状态图，如下图所示:



启动 JOG 运动后将速度百分比设定为 0 之运动状态图

当速度百分比设定为 0，目标速度将重新设定为 0，因此，单轴将由目前的速度减速到 0。当速度降为 0 之后，因为已经到达目标速度，所以运动状态中的 TAR(bit 8) 转为 1，且由于单轴已经静止，因此运动状态中的 CSTP (bit 9)会转为 1。虽然单轴已经降速为 0，但是 JOG 的运动命令仍然作用在单轴上，一旦速度百分比设定为大于 0 的某个数值，单轴将再重新运动起来，因此(OP) bit 13 仍然保持为 1，因为单轴仍在 JOG 的运作中。若使用者呼叫停止运动 NMC_AxisHalt()，单轴将从 JOG 的运作脱离出来，因此，OP(bit 13)会转变为 0。有关停止运动的相关行为，请参阅下面章节。

1.3.8. 运动停止

单轴在运动的过程中，若想终止目前正在进行的运动，可以呼叫 NMC_AxisHalt()或是 NMC_AxisStop()，两者的差别，主要是 NMC_AxisHalt()是在正常状况下呼叫，其降速的减加速度与速度曲线的形式乃依据单轴参数:0x34(Deceleration)与单轴参数:0x31(Profile type)的设定值，这些参数与单轴进行点对点(PTP)运动和 JOG 运动中所使用到的参数是一样的。但若在紧急状况下，欲使单轴停止目前正在进行的运动，则必须呼叫 NMC_AxisStop()，在降速的过程中所依据的减加速度与速度曲线形式为另一组参数，说明如下：

Param. Num.	Sub. Index	说明	备注
0x20	0	Profile type for AxisStop command	
0x21	0	Deceleration for AxisStop command (unit/sec2)	
0x22	0	Jerk for AxisStop command (unit/sec3)	

- 0x20: Profile type for AxisStop command

  呼叫 NMC_AxisStop()后，单轴在降速的过程中所依据的速度曲线形式。

- 0x21: Deceleration for AxisStop command (unit/sec2)

  呼叫 NMC_AxisStop()后，单轴在降速的过程中所使用的减加速度。此参数的设定值，不受限于软件极限保护中的加速度极限值(0x12, SubIndex = 0)。一般来说，若希望单轴在紧急状况下可以快一点降速到 0，0x21 的设定值会大于 0x34 的设定值。

- 0x22: Jerk for AxisStop command (unit/sec3)

  当参数 0x20 设定为 S 形速度曲线时，此参数可以指定 Jerk，与参数 0x35 可以区别开来。

以下，分别说明 NMC_AxisHalt()和 NMC_AxisStop()的使用方式：

- NMC_AxisHalt():

  单轴在进行点对点运动、JOG 运动或是 Homing 运动，皆可呼叫此函式让单轴进行正常停止运动，停止后，单轴状态回复到「NMC_AXIS_STATE_STAND_STILL」。由于此函式视为单轴在正常情况下的运动形式，有关呼叫前与呼叫后的单轴状态变化，如下表所示：

运动命令 / 目前状态	NMC_AxisHalt()
NMC_AXIS_STATE_DISABLE	禁止，回传错误
NMC_AXIS_STATE_STAND_STILL	状态无变化
NMC_AXIS_STATE_HOMING	当运动结束后，状态转变为 STAND_STILL
NMC_AXIS_STATE_DISCRETE_MOTION	依照单轴参数:0x36(Buffer mode)设定储存到运动队列或立即执行，当运动结束后，状态转变为 STAND_STILL
NMC_AXIS_STATE_CONTINUOUS_MOTION	依照单轴参数:0x36(Buffer mode)设定储存到运动队列或立即执行 当运动结束后，状态转变为 STAND_STILL
NMC_AXIS_STATE_STOPPING	禁止，回传错误
NMC_AXIS_STATE_STOPPED	禁止，回传错误
NMC_AXIS_STATE_ERROR	禁止，回传错误

- NMC_AxisStop()

单轴在进行点对点(PTP)运动、JOG 运动或是 Homing 运动，皆可呼叫此函式让单轴进行紧急停止运动，当成功启动此函式后，储存于运动队列中的运动命令会被清除，在停止过程中，单轴状态会切换到「NMC_AXIS_STATE_STOPPING」；待单轴完全停止后，单轴状态会切换到「NMC_AXIS_STATE_STOPPED」。当异常状况排除后，在重新启动单轴运动之前，必须呼叫 NMC_AxisResetState()，成功执行后，若单轴此时处于激磁状态，则单轴状态会回复到「NMC_AXIS_STATE_STAND_STILL」；若单轴处于非激磁状态，则单轴状态会切换到「NMC_AXIS_STATE_DISABLE」。有关呼叫 NMC_AxisStop()前与呼叫后的单轴状态变化，如下表所示：

运动命令 / 目前状态	NMC_AxisStop()
NMC_AXIS_STATE_DISABLE	状态无变化
NMC_AXIS_STATE_STAND_STILL	当运动结束后，状态转变为 NMC_AXIS_STATE_STOPPED
NMC_AXIS_STATE_HOMING	当运动结束后，状态转变为 NMC_AXIS_STATE_STOPPED
NMC_AXIS_STATE_DISCRETE_MOTION	当运动结束后，状态转变为 NMC_AXIS_STATE_STOPPED
NMC_AXIS_STATE_CONTINUOUS_MOTION	当运动结束后，状态转变为 NMC_AXIS_STATE_STOPPED
NMC_AXIS_STATE_STOPPING	状态无变化
NMC_AXIS_STATE_STOPPED	状态无变化
NMC_AXIS_STATE_ERROR	状态无变化

上述两个函式仅适用于单一个单轴的停止运动，因此，必须输入单轴的索引号码。若用户欲将整个系统中所有的单轴停止下来，可以呼叫相对应的 NMC_AxisHaltAll()或是 NMC_AxisStopAll()。

1.3.9. 运行中速度改变

当单轴在进行运动的过程中，可以呼叫相关函式改变原先设定的目标速度与加减速度，以下说明相关函式的使用方式与限制。

- NMC_AxisVelOverride()

只有当单轴正在进行点对点(PTP)运动或是 JOG 运动时，才可以呼叫此函式改变目标速度，否则将会回传错误码。此函式输入的参数为目标速度的绝对量值，并不受到速度百分比的影响，且不会改变单轴参数:0x32 (Max. velocity)的设定值。举例来说，假设单轴正在进行 JOG 运动，成功呼叫此函式后，不管此时的速度百分比为何，目标速度的量值都将更改为输入此函式的参数(TargetVel)，但单轴参数:0x32 (Max. velocity) 仍维持原先的设定值。由于 JOG 运动的主要目标乃是让单轴以输入的目标速度持续运动，因此，成功呼叫 NMC_AxisVelOverride()后，单轴必定会运动到指定的目标速度。但若单轴进行的是点对点(PTP)运动，且已经进入到降速阶段-单轴运动信息 (Status of axis)的 DEC(bit 11)为 1，则呼叫此函式对于目前正在进行的点对点运动不会有任何影响；若单轴是在加速或等速度区间，则会依据目前速度、指定的目标速度以及残余距离，判定是否可以到达用户指定的目标速度，若无法到达用户指定的目标速度，则会依据所需的加减速距离，合理修改目标速度。

- NMC_AxisAccOverride()

只有当单轴正在进行点对点(PTP)运动或是 JOG 运动时，才可以呼叫此函式改变目标加速度，否则将会回传错误码。此函式输入的参数为目标加速度的绝对量值，且会连动改变单轴参数:0x33 (Acceleration)的设定值。就单轴运动状态来看，只有当单轴处于加速到目标速度的状态下(Axis status 的 ACC-bit 10 变为 1)，呼叫此函式才会因为改变加速度而对目前的运动有所作用；若单轴已经到达目标速度(Axis status 的 MV-bit 12 变为 1)或是进入到降速阶段(Axis status 的 DEC-bit 11 变为 1)，则呼叫此函式对于目前的运动不会有任何作用。

- NMC_AxisDecOverride()

只有当单轴正在进行点对点运动或是停止运动(呼叫 NMC_AxisHalt)时，才可以呼叫此函式改变减加速度(Dec)，否则，将会回传错误码。此函式输入的参数为目标减加速度(Dec)的绝对量值，且会连动改变单轴参数:0x34 的设定值。就单轴运动形式来看，当单轴正在进行点对点运动，且运动状态处在加速到目标速度或是已经到达目标速度，呼叫此函式对于目前运动可能会有作用，主要是因为加减速距离改变而连带产生的作用，譬如：原先到达不了指定的目标速度，但因为呼叫此函式，新设定的 Dec 使得需要的加速速距离变短，因而可以到达指定的目标速度。但是，若单轴运动状态已经进入降速到末速度(Axis status 的 DEC-bit 11 变为 1)，则呼叫此函式对于目前的运动将不会有影响。当单轴因为呼叫 NMC_AxisHalt()正在进行停止运动，呼叫此函式将会改变目前的 Dec，因而影响停止所需的时间。

1.3.10. 单轴归零运动

一般来说，设定单轴归零的方式有两种应用情境，第一种应用情境为，将单轴用任何方式移动至原点位置(或参考位置)，然后在目前的位置直接设定坐标值。另一种为在单轴开始运作前，使用单轴的归原点程序，选择合适的归零模式回到原点位置，以此位置作为坐标基准点，再进行后续的运动；以下针对两种归零方式分别说明：

1.3.10.1. 手动设定原点位置

透过NMC_AxisSetHomePos()设定原点位置，只有在单轴状态为「NMC_AXIS_STATE_DISABLE」或是「NMC_AXIS_STATE_STAND_STILL」时，使用者才可以透过呼叫此函式，将单轴目前的位置指定为某坐标值，否则将回传错误码。成功呼叫此函式后，单轴的实际位置(Actual position)与命令位置(Command position)都会变更为指定的坐标值，且运动控制模块会自动计算伺服马达位置偏移量(position offset)并自动储存于单轴参数:0x08 (Position offset)之中。对于使用绝对式编码器的单轴(单轴参数:0x05 设定为 Absolute)来说，储存于单轴参数:0x08 中的 position offset 可以在系统开机后，当读取到编码器回传的数值时，透过此 position offset 计算出单轴此时的位置坐标。

1.3.10.2. 单轴归零运动

使用 NMC_AxisHomeDrive()启动驱动器所提供之归零运动(Homing)功能，该 API 只有当单轴状态为「NMC_AXIS_STATE_STAND_STILL」时，才可以呼叫此函式，启动单轴进行回原点的程序，否则将回传错误码。

驱动器归原点相关的单轴参数如下:

Param. Num.	Sub. Index	说明	备注
0x80	0	Number of home parameters	(*3)(*4)
	1	EtherCAT CiA HOME method	(*5)
	2	EtherCAT CiA HOME speed search switch	(*5)
	3	EtherCAT CiA HOME speed search zero	(*5)
	4	EtherCAT CiA HOME acceleration	(*5)
	5	EtherCAT CiA HOME offset	(*5)

(*3): 可设定范围根据控制器的规格而不同

(*4): Read only

(*5): 可设定范围根据受控装置的规格而不同

由于回原点的运动程序，主要由驱动器提供的功能规格为主，因此，用户在呼叫此函式进行归零运动时，必须先详阅驱动器使用手册，确认需要输入的参数，以及提供那些回原点模式(设定于上述单轴参数:0x80:1~5 中)。

若归零运动(Homing)使用到原点讯号或正负极限讯号，必须先确认传感器(Home sensor)、正负极限(Limit sensor)是否已正确连接到驱动器的 I/O 接点，可透过读取單軸運動資訊:RPEL(bit16)、RNEL(bit17)和 RHOM(bit18)讯号之变化确定传感器已经正确连接与设定。

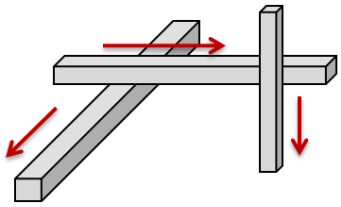
在正常情况下，当成功呼叫 NMC_AxisHomeDrive()函式后，控制器会将上述归原点相关参数传送给驱动器并自动进行用户设定的回原点程序，在运动的过程中，单轴状态将切换到「NMC_AXIS_STATE_HOMING」。一旦归零程序成功完成后，单轴状态将回复到「NMC_AXIS_STATE_STAND_STILL」，且 Axis status 中的 RHOM (bit 18)将转变为 1。在进行归零运动的过程中，若有发生不预期的错误，则单轴将停止归零运动，且单轴状态会切换到「NMC_AXIS_STATE_ERROR」，Axis status 的 ERR(bit 7) 会转变为 1。举例来说，若用户输入的归零模式，驱动器本身并不提供，则呼叫此函式并成功执行一段时间后，单轴状态会切换到「NMC_AXIS_STATE_ERROR」。另外，若归零运动的参数设定不当，例如：加速度设定过小，以至于当触发到外部极限传感器(Limit Sensor)后，因为减速距离过大，以至于在停止的过程中又脱离外部极限传感器(Limit Sensor)触发的位置范围，造成错误的状况。当单轴在归零过程中发生错误的状况，单轴状态进入「NMC_AXIS_STATE_ERROR」，使用者必须呼叫 NMC_AxisResetState()将单轴的状态回复到「NMC_AXIS_STATE_STAND_STILL」后，才可以再执行其他的运动命令。

归零运动视为单轴运动的一种，因此，在运动的过程中，可以呼叫 NMC_AxisHalt()、NMC_AxisDisable()或是 NMC_AxisStop()终止归零程序，呼叫后的单轴行为，与一般单轴在运动状况下呼叫相关函式的行为一样。

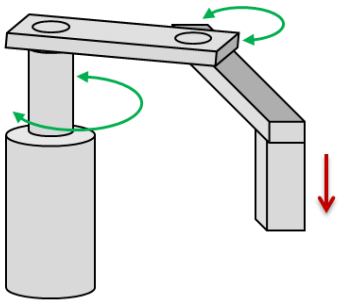
1.4. 群组(Group)控制

NexMotion 可定义若干轴为一个群组(Group)，而一群组代表一个有特定结构关系的机构或机器人(Robot)，目前已支持下列常见的工业型机器人:

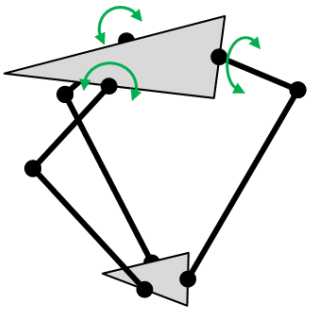
1. 直角坐标机构(Robot)
2. SCARA Robot
3. Delta Robot
4. 关节型(Arteculated) Robot



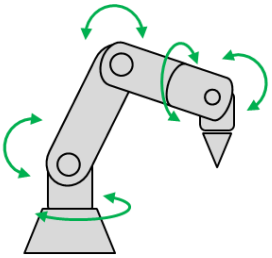
Linear Robot



SCARA Robot



Delta Robot



Articulated Robot

1.4.1. 设定群组

可透过 NexMotion Studio 汇入机构之设定，其步骤如下:

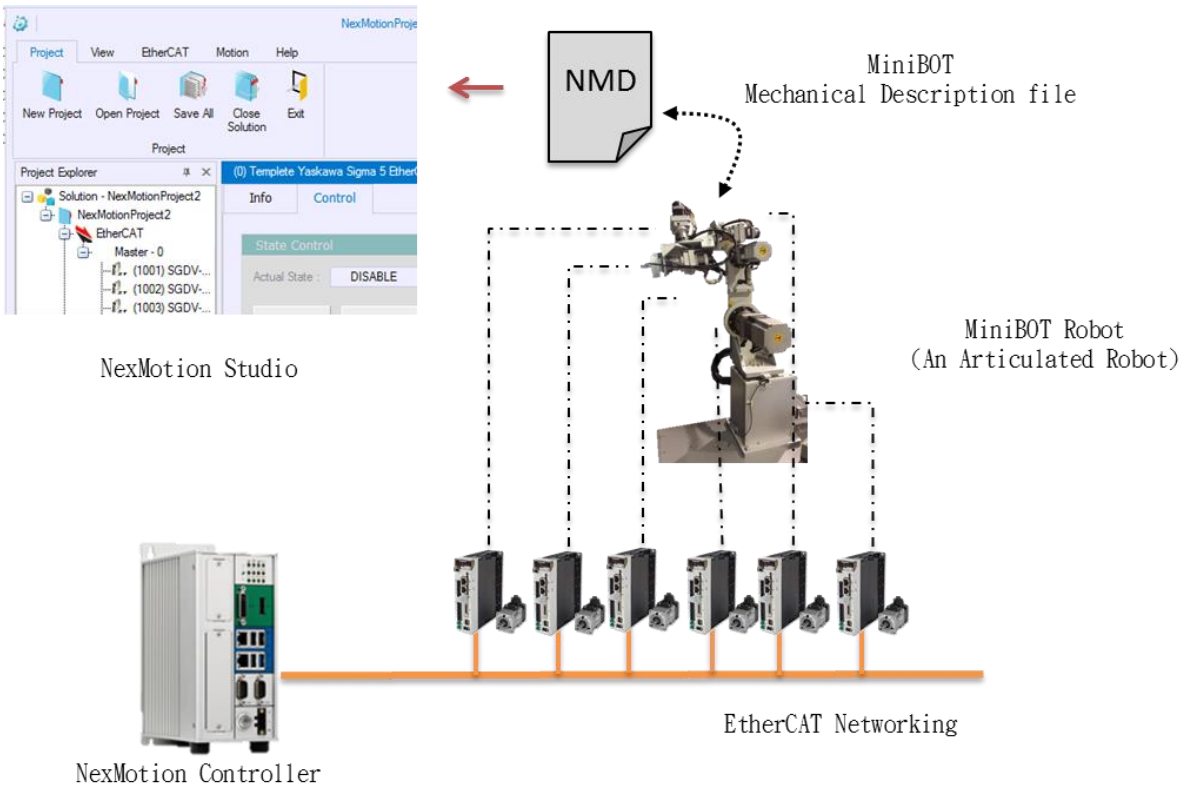
1. 透过 NexMotion Studio 汇入一个机构描述(Mechnical Description, NMD)档案
2. 映像伺服马达到群组各轴
3. 测试群组，同时产生受控系统配置档(NCF 档案)

详细步骤可参考 NexMotion Studio 使用手册。

也可以手动设定一个群组，可先利用 NexMotion Studio 先汇入一个类似的机构当作样板(Templete)，从现有样板再去修改调整成符合实际的机构型式，其步骤如下:

1. 透过 NexMotion Studio 汇入一个机构描述(Mechnical Description, NMD)档案，作为基础
2. 修改各轴单位参数(参考单轴单位设定)

3. 修改运动学参数(参考运动学设定)
4. 映像伺服马达到群组各轴
5. 测试群组，同时产生受控系统配置档(NCF 档案)



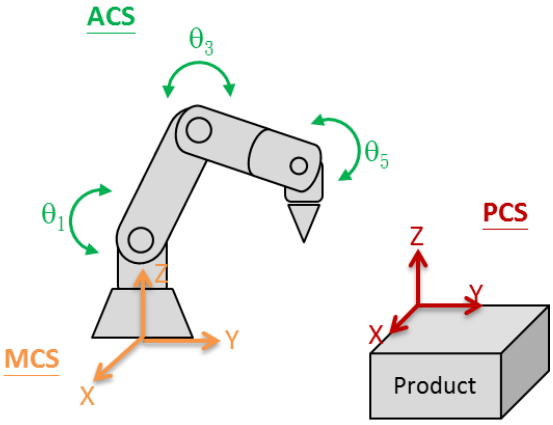
透过 NMD 档案汇入一机构(Robot)数据

1.4.2. 坐标系说明

吾欲对群组(或称 Robot)的各轴或其末端点(Tool center point,TCP)位置进行控制，NexMotion 提供了三种坐标系来描述 Robot 的位置，其中有

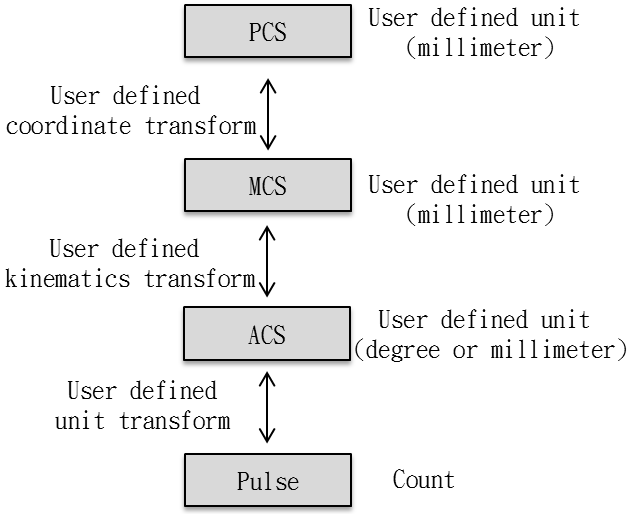
1. 轴坐标系(Axis Coordinate System, ACS)
2. 机械坐标系(Machine Coordinate system, MCS)或俗称大地坐标系
3. 工件坐标系(Product Coordinate System, PCS)

下图为各坐标系之示意图;



ACS、MCS 和 PCS 坐标系之定义

NexMotion 已经内建各坐标系间之单位转换，用户可透过参数设定来定义之转换关系，下图为各坐标系之单位转换关系图:



单位转换关系图

转换关系相关参数

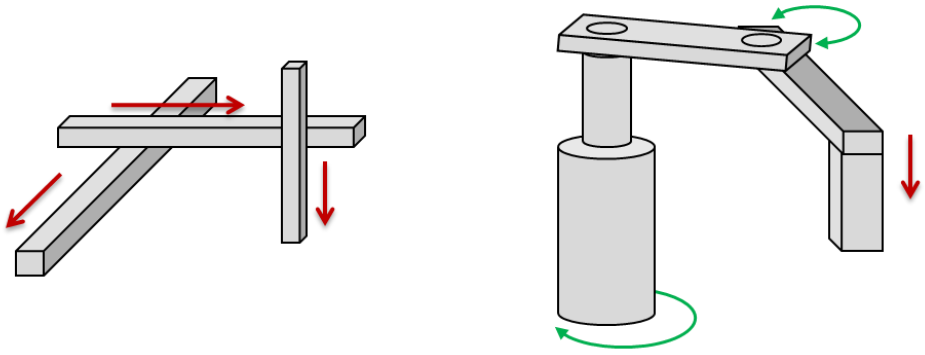
转换类别	参数类别	说明
单位转换	各群组轴参数 0x00~0x06	设定伺服的脉波数和 ACS 坐标系物理单位之转换
运动学转换	群组参数 0x00	设定 ACS 坐标系和 MCS 坐标系之转换
坐标系转换	群组参数 0xC0~DF	设定 MCS 和 PCS 坐标系之转换
TCP 坐标转换	群组参数	设定 Tool 之 TCP 坐标转换

	0x80~8F	

在程序方面，NexMotion 定义了「Pos_T 结构」和「Xyz_T 结构」来描述坐标数据。Pos_T 结构可用来描述 ACS, MCS, 或 PCS 坐标数据，而 Xyz_T 结构专门用来描述 MCS 和 PCS 的 x, y, z 轴位置。

1.4.2.1. ACS 坐标系(Axis Coordinate System)

ACS 坐标系为描述群组机构(或称 Robot)中各关节轴的位置，一 Robot 系统可能具有各种不同的活动关节，我们可将各关节视为一个轴(Axis)，并透过轴坐标系(ACS)来描述该轴的位置。较常见的关节可概分为直线关节(Linear joint)和旋转关节(Rotary joint)两种，使用者可以根据其机构形式来决定位置的单位定义，举例来说，若为直线关节可定义其物理单位为毫米(mm)，若为旋转关节可定义为角度(Degree)单位。

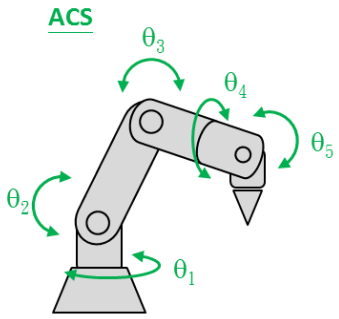


Robot 的关节型式

控制器提供从伺服的脉波单位(Pulse count)到机构之物理单位之转换，其设定方式请参考 单轴单位设定

一个群组最多可以有 8 个轴，其实际轴数依照机构的型态决定

下图为一关节型六轴机械手臂之 ACS 坐标系描述方式



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

范例:关节型机械手臂之

ACS 坐标参数表示法

下面是一个使用「Pos_T 结构」来表示轴坐标(ACS)的范例:

```
#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

}
```

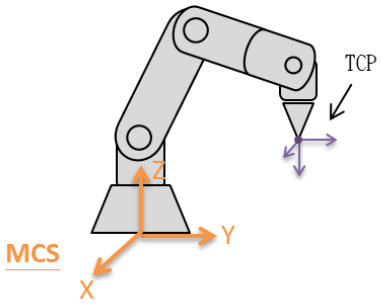
要读取群组之 ACS 坐标位置使用「Pos_T 结构」可用下列 API 达成

NMC_GroupGetCommandPosAcs() => 读取 ACS 命令位置

NMC_GroupGetActualPosAcs() => 读取ACS实际位置

1.4.2.2. MCS 坐标系(Machine Coordinate system)

MCS 坐标是用来表示为工具末端点(Tool Center Point, TCP)相对于机械原点的位置姿态，其坐标原点通常定义在机台安装点 ， 下图为一关节型六轴机械手臂之 ACS 坐标系描述方式



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

范例:关节型机械手臂之
MCS 坐标参数表示法

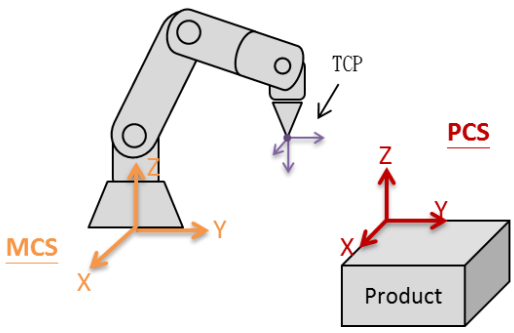
描述末端点最高为六个自由度，其表示法：X,Y,Z,A,B,C，

- 位置：X,Y,Z 为直角坐标系表示法，遵循右手定则(Right-hand rule)
- 姿态：A,B,C 为尤拉角 intrinsic Z-Y-X 表示法。分别依序对 Z，Y 和 X 轴旋转。

MCS 坐标系通常使用在归原点的阶段，归完原点后在应用时，习惯上直接使用 PCS 坐标系来表示群组在直角坐标系下的位置。

1.4.2.3. PCS 坐标系(Product Coordinate System):

PCS 坐标系是用来表示为工具末端点(Tool Center Point, TCP)相对于工件坐标原点的位置姿态，其坐标原点通常定义在待加工之对象上 ，下图为一关节型六轴机械手臂之 PCS 坐标系描述方式:



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

范例:关节型机械手臂之
PCS 坐标参数表示法

如同 MCS 坐标系，PCS 坐标系描述末端点最高为六个自由度，其表示法：X,Y,Z,A,B,C

- 位置：X,Y,Z 为直角坐标系表示法，遵循右手定则(Right-hand rule)
- 姿态：A,B,C 为尤拉角 intrinsic Z-Y-X 表示法。分别依序对 X，Y 和 X 轴旋转。

PCS 坐标系与 MCS 坐标系之差别在于坐标转换之关系，因此当转换关系为初始值 PCS 坐标系和 MCS 坐标系为重迭，换言之，当我们不特别设定工件坐标系时，PCS 坐标系就等于 MCS 坐标系。

下面是一个使用「Pos_T 结构」来表示机械坐标(PCS)的范例:

```
#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

}
```

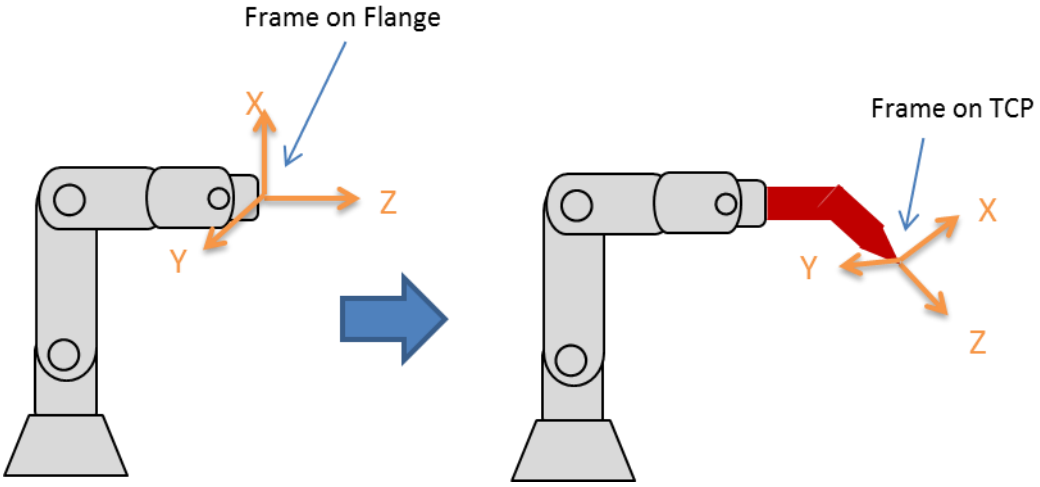
要读取群组之 PCS 坐标位置使用「Pos_T 结构」可用下列 API 达成

NMC_GroupGetCommandPosPcs() => 读取PCS命令位置

NMC_GroupGetActualPosPcs()=> 读取PCS实际位置

1.4.2.4. Tool 设定

当用户尚未设定 Tool 时，运动指令中的目标位置(Target position)代表机器人末端点法兰面(Flange)的位置与姿态，如下图。当 Tool 参数被设定使用后，TCP(Tool center point)坐标系根据转换关系附着于 Tool 之末端点。

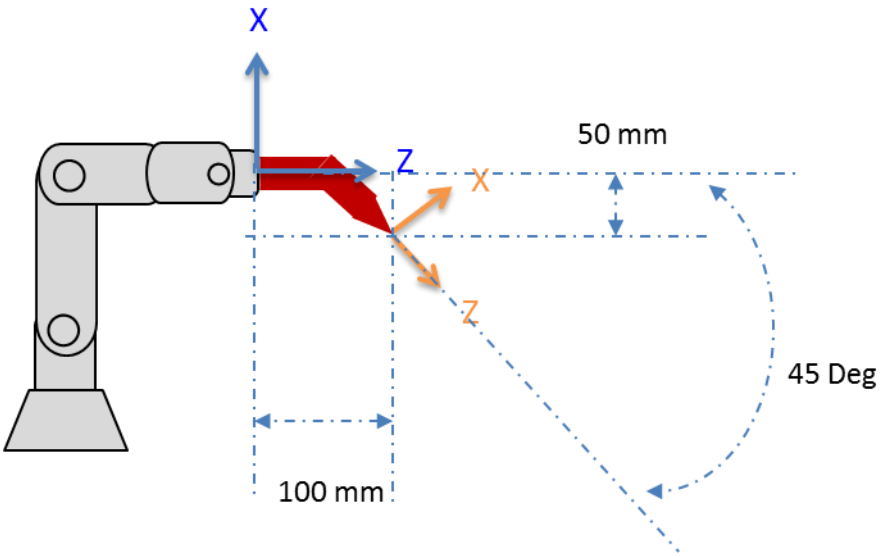


目前系统最多可支持 16 组 Tool 设定，其设定方式为设定群组参数 0x80~0x8F。定义如下:

Param. Num.	Sub. Index	资料型态	说明
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

设定范例:

吾欲设定一组工具 Tool (Index = 0)如下图，其 TCP 坐标原点相对于 Flange 坐标系 x 轴为-50 单位，y 轴为 0 单位，Z 轴为 100 单位，另外坐标系相对于 Flange 之 y 轴旋转-45 度:



其群组设定参数如下:

设定值如下:

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

使用方式为当运动指令的目标位置(Target postion)若该运动指令没有特别指定 Tool index，则系统会参考群组参数 0x40:0 之设定当作目前系统所使用的 Tool

Param. Num.	Sub. Index	资料型态	说明
0x40	0	I32_T	Tool index selection for motion target

1.4.2.5. Tool 教导

设定 Tool 的方式除了直接输入参数外，另外也提供数种使用姿态点位方式换算 Tool 的参数 APIs:

- TCP 平移教导法
- TCP 平移与 Z 方向设定教导法
- TCP 平移与姿态设定教导法
- 姿态设定教导法

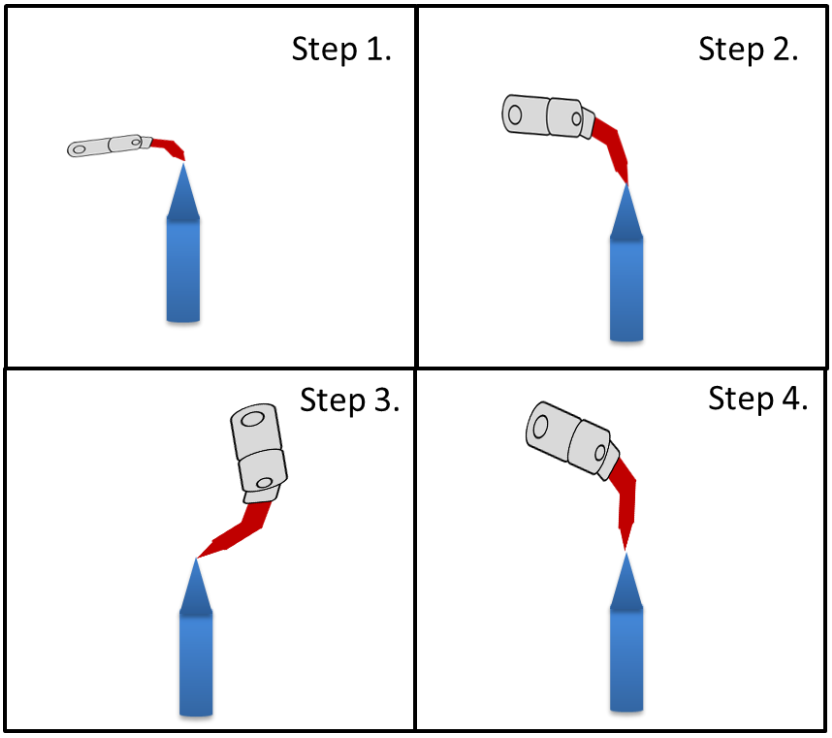
所对应使用的 API 如下:

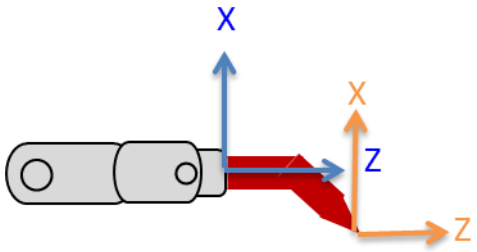
函数名称	说明
NMC_ToolCalib_4p	Tool 教导- TCP 平移教导法
NMC_ToolCalib_4pWithZ	Tool 教导- TCP 平移与 Z 方向设定教导法
NMC_ToolCalib_4pWithOri	Tool 教导- TCP 平移与姿态设定教导法
NMC_ToolCalib_Ori	Tool 教导- TCP 姿态设定教导法

注意,设定 Tool 过程中,可以选择在 MCS 坐标系中进行教导,或者在任何一个 Base 中进行教导。切勿教导过程中变更 Base 设定以避免教导错误。

- TCP 平移教导法

透过 TCP 位置固定，给定四个不同姿态找出 TCP 和手臂法兰面(Flange)之间的关系，此方法所找出的 TCP 只有平移关系， ABC 角无变化(为 0)

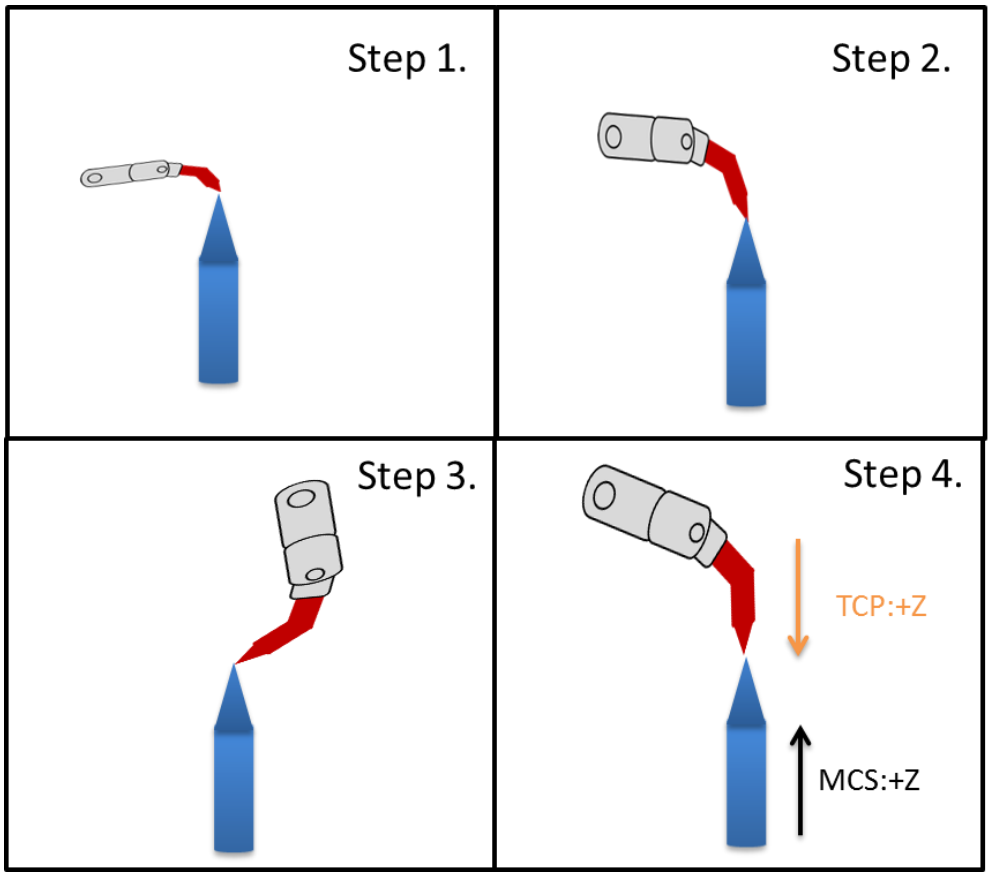


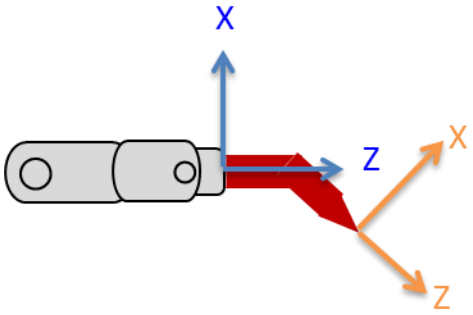


TCP 坐标为 Flange 坐标平移

- TCP 平移与 Z 方向设定教导法

透过 TCP 位置固定，给定四个不同姿态找出 TCP 和手臂法兰面(Flange)之间的关系，第四步骤 TCP 的 Z 方向必须指向 MCS 的负 Z 方向。此方法所找出的 TCP 有平移关系和对 Flange-Y 轴旋转

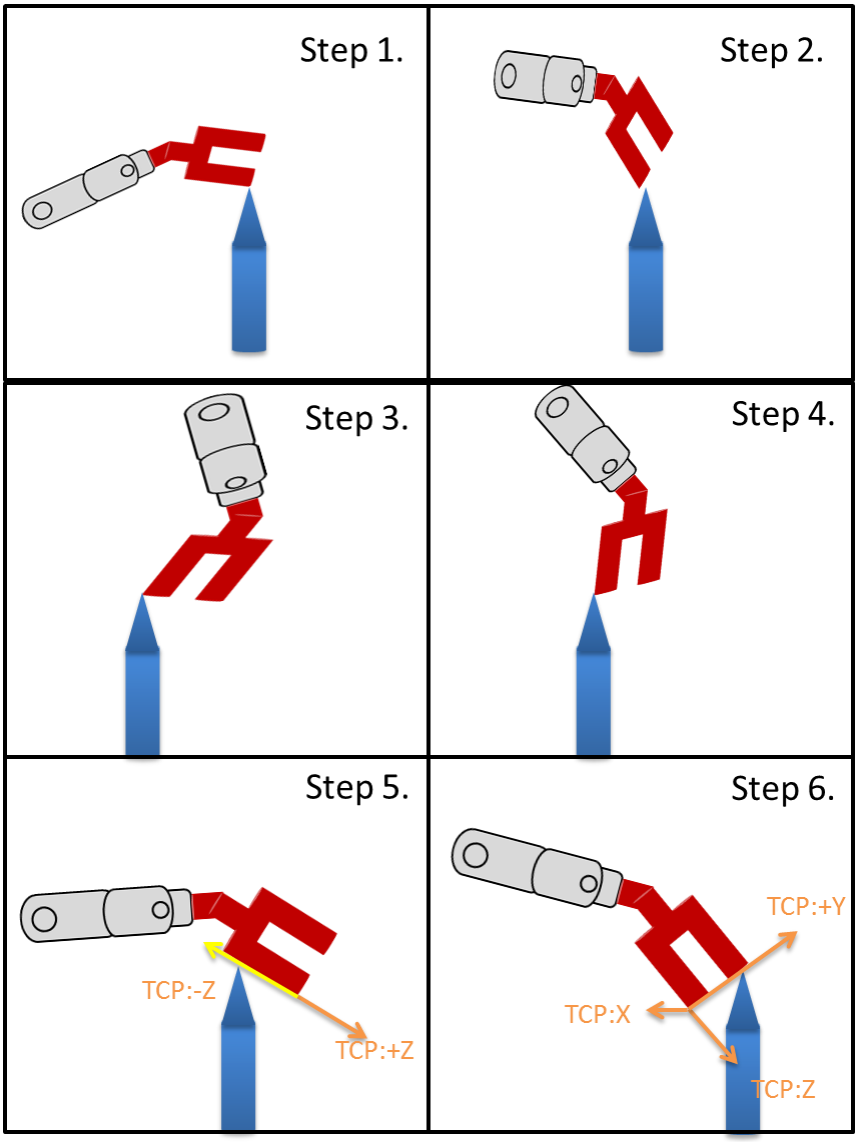


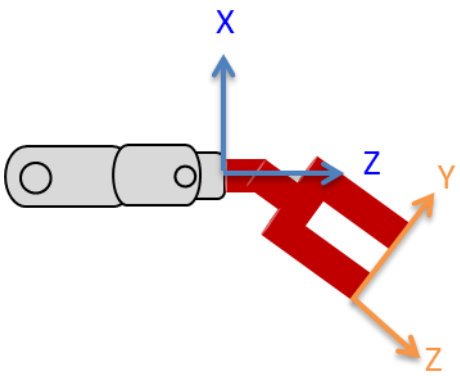


TCP 坐标为 Flange 坐标平移与对 Flange-y 轴旋转之方向

- TCP 平移与姿态设定教导法

  此方法之前面四个步骤与「TCP 平移教导法」相同，主要是找出 TCP 与 Flange 坐标之平移关系，步骤五与步骤六则是找出 TCP 坐标的姿态。

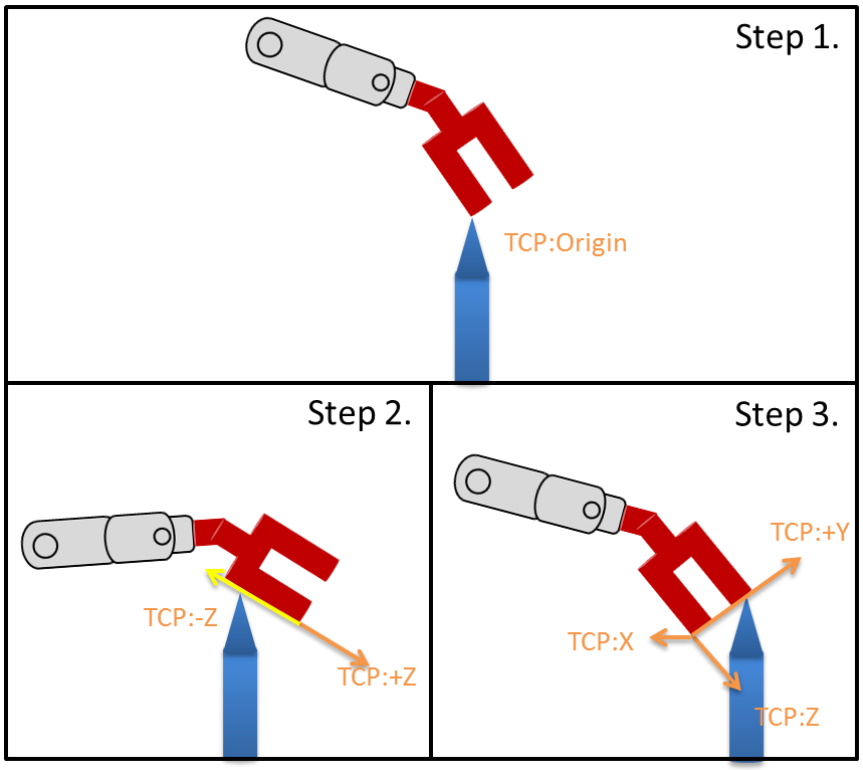




TCP 坐标与 Flange 坐标之关系

- 姿态设定教导法

这个方法是用来教导 TCP 姿态，可以配合「TCP 平移教导法」先找出平移关系，再使用本方法找出姿态设定。或者只想重新校正姿态时可使用本方法。一般不会独立使用。

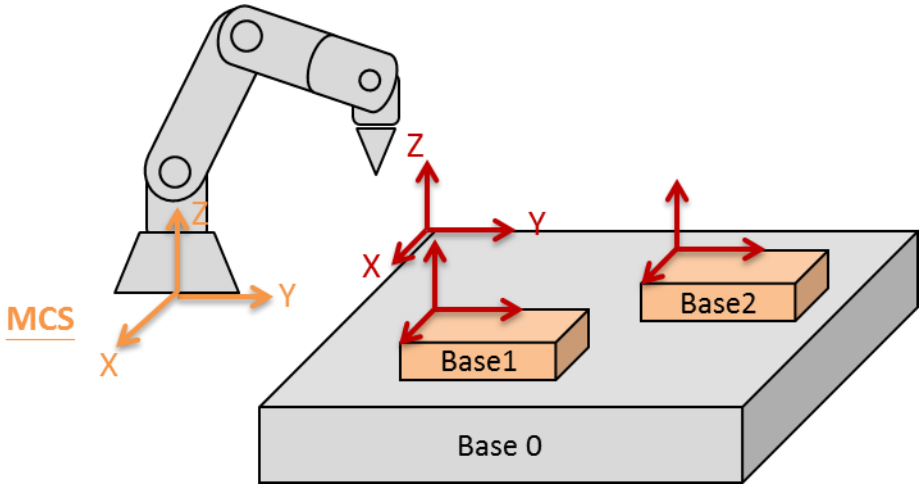


1.4.2.6. Base 设定

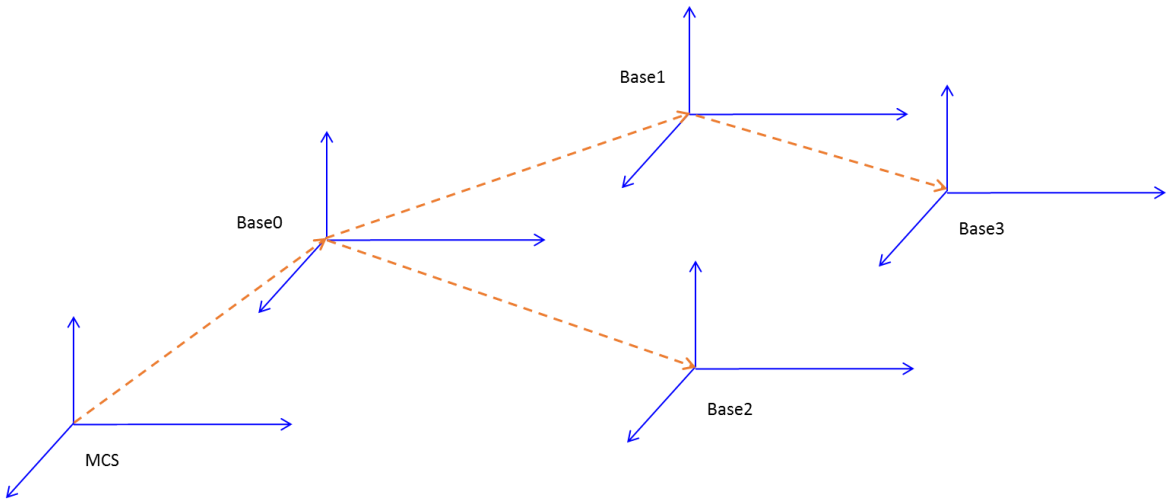
设定 Base 就是定义 PCS 坐标系，其最大好处在于，所有目标点位相对于参考坐标系(Base)，当参考坐标系变动时，只需从新设定参考坐标系，目标点位不需要从新教导即可利用。

NexMotin 之 PCS 坐标转换特色如下:

- 最高支持 32 组 Base
- 具阶层设定，最高 3 阶



Base 的设定具有阶层性，如上图,Base0 相对于 MCS，而 Base1 和 Base2 则相对于 Base0。当 Base0 有改变，Base1 和 Base2 也会跟着改变。

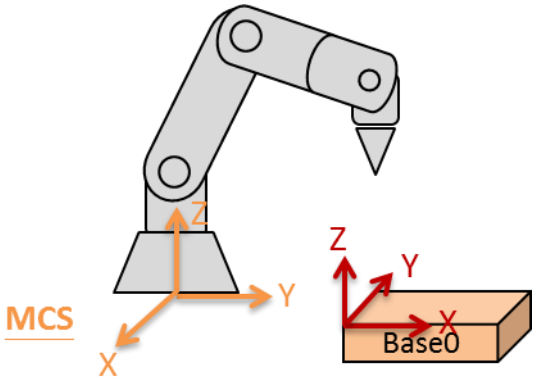


其设定方式为设定群组参数 0xC0~0xDF 共 32 组。定义如下:

Param. Num.	Sub. Index	资料型态	说明
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

设定范例:

吾欲设定 Base（Index = 0）之坐标原点相对于 MCS 坐标系 x 轴为 10 单位,y 轴为 15 单位，Z 轴为 5 单位，另外 Base0 坐标系相对于 MCS 之 Z 轴旋转 90 度



xyz@MCS =(10, 15, 5)

设定值如下:

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

使用方式为当运动指令的目标位置(Target postion)若该运动指令没有特别指定 Base index，则系统会参考群组参数 0x48:0 之设定当作目前系统所使用的 Base

Param. Num.	Sub. Index	资料型态	说明
0x48	0	I32_T	Base index selection for motion target

1.4.2.7. Base 教导

设定 Base 的方式除了直接输入外，另外提供教点位自动更正的方式来设定 Base，自动更正的方式包括:

1. 一点(1P)教导法
2. 两点(2P)教导法
3. 三点(3P)教导法

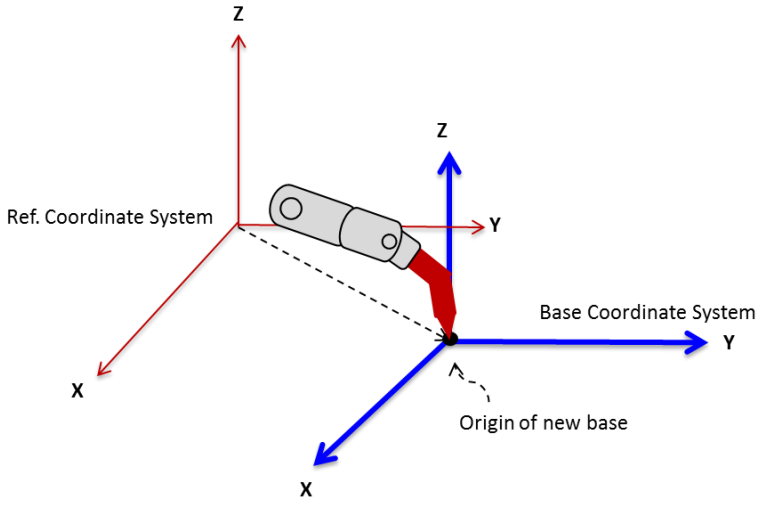
所对应使用的 API 如下:

函数名称	说明
NMC_BaseCalib_1p	Base 教导-1p 法
NMC_BaseCalib_2p	Base 教导-2p 法
NMC_BaseCalib_3p	Base 教导-3p 法

顾名思义，一点法只需要教导一点，两点法只需要交两点，三点法则需要教三点。，其中，两点法跟一点法的差别在于，两点法所教导的 base 多了 Z 轴的旋转，而三点法则可任意定义 X-Y-Z 轴。

- 「一点教导法」之步骤:

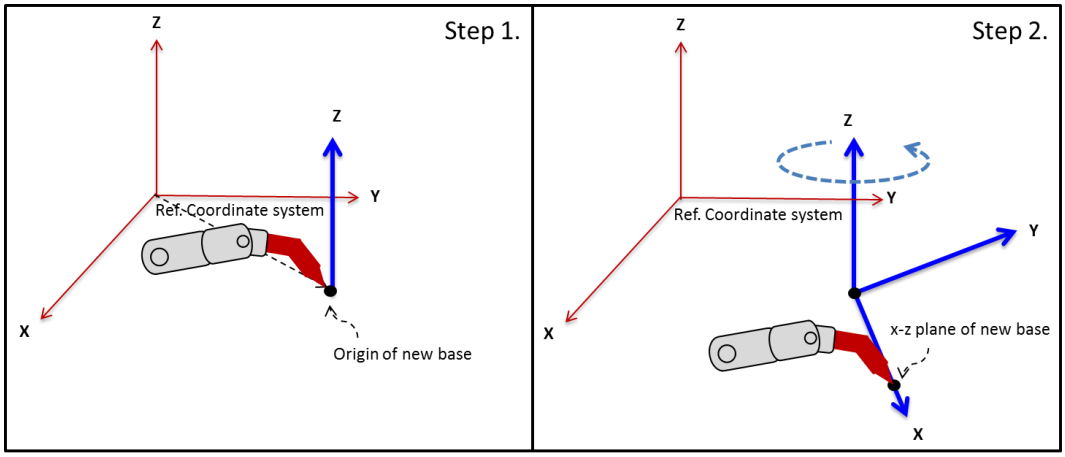
Step1: 定义 Base 坐标原点



此方法为相当于参考坐标系平移到新的坐标点上。

- 「两点教导法」之步骤:

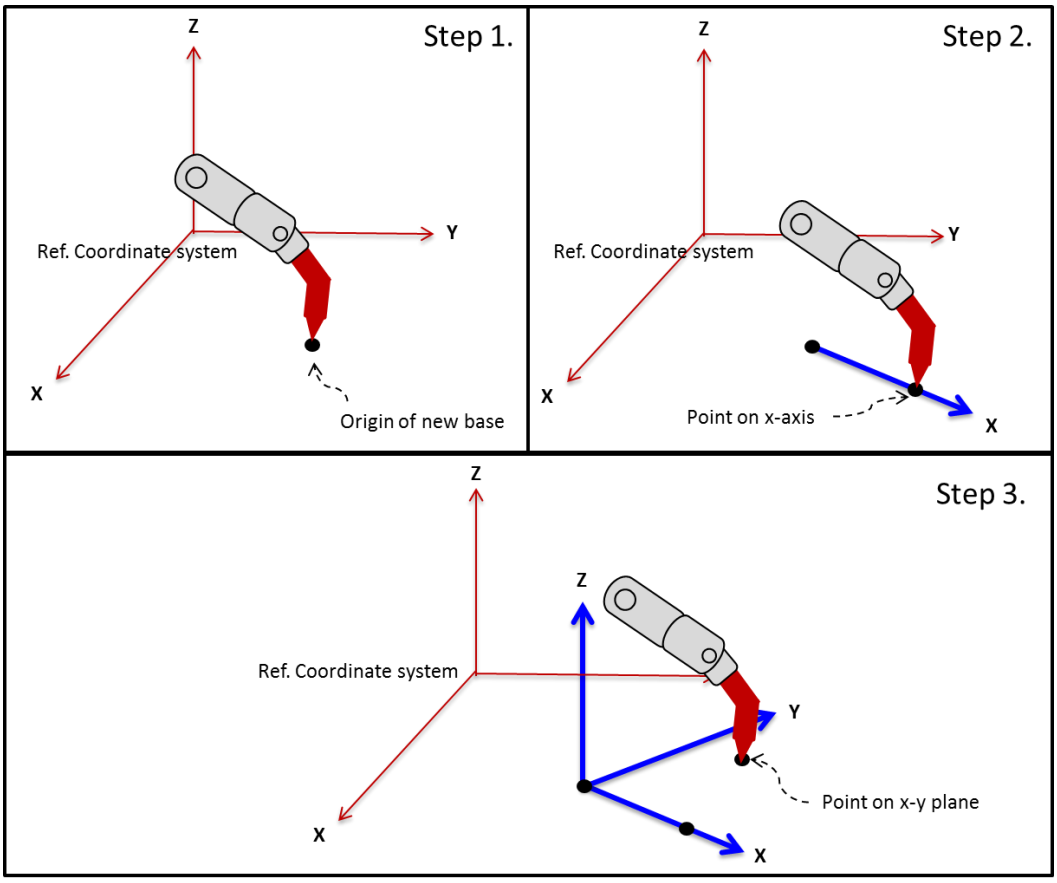
Step1: 定义 Base 坐标原点
Step2: 定义 X-Z 平面上一点，X 轴为正方向



此方法为相当于参考坐标系平移到新的坐标点上后再对 Z 轴做旋转。

- 「三点教导法」之步骤:

Step1: 定义 Base 坐标原点
Step2: 定义 X 轴上一点，X 轴为正方向
Step3: 定义 X-Y 平面上一点，Y 轴为正方向



1.4.3. 机构运动学设定

NexMotion 目前支持下列四种类型工业机器人

1. 线性型机器人(2~8 轴)
2. 六轴关节型(Arteculated) 机器人
3. Delta 型机器人
4. SCARA 型机器人

其设定方式于群组参数 0x00:0~48 中设定，其中定义如下:

参数 0x00:0 为构型选择

参数 0x00:1~24 为机构尺寸参数

参数意义如下表:

构型 / Sub	线性型	六轴关节型	Delta 型	SCARA 型
0	0	1	2	3
1	Axis number (value=2~8 integer)	0 (*1)	f(mm)	a1 (mm)
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

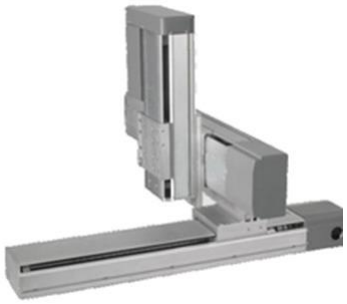
(*1) 保留，必须设定为0

- PEL: 正机构极限 (Positive End Limit)
- MEL: 负机构极限 (Minus End Limit)

1.4.3.1. 线性型机器人(2~8 轴)

线性型机器人为马达机构配置为线性轴，可定义轴数为 2~8 轴。线性机器人其特性为 ACS, MCS 和 PCS 坐标系为重迭。

最常见的为 X-Y Table (2 轴)或 X-Y-Z 直角坐标机器人(3 轴)如下图，直角坐标中的每一个方向都可以配置一颗马达，其马达运动则直接对应至卡式坐标中的方向运动。



X-Y-Z Linear Axis

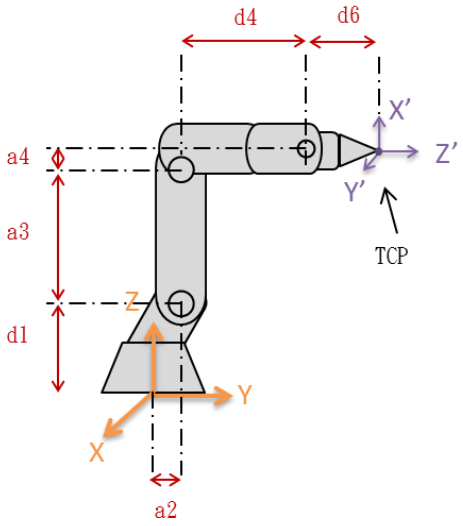
运动学参数(群组参数 0x00)定义如下表:

构型 / Sub	线性型 参数定义
0	0
1	轴数目 (value=2~8 integer)

1.4.3.2. 六轴关节型(Arteculated) 机器人

六轴关节型(Arteculated) 机器人其马达机构配置与运动学参数如下图，此系统可提供三维空间中的六个自由度。

运动学参数(群组参数 0x00)定义如下表:

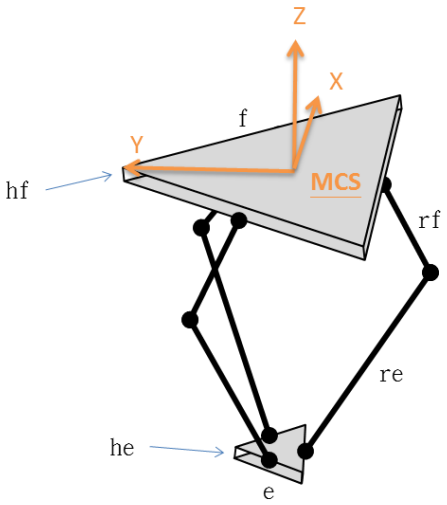
0x00 Sub. NO.	参数意义	图示
1	0 (*1)	
2	a2 (mm)	
3	a3 (mm)	
4	a4 (mm)	
5	0 (*1)	
6	0 (*1)	
7	d1 (mm)	
8	0 (*1)	
9	0 (*1)	
10	d4 (mm)	
11	0 (*1)	
12	d6 (mm)	
13	PEL-J1 (Deg)	Joint 1 正极限
14	PEL-J2 (Deg)	Joint 2 正极限
15	PEL-J3 (Deg)	Joint 3 正极限
16	PEL-J4 (Deg)	Joint 4 正极限
17	PEL-J5 (Deg)	Joint 5 正极限
18	PEL-J6 (Deg)	Joint 6 正极限
19	MEL-J1 (Deg)	Joint 1 负极限
20	MEL-J2 (Deg)	Joint 2 负极限
21	MEL-J3 (Deg)	Joint 3 负极限
22	MEL-J4 (Deg)	Joint 4 负极限
23	MEL-J5 (Deg)	Joint 5 负极限
24	MEL-J6 (Deg)	Joint 6 负极限

(*1) 保留，必须设定为0

1.4.3.3. Delta 型机器人

Delta Robot 为并联式机械手臂，其马达机构配置与运动学参数如下图，此系统可提供三维空间中的 XYZ 平移和 1 个 Z 轴方向旋转。

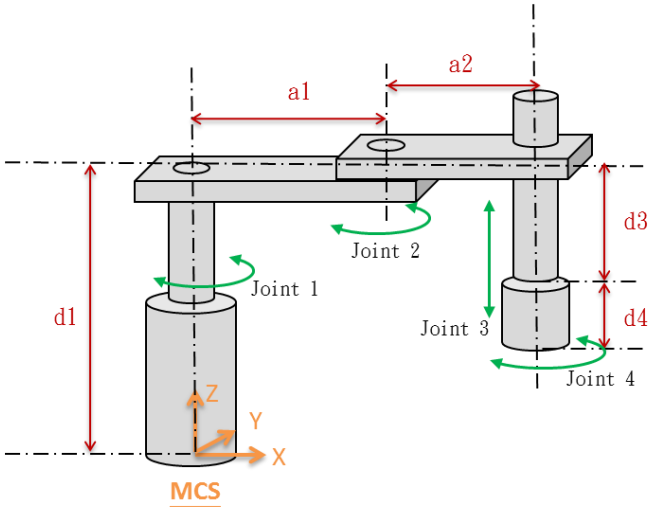
运动学参数(群组参数 0x00)定义如下表:

0x00 Sub. NO.	参数意义	图示
1	f 上三角形边长	 哪一个是第一轴，哪一个是第二轴?...
2	e 下三角形边长	
3	rf 上杆长	
4	re 下杆长	
5	hf 上三角形厚度	
6	he 下三角形厚度	
7	PEL-J1 (Deg)	Joint 1 正极限
8	PEL-J2 (Deg)	Joint 2 正极限
9	PEL-J3 (Deg)	Joint 3 正极限
10	PEL-J4 (Deg)	Joint 4 正极限
11	MEL-J1 (Deg)	Joint 1 负极限
12	MEL-J2 (Deg)	Joint 2 负极限
13	MEL-J3 (Deg)	Joint 3 负极限
14	MEL-J4 (Deg)	Joint 4 负极限

1.4.3.4. SCARA 型机器人

Scara Robot 为水平关节机器，其马达机构配置与运动学参数如下图，此系统可提供三维空间中的 XYZ 平移和 1 个 Z 轴方向旋转。

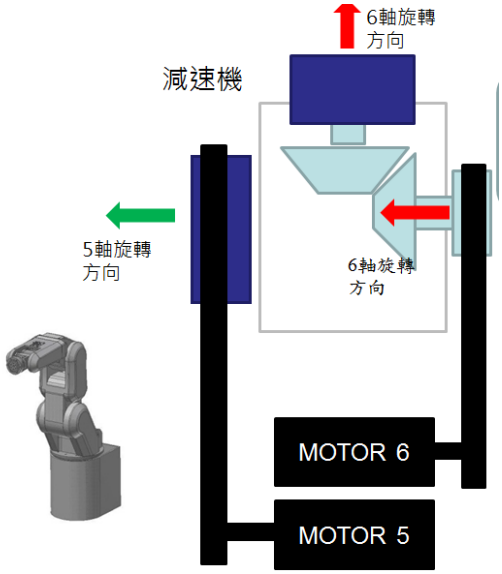
运动学参数(群组参数 0x00)定义如下表:

0x00 Sub. NO.	参数意义	图示
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 正极限
10	PEL-J2 (Deg)	Joint 2 正极限
11	PEL-J3 (mm)	Joint 3 正极限
12	PEL-J4 (Deg)	Joint 4 正极限
13	MEL-J1 (Deg)	Joint 1 负极限
14	MEL-J2 (Deg)	Joint 2 负极限
15	MEL-J3 (mm)	Joint 3 负极限
16	MEL-J4 (Deg)	Joint 4 负极限

(*1) 保留，必须设定为0

1.4.4. 结构耦合补偿功能

在群组的机构中，常见到轴于轴之间由于机构设计而产生耦合的情况，例如:六轴关节型机械手臂中第五与第六轴之间，由于传动机构的连动，造成第五轴马达在转动时，第六轴外部关节结构也会随之产生相对应的转动，如下图所示：



六轴关节行手臂第 5、6 轴结构耦合示意图

（图表文字修改成英文）

另一个范例为四轴 SCARA 机构，若其垂直轴是利用滚珠螺杆将马达转动变为垂直运动，则在末端的机构也会产生相对应的转动。为了补偿因为结构耦合产生的连动效应，使用者可以透过群组参数:0x01，将耦合的信息设定到运动控制核心模块，以计算所需的运动补偿命令，相关参数说明如下：

相关群组参数:0x01 说明如下:

Sub index	说明
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

  指定结构耦合关系中的主动轴，以上述六轴关节型手臂的例子来看，第 5 轴为主动轴。

- 0x01, SubIndex = 1:Mechanical couple slave axis

  指定结构耦合关系中的从动轴，以上述六轴关节型手臂的例子来看，第 6 轴为从动轴。

- 0x01, SubIndex = 2:Compensation source type

  用户可以指定从动轴的位置补偿命令是依据主动轴的 actual position 或是 command position。

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

  用户可以指定主动轴在甚么位置作为从动轴位置命令补偿的计算基准。

- 0x01, SubIndex = 4:Coupling ratio numerator(分子)
- 0x01, SubIndex = 5:Coupling ratio denominator(分母)

  上述两个参数必须搭配设定，主要利用分子分母的形式来设定从动轴位置命令补偿量计算所需之 Coupling ratio。分子，不可以为 0，负号代表反向补偿；分母，必须大于 0。以上图六轴关节型手臂来说，当第五轴(主动轴)机构端往正方向转动 90 度时，会造成第 6 轴(从动轴)马达端往负方向转动 90 度，假设第 6 轴马达减速比为 50，则第 6 轴机构端将转动-1.8 度。为了补偿因结构耦合造成的转动,则第 6 轴机构端的位置命令需要加上 1.8 度(从动轴位置命令补偿量)，对应到 Coupling ratio，则分母(SubIndex)设定为 50，分子(SubIndex 4)设定为 1，相关公式如下所示：

从动轴机构端位置命令补偿量 =(主动轴机构端位置 - 主动轴机构端位置基准(SubIndex 2)) * (SubIndex 4) / (SubIndex 5)

- 0x01, SubIndex = 6:Enable mechanical couple compensation

  启动结构耦合补偿之功能，开启后，控制器将依据相关结构耦合信息自动计算位置补偿命令到从动轴。

上述的参数当系统进入 OPERATION 的状态后，将无法更改。

1.4.5. 群组启动与状态

欲使某群组(Group)中所有轴向进行伺服启动(Servo enable)，则可使用NMC_GroupEnable()达成。因各厂牌驱动器之启动(Servo enable)时间不一，因此可撰写等待时间待驱动器状态切换至「NMC_AXIS_STATE_DISABLE」状态。若需要查看各群组状态(State of group)，可使用NMC_GroupGetState()来查看。

当所有轴向皆已启动，则此群组的状态会切换为「NMC_GROUP_STATE_STAND_STILL」；若群组中某一轴以上的状态为 DISABLE，则群组的总体状态会切换为「NMC_GROUP_STATE_DISABLE」；

若群组中有一轴以上出现错误，则群组的状态会切换为「NMC_GROUP_STATE_ERROR」。若欲使某群组中所有轴向进行伺服关闭(Servo disable)，则可使用 NMC_GroupDisable()达成。

```
#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...
    }
    // Controller is start up successfully.

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

    // To make sure state is transer to ENABLE
    Sleep( 3000 );  //3 secounds 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 enable successfully.
    // Do something...
    // ...


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

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

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

若控制器使用多个轴或群组，欲使所有轴或群组所有轴向同时伺服启动(Servo enable)，则可使用 NMC_DeviceEnableAll()。若需要查看群组的数量，可使用 NMC_DeviceGetGroupCount()来得知此控制器使用多少个群组。若欲使所有群组中的所有轴向一起 DISABLE，则可使用 NMC_DeviceDisableAll()即可关闭系统中所有轴向。

```
#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...
    }

    // Controller is start up successfully.

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

    // To make sure the state is transer to ENABLE
    Sleep( 3000 );  //3 secounds 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 transer to DISABLE
    Sleep( 1000 );  //1 second is a try value. Depends on servo drives.

    NMC_DeviceShutdown( devID );

    return 0;
}
```

除群组状态(State of group)外，若需查看更多群组的运动信息（Status of group），可使用NMC_GroupGetStatus()达成。目前提供14种状态信息供参考，包含外部急停讯号、警报、正向硬件极限、负向硬件极限、正向软件极限、负向软件极限、ENABLE、错误状态、命令停止、加速状态、减速状态、等速状态、运动中状态与停止状态。

有两个时机点需要使用NMC_GroupResetState()来清除群组状态:

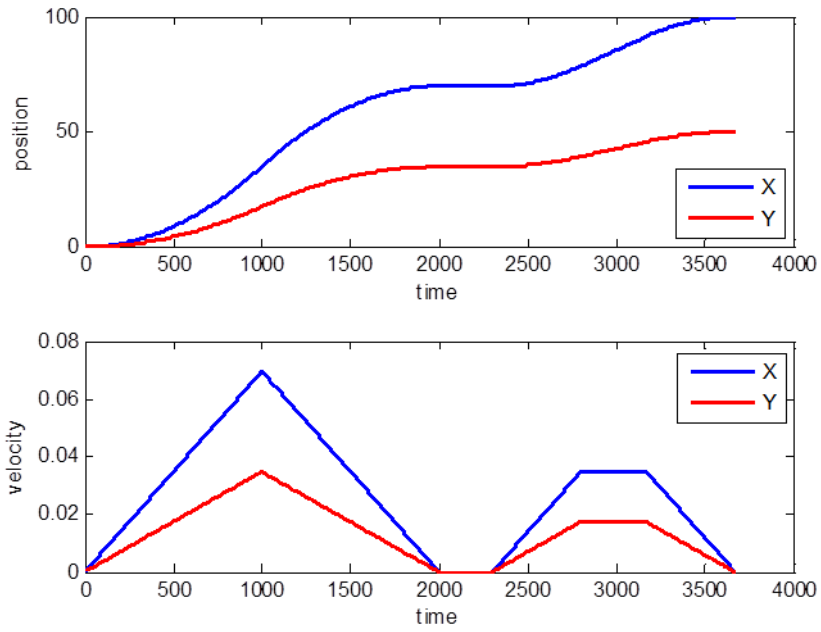
1. 于任何情况(或紧急情况)下当使用者呼叫NMC_GroupStop()使群组停止运动，则群组状态(STATE)的变化为STOPPED。当群组在STOPPED状态时，控制器将拒绝执行任何新的运动命令。当使用者确认状况排除后可让控制器接受新的运动命令时使用NMC_GroupResetState()来使群组状态切换为STAND STILL。
2. 若群组有任一轴驱动器发生警报，若此警报可由控制器清除，则亦可使用此API将警报清除，此时群组状态(STATE)的变化为ERROR STOP→DISABLE。

若群组有任一轴驱动器有警报，且此警报有开放权限给控制器来清除，则可使用NMC_GroupResetDriveAlm()或NMC_GroupResetDriveAlmAll()将此轴向的警报清除。

1.4.6. 群组速度百分比设定

每个群组有设置一速度百分比设定，可透过NMC_GroupSetVelRatio()进行设定，该群组的任何运动最大速度皆会参考此速度百分比的数值，例如，若最大速度为100；速度百分比为70%，则运动的最大速度将以 100 * 70% = 70的上限进行速度规划。如下图为一群组具有X和Y两轴执行

一点对点运动(PTP)，于时间为1000时设定速度百分比为0，设定后则开始减速为零；于时间为2300时设定速度百分比为50，设定后则开始加速，直至到达目标位置。



(摆放上图之范例程序)

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

int main()
{

}
```

若需得知目前群组的速度百分比的数值，则可使用NMC_GroupGetVelRatio()达成。

1.4.7. 群组点对点运动

与群组轴坐标之点对点运动(PTP)相关的API如下：

- 轴坐标系：

  NMC_GroupPtpAcs()

  NMC_GroupPtpAcsAll()

- 卡式坐标系：

  NMC_GroupPtpCart()

NMC_GroupPtpCartAll()

上述API之差别在于可控制的轴数与目标位置(Target position)所使用的坐标系不同，但其运动行为是一致的，换言之无论使用那一种API，当执行点对点(PTP)运动时，控制器会以各轴(轴坐标系)之最短路径进行运动规划，并自动以所需运动时间最长的轴为基准降低其他轴之速度使之所有运动轴向会同一时间到达目标位置。各轴之运动速度会以各群组轴之轴参数进行运算，与PTP相关之速度参数如下:

Param. Num.	Sub. Index	资料型态	说明
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)

设定或读取群组轴参数相关之API如下:

NMC_GroupAxSetParamI32()

NMC_GroupAxGetParamI32()

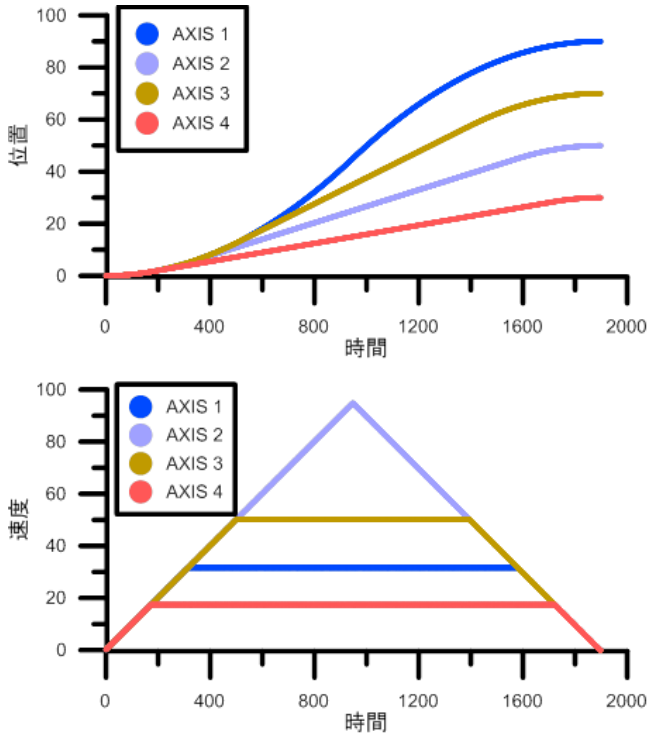
NMC_GroupAxSetParamF64()

NMC_GroupAxGetParamF64()

使用轴坐标系为单为，若仅需要移动群组中某一轴向到指定的轴坐标位置，则可使用NMC_GroupPtpAcs()达成。若需要移动一个以上的轴向到指定的轴坐标位置，则可使用NMC_GroupPtpAcsAll()达成。需指定欲移动轴向的屏蔽(Mask)与轴坐标位置，

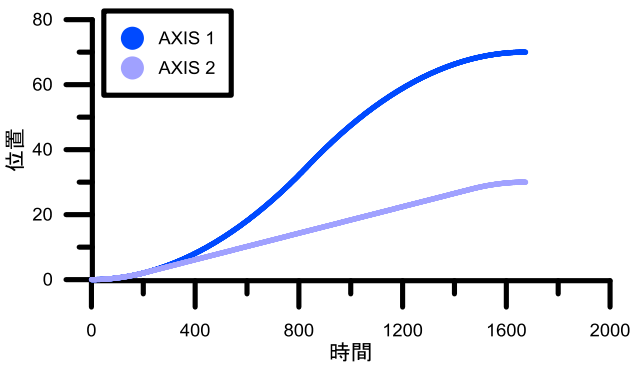
若使用空间坐标系为单位，需要延着移动某个卡氏空间坐标轴方向到指定的位置(于MCS或PCS空间中操作)，则可使用NMC_GroupPtpCart()达成。若需要移动一个以上的卡式坐标轴，则可使用NMC_GroupPtpCartAll()来达成，需指定卡氏空间坐标轴的屏蔽与位置。

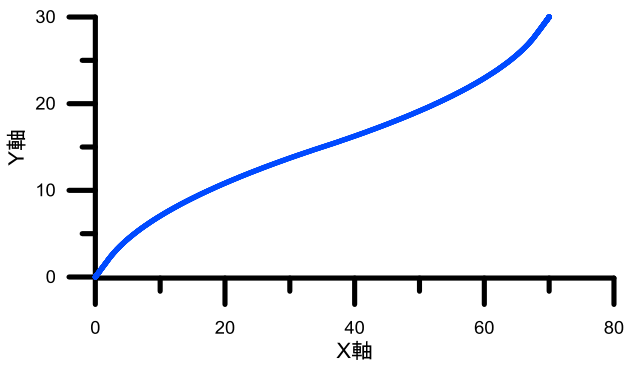
下图与范例程序为一使用NMC_GroupPtpAcsAll()之范例，使第1轴移动至90的位置；第2轴移动至50的位置；第3轴移动至70的位置；第4轴移动至30的位置，呼叫完此API后，群组将需要移动的轴向进行插补。




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

int main()
{
    RTN_ERR ret              = 0;
    I32_T   groupIndex        = 0;
    I32_T   groupAxesIdxMask = 15; //移动第1轴、第2轴、第3轴与第4轴
    I32_T   devID            = 0;
    Pos_T   acsPos           = { 0 };

    acsPos[0] = 90; //第1轴的目标位置
    acsPos[1] = 50; //第2轴的目标位置
    acsPos[2] = 70; //第3轴的目标位置
    acsPos[3] = 30; //第4轴的目标位置

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

    return 0;
}
```

下图与范例程序为一使用NMC_GroupPtpCartAll()之范例，使群组移动X方向至70的位置；Y方向至30的位置，呼叫完此API后，群组将需要移动的轴向进行插补。






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

int main()
{
    RTN_ERR ret          = 0;
    I32_T   groupIndex   = 0;
    I32_T   cartAxesMask = 5; //移动的坐标为X轴与Z轴
    I32_T   devID        = 0;
    Pos_T   targetPos    = { 0 };

    targetPos[0] = 10; //X轴的目标位置
    targetPos[2] = 20; //Z轴的目标位置

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

    return 0;
}
```

1.4.8. 群组 Jog 运动

若需要将群组中某一轴向以最大速度移动(于ACS空间中操作),则可使用NMC_GroupJogAcs()达成。当指令下达成功后，该群组轴将以设定之最大速运转。如要停止可呼叫NMC_GroupHalt()将或呼叫NMC_GroupHaltAll()将整个群组暂停。Jog的速度规划依下列轴参数定义:

Param. Num.	Sub. Index	资料型态	说明
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)

设定或读取群组轴参数相关之API如下:

NMC_GroupAxSetParamI32()

NMC_GroupAxGetParamI32()

NMC_GroupAxSetParamF64()

NMC_GroupAxGetParamF64()

若移动过程中遇到硬件或软件极限，则该轴会自动停止，且状态(STATE)的变化为MOVING→STOPPING→STOPPED，若需要将状态切换为STAND STILL，请使用NMC_GroupResetState()。

1.4.9. 群组停止运动

任何运动命令皆可透过下列指令将运动停止:

API	说明
NMC_GroupHalt() NMC_GroupStop()	停止单一群组
NMC_GroupHaltAll() NMC_GroupStopAll()	停止所有群组
NMC_DeviceHaltAll() NMC_DeviceStopAll()	停止所由轴与群组

Halt 与 Stop的行为比较:

	Halt	Stop
行为	将目前正在运行中的运动中止，若运动队列(Motion buffer)中无其他指运动指令则群组状态将回到 NMC_GROUP_STATE_STAND_STILL，反之若运动队列(Motion buffer)中尚有其他的运动指令则队列中之指令将会继续被执行。	将目前正在运行中的运动中止，并将运动队列(Motion buffer)中所有运动指令清空，群组状态会切换至 NMC_GROUP_STATE_STOPPED，直到使用者呼叫 NMC_GroupResetState()才能执行新的运动指令
参数	0x31:Halt profile type 0x34:Halt deceleration (unit/sec2) 0x35:Halt jerk (unit/sec3) 0x36-0:Buffer mode	0x20 :Stop profile type 0x21 :Stop deceleration (unit/sec2) 0x22 :Stop jerk (unit/sec3)
是否可队列?	指令可被放入运动队列中(Motion buffer)。配合群组参数 0x36 sub 0(Buffer Mode)来使用，若使用 0 的数值(Aborting)，则会立刻停止运动；若使用 1 的数值(Buffered)，则会将此命令加入至排序中。	无

1.4.10.群组直线补间

若要执行卡式坐标系(Cartisian space)下的直线运动，可使用下列 API:

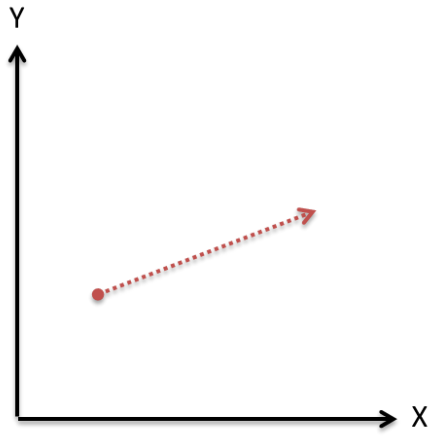
API	说明
NMC_GroupLineXY()	2D 直线补间(XY轴)
NMC_GroupLine()	3D 直线补间与3D姿态(Orientation)控制(最高六自由度)

直线补间运动功能可分为

1. 线性轴之直线补间
2. 线性轴之直线补间 + 姿态轴(ABC 轴)姿态控制

1.4.10.1. 多维度直线补间

一般二维的机构(如 XY Table)或机构超过二维可使用 NMC_GroupLineXY()下二维直线补间运动，如下图所示:



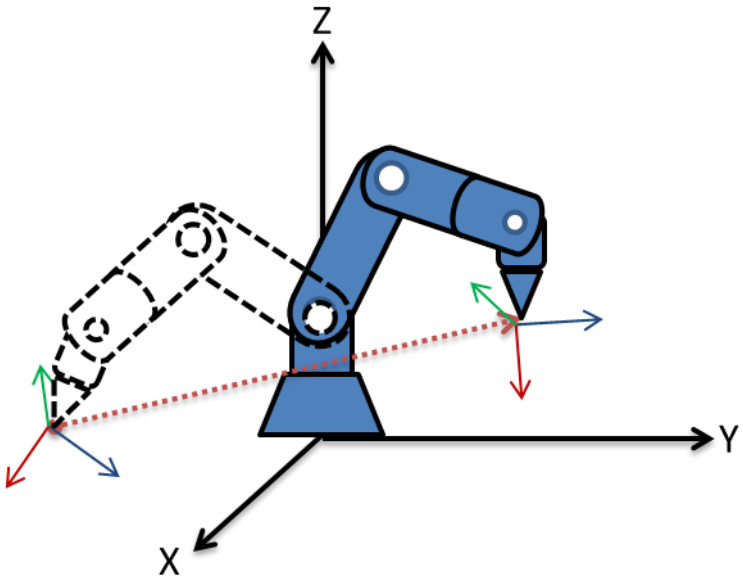
若为2轴以上的线性机构要进行补间运动则可采用NMC_GroupLine()下达，群组直线补间运动使用下列群组参数进行规划:

定义	常数值	说明
目标位置型式	0x30	0: 绝对位置; 1: 增量位置
速度规划方式	0x31	0: T curve; 1: S curve
最大速度	0x32	速度规划的最大速度(unit/sec)
加速度	0x33	速度规划的加速度(unit/sec^2)
减速度	0x34	速度规划的减速度(unit/sec^2)

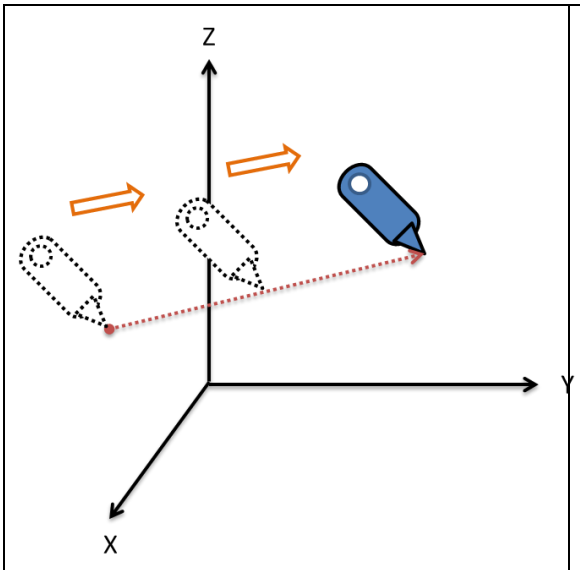
急冲量	0x35	速度规划的急冲量(unit/sec^3)(仅 S curve 有效)

1.4.10.2. 3D 直线补间与姿态控制

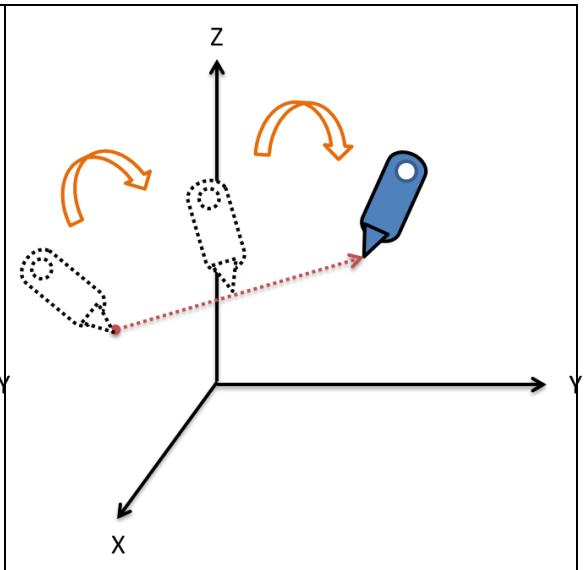
若设定的群组机构形式为机器人(Robot)，其ABC值则定义为姿态角(Oreintation angle)，下图为一六轴机械手臂执行直线补间伴随姿态控制



下图说明有无姿态控制之示意图:



直线补间(无姿态控制)



直线补间 + 姿态控制

群组直线补间与姿态控制使用下列群组参数进行规划:

- 线性轴直线捕间速度参数:

定义	常数值	说明
目标位置型式	0x30	0: 绝对位置; 1: 增量位置
速度规划方式	0x31	0: T curve; 1: S curve
最大速度	0x32	速度规划的最大速度(unit/sec)
加速度	0x33	速度规划的加速度(unit/sec^2)
减速度	0x34	速度规划的减速度(unit/sec^2)
急冲量	0x35	速度规划的急冲量(unit/sec^3)(仅 S curve 有效)

- 姿态控制速度参数:

定义	常数值	说明
姿态角独立插补	0x3A	0: 独立插补; 1: 非独立插补
姿态角最大速度	0x3B	姿态角的最大速度(unit/sec)
姿态角加速度	0x3C	姿态角的加速度(unit/sec^2)
姿态角减速度	0x3D	姿态角的减速度(unit/sec^2)
姿态角急冲量	0x3E	姿态角的急冲量(unit/sec^3)(仅 S curve 有效)

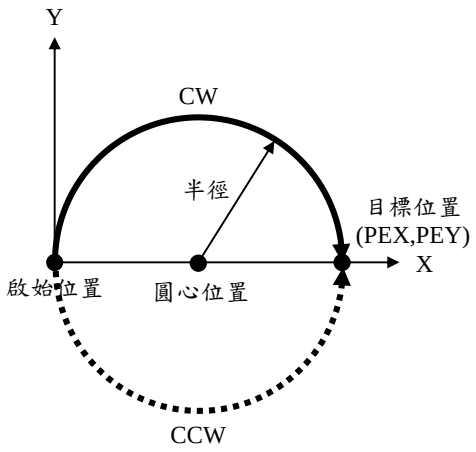
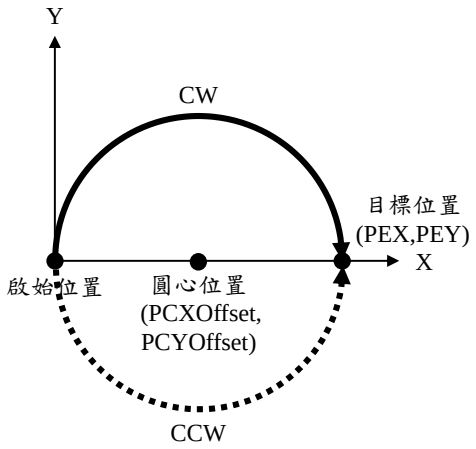
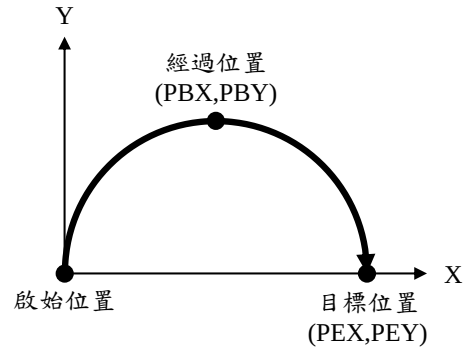
0x3A 参数决定姿态角运动是否使用独立之插补器运算:

- 当 0x3A 设定为 0(独立插补)，表示姿态控制采用独立插补器进行速度规划，其姿态角的速度规划参考 0x3B~0x3E;
- 当 0x3A 设定为 1(非独立差补)，表示姿态控角之变化量等比例跟随直线运动。

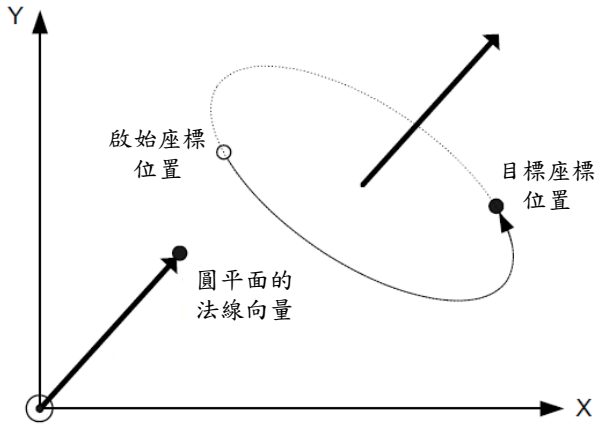
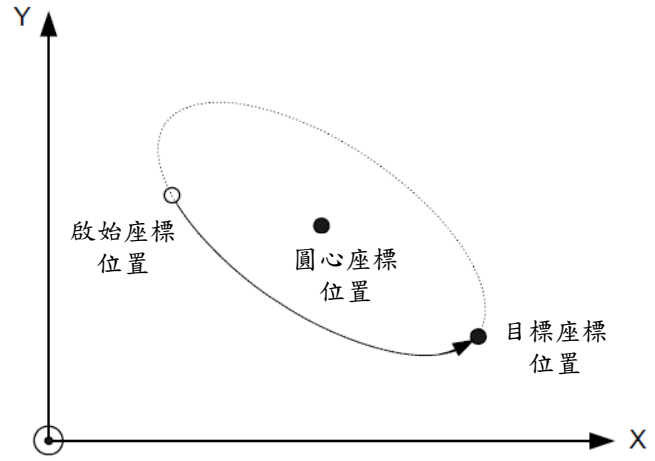
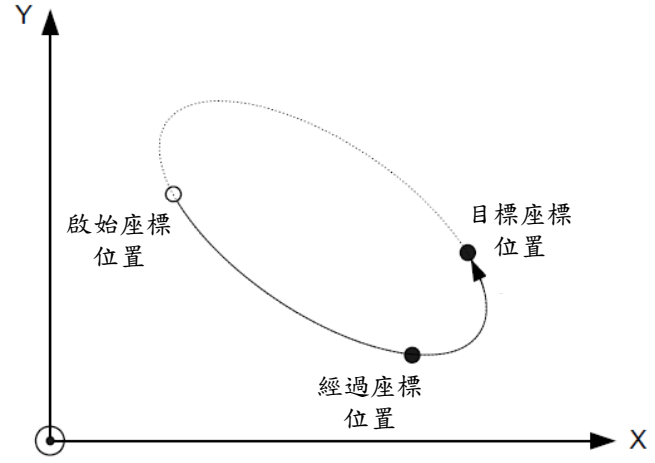
1.4.11. 群组圆弧补间

执行卡式坐标系(Cartisian space)下的圆弧运动，本控制器提供下列方式定义圆弧运动:

- 2D 圆弧

定义方式	相关 API	示意图
终点位置 半径 方向	NMC_GroupCirc2R()	
终点位置 圆心位置 方向	NMC_GroupCirc2C()	
终点位置 经过点位置	NMC_GroupCirc2B()	

- 3D 圆弧(含姿态)

定义方式	相关 API	示意图
终点位置 姿态 半径 方向 平面法向量	NMC_GroupCircR()	
终点位置 姿态 圆心位置 方向	NMC_GroupCircC()	
终点位置 姿态 经过点位置	NMC_GroupCircB()	

其相关速度规划参数如下:

- 圆弧切线速度参数:

定义	常数值	说明
目标位置型式	0x30	0: 绝对位置; 1: 增量位置
速度规划方式	0x31	0: T curve; 1: S curve

最大速度	0x32	圆弧切线最大速度(unit/sec)
加速度	0x33	圆弧切线加速度(unit/sec^2)
减速度	0x34	圆弧切线减速度(unit/sec^2)
急冲量	0x35	圆弧切线急冲量(unit/sec^3)(仅 S curve 有效)

- 姿态控制速度参数:

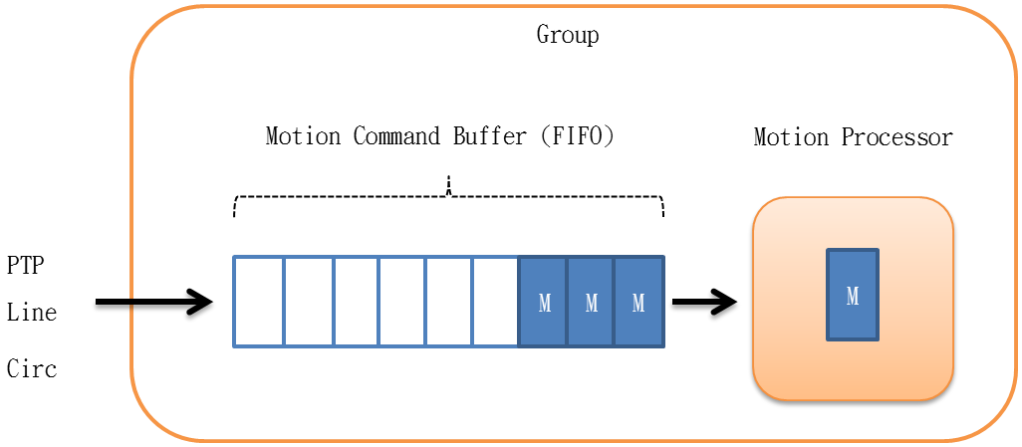
定义	常数值	说明
姿态角独立插补	0x3A	0: 独立插补; 1: 非独立插补
姿态角最大速度	0x3B	姿态角的最大速度(unit/sec)
姿态角加速度	0x3C	姿态角的加速度(unit/sec^2)
姿态角减速度	0x3D	姿态角的减速度(unit/sec^2)
姿态角急冲量	0x3E	姿态角的急冲量(unit/sec^3)(仅 S curve 有效)

0x3A 参数决定姿态角运动是否使用独立之插补器运算:

- 当 0x3A 设定为 0(独立插补)，表示姿态控制采用独立插补器进行速度规划，其姿态角的速度规划参考 0x3B~0x3E;
- 当 0x3A 设定为 1(非独立差补)，表示姿态控角之变化量等比例跟随直线运动。

1.4.12. 连续运动

每个群组设置有命令队列(Motion command buffer)，可以连续将运动命令设定存入队列中，此队列为先进先出(FIFO)，控制器会依序将运动队列中之运动命令依序执行，可透过NMC_GroupGetMotionBuffSpace()读取目前还有多少队列空间。默认的群组队列大小为 32。



可被暂存的运动命令有下列:

1. 点对点运动(PTP)
2. 直线补间运动(Line)
3. 圆弧补间运动(Circ)

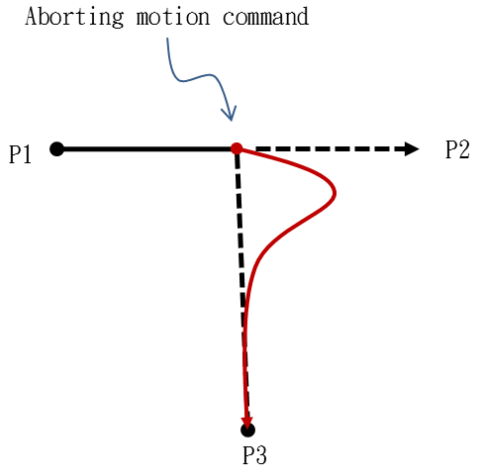
当用户呼叫上述运动指令时控制器会参考群组参数 0x36:0 (Buffer mode)来判断如何处里新的运动命令和当前(或前一个)运动命令的连接方式，下表为 Buffer Mode 的参数意义:

群组参数 0x36:0 (Buffer Mode)

数值	Buffer Mode	说明
0	Aborting	中断目前运动并立即执行此运动命令。
1	Buffered	暂存此运动命令待前一运动命令结束后再执行
2	Blending	混合此运动命令与前一个运动命令，使轨迹与速度平滑化， 平滑程度透过群组参数 0x36:1~3 设定之。

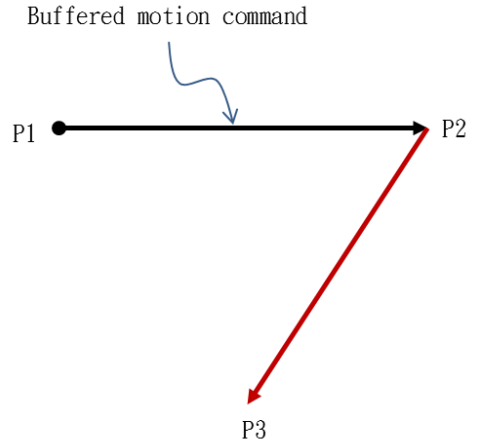
1.4.12.1. Buffer Mode: Aborting

系统默认为参数为 Aborting 模式，亦即当运动命令下达后会中断目前运动立刻执行，若队列中有其他运动命令也会一并清空。下图为 Aborting 之示意图:



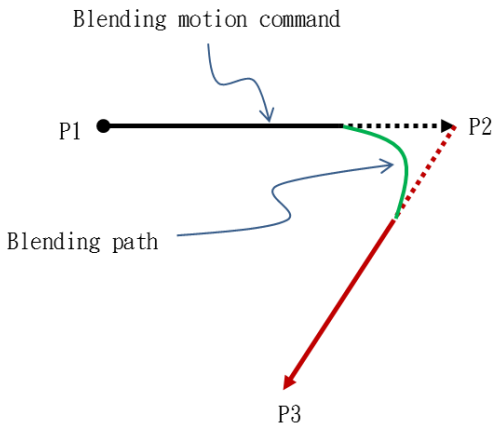
1.4.12.2. Buffer Mode: Buffered

若设定为 Buffered 模式，运动命令将被暂存至队列中待前一命令执行完毕后再被执行，前一个运动指令会减速至零，此模式通常被应用于希望轨迹规划能确实通过目标点位，下图为 Buffered 命令之示意图:



1.4.12.3. Buffer Mode: Blending

若设定路径连接方式为混合模式(Blending)，可将接续的两个路径进行位置与速度的混合，透过此功能可提升加工路径轨迹之平滑性，提升运动速率，缩短加工时间，但路径不通过设定点位。下图为 Blending 命令之示意图:



路径混合模式(Buffer Mode)包括: BlendingLow, BlendingPrevious, BlendingNext, BlendingHigh 和 Blending，而路径混合功能可另外设定混合方式 Transition Mode(群组参数 0x36 Sub 1)和 Transition Paratemer(群组参数 0x36 Sub 2)来微调其混合时的行为，参数意义如下表:

Transition Mode(参数 0x36 Sub 1):

数值	Transition Mode	说明
0	None	速度连续运动的速度由 Buffer Mode 决定，如 BlendingLow、BlendingHigh、BlendingPrevious、BlendingNext。
1	Start Velocity	速度连续运动的速度由当前运动的速度百分比决定
2	Constant Velocity	速度连续运动的速度由下一个运动的速度百分比决定
3	Corner Distance	速度连续运动的速度由转角的距离决定。
4	Corner Deviation	速度连续运动的速度由转角的误差决定。

Transition paratemer(参数 0x36 Sub 2):

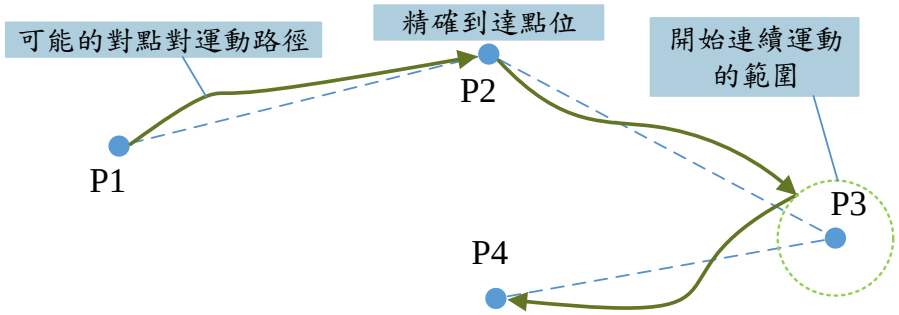
依 Buffer Mode 与 Transition Mode 设定的参数值不同，此参数可设定为速度百分比、转角的距离或误差。

范例:点对点运动(PTP)之间的连续运动

下表为一连续运动指令:

顺序	运动指令	连接方式 Buffer Mode	混合方式 Blending Mode	混合参数 Blending Parameter
1	PTP(P2)	忽略 (目前无运动)	忽略 (目前无运动)	忽略 (目前无运动)
2	PTP(P3)	Buffered	忽略	忽略

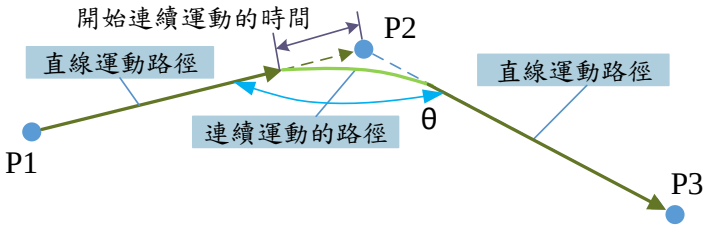
3	PTP(P4)	Blending	Corner Distance	Distance Value



当 PTP(P3)无使用连续运动功能，则前一运动指令会精确地到达 P2 点位;当 PTP(P4)使用连续运动功能，当插补点位接近 P3 点位时(到达指定的 Corner Distance 范围)，会立刻下达新的位置命令(P4)，且末端点的速度不会减速为零。

范例:两个直线补间(Line)的连续运动：

顺序	运动指令	连接方式 Buffer Mode	混合方式 Blending Mode	混合参数 Blending Parameter
1	Line(P2)	忽略 (目前无运动)	忽略 (目前无运动)	忽略 (目前无运动)
2	Line(P3)	Blending	Corner Distance	Distance value

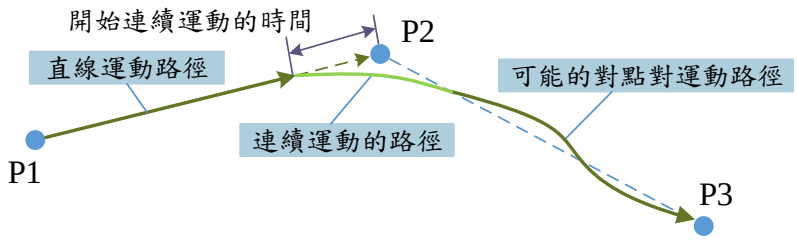


若 Line(P3) 使用路径混合模式(Buffer mode = Blendig)，当插补点位到达指定的 Corner Distance 时，控制器会将 Line(P2) 的路径与 Line(P3)之路径进行速度与路径平滑化，而末端点的速度不会减速为零，路径规划将不会通过 P2 点位。

范例:直线补间(Line)和点对点运动(PTP)的连续运动：如下图，

顺序	运动指令	连接方式 Buffer Mode	混合方式 Blending Mode	混合参数 Blending Parameter
1	Line(P2)	忽略	忽略	忽略

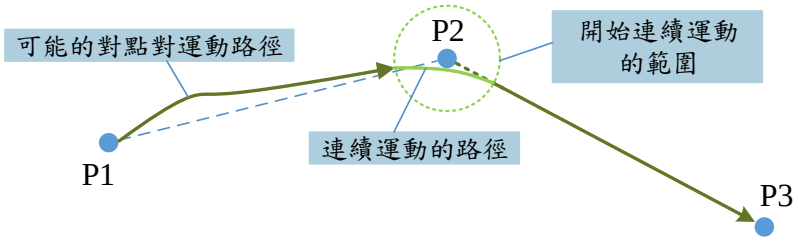
		(目前无运动)	(目前无运动)	(目前无运动)
2	PTP(P3)	Blending	Corner Distance	Distance value



若 PTP(P3) 使用路径混合模式(Buffer mode = Blendig)，当插补点位到达指定的 Corner Distance 时，控制器会将 Line(P2) 的路径与 PTP(P3)之路径进行速度与路径平滑化，而末端点的速度不会减速为零，路径规划将不会通过 P2 点位。

范例:点对点运动(PTP)与直线补间(Line)的连续运动:

顺序	运动指令	连接方式 Buffer Mode	混合方式 Blending Mode	混合参数 Blending Parameter
1	PTP(P2)	忽略 (目前无运动)	忽略 (目前无运动)	忽略 (目前无运动)
2	Line(P3)	Blending	Corner Distance	Distance value



若 Line(P3) 使用路径混合模式(Buffer mode = Blendig)，当 PTP 的路径进入到达指定的 Corner Distance 范围内，控制器会将 Line(P3) 的路径与 PTP(P2)之路径进行速度与路径平滑化，路径规划将不会通过 P2 点位且末端点的速度不会减速为零。

不同的运动命令的组合有其使用规则，请参考下表(○代表支持；✕代表不支持)。

直线补间-直线补间(或 直线补间-圆弧补间 / 圆弧补间-直线补间 / 圆弧补间-圆弧补间)

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

Corner Distance	×	×	×	×	×	×	○
Corner Deviation	×	×	×	×	×	×	○

直线补间-点对点运动(或 圆弧补间-点对点运动)

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

点对点运动-直线补间(或 点对点运动-圆弧补间)

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

点对点运动-点对点运动

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

Halt()/Stop() command 的作用?

Motion state/Motion satus 的变化?

简易版(所有)

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. 群组归零运动

由于群组归零运动和单轴归零运动功能相同，相关的使用行为与使用时机，请参考单轴归零运动章节中的说明。

与群组归零运动相关的函式，说明如下：

- NMC_GroupSetHomePos():

  在此函式中，使用者必须输入在目前各轴的位置下，所欲设定的各轴之坐标位置(ACS)。

- NMC_GroupAxesHomeDrive():

  在此函式中，使用者可以指定要进行单轴归零运动的群组轴，成功呼叫后，指定的群组轴将依据单轴参数中与归零运动相关的参数，进行各自的单轴归零程序。只要有一轴仍在进行归零程序时，群组的状态会停留在HOMING。在归零程序中，若有任一轴发生错误的状况，群组状态将切换到ERROR。

2. 控制器参数

2.1. 系统参数

Param. Num.	Sub. Index	资料型态	说明	设定范围	备注
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): 系统启动后参数生效，启动中无法修改该参数(Return error)

(*2): Enable 该功能时参数生效, 其他时间修改不会立刻变更

(*3): 可设定范围根据控制器的规格而不同

(*4): Read only

2.2. 单轴参数

Param. Num.	Sub. Index	资料型态	说明	设定范围	备注
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 external 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)
0x09	0	I32_T	Digital input configuration	该值以 bit 方式设定, 各 bit 设定意义如下: Bit 0: Set 1 to ignore PEL signal Bit 1: Set 1 to ignore MEL signal Bit 2~15: Reserved, set 0 Bit 16: Set 1 to inverse PEL signal Bit 17: Set 1 to inverse MEL signal Bit 18~31: Reserved, set 0	(*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 speed search switch	Refer to drive user manual	(*5)
	3	F64_T	EtherCAT CiA HOME speed 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): 系统启动后参数生效，启动中无法修改该参数(Return error)

(*2): Enable 该功能时参数生效, 其他时间修改不会立刻变更

(*3): 可设定范围根据控制器的规格而不同

(*4): Read only

(*5): 可设定范围根据受控装置的规格而不同

2.3. 群组参数

Param. Num.	Sub. Index	资料型态	说明	设定范围	备注
0x00	0	I32_T	Kinematics type	0:Linear (2~8 Axes) 1:Articulated Robot (AR6) 2:Delta 3:SCARA	(*1)
	1~48	F64_T	Kinematics parameter 1~48	参考机构运动学设定章节	(*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 (负号代表反方向补偿)	(*1)
	5	F64_T	Coupling ratio denominator	Value > 0	(*1)
	6	I32_T	Enable mechanical couple compensation	0:Disable, 1:Enable	(*1)
0x20	0	I32_T	Profile type for GroupStop command	0: T-Curve, 1:S-Curve	
0x21	0	F64_T	Deceleration for GroupStop command (unit/sec^2)	1 ~ 2147483647	
0x22	0	F64_T	Jerk for GroupStop command (unit/sec^3)	1 ~ 2147483647	
0x30	0	I32_T	Absolute/relative programming(Cartesian coord. system)	0:Absolute, 1:Relative	
0x31	0	I32_T	Profile type (Cartesian coord. system)	0:T-Curve, 1:S-Curve	
0x32	0	F64_T	Max. velocity (Cartesian coord. System, unit/sec)	1 ~ 2147483647	
0x33	0	F64_T	Acceleration (Cartesian coord. system, unit/sec^2)	1 ~ 2147483647	
0x34	0	F64_T	Deceleration (Cartesian coord. system, unit/sec^2)	1 ~ 2147483647	
0x35	0	F64_T	Jerk (Cartesian coord. system, unit/sec^3)	1 ~ 2147483647	
0x36	0	I32_T	Buffer mode (Cartesian coord. system)	0:Aborting, 1:Buffered 2:BlendingLow 3:BlendingPrevious 4:BlendingNext 5:BlendingHigh 6:Blending	
	1	I32_T	Transition mode selection (Cartesian coord. system)	0: None 1: StartVelocity 2: ConstantVelocity 3: CornerDistance 4: MaxCornerDeviation	
	2	F64_T	Transition parameter	Refer to transition parameter description	

0x38	0	I32_T	Coord. system selection	0:MCS, 1:PCS	
0x3A	0	I32_T	Orientation motion independent	0: Not independent 1:Independent	
0x3B	0	F64_T	Max. orientation velocity (Cartesian coord. System, unit/sec)	1 ~ 2147483647	
0x3C	0	F64_T	Orientation acceleration (Cartesian coord. System, unit/sec^2)	1 ~ 2147483647	
0x3D	0	F64_T	Orientation deceleration (Cartesian coord. System, unit/sec^2)	1 ~ 2147483647	
0x3E	0	F64_T	Orientation jerk (Cartesian coord. System, unit/sec^3)	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	I32_T	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): 系统启动后参数生效，启动中无法修改该参数(Return error)

(*2): Enable 该功能时参数生效, 其他时间修改不会立刻变更

3. C/C++函式库

3.1. APIs 总览

下表列出 NexMotion 函式库 API 的列表，所有 API 定义于 NexMotion.h 头文件之中。

版本及错误信息读取相关函式

函数名称	说明
NMC_GetLibVersion	回传函式库(Library)的版本编号
NMC_GetLibVersionString	回传函式库(Library)的版本编号 c 字符串
NMC_GetErrorDescription	回传错误码描述 c 字符串

控制器启动相关函式

函数名称	说明
NMC_DeviceOpenUp	开启控制器 (阻塞式呼叫)
NMC_DeviceShutdown	关闭控制器 (阻塞式呼叫)
NMC_DeviceOpenUpRequest	送出开启控制器请求(非阻塞式呼叫)
NMC_DeviceWaitOpenUpRequest	等待控制器启动程序(阻塞式呼叫)
NMC_DeviceShutdownRequest	送出关闭控制器请求(非阻塞式呼叫)
NMC_DeviceWaitShutdownReque	等待控制器关闭程序(阻塞式呼叫)

进阶控制器启动相关函式

函数名称	说明
NMC_DeviceCreate	取得控制器标识符
NMC_DeviceDelete	移除控制器标识符
NMC_DeviceLoadIniConfig	加载控制器之设定参数
NMC_DeviceResetConfig	清除控制器之设定参数
NMC_DeviceStart	启动控制器(阻塞式呼叫)
NMC_DeviceStop	停止控制器(阻塞式呼叫)
NMC_DeviceStartRequest	送出启动控制器的请求(非阻塞式呼叫)
NMC_DeviceStopRequest	送出停止控制器的请求(非阻塞式呼叫)
NMC_DeviceGetState	读取控制器的状态

看门狗相关函式

函数名称	说明
NMC_DeviceWatchdogTimerEnable	启动看门狗定时器
NMC_DeviceWatchdogTimerDisable	停止看门狗定时器

NMC_DeviceWatchdogTimerReset	清除看门狗计数器

系统参数设定相关函式

函数名称	说明
NMC_DeviceSetParam	设定控制器之系统参数
NMC_DeviceGetParam	读取控制器之系统参数
NMC_SetIniPath	设定 INI 档案所在位置

I/O 控制相关函式

函数名称	说明
NMC_GetInputMemorySize	读取数字输入(Input)映像内存之大小
NMC_GetOutputMemorySize	读取数字输出(Output)映像内存之大小
NMC_ReadInputMemory	读取数字输入(Input)映像内存
NMC_ReadOutputMemory	读取数字输出(Output)映像内存
NMC_WriteOutputMemory	写入数字输出(Output)映像内存

轴或群组数量信息

函数名称	说明
NMC_DeviceGetAxisCount	读取轴数量信息
NMC_DeviceGetGroupCount	读取群组数量信息
NMC_DeviceGetGroupAxisCount	读取群组中轴数量信息

轴或组名信息

函数名称	说明
NMC_AxisGetDescription	读取指定轴的名称描述信息
NMC_GroupGetDescription	读取指定群组的名称描述信息

所有轴与群组启用/禁用函式

函数名称	说明
NMC_DeviceEnableAll	启用系统中所有轴与群组
NMC_DeviceDisableAll	停用系统中所有轴与群组

所有轴与群组运动中止函式

函数名称	说明
NMC_DeviceHaltAll	系统中所有轴与群组运动终止 (Stand still 状态)
NMC_DeviceStopAll	系统中所有轴与群组运动终止 (Stopped 状态)

系统讯息相关

函数名称	说明
NMC_MessagePopFirst	读取系统消息队列
NMC_MessageOutputEnable	讯息转发一份至 MS Windows 系统讯息

函式追踪相关

函数名称	说明
NMC_DebugSetTraceMode	设定 API 追踪模式
NMC_DebugSetHookData	设定传入嵌入函式(Hook function)料结构指针
NMC_DebugSetHookFunction	设定嵌入函式(Hook function)
NMC_DebugGetApiAddress	读取 API 的 Address

单轴参数设定相关函式

函数名称	说明
NMC_AxisSetParamI32	设定轴参数(I32_T 数据型态)
NMC_AxisGetParamI32	读取轴参数(I32_T 数据型态)
NMC_AxisSetParamF64	设定轴参数(F64_T 数据型态)
NMC_AxisGetParamF64	读取轴参数(F64_T 数据型态)

单轴状态控制相关函式

函数名称	说明
NMC_AxisEnable	单轴启用
NMC_AxisDisable	单轴停用
NMC_AxisGetStatus	读取单轴运动信息
NMC_AxisGetState	读取单轴状态
NMC_AxisResetState	清除单轴错误状态
NMC_AxisResetDriveAlm	清除单轴伺服警报

单轴运动信息读取相关函式

函数名称	说明
NMC_AxisGetCommandPos	读取单轴命令位置
NMC_AxisGetActualPos	读取单轴实际位置
NMC_AxisGetCommandVel	读取单轴命令速度
NMC_AxisGetActualVel	读取单轴实际速度
NMC_AxisGetMotionBuffSpace	读取单轴运动指令暂存空间大小

单轴运动控制相关函式

函数名称	说明
NMC_AxisPtp	启动单轴点对点运动
NMC_AxisJog	启动单轴速度运动

单轴归原点运动函式

函数名称	说明
NMC_AxisSetHomePos	设定原点位置
NMC_AxisHomeDrive	启动单轴归原点运动(驱动器执行)

单轴运动中止函式

函数名称	说明
NMC_AxisHalt	单轴运动终止 (Stand still 状态)
NMC_AxisStop	单轴运动终止 (Stopped 状态)
NMC_AxisHaltAll	所有单轴运动终止 (Stand still 状态)
NMC_AxisStopAll	所有单轴运动终止 (Stopped 状态)

单轴运行中变动相关函式

函数名称	说明
NMC_AxisVelOverride	单轴运行中速度改变
NMC_AxisAccOverride	单轴运行中加速度改变
NMC_AxisDecOverride	单轴运行中减速度改变

单轴总体速率设定相关函式

函数名称	说明
NMC_AxisSetVelRatio	设定单轴速度百分比
NMC_AxisGetVelRatio	读取单轴速度百分比

群组参数设定相关函式

函数名称	说明
NMC_GroupSetParamI32	设定群组参数(I32_T 数据型态)
NMC_GroupGetParamI32	读取群组参数(I32_T 数据型态)
NMC_GroupSetParamF64	设定群组参数(F64_T 数据型态)
NMC_GroupGetParamF64	读取群组参数(F64_T 数据型态)
NMC_GroupAxSetParamI32	设定群组轴参数(I32_T 数据型态)
NMC_GroupAxGtParamI32	读取群组轴参数(I32_T 数据型态)
NMC_GroupAxSetParamF64	设定群组轴参数(F64_T 数据型态)
NMC_GroupAxGetParamF64	读取群组轴参数(F64_T 数据型态)

群组状态控制相关函式

函数名称	说明
NMC_GroupEnable	群组启用
NMC_GroupDisable	群组停用
NMC_GroupGetStatus	读取群组运动信息
NMC_GroupGetState	读取群组状态
NMC_GroupResetState	清除群组错误状态
NMC_GroupResetDriveAlm	清除群组轴伺服警报
NMC_GroupResetDriveAlmAll	清除群组中所有轴伺服警报

群组总体速率设定相关函式

函数名称	说明
NMC_GroupSetVelRatio	设定单轴速度百分比
NMC_GroupGetVelRatio	读取单轴速度百分比

群组点对点运动(轴坐标系)相关函式

函数名称	说明
NMC_GroupPtpAcs	启动群组单轴之点对点运动
NMC_GroupPtpAcsAll	启动群组所有轴之点对点运动

群组轴 Jog 运动(轴坐标系)相关函式

函数名称	说明
NMC_GroupJogAcs	启动群组轴之速度运动

群组点对点运动(卡氏坐标)相关函式

函数名称	说明
NMC_GroupPtpCart	启动群组卡氏坐标单轴之点对点运动
NMC_GroupPtpCartAll	启动群组卡氏坐标之点对点运动

群组轴 Jog 运动(卡氏坐标)相关函式

函数名称	说明
GroupJogCartFrame	启动群组轴卡式坐标之速度运动

群组运动中止函式

函数名称	说明
NMC_GroupHalt	群组运动终止 (Stand still 状态)

NMC_GroupStop	群组运动终止（Stopped 状态）
NMC_GroupHaltAll	所有群组运动终止（Stand still 状态）
NMC_GroupStopAll	所有群组运动终止（Stopped 状态）

群组运动信息读取相关函式

函数名称	说明
NMC_GroupGetCommandPosAcs	读取群组轴坐标(ACS)命令位置
NMC_GroupGetActualPosAcs	读取群组轴坐标(ACS)实际位置
NMC_GroupGetCommandPosPcs	读取群组卡式坐标(PCS)命令位置
NMC_GroupGetActualPosPcs	读取群组卡式坐标(PCS)实际位置
NMC_GroupGetCommandPos	读取群组命令位置
NMC_GroupGetActualPos	读取群组实际位置
NMC_GroupGetMotionBuffSpace	读取群组运动指令暂存空间大小

群组归原点运动函式

函数名称	说明
NMC_GroupSetHomePos	设定群组轴原点位置
NMC_GroupAxesHomeDrive	启动群组轴归原点运动(驱动器执行)

群组 2D 直线圆弧补间运动相关函式

函数名称	说明
NMC_GroupLineXY	群组 2D 直线补间运动
NMC_GroupCirc2R	群组 2D 圆弧补间运动给定终点和半径
NMC_GroupCirc2C	群组 2D 圆弧补间运动给定终点和圆心
NMC_GroupCirc2B	群组 2D 圆弧补间运动给定终点和经过点

群组 3D 直线圆弧补间运动相关函式

函数名称	说明
NMC_GroupLine	群组 3D 直线补间运动
NMC_GroupCircR	群组 3D 圆弧补间运动给定终点和半径
NMC_GroupCircC	群组 3D 圆弧补间运动给定终点和圆心
NMC_GroupCircB	群组 3D 圆弧补间运动给定终点和经过点

Tool 教导相关函式

函数名称	说明
NMC_ToolCalib_4p	Tool 教导- TCP 平移教导法
NMC_ToolCalib_4pWithZ	Tool 教导- TCP 平移与 Z 方向设定教导法

NMC_ToolCalib_4pWithOri	Tool 教导- TCP 平移与姿态设定教导法
NMC_ToolCalib_Ori	Tool 教导- TCP 姿态设定教导法

Base 教导相关函式

函数名称	说明
NMC_BaseCalib_1p	Base 教导-1p 法
NMC_BaseCalib_2p	Base 教导-2p 法
NMC_BaseCalib_3p	Base 教导-3p 法

3.2. 系统相关 APIs

3.2.1. 版本及错误信息读取相关函式

3.2.1.1. NMC_GetLibVersion

回传函式库(Library)的版本编号: (Major.Minor.Stage.Build)

- **C/C++语法:**

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

- **参数:**

I32_T *PRetMajor: 输入一指针变量，回传主版本号码, 可以输入 NULL (0)忽略此参数

I32_T *PRetMinor: 输入一指针变量，回传次版本号码, 可以输入 NULL (0)忽略此参数

I32_T *PRetStage: 输入一指针变量，回传阶段版本号码, 可以输入 NULL (0)忽略此参数

1~4:测试阶段版本，5:正式发行版

I32_T *PRetBuild: 输入一指针变量，回传编译版本号码, 可以输入 NULL (0)忽略此参数

- **回传值:**

以I32_T 数据型态回传版本编号，回传值意义如下:

Version = (Major x 10,000,000) + (Minor x 100,000) + (Stage x 10,000) + Build

- **用法:**

- **范例:**

```
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 );
```

- **参阅:**

<无>

3.2.1.2. NMC_GetLibVersionString

回传函式库(Library)的版本编号 C-Style 字符串: “Major.Minor.Stage.Build”

- **C/C++语法:**

void NMC_GetLibVersionString(char *PRetVersionString, U32_T StringSize);

- **参数:**

char *PRetVersionString: 输入 C-Style 字符数组，建议大小至少超过 32 char

U32_T StringSize: char 字符数组大小

- **回传值:**

无回传值

- **用法:**

- **范例:**

```
char versionString[32];

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

- **参阅:**

<无>

3.2.1.3. NMC_GetErrorDescription

回传错误码描述 c 字符串

- **C/C++语法:**

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

- **参数:**

RTN_ERR ErrorCode: 欲查询之错误码

_opt_null_ char *PRetErrorDesc: 输入 char 数组，大小建议 512 以上

U32_T StringSize: char 数组大小

- **回传值:**

Const char* : 回传错误代码描述，若 ErrorCode 无定义则回传 NULL.

- **用法:**

呼叫本函式成功后存入错误代码描述。

- **范例:**

- **参阅:**

<无>

3.2.2. 控制器启动相关函式

3.2.2.1. NMC_DeviceOpenUp

开启控制器（阻塞式呼叫）

- **C/C++语法:**

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

- **参数:**

I32_T DevType: 指定驱动装置型态(Device Type)

0:启动为仿真器(Simulator)

1:启动为 EtherCAT 装置

I32_T DevIndex: 指定装置序号，设定为 0

I32_T *PRetDevID:呼叫成功后回传装置识别号(DevID)

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

呼叫其他控制 API 前必须先呼叫 NMC_DeviceOpenUp()来启动控制器，本函数为阻塞式(blocked)呼叫，函数会进行一系列初始化动作包含加载配置文件案等需耗费一段时间，待控制器状态进入操作状态(Operation state)后返回。

- **范例:**

- **参阅:**

NMC_DeviceShutdown()

3.2.2.2. NMC_DeviceShutdown

关闭控制器（阻塞式呼叫）

- **C/C++语法:**

```
RTN_ERR NMC_DeviceShutdown( I32_T DevID );
```

- **参数:**

I32_T DevID: 装置识别号

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”（0），反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

整个应用程序结束前必须呼叫本函式来关闭控制器并释放系统资源，呼叫本函式后会阻塞(blocked)直到关闭程序结束。

- **范例:**

```
```

- **参阅:**

NMC_DeviceOpenUp()

3.2.2.3. NMC_DeviceOpenUpRequest

送出开启控制器请求(非阻塞式呼叫)

- **C/C++语法:**

```
RTN_ERR NMC_DeviceOpenUpRequest( I32_T DevType, I32_T DevIndex );
```

- **参数:**

I32_T DevType: 指定驱动装置型态(Device Type)

I32_T DevIndex: 指定装置序号，设定为 0

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

本函数调用后立即返回，必须使用 NMC_DeviceWaitOpenUpRequest()或使用 NMC_DeviceGetState()读取控制器状态确认是否进入可操作状态。

- **范例:**

```
```

- **参阅:**

NMC_DeviceWaitOpenUpRequest()

3.2.2.4. NMC_DeviceWaitOpenUpRequest

等待控制器启动程序(阻塞式呼叫)

- **C/C++语法:**

```
RTN_ERR NMC_DeviceWaitOpenUpRequest( U32_T WaitMs, I32_T *PRetDevID  );
```

- **参数:**

U32_T WaitMs: 等待启动时间，单位ms，可设定NMC_WAIT_TIME_INFINITE等待程序结束

I32_T *PRetDevID:呼叫成功后回传装置识别号(DevID)

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

本函是一般用于确认控制器是否已成功启动，通常在 NMC_DeviceOpenUpRequest()之后呼叫使用。

呼叫后会阻塞(Blocked)直到控制器进入「OPERATION」或者是「ERROR」状态。

- **范例:**

```
```

- **参阅:**

NMC_DeviceOpenUpRequest()

3.2.2.5. NMC_DeviceShutdownRequest

送出关闭控制器请求(非阻塞式呼叫)

- **C/C++语法:**

```
RTN_ERR NMC_DeviceShutdownRequest( I32_T DevID );
```

- **参数:**

I32_T DevID: 装置识别号

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

本函用于送出关闭控制器请求，呼叫后立即返回，必须使用 NMC_DeviceWaitShutdownRequest() 或使用 NMC_DeviceGetState()读取控制器状态确认是否已离开可操作状态。

- **范例:**

- **参阅:**

3.2.2.6. NMC_DeviceWaitShutdownRequet

等待控制器关闭程序(阻塞式呼叫)

- **C/C++语法:**

RTN_ERR NMC_DeviceWaitShutdownRequest(I32_T DevID, U32_T WaitMs);

- **参数:**

I32_T DevID: 装置识别号

U32_T WaitMs: 等待启动时间，单位ms，可设定NMC_WAIT_TIME_INFINITE (0xFFFFFFFF)等待程序结束

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

本函是一般用于确认控制器是否已成功关闭，通常在 NMC_DeviceShutdownRequest()之后呼叫使用。呼叫后会阻塞(Blocked)直到控制器进入「OPERATION」或者是「ERROR」状态。

- **范例:**

```
```

- **参阅:**

NMC_DeviceShutdownRequest()

3.2.3.1. NMC_DeviceCreate

取得控制器标识符

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T DevIndex: 指定装置序号，设定为 0

I32_T *PRetDevID: 指呼叫成功后回传装置识别号(DevID)

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

参考进阶系统初始化章节

- **范例:**

- **参阅:**

3.2.3.2. NMC_DeviceDelete

移除控制器标识符

- **C/C++语法:**

```
RTN_ERR NMC_DeviceDelete( I32_T DevID );
```

- **参数:**

I32_T DevID: 装置识别号

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

参考进阶系统初始化章节

- **范例:**

```
```

- **参阅:**

3.2.3.3. NMC_DeviceLoadIniConfig

加载控制器之设定参数

- **C/C++语法:**

```
RTN_ERR NMC_DeviceLoadIniConfig( I32_T DevID );
```

- **参数:**

I32_T DevID: 装置识别号

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

本函式会根据「NexMotionLibConfig.ini」档案内容加载控制器相关设定。

「NexMotionLibConfig.ini」预设安装为 C:\Nexcom，使用者不需要自行建立此档案，也应避免修改文件名和其内容避免档案载入错误。

NMC_DeviceLoadIniConfig()会依下列目录顺序搜寻「NexMotionLibConfig.ini」档案:

1. NexMotion.dll 文件夹
2. C:\Nexcom
3. 系统目录(C:\Windows\System32)

若使用者想要自定义「NexMotionLibConfig.ini」的路径位置可另外可呼叫 NMC_SetIniPath() 来设定 ini 档案之路径。

当函数 NMC_DeviceLoadIniConfig()成功呼叫后，控制器状态进入「READY」状态，可准备启动。

- **范例:**

```
```

- **参阅:**

NMC_SetIniPath()

3.2.3.4. NMC_DeviceResetConfig

清除控制器之设定参数

- **C/C++语法:**

```
RTN_ERR NMC_DeviceResetConfig( I32_T DevID );
```

- **参数:**

I32_T DevID: 装置识别号

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”（0），反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

NMC_DeviceResetConfig()用于清除控制器之设定，成功呼叫后控制器状态返回「INIT」状态。

若控制器处于操作中「OPERATION」状态，不可呼叫本函式。

- **范例:**

```
```

- **参阅:**

3.2.3.5. NMC_DeviceStart

启动控制器(阻塞式呼叫)

- **C/C++语法:**

```
RTN_ERR NMC_DeviceStart( I32_T DevID );
```

- **参数:**

I32_T DevID: 装置识别号

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

当控制器状态处于「READY」状态时，呼叫本函式启动控制器，当函式成功返回后控制器进入可操作「OPERATION」状态。

参考进阶系统初始化章节

- **范例:**

```
```

- **参阅:**

3.2.3.6. NMC_DeviceStop

停止控制器(阻塞式呼叫)

- **C/C++语法:**

```
RTN_ERR NMC_DeviceStop( I32_T DevID );
```

- **参数:**

I32_T DevID: 装置识别号

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

当控制器状态处于于「OPERATION」状态，呼叫本函数来停止控制器，当函式成功返回控制器始进入「READY」状态。

- **范例:**

```
```

- **参阅:**

3.2.3.7. NMC_DeviceStartRequest

送出启动控制器的请求(非阻塞式呼叫)

- **C/C++语法:**

```
RTN_ERR NMC_DeviceStartRequest( I32_T DevID );
```

- **参数:**

I32_T DevID: 装置识别号

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

当控制器状态于「READY」状态，呼叫本函式送出启动控制器之请求，送出请求后立刻返回，可使用 NMC_DeviceGetState()读取控制器状态以确认控制器是否进入「OPERATION」状态

- **范例:**

```

```

- **参阅:**

3.2.3.8. NMC_DeviceStopRequest

送出停止控制器的请求(非阻塞式呼叫)

- **C/C++语法:**

```
RTN_ERR NMC_DeviceStopRequest( I32_T DevID );
```

- **参数:**

I32_T DevID: 装置识别号

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

当控制器状态于「OPERATION」状态，呼叫本函式送出停止控制器之请求，送出请求后立刻返回，可使用 NMC_DeviceGetState()读取控制器状态以确认控制器是否进入「READY」状态

- **范例:**

```
```

- **参阅:**

3.2.3.9. NMC_DeviceGetState

读取控制器的状态

- **C/C++语法:**

```
RTN_ERR NMC_DeviceGetState( I32_T DevID, I32_T *PRetDeviceState );
```

- **参数:**

I32_T DevID: 装置识别号

I32_T *PRetDeviceState:呼叫成功后回传装置状态(Device State)

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

- **范例:**

```
```

- **参阅:**

3.2.4.1. NMC_DeviceWatchdogTimerEnable

启动看门狗定时器

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T TimeoutMs: 设定看门狗计数器过时(Timeout)值，单位 milliseconds.

设定范围 20~200000 ms

I32_T WDTMode:计数器到时(Timeout)模式

0:系统状态切换至 READY(若系统状态为 OPERATION)

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

参考看门狗定时器(Watch Dog Timer)章节说明

- **范例:**

参阅:

3.2.4.2. NMC_DeviceWatchdogTimerDisable

停止看门狗定时器

- **C/C++语法:**

```
RTN_ERR NMC_DeviceWatchdogTimerDisable( I32_T DevID );
```

- **参数:**

I32_T DevID: 装置识别号

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

参考看门狗定时器(Watch Dog Timer)章节说明

- **范例:**

参阅:

3.2.4.3. NMC_DeviceWatchdogTimerReset

清除看门狗计数器

- **C/C++语法:**

```
RTN_ERR NMC_DeviceWatchdogTimerReset( I32_T DevID );
```

- **参数:**

I32_T DevID: 装置识别号

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

参考看门狗定时器(Watch Dog Timer)章节说明

- **范例:**

参阅:

3.2.5. 系统参数设定相关函式

3.2.5.1. NMC_DeviceSetParam

3.2.5.2. NMC_DeviceGetParam

设定/读取控制器之系统参数

- **C/C++语法:**

```
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 );
```

- **参数:**

I32_T DevID: 装置识别号

I32_T ParamNum: 参数类别编号

I32_T SubIndex: 参数序号

I32_T ParaValue: 设定之参数数值

I32_T *PRetParaValue: 呼叫成功后回传

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”（0），反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

参数内容请参阅:系统参数章节

- **范例:**

- **参阅:**

系统参数章节

3.2.5.3. NMC_SetIniPath

设定 INI 档案所在位置

- **C/C++语法:**

RTN_ERR NMC_SetIniPath(_opt_null_ const char *PIniPath);

- **参数:**

I32_T DevID: 装置识别号

const char *PIniPath: 指定 ini 档案之路径，c-style 字符串，可给定 NULL(0)代表清除路径设定使用系统默认路径

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

系统启动过程中，会根据「NexMotionLibConfig.ini」加载设定值，使用者可以透过呼叫本函式来指定 INI 文件的路径。系统默认路径为 C:\NEXCOM\。

- **范例:**

- **参阅:**

NMC_DeviceOpenUp()

NMC_DeviceOpenUpRequest()

3.2.6.1. NMC_GetInputMemorySize

读取数字输入(Input)映像内存之大小

- **C/C++语法:**

```
RTN_ERR NMC_GetInputMemorySize ( I32_T DevID, U32_T *PRetSizeByte );
```

- **参数:**

I32_T DevID: 装置识别号

U32_T *PRetSizeByte: 呼叫成功后回传 I/O 输入内存大小，单位 Byte

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

控制器启动后，可调用此函式。

- **范例:**

```
```

- **参阅:**

3.2.6.2. NMC_GetOutputMemorySize

读取数字输出(Output)映像内存之大小

- **C/C++语法:**

```
RTN_ERR NMC_GetOutputMemorySize( I32_T DevID, U32_T *PRetSizeByte );
```

- **参数:**

I32_T DevID: 装置识别号

U32_T *PRetSizeByte: 呼叫成功后回传 I/O 输出内存大小，单位 Byte

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

控制器启动后，可调用此函式。

- **范例:**

```
```

- **参阅:**

3.2.6.3. NMC_ReadInputMemory

读取数字输入(Input)映像内存

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

U32_T OffsetByte: 内存偏移量，从 0 开始

U32_T SizeByte: 欲读取内存之大小

void *PRetValues:成功呼叫后存入指定的内存区段值于变量中

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

控制器启动后，可调用此函式。

- **范例:**

```
```

- **参阅:**

3.2.6.4. NMC_ReadOutputMemory

读取数字输出(Output)映像内存

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

U32_T OffsetByte: 内存偏移量，从 0 开始

U32_T SizeByte: 欲读取内存之大小

void *PRetValues:成功呼叫后存入指定的内存区段值于变量中

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

控制器启动后，可调用此函式。

- **范例:**

- **参阅:**

3.2.6.5. NMC_WriteOutputMemory

写入数字输出(Output)映像内存

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

U32_T OffsetByte: 内存偏移量，从 0 开始

U32_T SizeByte: 欲写入内存之大小

void *PRetValues:成功呼叫后将此变量的值存入指定的内存区段中

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

控制器启动后，可调用此函式。

- **范例:**

- **参阅:**

3.2.7.1. NMC_DeviceGetAxisCount

读取轴数量信息

- **C/C++语法:**

```
RTN_ERR NMC_DeviceGetAxisCount( I32_T DevID, I32_T *PRetAxisCount );
```

- **参数:**

I32_T DevID: 装置识别号

I32_T *PRetAxisCount: 成功呼叫回传启用单轴数量

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

加载配置文件后，可呼叫此函数确认启用单轴数量。呼叫清除控制器函式 NMC_DeviceResetConfig()后单轴数量为 0

- **范例:**

```
```

- **参阅:**

3.2.7.2. NMC_DeviceGetGroupCount

读取群组数量信息

- **C/C++语法:**

```
RTN_ERR NMC_DeviceGetGroupCount( I32_T DevID, I32_T *PRetGroupCount );
```

- **参数:**

I32_T DevID: 装置识别号

I32_T *PRetGroupCount: 成功呼叫回传启用群组数量

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

加载配置文件后，可呼叫此函数确认启用群组数量。呼叫清除控制器函式 NMC_DeviceResetConfig()后群组数量为 0

- **范例:**

- **参阅:**

3.2.7.3. NMC_DeviceGetGroupAxisCount

读取群组中轴数量信息

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 欲取得信息之群组序号

I32_T *PRetGroupAxisCount:成功呼叫回传该群组的轴数

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

加载配置文件后，可呼叫此函数确认某一启用群组中实际的轴数。

- **范例:**

```
```

- **参阅:**

3.2.8. 轴或组名信息

3.2.8.1. NMC_AxisGetDescription

读取指定轴的名称描述信息

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T AxisIndex: 单轴识别号

U32_T DescStrSize: C-字符串缓存空间大小, byte 单位

char *PRetAxisDescription: 回传轴描述 C-字符串

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

本函数用于读取指定单轴之名称描述，其信息来源根据系统所加载之 NCF 档案，因此必须在加载后方才能正常读取，换言之系统状态必须为 READY 状态。

- **范例:**

```
```

- **参阅:**

3.2.8.2. NMC_GroupGetDescription

读取指定群组的名称描述信息

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

U32_T DescStrSize: C-字符串缓存空间大小, byte 单位

char *PRetAxisDescription: 回传轴描述 C-字符串

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

本函数用于读取指定群组之名称描述，其信息来源根据系统所加载之 NCF 档案，因此必须在加载后方才能正常读取，换言之系统状态必须为 READY 状态。

- **范例:**

- **参阅:**

3.2.9.1. NMC_DeviceEnableAll

启用系统中所有轴与群组

- **C/C++语法:**

```
RTN_ERR NMC_DeviceEnableAll( I32_T DevID );
```

- **参数:**

I32_T DevID: 装置装置识别号

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

控制器启动后，方可调用此函式。

呼叫此函数会将系统中所有单轴与所有群组设定为启动状态(伺服启动 Servo On)。启动过程中会依序先由单轴先启动之后接续各群组之启动，若过程中如遇启动失败则程序停止并回传错误。

- **范例:**

- **参阅:**

3.2.9.2. NMC_DeviceDisableAll

停用系统中所有轴与群组

- **C/C++语法:**

```
RTN_ERR NMC_DeviceDisableAll( I32_T DevID );
```

- **参数:**

I32_T DevID: 装置装置识别号

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

控制器启动后，方可调用此函式。

呼叫此函数会将系统中所有单轴与所有群组设定为停用状态(伺服启动 Servo Off)。停用过程中会依序先由单轴先停用之后接续各群组之停用，若过程中如遇停用失败则程序停止并回传错误。

- **范例:**

- **参阅:**

3.2.10.1. NMC_DeviceHaltAll

系统中所有轴与群组运动终止（Stand still 状态）

- **C/C++语法：**

```
RTN_ERR NMC_DeviceHaltAll( I32_T DevID );
```

- **参数：**

I32_T DevID：装置装置识别号

- **回传值：**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”（0），反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法：**

控制器启动后，方可调用此函式。

呼叫此函数会将系统中所有单轴与所有群组下达减速停止(Halt)指令，其效果等同单一 Halt 指令，过程中会依序先由单轴命令下达之后接续各群组，若过程中如命令下达失败则程序停止并回传错误。

- **范例：**

```
```

- **参阅：**

3.2.10.2. NMC_DeviceStopAll

系统中所有轴与群组运动终止（Stopped 状态）

- **C/C++语法：**

```
RTN_ERR NMC_DeviceStopAll( I32_T DevID );
```

- **参数：**

I32_T DevID：装置装置识别号

- **回传值：**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”（0），反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法：**

控制器启动后，方可调用此函式。

呼叫此函数会将系统中所有单轴与所有群组下达减速停止(Stop)指令，其效果等同单一 Stop 指令，过程中会依序先由单轴命令下达之后接续各群组，若过程中如命令下达失败则程序停止并回传错误。

- **范例：**

- **参阅：**

3.2.11.1. NMC_MessagePopFirst

读取系统消息队列

- **C/C++语法:**

```
RTN_ERR NMC_MessagePopFirst( _opt_null_ NmcMsg_T *PRetMsg );
```

- **参数:**

_opt_null_ NmcMsg_T *PRetMsg: 系统讯息数据结构，可为 NULL 表示从消息队列中移除讯息

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

若系统中已无讯息，回传 ERR_NEXMOTION_QUEUE_EMPTY

- **用法:**

NMC_MessagePopFirst()读取消息队列中之讯息,会依最早(最旧)之讯息依序读取,读取后删除。

- **范例:**

```
```

- **参阅:**

NMC_MessageOutputEnable()

3.2.11.2. NMC_MessageOutputEnable

讯息转发一份至 MS Windows 系统讯息

- **C/C++语法:**

```
void NMC_MessageOutputEnable( BOOL_T Enable );
```

- **参数:**

BOOL_T Enable: False (0) 不转发，True(1)转发

- **回传值:**

无回传

- **用法:**

当呼叫NMC_MessageOutputEnable(True) 后系统讯息将转发至MS Windows之系统讯息中

- **范例:**

```
```

- **参阅:**

NMC_MessagePopFirst()

3.2.12.1. NMC_DebugSetTraceMode

设定 API 追踪模式

- **C/C++语法:**

```
void NMC_DebugSetTraceMode( I32_T TraceMode );
```

- **参数:**

I32_T TraceMode:

0:关闭 trace 功能,不输出

1:只输出发生错误之 API

2:输出所有 API

- **回传值:**

无回传

- **用法:**

参考函式追踪章节

3.2.12.2. NMC_DebugSetHookData

设定传入嵌入函数(Hook function)料结构指针

- **C/C++语法:**

```
void NMC_DebugSetHookData( void *PHookUserData );
```

- **参数:**

void *PHookUserData: 传入指标(Pointer)指向用户定义之变量或数据结构

- **回传值:**

无回传

- **用法:**

参考函式追踪章节

3.2.12.3. NMC_DebugSetHookFunction

设定嵌入函式(Hook function)

- **C/C++语法:**

```
void    NMC_DebugSetHookFunction( PF_NmcHookAPI PFHookFuncPtr );
```

- **参数:**

PF_NmcHookAPI PFHookFuncPtr: 设定嵌入函式(Hook function)之函式指标,

- **回传值:**

无回传

- **用法:**

参考函式追踪章节

3.2.12.4. NMC_DebugGetApiAddress

读取 API 的 Address

- **C/C++语法:**

```
const void* NMC_DebugGetApiAddress( const char *PApiName );
```

- **参数:**

const char *PApiName: 输入 NexMotion API 之函式名称

- **回传值:**

const void*: 回传 NexMotion API 之函式指标

- **用法:**

参考函式追踪章节

3.3. 单轴相关 APIs

3.3.1. 单轴参数设定相关函式

3.3.1.1. NMC_AxisSetParamI32

3.3.1.2. NMC_AxisGetParamI32

3.3.1.3. NMC_AxisSetParamF64

3.3.1.4. NMC_AxisGetParamF64

设定/读取轴参数(I32_T/F64_T 数据型态)

- **C/C++语法:**

```
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 );
```

- **参数:**

I32_T DevID: 装置识别号
I32_T AxisIndex: 单轴识别号
I32_T ParamNum: 参数组别索引编号
I32_T SubIndex: 个别参数索引编号
I32_T ParaValueI32: 设定之参数数值(有号整数)
F64_T ParaValueF64: 设定之参数数值(双精浮点数)
I32_T *PRetParaValueI32: 回传之参数数值(有号整数)
F64_T *PRetParaValueF64: 回传之参数数值(双精浮点数)

- **回传值:**

回传错误代码。
呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

参数内容请参阅：单轴参数章节。

控制器启动过程中会依照配置文件案加载参数值，可调用本函式设定或读取部分单轴参数，在呼叫函数进行参数设定与读取参数值时，必须先确认参数之数据型态为有号整数(I32_T)或双精浮点数(F64_T)，以选用适当之函数。

- **范例：**

- **参阅：**

单轴参数章节

3.3.2.1. NMC_AxisEnable

单轴启用

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T AxisIndex: 单轴识别号

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

1. 在进行单轴运动前，必须先呼叫此函数让单轴进入激磁状态。
2. 若单轴已经处于激磁状态，呼叫此函数会成功回传。
3. 若单轴处于错误状态，呼叫此函数会回传错误代码。此时，必须先解除错误状况后，呼叫 NMC_AxisResetState()后，方可呼叫此函数让单轴重新进入激磁状态。

- **范例:**

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

- **参阅:**

NMC_AxisDisable()

NMC_AxisGetState()

NMC_AxisResetState()

3.3.2.2. NMC_AxisDisable

单轴停用

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T AxisIndex: 单轴识别号

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

1. 单轴在激磁状态下，可以呼叫此函数解除激磁状态。
2. 单轴在运动状态中，也可以呼叫此函数，立即中止目前正在进行的运动，并解除激磁状态。此时，若有运动储存在 Motion Queue 中，将会被清除。

- **范例:**

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

- **参阅:**

NMC_AxisEnable()

NMC_AxisGetState()

3.3.2.3. NMC_AxisGetStatus

读取单轴运动信息

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T AxisIndex: 单轴识别号

I32_T *PRetAxisStatus: 回传单轴运动信息(Status of axis)

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

- **范例:**

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

- **参阅:**

NMC_AxisGetState()

3.3.2.4. NMC_AxisGetState

读取单轴状态

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T AxisIndex: 单轴识别号

I32_T *PRetAxisState: 回传单轴状态

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

1. 此函数可以在周期性的处理程序中呼叫，以确认单轴目前的状态。
2. 单轴在进行运动或是 Homing 程序前，必须先处于激磁状态，当呼叫 NMC_AxisEnable()后，可以透过呼叫 NMC_AxisGetState()确认单轴是否已进入激磁状态。
3. 当单轴处于错误状态或非正常停止状态，呼叫 NMC_AxisResetState()后，可以透过呼叫 NMC_AxisGetState()确认单轴是否已解除错误状态。

- **范例:**

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

- **参阅:**

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

3.3.2.5. NMC_AxisResetState

清除单轴错误状态

- **C/C++语法:**

RTN_ERR NMC_AxisResetState(I32_T DevID, I32_T AxisIndex);

- **参数:**

I32_T DevID: 装置识别号

I32_T AxisIndex: 单轴识别号

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

1. 当单轴处于错误状态，或因启动非正常停止运动且已停止状态下，可以呼叫此函数将单轴回复到非激磁或激磁等正常状态，使用者可以呼叫 NMC_AxisGetState()确认单轴是否已回复到正常状态。
2. 呼叫此函数时，若单轴驱动器处于报警状态(Alarm)，控制器会自动清除驱动器错误，当错误清除后，会将单轴状态从错误状态(AXIS_STATE_ERROR)回复到非激磁状态(AXIS_STATE_DISABLE)。
3. 若单轴因启动非正常停止运动且已停止状态下，呼叫此函数可将单轴回复到激磁状态(AXIS_STATE_STAND_STILL)。

- **范例:**

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

- **参阅:**

1. NMC_AxisGetState()
2. NMC_AxisResetDriveAlm()

3.3.2.6. NMC_AxisResetDriveAlm

清除单轴伺服警报

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T AxisIndex: 单轴识别号

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

1. 当单轴驱动器发生报警(Alarm)，通常驱动器会自动解除激磁，使用者可以在排除错误状况后，呼叫此函数清除驱动器报警，并呼叫 NMC_AxisGetStatus()读取单轴 Status，确认驱动器报警是否已经清除成功。
2. 并非所有驱动器报警都可以透过此函数加以清除，某些驱动器报警必须利用重新上电的方式加以清除。
3. 呼叫此函数成功清除驱动器报警后，单轴状态并不会自动从错误状态中回复，必须接着呼叫 NMC_AxisResetState()将单轴回复到非激磁状态(AXIS_STATE_DISABLE)。

- **范例:**

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

- **参阅:**

1. NMC_AxisGetStatus()
2. NMC_AxisResetState()

3.3.3. 单轴运动信息读取相关函式

3.3.3.1. NMC_AxisGetCommandPos

3.3.3.2. NMC_AxisGetActualPos

- **C/C++语法:**

```
TN_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 );
```

- **参数:**

I32_T DevID: 装置识别号

I32_T AxisIndex: 单轴识别号

- **回传值:**

1. 回传错误代码， 若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”（0），反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。
2. F64_T *PRetCmdPos: 回传命令位置，单位为 user unit
3. F64_T *PRetActPos: 回传编码器位置，单位为 user unit。

- **用法:**

呼叫此函数可以读取单轴的命令位置与编码器的回授位置。

- **范例:**

```
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 );
```

- **参阅:**

3.3.3.3. NMC_AxisGetCommandVel

3.3.3.4. NMC_AxisGetActualVel

- **C/C++语法:**

```
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 );
```

- **参数:**

I32_T DevID: 装置识别号

I32_T AxisIndex: 单轴识别号

- **回传值:**

1. 回传错误代码， 若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”（0），反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。
2. F64_T *PRetCmdVel: 回传单轴命令速度，单位为 user unit/sec。
3. F64_T *PRetActVel: 回传单轴编码器回授速度，单位为 user unit/sec。

- **用法:**

呼叫相关函数以取得单轴命令速度与编码器回授速度。

- **范例:**

```
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 );
```

- **参阅:**

3.3.3.5. NMC_AxisGetMotionBuffSpace

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T AxisIndex: 单轴识别号

- **回传值:**

1. 回传错误代码， 若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”（0），反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。
2. I32_T *PRetFreeSpace: 回传单轴 Motion Queue 中尚可以储存的运动数量。

- **用法:**

由于单轴提供 Motion Queue 的功能，用户可以呼叫此函数确认尚可存入 Motion Queue 的运动数量。

- **范例:**

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

- **参阅:**

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

3.3.4. 单轴运动控制相关函式

3.3.4.1. NMC_AxisHomeDrive

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T AxisIndex: 单轴识别号

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

1. 在进行单轴运动前，用户通常会呼叫此函数启动单轴 Homing 程序，让单轴回到定义的原点位置。
2. 与 Homing 相关的参数定义于单轴参数表中，包含：Homing method、加速度、速度与 Offset 等。其中，有关 Homing 程序的方法，必须视驱动器提供的方法而定。
3. 当单轴在进行 Homing 程序的过程中发生错误，单轴 Status 的 Bit 7 会变为 1，且单轴的状态会切换到错误状态(AXIS_STATE_ERROR)，此时，当单轴完全停止后(可读取单轴 Status 确认 Bit 9 是否为 1)，可以呼叫 NMC_AxisResetDriveAlm()或是 NMC_AxisResetState()以回复到正常状态。
4. 当单轴顺利完成 Homing 程序，单轴 Status 的 Bit 18 会变为 1，且单轴的状态回切换到正常激磁状态(AXIS_STATE_STAND_STILL)。

- **范例:**

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

- **参阅:**

3.3.4.2. NMC_AxisPtp

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T AxisIndex: 单轴识别号

F64_T TargetPos: 目标位置(单位: user unit)，本设定值将根据轴参数 0x30 (Absolute or relative programming)解释为绝对位置或相对距离。

F64_T *PMaxVel: 利用指针的方式输入目标速度，或直接输入 0，代表不输入目标速度，此时，控制器会以单轴参数 AXP_VM 的速度设定值进行运动规划。

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

1. 单轴必须处于激磁状态下，方可呼叫此函数进行点对点运动。
2. 单轴正在进行 Homing 程序时，呼叫此函数将回传错误码。
3. 当单轴处于 AXIS_STATE_STOPPING、AXIS_STATE_STOPPED 与 AXIS_STATE_ERROR 等状态下，呼叫此函数将回传错误码。使用者必须呼叫 NMC_AxisResetState()将单轴回复到正常激磁状态(AXIS_STATE_STAND_STILL)后，才可顺利呼叫此函数启动点对点运动。
4. 若单轴正在进行其他运动，呼叫此函数后，会依据单轴参数 AXP_BUFF_PARAM 的设定，而有相对应的行为。
5. 呼叫此函数可启动单轴进行点对点运动，运动结束后，单轴将运动到输入的目标位置。若单轴参数 0x30 (Absolute or relative programming)设定为 1，则输入此函数的目标位置为与目前位置的相对距离；若设定为 0，则输入此函数的目标位置为绝对坐标位置。
6. 当单轴在进行点对点运动时，单轴状态会切换到 AXIS_STATE_DISCRETE_MOTION；到达目标位置后，若没有接续新的运动，则单轴 Status 的 Bit 9 会变为 1，状态会回复到正常激磁状态(AXIS_STATE_STAND_STILL)。
7. 控制器会依照单轴参数中的 AXP_PROF_TYPE、AXP_ACC、AXP_DEC 与 AXP_JERK 进行速度曲线规划。
8. 用户可以透过 PMaxVel 利用指针方式输入最大速度，此时，相对应的单轴参数 AXP_VM 会自动更改为输入的最大速度，并依此进行速度规划。

9. 若输入的指标 PMaxVel 为 0，控制器将径行依照单轴参数 AXP_VM 做为目标速度进行速度规划。
10. 若单轴状态处于 AXIS_STATE_STAND_STILL，则不管单轴参数 AXP_BUFF_PARAM 为何，呼叫此函数后都将立即启动。
11. 若单轴状态处于 AXIS_STATE_WAIT_SYNC，若单轴参数 AXP_BUFF_PARAM 为 Aborting，呼叫此函数后，原先储存在 Motion Queue 的运动将被清除，而此次呼叫的点对点运动将被储存到 Motion Queue，等待 Trigger 讯号的启动。
12. 在进行点对点运动的过程中，在尚未到达目标位置前，可以呼叫 NMC_AxisHalt()以停止运动。

- **范例:**

```
RTN_ERR  ret = 0;
ret = NMC_AxisPtp( 0, 0, 100, 0 ) // 依照单轴参数AXP_VM作为目标速度进行速度规划
```

- **参阅:**

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

3.3.4.3. NMC_AxisJog

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T AxisIndex: 单轴识别号

I32_T Dir: 目标速度方向，1 代表往正方向；-1 代表往负方向。

F64_T *PMaxVel: 利用指针的方式输入目标速率，或直接输入 0，代表不输入目标速率，此时，控制器会以单轴参数中 AXP_VM 的设定值做为目标速率。

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

1. 单轴必须处于激磁状态下，方可呼叫此函数进行点对点运动。
2. 单轴正在进行 Homing 程序时，呼叫此函数将回传错误码。
3. 当单轴处于 AXIS_STATE_STOPPING、AXIS_STATE_STOPPED 与 AXIS_STATE_ERROR 等状态下，呼叫此函数将回传错误码。使用者必须呼叫 NMC_AxisResetState()将单轴回复到正常激磁状态(AXIS_STATE_STAND_STILL)后，才可顺利呼叫此函数启动点对点运动。
4. 呼叫此函数后，控制器将依据单轴参数 AXP_ACC 设定的加速度，藉以加速或减速到目标速度，单轴状态会切换到 AXIS_STATE_CONTINUOUS_MOTION。
5. 利用指针 PMaxVel 输入的目标速率会连带修改单轴参数 AXP_VM。输入的目标速率可以为 0。
6. 呼叫此函数后，当到达目标速度，此时单轴 Status 的 Bit 8 会变为 1，且以目标速度持续运动。
7. 若单轴正在进行其他运动，呼叫此函数后，会依据单轴参数 AXP_BUFF_PARAM 的设定，而有相对应的行为。
8. 呼叫此函数后，可以呼叫 NMC_AxisHalt()停止单轴运动。

- **范例:**

```
RTN_ERR  ret = 0;
F64_T    maxVel = 100;
ret = NMC_AxisJog( 0, 0, -1, &maxVel ); // 设定单轴往负方向运动；单轴参数AXP_VM
                                        修改为100，且作为目标速率进行速度规划。
```

- **参阅:**

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

3.3.5. 单轴运动中止函式

3.3.5.1. NMC_AxisHalt

- **C/C++语法:**

RTN_ERR NMC_AxisHalt(I32_T DevID, I32_T AxisIndex);

- **参数:**

I32_T DevID: 装置识别号

I32_T AxisIndex: 单轴识别号

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

1. 当单轴在进行点对点运动或是速度运动时，可以呼叫此函数停止单轴运动，控制器会依据单轴参数 AXP_DEC 降速到 AXP_V_BASE 设定的速率。
2. 单轴正在进行 Homing 程序时，呼叫此函数将回传错误码。
3. 当单轴处于 AXIS_STATE_STOPPING、AXIS_STATE_STOPPED 与 AXIS_STATE_ERROR 等状态下，呼叫此函数将回传错误码。
4. 当单轴处于 AXIS_STATE_DISABLE 或是 AXIS_STATE_STAND_STILL 等状态，呼叫此函数不会回传错误码。
5. 若单轴参数 AXP_BUFF_PARAM 设定为 Aborting，呼叫此函数后将立即启动正常停止运动；若 AXP_BUFF_PARAM 设定为 Buffered，则会等单轴 Status Bit 8 变为 1 后(前一个运动完成目标后)，才会启动正常停止运动。
6. 呼叫此函数且成功启动正常停止运动后，单轴状态会切换到 AXIS_STATE_DISCRETE_MOTION，当单轴停止运动后，单轴 Status Bit 8、9 变为 1，且状态将切换到正常激磁状态 AXIS_STATE_STAND_STILL。

- **范例:**

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

- **参阅:**

3.3.5.2. NMC_AxisStop

- **C/C++语法:**

RTN_ERR NMC_AxisStop(I32_T DevID, I32_T AxisIndex);

- **参数:**

I32_T DevID: 装置识别号

I32_T AxisIndex: 单轴识别号

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

1. 单轴不管在进行任何运动时，当有非预期状况发生，需要停止目前正在进行的运动，都可以呼叫此函数启动单轴进行非正常停止运动，控制器会依据单轴参数 AXP_STOP_PROF_DEC 降速到 AXP_V_BASE 设定的速率。
2. 单轴正在进行 homing 程序时，也可以呼叫此函数终止 homing 程序。
3. 呼叫此函数启动非正常停止运动后，若单轴尚未停止，此时状态会切换到 AXIS_STATE_STOPPING；停止后，单轴 Status Bit 9 变为 1，状态会切换到 AXIS_STATE_STOPPED。
4. 当单轴状态处于 AXIS_STATE_STAND_STILL，呼叫此函数后，单轴状态将切换到 AXIS_STATE_STOPPED。
5. 呼叫此函数且单轴停止运动后，不允许再启动单轴运动，使用者必须呼叫 NMC_AxisResetState()将单轴回复到正常激磁状态 AXIS_STATE_STAND_STILL 后，方可启动单轴运动。

- **范例:**

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

- **参阅:**

NMC_AxisResetState()

3.3.5.3. NMC_AxisHaltAll

- **C/C++语法:**

RTN_ERR NMC_AxisHaltAll(I32_T DevID);

- **参数:**

I32_T DevID: 装置识别号

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”（0），反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

与 NMC_AxisHalt()行为一致，呼叫此函数可以正常停止指定 Device 所有单轴之运动。

- **范例:**

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

- **参阅:**

NMC_AxisHalt()

3.3.5.4. NMC_AxisStopAll

- **C/C++语法:**

RTN_ERR NMC_AxisStopAll(I32_T DevID);

- **参数:**

I32_T DevID: 装置识别号

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”（0），反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

与 NMC_AxisStop()行为一致，呼叫此函数可以启动指定 Device 所有单轴进行非正常停止运动。

- **范例:**

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

- **参阅:**

NMC_AxisStop()

3.3.6. 单轴运行中变动相关函式

3.3.6.1. NMC_AxisVelOverride

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T AxisIndex: 单轴识别号

F64_T TargetVel: 目标速度(单位: user unit / sec)

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

1. 只有单轴启动点对点运动(NMC_AxisPtp)或是速度运动(NMC_AxisJog)后，才可以呼叫此函数改变目标速度。
2. 呼叫此函数输入的目标速度不会改变单轴参数 AXP_VM，只对于目前正在进行的运动有效。
3. 若启动的是速度运动，则输入的目标速度一定可以达到，此时单轴 Status 的 Bit 8 会变为 1。
4. 若启动的是点对点运动，则输入的目标速度不一定可以达到，在满足用户设定的目标位置、加减速度下，控制器会依据输入的目标速度自行规画可行的速度曲线。

- **范例:**

```
RTN_ERR  ret = 0;
F64_T    maxVel = 100.0;
ret = NMC_AxisJog( 0, 0, 1, &maxVel );  // 启动单轴进行速度运动，目标速度为100。
Sleep(100);
ret = NMC_AxisVelOverride( 0, 0, 50.0 );  // 将目标速度调降为50。
```

- **参阅:**

1. NMC_AxisPtp()
2. NMC_AxisJog()

3.3.6.2. NMC_AxisAccOverride

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T AxisIndex: 单轴识别号

F64_T TargetAcc: 目标加速度

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

1. 只有单轴启动点对点运动(NMC_AxisPtp)或是速度运动(NMC_AxisJog)后，才可以呼叫此函数改变加速度。
2. 输入的加速度将会设定到单轴参数 AXP_ACC。
3. 若单轴已经到达目标速度(单轴 Status 的 Bit 12 变为 1)，则输入的加速度对于目前的运动将不具影响。

- **范例:**

```
RTN_ERR  ret = 0;
I32_T    status = 0;
F64_T    maxVel = 50.0;
ret = NMC_AxisJog( 0, 0, 1, &maxVel );  // 启动单轴进行速度运动，目标速度为50。
Sleep(100);
ret = NMC_AxisGetStatus( 0, 0, &status );
if( !( status & 0x1000 ) )
{
    ret = NMC_AxisAccOverride ( 0, 0, 100.0 );  // 改变加速度为100.0
}
```

- **参阅:**

3.3.6.3. NMC_AxisDecOverride

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T AxisIndex: 单轴识别号

F64_T TargetDec: 目标减加速度

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

1. 只有单轴启动点对点运动(NMC_AxisPtp)或是正常停止运动(NMC_AxisHalt)后，才可以呼叫此函数改变减加速度。
2. 输入的减加速度将会设定到单轴参数 AXP_DEC。
3. 若单轴启动点对点运动，且已从目标速度逐渐进入设定末速(单轴Status的Bit 11变为1)，则输入的减加速度对于目前的运动将不具影响。
4. 若单轴启动正常停止运动 NMC_AxisHalt，在单轴还没停止前，输入的减加速度会影响目前运动。

- **范例:**

```
RTN_ERR  ret = 0;
I32_T    status = 0;
F64_T    maxVel = 50.0;
ret = NMC_AxisPtp( 0, 0, 100, 0 );  // 启动单轴进行点对点运动，目标速度依据AXP_VM的设定值。
ret = NMC_AxisDecOverride ( 0, 0, 100.0 );  // 改变减加速度为100.0
```

- **参阅:**

3.3.7. 单轴总体速率设定相关函式

3.3.7.1. NMC_AxisSetVelRatio

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T AxisIndex: 单轴识别号

F64_T Percentage: 输入速度比例

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

1. 不管单轴是否有在进行运动，皆可呼叫此函数设定速度比例，此速度比例的 default 设定值为 1.0。
2. 输入的速度比例必须满足大于等于 0，小于等于 1。
3. 当单轴处于 AXIS_STATE_STOPPING、AXIS_STATE_STOPPED 与 AXIS_STATE_ERROR 等状态下，呼叫此函数将回传错误码。
4. 当单轴正在进行 homing 时，呼叫此函数将回传错误码。
5. 当呼叫此函数设定速度比例后，尔后启动的单轴运动，输入的速度最大值将会乘上此速度比例做为目标速度。
6. 若呼叫此函数时，单轴正在进行点对点运动或是速度运动，输入的速度比例会改变目前运动的目标速度。

- **范例:**

```
RTN_ERR  ret = 0;
F64_T    maxVel = 50.0;
ret = NMC_AxisJog( 0, 0, 1, &maxVel );  // 启动单轴进行速度运动，目标速度为50。
ret = NMC_AxisSetVelRatio( 0, 0, 0.5 ); // 改变速度比例，目标速度变为25。
```

- **参阅:**

3.3.7.2. NMC_AxisGetVelRatio

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T AxisIndex: 单轴识别号

- **回传值:**

1. 回传错误代码， 若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”（0），反之函数呼叫失败
   回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。
2. F64_T *PPercentage: 回传速度比例

- **用法:**

  呼叫此函数读取目前设定的速度比例。

- **范例:**

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

- **参阅:**

3.4. 群组相关 APIs

3.4.1. 群组参数设定相关函式

3.4.1.1. NMC_GroupSetParamI32

设定群组参数（I32_T 数据型态）

- **C/C++语法：**

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

- **参数：**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

I32_T ParamNum: 参数的编号

I32_T SubIndex: 参数的次编号

I32_T ParaValueI32: 设定的数值

- **回传值：**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”（0），反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例：**

```
I32_T   devID        = 0;
I32_T   groupIndex   = 0;
I32_T   paramNum     = 0x30; //命令为绝对式或增量式的参数
I32_T   subIndex     = 0;
I32_T   paraValueI32 = 1;    //命令设定为增量式
RTN_ERR ret          = 0;

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

- **参阅：**

<无>

3.4.1.2. NMC_GroupGetParamI32

读取群组参数（I32_T 数据型态）

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

I32_T ParamNum: 参数的编号

I32_T SubIndex: 参数的次编号

I32_T *PRetParaValueI32: 输入指针变量，回传参数的数值

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”（0），反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
I32_T   devID        = 0;
I32_T   groupIndex   = 0;
I32_T   paramNum     = 0x30; //读取命令为绝对式或增量式的参数
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;
```

- **参阅:**

<无>

3.4.1.3. NMC_GroupSetParamF64

设定群组参数（F64_T 数据型态）

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

I32_T ParamNum: 参数的编号

I32_T SubIndex: 参数的次编号

F64_T ParaValueF64: 设定的数值

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”（0），反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
I32_T   devID        = 0;
I32_T   groupIndex   = 0;
I32_T   paramNum     = 0x21; //设定停止运动的减速度
I32_T   subIndex     = 0;
F64_T   paraValueF64 = 80.5; //停止运动的减速度的数值
RTN_ERR ret          = 0;

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

- **参阅:**

<无>

3.4.1.4. NMC_GroupGetParamF64

设定群组参数（F64_T 数据型态）

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

I32_T ParamNum: 参数的编号

I32_T SubIndex: 参数的次编号

F64_T *PRetParaValueF64: 输入指针变量，回传参数的数值

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”（0），反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
I32_T   devID        = 0;
I32_T   groupIndex   = 0;
I32_T   paramNum     = 0x21; //设定停止的减速度
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;
```

- **参阅:**

<无>

3.4.1.5. NMC_GroupAxSetParamI32

设定群组轴参数(I32_T 数据型态)

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

I32_T GroupAxisIndex: 群组轴识别号

I32_T ParamNum: 参数的编号

I32_T SubIndex: 参数的次编号

I32_T ParaValueI32: 设定的数值

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
I32_T   devID          = 0;
I32_T   groupIndex     = 0;
I32_T   groupAxisIndex = 2;
I32_T   paramNum       = 0x30; //命令为绝对式或增量式的参数
I32_T   subIndex       = 0;
I32_T   paraValueI32   = 1;    //命令设定为增量式
RTN_ERR ret            = 0;

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

- **参阅:**

<无>

3.4.1.6. NMC_GroupAxGtParamI32

设定群组轴参数(I32_T 数据型态)

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

I32_T GroupAxisIndex: 群组轴识别号

I32_T ParamNum: 参数的编号

I32_T SubIndex: 参数的次编号

I32_T *PRetParaValueI32: 输入指针变量，回传参数的数值

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”（0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
I32_T   devID          = 0;
I32_T   groupIndex     = 0;
I32_T   groupAxisIndex = 2
I32_T   paramNum       = 0x30; //命令为绝对式或增量式的参数
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;
```

- **参阅:**

<无>

3.4.1.7. NMC_GroupAxSetParamF64

设定群组轴参数(F64_T 数据型态)

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

I32_T GroupAxisIndex: 群组轴识别号

I32_T ParamNum: 参数的编号

I32_T SubIndex: 参数的次编号

F64_T ParaValueF64: 设定的数值

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”（0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
I32_T   devID          = 0;
I32_T   groupIndex     = 0;
I32_T   groupAxisIndex = 2;
I32_T   paramNum       = 0x21; //设定停止运动的减速度
I32_T   subIndex       = 0;
F64_T   paraValueF64   = 80.5; //停止运动的减速度的数值
RTN_ERR ret            = 0;

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

- **参阅:**

<无>

3.4.1.8. NMC_GroupAxGetParamF64

设定群组轴参数(F64_T 数据型态)

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

I32_T GroupAxisIndex: 群组轴识别号

I32_T ParamNum: 参数的编号

I32_T SubIndex: 参数的次编号

F64_T *PRetParaValueF64: 设定的数值

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”（0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
I32_T   devID          = 0;
I32_T   groupIndex     = 0;
I32_T   groupAxisIndex = 2;
I32_T   paramNum       = 0x21; //设定停止运动的减速度
I32_T   subIndex       = 0;
F64_T   paraValueF64   = 0;    //停止运动的减速度的数值
RTN_ERR ret            = 0;

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

- **参阅:**

<无>

3.4.2.1. NMC_GroupEnable

启用(Servo On)群组中所有的群组轴，若所有的群组轴成功启用，则群组的状态(state)将由 GROUP_DISABLE 切换为 GROUP_ENABLE。

- **C/C++语法:**

RTN_ERR NMC_GroupEnable(I32_T DevID, I32_T GroupIndex);

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号。

- **回传值:**

以RTN_ERR数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

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

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

- **参阅:**

<无>

3.4.2.2. NMC_GroupDisable

停用(Servo Off)群组中所有的群组轴，若所有的群组轴成功停用，则群组的状态(state)切换为GROUP_DISABLE。

- **C/C++语法:**

RTN_ERR NMC_GroupDisable(I32_T DevID, I32_T GroupIndex);

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号。

- **回传值:**

以RTN_ERR数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于NexMotionError.h头文件中。

- **范例:**

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

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

- **参阅:**

<无>

3.4.2.3. NMC_GroupGetStatus

读取群组的以 bit 为单位的状态(status)，各 bit 所代表的意义如下表，若读取到的 bit 数值为 1，则触发此事件。

bit	说明
0	触发外部紧急开关
1	驱动器有警报
2	超出硬件的正极限
3	超出硬件的负极限
4	超出软件的正极限
5	超出软件的负极限
6	所有群组轴皆启用中(即状态(state)为 GROUP_STAND_STILL)
7	任一群组轴有发生错误(即状态(state)为 GROUP_ERROR_STOP)
9	所有的群组轴为没有位置的变动
10	卡式坐标运动(直线与圆弧)中，正在加速(或减速)至最高速度的阶段。 (PTP 或 JOG 运动时，此 bit 为 0)
11	卡式坐标运动(直线与圆弧)中，正在减速至目标位置或停止的阶段。 (PTP 或 JOG 运动时，此 bit 为 0)
12	卡式坐标运动(直线与圆弧)中，正在最高速度的阶段。 (PTP 或 JOG 运动时，此 bit 为 0)
13	群组正在运动中(即状态(state)为 GROUP_MOVING、GROUP_HOMING 或 GROUP_STOPPING)
14	群组已停止下来(即状态(state)为 GROUP_STOPPED)

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号。

I32_T *PRetStatusInBit: 输入指针变量，回传以 bit 为单位的状态。

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
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;
```

- **参阅:**

<无>

3.4.2.4. NMC_GroupGetState

读取群组的状态(state)，详细请参考下表。

数值	状态(state)	说明
0	GROUP_DISABLE	有任一群组轴为停用状态
1	GROUP_STAND_STILL	所有群组轴为启用状态
2	GROUP_STOPPED	下达 NMC_GroupStop()后，群组处于停止阶段
3	GROUP_STOPPING	下达 NMC_GroupStop()后，群组处于正在停止阶段
4	GROUP_MOVING	群组运动中
5	GROUP_HOMING	群组归零中
6	GROUP_ERROR_STOP	任一群组轴有发生错误

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

I32_T *PRetState: 输入指针变量，回传群组的状态。

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
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;
```

- **参阅:**

<无>

3.4.2.5. NMC_GroupResetState

重置群组的状态(state)，可重置群组状态 GROUP_STOPPED 成状态 GROUP_STAND_STILL；若群组状态为 GROUP_ERROR_STOP，此时下达此 API 会自动清除驱动器的警报，所有群组轴皆无警报后，会切换为 GROUP_STAND_STILL。

- **C/C++语法:**

RTN_ERR NMC_GroupResetState(I32_T DevID, I32_T GroupIndex);

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号。

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

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

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

- **参阅:**

<无>

3.4.2.6. NMC_GroupResetDriveAlm

清除指定的群组轴的驱动器的警报。

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

I32_T GroupAxisIndex: 欲清除警报的群组轴识别号

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
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;
```

- **参阅:**

<无>

3.4.2.7. NMC_GroupResetDriveAlmAll

清除所有群组轴的驱动器的警报。

- **C/C++语法:**

RTN_ERR NMC_GroupResetDriveAlmAll(I32_T DevID, I32_T GroupIndex);

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

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

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

- **参阅:**

<无>

3.4.3. 群组总体速率设定相关函式

3.4.3.1. NMC_GroupSetVelRatio

设定群组的运动速率百分比，范围为 0.0 ~ 100.0%。

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

F64_T Percentage: 欲设定的运动速率百分比(0% ~ 100.0%)

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
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;
```

- **参阅:**

<无>

3.4.3.2. NMC_GroupGetVelRatio

读取群组的运动速率百分比，范围为 0.0 ~ 100.0%。

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

F64_T *PRetPercentage: 输入指针变量，回传运动速率百分比

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
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;
```

- **参阅:**

<无>

3.4.4. 群组点对点运动(轴坐标系)相关函式

3.4.4.1. NMC_GroupPtpAcs

执行群组的单群组轴的点对点运动，其输入点位的坐标系统为 ACS(Axis coordinate system)，默认的最高速度将参考参数中的数值,若需要,可在此 API 中设定,设定后将自动储存于参数中。

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

I32_T GroupAxisIndex: 欲执行运动的群组轴识别号

F64_T AcsPos: 欲执行的点位位置值

_opt_null_ const F64_T *PAcsMaxVel: 输入指针变量，可指定最高速度，输入 NULL (0)则忽略此参数

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
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;
```

- **参阅:**

<无>

3.4.4.2. NMC_GroupPtpAcsAll

执行群组的多群组轴的点对点运动，其输入点位的坐标系统为 ACS(Axis coordinate system)，指定运动的群组轴将同时启动同时到达。

- **C/C++语法:**

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

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

I32_T GroupAxesIdxMask: 欲执行运动的群组轴组合码，请参考下表。

Pos_T *PAcsPos: 输入指针变量，为目标坐标位置

群组轴	8	7	6	5	4	3	2	1
位数	7	6	5	4	3	2	1	0
数值（次方为位数）	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

群组轴组合码(GroupAxesIdxMask)的使用方式如下述：

若移动的群组轴有第 1 轴、第 3 轴与第 8 轴，则群组轴组合码为 $2^0 + 2^2 + 2^7 = 133$。

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
I32_T   devID           = 0;
I32_T   groupIndex      = 0;
I32_T   groupAxesIdxMask = 133; //移动第1、3与8轴
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;
```

- **参阅：**

<无>

3.4.5. 群组轴 Jog 运动(轴坐标系)相关函式

3.4.5.1. NMC_GroupJogAcs

执行群组的单群组轴的 Jog 运动，默认的最高速度将参考参数中的数值，若需要，可在此 API 中设定，设定后将自动储存于参数中。

- **C/C++语法:**

RTN_ERR NMC_GroupJogAcs(I32_T DevID, I32_T GroupIndex, I32_T GroupAxisIndex, I32_T Dir, _opt_null_ const F64_T *PAcsMaxVel);

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

I32_T GroupAxisIndex: 群组轴识别号

I32_T Dir: 旋转方向

_opt_null_ const F64_T *PAcsMaxVel: 输入指针变量，可指定最高速度，输入 NULL (0)则忽略此参数

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
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;
```

- **参阅:**

<无>

3.4.6. 群组点对点运动(卡氏坐标)相关函式

3.4.6.1. NMC_GroupPtpCart

执行群组的单群组轴的点对点运动，其点位单位为卡氏空间坐标值。

- **C/C++语法:**

RTN_ERR NMC_GroupPtpCart(I32_T DevID, I32_T GroupIndex, I32_T CartAxis, F64_T CartPos);

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

I32_T CartAxis: 欲执行运动的卡氏空间坐标轴的代号，请参考下表

F64_T CartPos: 欲执行的点位位置值

卡氏空间坐标	V	U	C	B	A	Z	Y	X
数值	7	6	5	4	3	2	1	0

卡氏空间坐标轴的代号(CartAxis)的使用方式如下述:

若移动的卡氏空间坐标为 Z 轴，则卡氏空间坐标轴的代号为 2。

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
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;
```

- **参阅:**

<无>

3.4.6.2. NMC_GroupPtpCartAll

执行群组的多群组轴的点对点运动，其点位单位为卡氏空间坐标值。

- **C/C++语法:**

RTN_ERR NMC_GroupPtpCartAll(I32_T DevID, I32_T GroupIndex, I32_T CartAxesMask, const Pos_T *PTargetPos);

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

I32_T CartAxesMask: 欲执行运动的卡氏空间坐标轴的组合码，请参考下表

const Pos_T *PTargetPos: 输入指针变量，为目标坐标位置

卡氏空间坐标	V	U	C	B	A	Z	Y	X
位数	7	6	5	4	3	2	1	0
数值（次方为位数）	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

卡氏空间坐标轴的组合码(CartAxesMask)的使用方式如下述:

若移动的卡氏空间坐标为 X，Z 与 A 轴，则卡氏空间坐标轴的代号为 $2^0 + 2^2 + 2^3 = 13$。

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
I32_T   devID        = 0;
I32_T   groupIndex   = 0;
I32_T   cartAxesMask = 13; //移动X、Z与A轴
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;
```

- **参阅：**

<无>

3.4.7.1. NMC_GroupJogCartFrame

启动群组轴卡式坐标之速度运动

- **C/C++语法:**

RTN_ERR NMC_GroupJogCartFrame(I32_T DevID, I32_T GroupIndex, I32_T CartAxis, I32_T Dir, _opt_null_ const F64_T *PMaxVel);

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

I32_T CartAxis: 欲执行运动的卡氏空间坐标轴的代号，请参考下表

卡氏空间坐标	V	U	C	B	A	Z	Y	X
数值	7	6	5	4	3	2	1	0

卡氏空间坐标轴的代号(CartAxis)的使用方式如下述:

若移动的卡氏空间坐标为 Z 轴，则卡氏空间坐标轴的代号为 2。

I32_T Dir: 移动方向，0:正向，1:负向

_opt_null_ const F64_T *PMaxVel: 输入指针变量，指定 Jog 最高速度，输入 NULL (0)则忽略此参数

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
```

- **参阅:**

<无>

3.4.8.1. NMC_GroupHalt

执行群组的停止运动，此 API 会将任何的运动立即减速，而群组的状态(state)将会由 GROUP_MOVING 变成 GROUP_STOPPING(减速中)，若速度减至零，则状态(state)变为 GROUP_STAND_STILL。

此 API 需搭配 Buffer mode 来指定此停止运动的行为：若 Buffer mode 为 Aborting，则将目前的运动立即减速；若 Buffer mode 为 Buffered，则会将此 API 加入至 Motion Buffer 中，不会立即生效。

- **C/C++语法:**

RTN_ERR NMC_GroupHalt(I32_T DevID, I32_T GroupIndex);

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

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

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

- **参阅:**

<无>

3.4.8.2. NMC_GroupStop

执行群组的停止运动，此 API 会将任何的运动立即减速，而群组的状态(state)将会由 GROUP_MOVING 变成 GROUP_STOPPING(减速中)，若速度减至零，则状态(state)变为 GROUP_STOPPED。若需要执行新的运动命令时，则需使用 NMC_GroupResetState()来重置目前的 GROUP_STOPPED 状态。

- **C/C++语法:**

RTN_ERR NMC_GroupStop(I32_T DevID, I32_T GroupIndex);

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

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

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

- **参阅:**

<无>

3.4.8.3. NMC_GroupHaltAll

执行所有群组的停止运动，此 API 会将任何的运动立即减速，而群组的状态(state)将会由 GROUP_MOVING 变成 GROUP_STOPPING(减速中)，若速度减至零，则状态(state)变为 GROUP_STAND_STILL。

此 API 需搭配 Buffer mode 来指定此停止运动的行为：若 Buffer mode 为 Aborting，则将目前的运动立即减速；若 Buffer mode 为 Buffered，则会将此 API 加入至 Motion Buffer 中，不会立即生效。

- **C/C++语法:**

RTN_ERR NMC_GroupHaltAll(I32_T DevID);

- **参数:**

I32_T DevID: 装置识别号

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

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

ret = NMC_GroupHaltAll( devID );
if( ret != 0 ) return ret;
```

- **参阅:**

<无>

3.4.8.4. NMC_GroupStopAll

执行所有群组的停止运动，此 API 会将任何的运动立即减速，而群组的状态(state)将会由 GROUP_MOVING 变成 GROUP_STOPPING(减速中)，若速度减至零，则状态(state)变为 GROUP_STOPPED。若需要执行新的运动命令时，则需使用 NMC_GroupResetState()来重置目前的 GROUP_STOPPED 状态。

- **C/C++语法:**

RTN_ERR NMC_GroupStopAll(I32_T DevID);

- **参数:**

I32_T DevID: 装置识别号

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:
若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

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

ret = NMC_GroupStopAll( devID );
if( ret != 0 ) return ret;
```

- **参阅:**

<无>

3.4.9. 群组运动信息读取相关函式

3.4.9.1. NMC_GroupGetCommandPosAcs

读取群组的各群组轴的位置命令值，其坐标系统为 ACS(Axis coordinate system)。

- **C/C++语法:**

RTN_ERR NMC_GroupGetCommandPosAcs(I32_T DevID, I32_T GroupIndex, Pos_T *PRetCmdPosAcs);

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

Pos_T *PRetCmdPosAcs: 输入指针变量，回传各群组轴的位置命令值

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
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;
```

- **参阅:**

<无>

3.4.9.2. NMC_GroupGetActualPosAcs

读取群组的各群组轴的位置实际值，此实际值由编码器的 count 值经过齿轮比或导螺杆节距转换后得到，其坐标系统为 ACS(Axis coordinate system)。

- **C/C++语法：**

RTN_ERR NMC_GroupGetActualPosAcs(I32_T DevID, I32_T GroupIndex, Pos_T *PRetActPosAcs);

- **参数：**

I32_T DevID：装置识别号

I32_T GroupIndex：群组识别号

Pos_T *PRetActPosAcs：输入指针变量，回传各群组轴的位置实际值

- **回传值：**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下：

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例：**

```
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;
```

- **参阅：**

<无>

3.4.9.3. NMC_GroupGetCommandPosPcs

读取群组的工件坐标的位置命令值，其坐标系统为 PCS(Product coordinate system)。

- **C/C++语法:**

RTN_ERR NMC_GroupGetCommandPosPcs(I32_T DevID, I32_T GroupIndex, Pos_T *PRetCmdPosPcs);

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

Pos_T *PRetCmdPosPcs: 输入指针变量，回传各群组轴的位置命令值

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
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;
```

- **参阅:**

<无>

3.4.9.4. NMC_GroupGetActualPosPcs

读取群组的工件坐标的位置实际值，此实际值由编码器的 count 值经过齿轮比或导螺杆节距转换后得到，其坐标系统为 PCS(Product coordinate system)。

- **C/C++语法:**

RTN_ERR NMC_GroupGetActualPosPcs(I32_T DevID, I32_T GroupIndex, Pos_T *PRetActPosPcs);

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

Pos_T *PRetActPosPcs: 输入指针变量，回传各群组轴的位置命令值

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
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;
```

- **参阅:**

<无>

3.4.9.5. NMC_GroupGetCommandPos

读取群组的位置命令值，回传的数值由 CoordSys(坐标系统)指定。

- **C/C++语法:**

RTN_ERR NMC_GroupGetCommandPos(I32_T DevID, I32_T GroupIndex, I32_T CoordSys, Pos_T *PRetCmdPos);

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

I32_T CoordSys: 指定坐标系(0:MCS, 1:PCS, 2:ACS)

Pos_T *PRetCmdPosPcs: 输入指针变量，回传指定的坐标系统的位置命令值

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
I32_T   devID      = 0;
I32_T   groupIndex = 0;
I32_T   coordSys   = 2; //读取ACS坐标系统的位置命令值
Pos_T   cmdPos     = { 0 };
RTN_ERR ret        = 0;

ret = NMC_GroupGetCommandPos( devID, groupIndex, coordSys, &cmdPos );
if( ret != 0 ) return ret;
```

- **参阅:**

<无>

3.4.9.6. NMC_GroupGetActualPos

读取群组的位置实际值，回传的数值由 CoordSys(坐标系统)指定。

- **C/C++语法:**

RTN_ERR NMC_GroupGetActualPos(I32_T DevID, I32_T GroupIndex, I32_T CoordSys, Pos_T *PRetActPos);

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

I32_T CoordSys: 指定坐标系(0:MCS, 1:PCS, 2:ACS)

Pos_T *PRetActPos: 输入指针变量，回传指定的坐标系统的位置实际值

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
I32_T   devID      = 0;
I32_T   groupIndex = 0;
I32_T   coordSys   = 2; //读取ACS坐标系统的位置命令值
Pos_T   actPos     = { 0 };
RTN_ERR ret        = 0;

ret = NMC_GroupGetActualPos( devID, groupIndex, coordSys, &actPos );
if( ret != 0 ) return ret;
```

- **参阅:**

<无>

3.4.9.7. NMC_GroupGetMotionBuffSpace

读取单轴运动指令暂存空间大小

- **C/C++语法:**

RTN_ERR NMC_GroupGetMotionBuffSpace(I32_T DevID, I32_T GroupIndex, I32_T *PRetFreeSpace);

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

F64_T *PRetFreeSpace: 输入指针变量，回传 Motion Buffer 的剩余空间

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
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;
```

- **参阅:**

<无>

3.4.10.1. NMC_GroupSetHomePos

设定群组轴原点位置

- **C/C++语法:**

RTN_ERR NMC_GroupSetHomePos(I32_T DevID, I32_T GroupIndex, I32_T GroupAxesIdxMask, const Pos_T *PHomePosAcs);

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

I32_T GroupAxesIdxMask: 欲执行运动的群组轴组合码，请参考下表。

const Pos_T *PHomePosAcs: 输入指针变量，原点坐标(ACS)位置

群组轴	8	7	6	5	4	3	2	1
位数	7	6	5	4	3	2	1	0
数值（次方为位数）	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

群组轴组合码(GroupAxesIdxMask)的使用方式如下述:

若移动的群组轴有第 1 轴、第 3 轴与第 8 轴，则群组轴组合码为 $2^0 + 2^2 + 2^7 = 133$。

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS” (0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

- **参阅:**

<无>

3.4.10.2. NMC_GroupAxesHomeDrive

启动群组轴归原点运动(驱动器执行)

- **C/C++语法:**

RTN_ERR NMC_GroupAxesHomeDrive(I32_T DevID, I32_T GroupIndex, I32_T GroupAxesIdxMask);

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

I32_T GroupAxesIdxMask: 欲执行运动的群组轴组合码，请参考下表。

群组轴	8	7	6	5	4	3	2	1
位数	7	6	5	4	3	2	1	0
数值（次方为位数）	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

群组轴组合码(GroupAxesIdxMask)的使用方式如下述：

若移动的群组轴有第 1 轴、第 3 轴与第 8 轴，则群组轴组合码为 $2^0 + 2^2 + 2^7 = 133$。

- **回传值:**

回传错误代码。

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”（0），反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
```

- **参阅:**

<无>

3.4.11.1. NMC_GroupLineXY

执行 XY 平面上的直线运动，将从目前的卡氏空间坐标位置直线运动至目标坐标位置，而状态(state)将会由 GROUP_STAND_STILL 变成 GROUP_MOVING(移动中)，若到达目标位置即速度减至零，则状态(state)变为 GROUP_STAND_STILL。默认的最高速度将参考参数中的数值，若需要，可在此 API 中设定，设定后将自动储存于参数中。

- **C/C++语法:**

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);

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

_opt_null_ const F64_T *PX: 输入指针变量，设定 X 轴目标坐标值，输入 NULL(0)则忽略此参数(不移动)

_opt_null_ const F64_T *PY: 输入指针变量，设定 Y 轴目标坐标值，输入 NULL(0)则忽略此参数(不移动)

_opt_null_ const F64_T *PMaxVel: 输入指针变量，设定最高速度，输入 NULL(0)则忽略此参数

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
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;
```

- **参阅：**

<无>

3.4.11.2. NMC_GroupCirc2R

执行 XY 平面上的圆弧运动(半径法)，将从目前的卡氏空间坐标位置圆弧运动至目标坐标位置，而状态(state)将会由 GROUP_STAND_STILL 变成 GROUP_MOVING(移动中)，若到达目标位置即速度减至零，则状态(state)变为 GROUP_STAND_STILL。默认的最高速度将参考参数中的数值，若需要，可在此 API 中设定，设定后将自动储存于参数中。

- **C/C++语法:**

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);

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

_opt_null_ const F64_T *PEX: 输入指针变量，设定 X 轴目标坐标值，输入 NULL(0)则忽略此参数(不移动)

_opt_null_ const F64_T *PEY: 输入指针变量，设定 Y 轴目标坐标值，输入 NULL(0)则忽略此参数(不移动)

F64_T Radius: 圆弧的半径(负值代表选择弧度较大之路径)

I32_T CW_CCW: 圆弧的旋转方向(0=CW; 1=CCW)

_opt_null_ const F64_T *PMaxVel: 输入指针变量，设定最高速度，输入 NULL(0)则忽略此参数

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
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;
```

- **参阅:**

<无>

3.4.11.3. NMC_GroupCirc2C

执行 XY 平面上的圆弧运动(圆心法)，将从目前的卡氏空间坐标位置圆弧运动至目标坐标位置，而状态(state)将会由 GROUP_STAND_STILL 变成 GROUP_MOVING(移动中)，若到达目标位置即速度减至零，则状态(state)变为 GROUP_STAND_STILL。默认的最高速度将参考参数中的数值，若需要，可在此 API 中设定，设定后将自动储存于参数中。

- **C/C++语法:**

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);

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

_opt_null_ const F64_T *PEX: 输入指针变量，设定 X 轴目标坐标值，输入 NULL(0)则忽略此参数(不移动)

_opt_null_ const F64_T *PEY: 输入指针变量，设定 Y 轴目标坐标值，输入 NULL(0)则忽略此参数(不移动)

_opt_null_ const F64_T *PCXOffset:输入指针变量，设定 X 轴圆心坐标值，输入 NULL(0)则忽略此参数(不移动)

_opt_null_ const F64_T *PCYOffset:输入指针变量，设定 Y 轴圆心坐标值，输入 NULL(0)则忽略此参数(不移动)

I32_T CW_CCW: 圆弧的旋转方向(0=CW; 1=CCW)

_opt_null_ const F64_T *PMaxVel: 输入指针变量，设定最高速度，输入 NULL(0)则忽略此参数

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
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;
```

- **参阅:**

<无>

3.4.11.4. NMC_GroupCirc2B

执行 XY 平面上的圆弧运动(经过法)，将从目前的卡氏空间坐标位置圆弧运动至目标坐标位置，而状态(state)将会由 GROUP_STAND_STILL 变成 GROUP_MOVING(移动中)，若到达目标位置即速度减至零，则状态(state)变为 GROUP_STAND_STILL。默认的最高速度将参考参数中的数值，若需要，可在此 API 中设定，设定后将自动储存于参数中。

- **C/C++语法:**

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);

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

_opt_null_ const F64_T *PEX: 输入指针变量，设定 X 轴目标坐标值，输入 NULL(0)则忽略此参数(不移动)

_opt_null_ const F64_T *PEY: 输入指针变量，设定 Y 轴目标坐标值，输入 NULL(0)则忽略此参数(不移动)

_opt_null_ const F64_T *PBX: 输入指针变量，设定 X 轴经过坐标值，输入 NULL(0)则忽略此参数(不移动)

_opt_null_ const F64_T *PBY: 输入指针变量，设定 Y 轴经过坐标值，输入 NULL(0)则忽略此参数(不移动)

_opt_null_ const F64_T *PMaxVel: 输入指针变量，设定最高速度，输入 NULL(0)则忽略此参数

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
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;
```

- **参阅:**

<无>

3.4.12.1. NMC_GroupLine

执行卡氏空间中的直线运动，将从目前的坐标位置直线运动至目标坐标位置，而状态(state)将会由 GROUP_STAND_STILL 变成 GROUP_MOVING(移动中)，若到达目标位置即速度减至零，则状态(state)变为 GROUP_STAND_STILL。

- **C/C++语法:**

RTN_ERR NMC_GroupLine(I32_T DevID, I32_T GroupIndex, I32_T CartAxisMask, const Pos_T *PCartPos, _opt_null_ const F64_T *PMaxVel);

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

I32_T CartAxisMask: 欲执行运动的卡氏空间坐标轴的组合码，请参考下表

const Pos_T *PCartPos: 输入指针变量，为目标坐标位置

_opt_null_ const F64_T *PMaxVel: 输入指针变量，设定最高速度，输入 NULL(0)则忽略此参数

卡氏空间坐标	V	U	C	B	A	Z	Y	X
位数	7	6	5	4	3	2	1	0
数值 (次方为位数)	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

卡氏空间坐标轴的组合码(CartAxesMask)的使用方式如下述：

若移动的卡氏空间坐标为 X，Z 与 A 轴，则卡氏空间坐标轴的代号为 $2^0 + 2^2 + 2^3 = 13$。

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
I32_T   devID        = 0;
I32_T   groupIndex   = 0;
I32_T   cartAxesMask = 13; //移动X、Z与A轴
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;
```

- **参阅:**

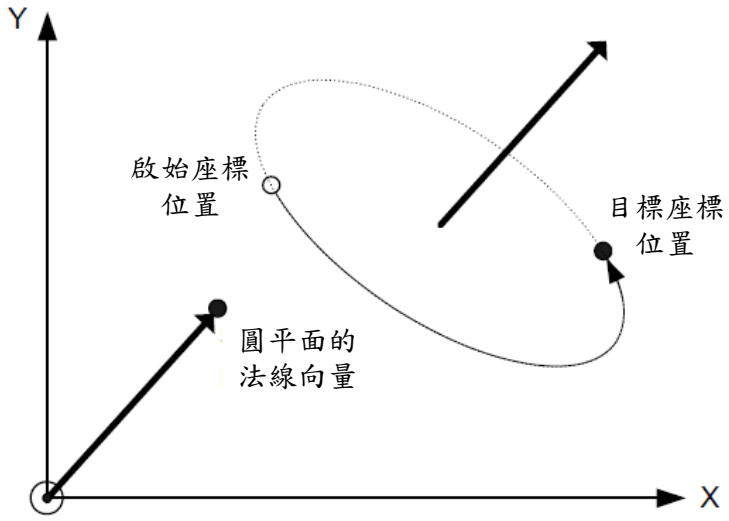
<无>

3.4.12.2. NMC_GroupCircR

执行卡氏空间中的圆弧运动(半径法)，将从目前的卡氏空间坐标位置圆弧运动至目标坐标位置，而状态(state)将会由 GROUP_STAND_STILL 变成 GROUP_MOVING(移动中)，若到达目标位置即速度减至零，则状态(state)变为 GROUP_STAND_STILL。

此圆弧运动(半径法)的路径算法为参考 PLCopen 规范所制作，其注意事项如下：

- 圆平面的法线向量的起点为卡氏空间坐标的原点，可不必输入单位向量。
- 启始坐标位置与目标坐标位置所形成的向量，必须与圆平面的法线向量垂直，否则会回传错误代码。
- 半径法所规划出来的路径可分为弧度较大与较小的两种，使用者需选择其一来执行，此时则使用半径(Radius)的正负号来决定，正号为选择弧度较小的路径；负号为选择弧度较大的路径。
- 若圆弧为 2D 路径，可不必输入圆平面的法线向量，但需设定旋转方向(CW_CCW)，因可在 XY、YZ 或 ZX 平面上使用右手定则。
- 若圆弧为 3D 路径，则不会参考旋转方向(CW_CCW)，因 3D 路径无法由右手定则决定。



- **C/C++语法：**

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);

- **参数：**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

I32_T CartAxisMask: 欲执行运动的卡氏空间坐标轴的组合码，请参考下表

const Pos_T *PCartPos: 输入指针变量，为目标坐标位置

const Xyz_T *PNormalVector: 输入指针变量，为圆平面的法线向量

F64_T Radius: 圆弧的半径(负值代表选择弧度较大之路径)

I32_T CW_CCW: 圆弧的旋转方向(0=CW; 1=CCW)

_opt_null_ const F64_T *PMaxVel: 输入指针变量，设定最高速度，输入 NULL(0)则忽略此参数

卡氏空间坐标	V	U	C	B	A	Z	Y	X
位数	7	6	5	4	3	2	1	0
数值 (次方为位数)	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

卡氏空间坐标轴的组合码(CartAxesMask)的使用方式如下述：

若移动的卡氏空间坐标为 X，Z 与 A 轴，则卡氏空间坐标轴的代号为 $2^0 + 2^2 + 2^3 = 13$。

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例:**

```
I32_T   devID        = 0;
I32_T   groupIndex   = 0;
I32_T   cartAxesMask = 3; //移动X与Y轴
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;
```

- **参阅:**

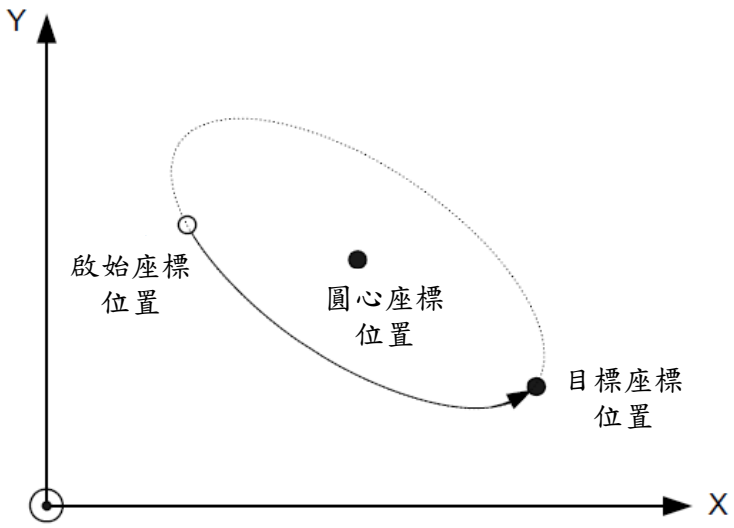
<无>

3.4.12.3. NMC_GroupCircC

执行卡氏空间中的圆弧运动(圆心法)，将从目前的卡氏空间坐标位置圆弧运动至目标坐标位置，而状态(state)将会由 GROUP_STAND_STILL 变成 GROUP_MOVING(移动中)，若到达目标位置即速度减至零，则状态(state)变为 GROUP_STAND_STILL。

此圆弧运动(圆心法)的路径算法为参考 PLCopen 规范所制作，其注意事项如下：

- 若圆心法所规划出来的圆弧为 3D 路径，则可分为弧度较大与较小的两种，使用者需选择其一来执行，此时则使用旋转方向(CW_CCW)来决定，CW 为选择弧度较小的路径；CCW 为选择弧度较大的路径。
- 若圆弧为 2D 路径，因可在 XY、YZ 或 ZX 平面上使用右手定则，故可利用旋转方向(CW_CCW)来决定弧度为选择较大或较小的路径。



- **C/C++语法：**

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);

- **参数：**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

I32_T CartAxisMask: 欲执行运动的卡氏空间坐标轴的组合码，请参考下表

const Pos_T *PCartPos: 输入指针变量，为目标坐标位置

I32_T CenOfsMask: 圆心坐标位置的组合码，请参考下表

const Xyz_T *PCenOfs: 输入指针变量，为圆心坐标位置

I32_T CW_CCW: 圆弧的旋转方向(0=CW; 1=CCW)

_opt_null_ const F64_T *PMaxVel: 输入指针变量，设定最高速度，输入 NULL(0)则忽略此参数

卡氏空间坐标	V	U	C	B	A	Z	Y	X
位数	7	6	5	4	3	2	1	0
数值（次方为位数）	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

卡氏空间坐标轴的组合码(CartAxesMask)的使用方式如下述：
若移动的卡氏空间坐标为 X，Z 与 A 轴，则卡氏空间坐标轴的代号为 $2^0 + 2^2 + 2^3 = 13$。

- **回传值：**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下：
若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例：**

```
I32_T   devID        = 0;
I32_T   groupIndex   = 0;
I32_T   cartAxesMask = 7; //移动X、Y与Z轴
Pos_T   targetPos    = { 50, 50, 0, 0, 0, 0, 0 };
I32_T   cenOfsMask   = 7; //移动X、Y与Z轴
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;
```

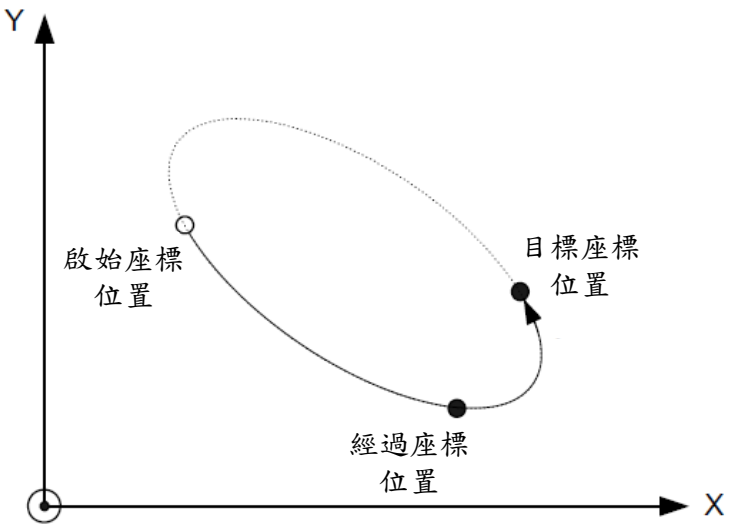
- **参阅：**

<无>

3.4.12.4. NMC_GroupCircB

执行卡氏空间中的圆弧运动(经过法)，将从目前的卡氏空间坐标位置圆弧运动至目标坐标位置，而状态(state)将会由 GROUP_STAND_STILL 变成 GROUP_MOVING(移动中)，若到达目标位置即速度减至零，则状态(state)变为 GROUP_STAND_STILL。

此圆弧运动(圆心法)的路径算法为参考 PLCopen 规范所制作，其坐标位置如下图。



- **C/C++语法:**

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);

- **参数:**

I32_T DevID: 装置识别号

I32_T GroupIndex: 群组识别号

I32_T CartAxisMask: 欲执行运动的卡氏空间坐标轴的组合码，请参考下表

const Pos_T *PCartPos: 输入指针变量，为目标坐标位置

I32_T BorPosMask: 经过坐标位置的组合码，请参考下表

const Xyz_T *PBorPoint: 输入指针变量，为经过坐标位置

_opt_null_ const F64_T *PMaxVel: 输入指针变量，设定最高速度，输入 NULL(0)则忽略此参数

卡氏空间坐标	V	U	C	B	A	Z	Y	X
位数	7	6	5	4	3	2	1	0
数值 (次方为位数)	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

卡氏空间坐标轴的组合码(CartAxesMask)的使用方式如下述：

若移动的卡氏空间坐标为 X，Z 与 A 轴，则卡氏空间坐标轴的代号为 $2^0 + 2^2 + 2^3 = 13$。

- **回传值：**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下：

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **范例：**

```
I32_T   devID        = 0;
I32_T   groupIndex   = 0;
I32_T   cartAxesMask = 7; //移动X、Y与Z轴
Pos_T   targetPos    = { 50, 50, 0, 0, 0, 0, 0 };
I32_T   borPosMask   = 7; //移动X、Y与Z轴
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;
```

- **参阅：**

<无>

3.4.13.1. NMC_ToolCalib_4p

Tool 教导- TCP 平移教导法

- **C/C++语法:**

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);

- **参数:**

const Pos_T *PMcsKinP1: 第一步骤位置，TCP 必须(尽可能)落在参考位置上
const Pos_T *PMcsKinP2: 第二步骤位置，TCP 必须(尽可能)落在参考位置上
const Pos_T *PMcsKinP3: 第三步骤位置，TCP 必须(尽可能)落在参考位置上
const Pos_T *PMcsKinP4: 第四步骤位置，TCP 必须(尽可能)落在参考位置上
CoordTrans_T *PRetToolCoordTrans:回传 Tool 坐标转换设定
F64_T *PRetTolerance:回传重复误差

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:
若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

- **范例:**

```
```

- **参阅:**

<无>

3.4.13.2. NMC_ToolCalib_4pWithZ

Tool 教导- TCP 平移与 Z 方向设定教导法

- **C/C++语法:**

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);

- **参数:**

const Pos_T *PMcsKinP1: 第一步骤位置，TCP 必须(尽可能)落在参考位置上

const Pos_T *PMcsKinP2: 第二步骤位置，TCP 必须(尽可能)落在参考位置上

const Pos_T *PMcsKinP3: 第三步骤位置，TCP 必须(尽可能)落在参考位置上

const Pos_T * PMcsKinP4ZDir: 第四步骤位置，TCP 正 Z 方向须指向 MCS 负 Z 方向

CoordTrans_T *PRetToolCoordTrans:回传 Tool 坐标转换设定

F64_T *PRetTolerance:回传重复误差

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

- **范例:**

- **参阅:**

<无>

3.4.13.3. NMC_ToolCalib_4pWithOri

Tool 教导- TCP 平移与姿态设定教导法

- **C/C++语法:**

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);

- **参数:**

const Pos_T *PMcsKinP1: 第一步骤位置，TCP 必须(尽可能)落在参考位置上

const Pos_T *PMcsKinP2: 第二步骤位置，TCP 必须(尽可能)落在参考位置上

const Pos_T *PMcsKinP3: 第三步骤位置，TCP 必须(尽可能)落在参考位置上

const Pos_T *PMcsKinP4: 第四步骤位置，TCP 必须(尽可能)落在参考位置上

const Pos_T *PMcsKinMinusZAxisPt: 第五步骤位置,参考位置必须落在 TCP 负 Z 轴上

const Pos_T *PMcsKinYZPlanPt: 第六步骤位置,参考位置必须落在 TCP 正 Y 轴上

CoordTrans_T *PRetToolCoordTrans:回传 Tool 坐标转换设定

F64_T *PRetTolerance:回传重复误差

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

- **范例:**

```
```

- **参阅:**

<无>

3.4.13.4. NMC_ToolCalib_Ori

Tool 教导- TCP 平移教导法

- **C/C++语法:**

RTN_ERR NMC_ToolCalib_Ori (const Pos_T *PMcsKinOrg, const Pos_T *PMcsKinMinusZAxisPt, const Pos_T *PMcsKinYZPt, CoordTrans_T *PRetToolCoordTrans);

- **参数:**

const Pos_T * PMcsKinOrg: 第一步骤位置，TCP 必须(尽可能)落在参考位置上

const Pos_T *PMcsKinMinusZAxisPt: 第二步骤位置,参考位置必须落在 TCP 负 Z 轴上

const Pos_T *PMcsKinYZPlanPt: 第三步骤位置,参考位置必须落在 TCP 正 Y 轴上

CoordTrans_T *PRetToolCoordTrans:回传 Tool 坐标转换设定，(只修改姿态转换参数)

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

- **范例:**

- **参阅:**

<无>

3.4.14.1. NMC_BaseCalib_1p

Base 教导-1p 法

- **C/C++语法:**

RTN_ERR NMC_BaseCalib_1p(const Pos_T *PRefBaseP1, CoordTrans_T *PRetBaseCoordTrans);

- **参数:**

const Pos_T *PRefBaseP1: 第一步骤位置

const Pos_T *PRefBaseP2: 第二步骤位置

const Pos_T *PRefBaseP3: 第三步骤位置

CoordTrans_T *PRetBaseCoordTrans: 回传相对于参考坐标系之坐标转换关系

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

- **范例:**

```
```

- **参阅:**

<无>

3.4.14.2. NMC_BaseCalib_2p

Base 教导-2p 法

- **C/C++语法:**

RTN_ERR NMC_BaseCalib_2p(const Pos_T *PRefBaseP1, const Pos_T *PRefBaseP2, CoordTrans_T *PRetBaseCoordTrans);

- **参数:**

const Pos_T *PRefBaseP1: 第一步骤位置

const Pos_T *PRefBaseP2: 第二步骤位置

CoordTrans_T *PRetBaseCoordTrans: 回传相对于参考坐标系之坐标转换关系

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

- **范例:**

```
```

- **参阅:**

<无>

3.4.14.3. NMC_BaseCalib_3p

Base 教导-3p 法

- **C/C++语法:**

RTN_ERR NMC_BaseCalib_3p(const Pos_T *PRefBaseP1, const Pos_T *PRefBaseP2, const Pos_T *PRefBaseP3, CoordTrans_T *PRetBaseCoordTrans);

- **参数:**

const Pos_T *PRefBaseP1: 第一步骤位置

const Pos_T *PRefBaseP2: 第二步骤位置

const Pos_T *PRefBaseP3: 第三步骤位置

CoordTrans_T *PRetBaseCoordTrans: 回传相对于参考坐标系之坐标转换关系

- **回传值:**

以 RTN_ERR 数据型态回传错误代码，回传值意义如下:

若呼叫函数成功回传“ERR_NEXMOTION_SUCCESS”(0)，反之函数呼叫失败回传错误代码，所有错误代码定义于 NexMotionError.h 头文件中。

- **用法:**

- **范例:**

```
```

- **参阅:**

<无>

4. 函式库定义

4.1. 数据型态定义

4.1.1. 基本数据型态

API 所使用的 C/C++数据型态定义于 nex_type.h 中，说明如下表:

型别	C/C++ 原型	说明	byte	范围
BOOL_T	int	布尔型别	4	0:False, 1:True
U8_T	unsigned char	无号整数	1	0 ~ 255
U16_T	unsigned short	无号整数	2	0 ~ 65535
U32_T	unsigned int	无号整数	4	0 ~ 4294967295
U64_T	unsigned __int64	无号整数	8	0 ~ 18446744073709551615
I8_T	char	有号整数	1	-128 ~ 127
I16_T	short	有号整数	2	-32768 ~ 32767
I32_T	int	有号整数	4	-2147483648 ~ 2147483647
I64_T	__int64	有号整数	8	-9223372036854775808 ~ 9223372036854775807
F32_T	float	浮点数	4	IEEE-754, 有效小数后 7 位
F64_T	double	双精浮点数	8	IEEE-754, 有效小数后 15 位
RTN_ERR	int	错误代码	4	-2147483648 ~ 2147483647

4.1.2. Motion 相关数据型态定义

Motion 相关数据型态定义于 NexMotionDef.h

4.1.2.1. Structure: Pos_T

描述群组(Group)之坐标位置，视使用的 API 之定义内容可以是 ACS 坐标信息或 MCS/PCS 坐标信息

- **C/C++语法定义:**

```
typedef struct
{
    F64_T pos[MAX_POS_SIZE];
} Pos_T;
```

- **成员:**

F64_T pos[MAX_POS_SIZE]: 大小为 8 的位置数组。

若描述 ACS 坐标系 pos[0~7]代表群组中 axis 0~7 之轴坐标位置。

若描述为卡式坐标系统(MCS/PCS) pos[0~7]代表卡式坐标系统 X 轴、Y 轴、Z 轴、A 轴、B 轴、C 轴、U 轴和 V 轴之坐标位置

4.1.2.2. Structure: Xyz_T

描述群组(Group)之卡式坐标 X 轴、Y 轴和 Z 轴之位置

- **C/C++语法定义:**

```
typedef struct
{
    F64_T pos[MAX_XYZ_SIZE];
} Xyz_T;
```

- **成员:**

F64_T pos[MAX_XYZ_SIZE]: 大小为 3 的位置数组。

pos[0]、pos[1]和 pos[2]分别代表卡式坐标系统 X 轴、Y 轴和 Z 轴之坐标位置

4.1.2.3. Structure: CoordTrans_T

描述两个标系之间之坐标转换关系

- **C/C++语法定义:**

```
typedef struct
{
    F64_T pose[NMC_MAX_POSE_DATA_SIZE];
} CoordTrans_T;
```

- **成员:**

F64_T pose[NMC_MAX_POSE_DATA_SIZE]: 大小为 6 的 F64_T 数组。

pose[0]、pose[1]和 pose[2]分别代表卡式坐标系统 X 轴、Y 轴和 Z 轴之相对位置(平移)

pose[3]、pose[4]和 pose[5]分别代表卡式坐标系统 Z 轴、Y 轴和 X 轴之旋转角度(degree)

4.1.3. 其他型态定义

4.1.3.1. 嵌入函式型态:PF_NmcHookAPI

- **C/C++语法定义:**

```
typedef void(*PF_NmcHookAPI)( const void *FPFuncAddress , const char *PFuncName, RTN_ERR
ReturnCode, void *PUserData );
```

- **成员:**

const void *FPFuncAddress: 目前被呼叫之函式指标

const char *PFuncName:目前被呼叫之函式名称

RTN_ERR ReturnCode: 目前被呼叫之函式回传值

void *PUserData:用户数据指针，由NMC_DebugSetHookData()设定

4.1.3.2. Structure: NmcTime_T

描述系统时间之数据结构

- **C/C++语法定义:**

```
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;
```

- **成员:**

U32_T year: 公元年

U32_T month: 月, 1 ~ 12.

U32_T day:日, 1 ~ 31.

U32_T hour:时, 1~ 23.

U32_T minute:分 , 0 ~ 59.

U32_T second:秒 , 0 ~ 59

U32_T milliseconds:毫秒 , 0 ~ 999

4.1.3.3. Structure: NmcMsg_T

描述系统时间之数据结构

- **C/C++语法定义:**

```
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;
```

- **成员:**

U32_T sizeOfStruct: 纪录检验NmcMsg_T 数据结构之大小, 相当于sizeof(NmcMsg_T)

NmcTime_T localTime: 讯息产生当下之系统时间纪录

U32_T index:讯息序号

I32_T type:讯息类别: 0:一般讯息(Normal), 1:警告讯息(Warning), 2:错误讯息(Error)

char source[NMC_MAX_MSG_SOURCE_SIZE]: 讯息来源(保留)

I32_T id;讯息识别(保留)

I32_T code;讯息代码

char text[NMC_MAX_MSG_TEXT_SIZE]:讯息内容

4.2. 常数定义

NexMotion 相关常数定义于 NexMotionDef.h

4.2.1. 装置型态(Device Type) 选择

定义	常数值	说明
NMC_DEVICE_TYPE_SIMULATOR	0	仿真器装置型态
NMC_DEVICE_TYPE_ETHERCAT	1	EtherCAT 装置型态

4.2.2. Wait 函式 Timeout 设定

定义	常数值	说明
NMC_WAIT_TIME_INFINITE	0xFFFFFFFF	等待至该程序执行完毕

4.2.3. 控制器状态(Device State)

定义	常数值	说明
NMC_DEVICE_STATE_INIT	1	初始化状态(Init)，控制器参数尚未设定或已清除
NMC_DEVICE_STATE_READY	2	预备状态(Ready)，参数已设定完成
NMC_DEVICE_STATE_ERROR	3	错误状态(Error)，控制器进入重大错误状态(如通讯中断)
NMC_DEVICE_STATE_OPERATION	4	可操作状态(Operation)

4.2.4. 坐标系统 (Coordinate system)选择

定义	常数值	说明
NMC_COORD_MCS	0	机构坐标系(Mechanical coord. system)
NMC_COORD_PCS	1	程序坐标系(Programming coord. System)
NMC_COORD_ACS	2	轴坐标系 (Axis coord. System)

4.2.5. 单轴轴状态(State of axis)

透过 NMC_AxisGetState()读取单轴运动状态，其变量值所代表意义如下示:

定义	常数值	说明
NMC_AXIS_STATE_DISABLE	0	禁用(Disable)状态，伺服未启动
NMC_AXIS_STATE_STAND_STILL	1	启用(Enable)状态，伺服启动整定状态
NMC_AXIS_STATE_HOMING	2	执行回原点(Homing)运动中
NMC_AXIS_STATE_DISCRETE_MOTION	3	执行点对点运动中
NMC_AXIS_STATE_CONTINUOUS_MOTION	4	执行连续运动中
NMC_AXIS_STATE_STOPPING	5	收到停止命令，减速停运动中
NMC_AXIS_STATE_STOPPED	6	收到停止命令，停止状态
NMC_AXIS_STATE_WAIT_SYNC	7	收到 Wait 命令，等待 SYNC 讯号状态中

NMC_AXIS_STATE_ERROR	10	发生错误，错误停止状态

4.2.6. 单轴运动信息(Status of Axis)位编号

透过 NMC_AxisGetStatus()读取单轴运动信息，其变量之各位编号所代表意义如下示:

定义	常数值	说明
NMC_AXIS_STATUS_EMG	0	EMG 讯号发生栓锁指示(*1)
NMC_AXIS_STATUS_ALM	1	伺服警报(Alarm)发生栓锁指示 (*1)
NMC_AXIS_STATUS_PEL	2	正极限讯号发生栓锁指示 (*1)
NMC_AXIS_STATUS_NEL	3	负极限讯号发生栓锁指示 (*1)
NMC_AXIS_STATUS_PSEL	4	软件正极限讯号发生栓锁指示 (*1)
NMC_AXIS_STATUS_NSEL	5	软件负极限讯号发生栓锁指示 (*1)
NMC_AXIS_STATUS_ENA	6	单轴已启用(Enable)停用(Disable)指示
NMC_AXIS_STATUS_ERR	7	单轴错误指示
NMC_AXIS_STATUS_TAR	8	单轴到达目标指示
NMC_AXIS_STATUS_CSTP	9	单轴命令停止指示
NMC_AXIS_STATUS_ACC	10	单轴于加速段指示
NMC_AXIS_STATUS_DEC	11	单轴于减速段指示
NMC_AXIS_STATUS_MV	12	单轴于最大速指示
NMC_AXIS_STATUS_OP	13	单轴于运动中指示
NMC_AXIS_STATUS_STOP	14	单轴于停止状态中指示
NMC_AXIS_STATUS_RPEL	16	正极限讯号指示: 1:触发, 0:未触发
NMC_AXIS_STATUS_RNEL	17	负极限讯号指示: 1:触发, 0:未触发
NMC_AXIS_STATUS_RHOM	18	Home 讯号指示: 1:高准位(high), 0:低准位(Low)

(*1)栓锁(Latched)讯号当错误状态排除后，呼叫 NMC_AxisResetState()后只讯号才会清除为 0

4.2.7. 单轴运动信息(Motion Status)位掩码

透过 NMC_AxisGetStatus()读取单轴运动信息,其变量之各位掩码(Bit Mask)所代表意义如下示:

定义	常数值	说明
NMC_AXIS_STATUS_MASK_EMG	0x00000001	EMG 讯号发生栓锁指示屏蔽(*1)
NMC_AXIS_STATUS_MASK_ALM	0x00000002	伺服警报(Alarm)发生栓锁指示屏蔽(*1)
NMC_AXIS_STATUS_MASK_PEL	0x00000004	正极限讯号发生栓锁指示屏蔽(*1)
NMC_AXIS_STATUS_MASK_NEL	0x00000008	负极限讯号发生栓锁指示屏蔽(*1)
NMC_AXIS_STATUS_MASK_PSEL	0x00000010	软件正极限讯号发生栓锁指示屏蔽(*1)
NMC_AXIS_STATUS_MASK_NSEL	0x00000020	软件负极限讯号发生栓锁指示屏蔽(*1)
NMC_AXIS_STATUS_MASK_ENA	0x00000040	单轴已启用(Enable)停用(Disable)指示屏蔽
NMC_AXIS_STATUS_MASK_ERR	0x00000080	单轴错误指示屏蔽
NMC_AXIS_STATUS_MASK_TAR	0x00000100	单轴到达目标指示屏蔽

NMC_AXIS_STATUS_MASK_CSTP	0x00000200	单轴命令停止指示屏蔽
NMC_AXIS_STATUS_MASK_ACC	0x00000400	单轴于加速段指示屏蔽
NMC_AXIS_STATUS_MASK_DEC	0x00000800	单轴于减速段指示屏蔽
NMC_AXIS_STATUS_MASK_MV	0x00001000	单轴于最大速指示屏蔽
NMC_AXIS_STATUS_MASK_OP	0x00002000	单轴于运动中指示屏蔽
NMC_AXIS_STATUS_MASK_STOP	0x00004000	单轴于停止状态中指示屏蔽
NMC_AXIS_STATUS_MASK_RPEL	0x00010000	正极限讯号指示屏蔽 1:触发, 0:未触发
NMC_AXIS_STATUS_MASK_RNEL	0x00020000	负极限讯号指示屏蔽 1:触发, 0:未触发
NMC_AXIS_STATUS_MASK_RHOM	0x00040000	Home 讯号指示屏蔽 1:高准位(high), 0:低准位(Low)

(*1)栓锁(Latched)讯号当错误状态排除后，呼叫 NMC_AxisResetState()后只讯号才会清除为 0

4.2.8. 群组坐标轴编号

定义	常数值	说明
GROUP_AXIS_X	0	坐标系统 X 轴
GROUP_AXIS_Y	1	坐标系统 Y 轴
GROUP_AXIS_Z	2	坐标系统 Z 轴
GROUP_AXIS_A	3	坐标系统 A 轴
GROUP_AXIS_B	4	坐标系统 B 轴
GROUP_AXIS_C	5	坐标系统 C 轴
GROUP_AXIS_U	6	坐标系统 U 轴
GROUP_AXIS_V	7	坐标系统 V 轴

4.2.9. 群组坐标轴号屏蔽

定义	常数值	说明
GROUP_AXIS_MASK_X	0x00000001	坐标系统 X 轴屏蔽
GROUP_AXIS_MASK_Y	0x00000002	坐标系统 Y 轴屏蔽
GROUP_AXIS_MASK_Z	0x00000004	坐标系统 Z 轴屏蔽
GROUP_AXIS_MASK_A	0x00000008	坐标系统 A 轴屏蔽
GROUP_AXIS_MASK_B	0x00000010	坐标系统 B 轴屏蔽
GROUP_AXIS_MASK_C	0x00000020	坐标系统 C 轴屏蔽
GROUP_AXIS_MASK_U	0x00000040	坐标系统 U 轴屏蔽
GROUP_AXIS_MASK_V	0x00000080	坐标系统 V 轴屏蔽

4.2.10. 群组状态(State of group)

透过 NMC_GroupGetState()读取群组运动状态，其变量值所代表意义如下示:

定义	常数值	说明
NMC_GROUP_STATE_DISABLE	0	群组为禁用(Disable)状态，群组伺服未启动
NMC_GROUP_STATE_STAND_STILL	1	群组启用(Enable)状态，群组伺服启动整定状态
NMC_GROUP_STATE_STOPPED	2	收到停止命令，停止状态
NMC_GROUP_STATE_STOPPING	3	收到停止命令，减速运动中
NMC_GROUP_STATE_MOVING	4	执行运动命令中
NMC_GROUP_STATE_HOMING	5	执行回原点(Homing)运动中
NMC_GROUP_STATE_ERROR	6	发生错误，错误停止状态

4.2.11. 群组运动信息(status of group)位编号

透过 NMC_GroupGetStatus()读取单轴运动信息，其变量之各位编号所代表意义如下示:

定义	常数值	说明
NMC_GROUP_STATUS_EMG	0	外部 EMG 讯号发生栓锁指示(*1)
NMC_GROUP_STATUS_ALM	1	任意群组轴伺服警报(Alarm)发生栓锁指示 (*1)
NMC_GROUP_STATUS_PEL	2	任意群组轴正极限讯号发生栓锁指示 (*1)
NMC_GROUP_STATUS_NEL	3	任意群组轴负极限讯号发生栓锁指示 (*1)
NMC_GROUP_STATUS_PSEL	4	任意群组轴软件正极限讯号发生栓锁指示 (*1)
NMC_GROUP_STATUS_NSEL	5	任意群组轴软件负极限讯号发生栓锁指示 (*1)
NMC_GROUP_STATUS_ENA	6	群组已启用(Enable, 1)或停用(Disable, 0)指示
NMC_GROUP_STATUS_ERR	7	群组(任意群组轴)错误指示
NMC_GROUP_STATUS_CSTP	9	所有的群组轴无位置输出(位置变化量为 0)
NMC_GROUP_STATUS_ACC	10	卡式坐标运动(直线与圆弧)中,正在加速(或减速)至最高速度的阶段，PTP 或 JOG 运动时，此位为 0
NMC_GROUP_STATUS_DEC	11	卡式坐标运动(直线与圆弧)中,正在减速至目标位置或停止的阶段，PTP 或 JOG 运动时，此位为 0
NMC_GROUP_STATUS_MV	12	卡式坐标运动(直线与圆弧)中,正在最高速度的阶段，PTP 或 JOG 运动时，此位为 0
NMC_GROUP_STATUS_OP	13	群组运动中，即状态(state)为 GROUP_MOVING、GROUP_HOMING 或 GROUP_STOPPING
NMC_GROUP_STATUS_STOP	14	群组为停止状态，即状态(state)为 GROUP_STOPPED

(*1)栓锁(Latched)讯号当错误状态排除后,呼叫 NMC_GroupResetState()后只讯号才会清除为 0

4.3. 错误代码(Error Code)

错误代码定义于 NexMotionError.h 中，说明如下表:

定义	常数值	说明
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_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