NATORS

纳米定位系统编程指南

NPS - Nano Positioning System

Programmer's Guide

纳特斯(苏州)科技有限公司

Tel：13166018168
eMail：info@nators.com
URL: http://www.nators.com/

Documents Version 2.4.16

目录

1 概述

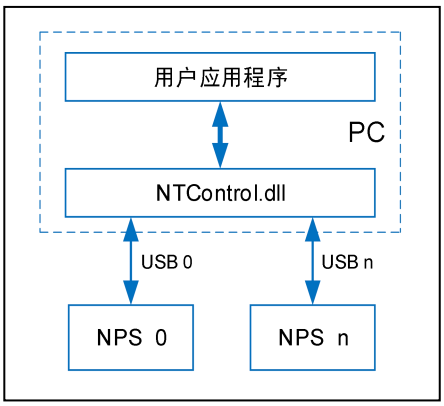
为了让用户可以自己编写应用程序来操控 NATORS 纳米定位系统，我们提供了符合 C 语言编译和链接规约的动态链接库 NTControl.dll（以下简称 DLL），调用约定为 cdecl，**能够被跨语言调用**。

本文档主要介绍纳米定位系统 DLL 的用法。DLL 可以看做是应用程序与定位系统控制器之间的“桥梁”。用户无需了解 PC 与控制器之间复杂的通信协议和数据格式，对于想要实现的功能，只需要选择对应的 API 函数，并传入正确的参数即可。

针对使用 LabVIEW 软件的用户，我们将 DLL 中的所有 API 函数都封装成了子 VI，并打包成了一个 VI 库。VI 库的安装请参阅《NPS Software Installation Guide》，VI 库的用法请参阅《NPS LabVIEW-Integration》。

用户可以将多个控制器连接到 PC，并通过 DLL，同时给多台控制器发送命令。在 DLL 中，将每一个纳米定位系统（不限制通道数）都称为“系统”。一个系统中包含一个控制器和若干个定位台。当 PC 上连接多个系统时，为了方便区分，每一个系统会由 DLL 分配一个唯一的系统索引 systemIndex，可以将它理解为一个由 DLL 分配的“位置”，对于大多数 API 函数，都需要将它作为参数来定位目标系统。注意，由于 systemIndex 由 DLL 生成，因此需要将其保存在应用程序中。

下面给出了系统、DLL、PC 及用户应用程序之间的关系连接简图。



其中，箭头的方向表示通信的方向。**注意，各个系统之间是独立的。**

在一个系统中，每个定位台都占用一个“通道”，为了方便区分，每一个通道都有一个唯一的通道索引 channelIndex，与 systemIndex 类似，大部分 API 函数都需要 channelIndex 作为参数来定位目标通道。注意，通道索引值从 0 开始，以 1 为单位递增，直到最大通道数减一。假设一个系统的最大通道数为 N，那么其通道索引依次为 0，1，2，... N-1。

有两点需要注意。第一点，通道的命名形式为“CHx”，x 为通用数字，从 1 开始，以 1 为单位递增，如 CH1 表示通道 1，其通道索引为 0。第二点，与 systemIndex 不同的是，channelIndex 不是由 DLL 分配的，而是由控制器的硬件决定的。

库附带的头文件（NTControl.h）总结了 DLL 的功能，并列出了所有错误代码和状态代码的定义（参阅第 4 章）。第 2 章对 DLL 的使用进行了简单的介绍。第 3 章对 DLL 中的所有 API 函数进行了详细介绍，并提供了示例程序。

注意，所有库函数都使用 cdecl 调用约定。某些开发环境（例如 Delphi）默认使用 stdcall，导入库函数时必须考虑到这一点。

2 功能文档

在使用纳米定位系统的 DLL 前，用户需要先了解 DLL 的使用流程。本章将详细地说明这些信息，并介绍一些 DLL 及定位系统控制器的基本概念。

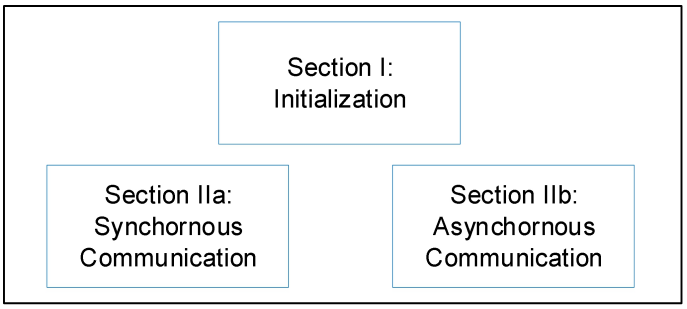
请用户在阅读第三章前，一定要仔细阅读本章的内容。

2.1 总览

DLL 中的 API 函数分为两类。第一类用于初始化系统，第二类用于应用程序与硬件的实际通信。第二类中的函数又分为两部分（IIa 和 IIb）。

注意，IIa 的函数与 IIb 的函数不可同时使用。

在初始化时，用户必须在同步通信模式与异步通信模式之间进行选择（请参阅 2.2.2 节），并且根据这个选择，调用 IIa 节或 IIb 节的函数。若调用的函数与选择的模式不匹配，函数将终止运行并返回错误代码。



为了便于区分，IIa 部分的函数名称后缀为_S(同步)，IIb 部分的函数名称后缀为_A(异步)。所有 API 函数被调用后均会返回 NT_STATUS 类型的状态码。状态码用于指示函数是否调用成功。若函数调用成功，将返回 NT_OK。否则，函数将根据调用失败的原因返回相应的错误代码。有关错误代码的完整列表，请参阅 4.1 节。

为简单起见，所有 API 函数的参数均为 32 位宽（有符号或无符号）。库函数的大多数参数都有范围限制，若输入超出有效范围，函数将终止执行并返回错误代码。有关 API 函数的更多信息，请参阅第 3 章。

2.2 初始化

在使用系统之前，必须先调用函数 NT_OpenSystem 初始化系统。注意，除以下函数外，所有 API 函数都必须在系统初始化以后才能被调用。

- NT_FindSystems。

函数 NT_OpenSystem 用于打开由 systemLocator 指定的系统，以建立应用程序与系统之间的连接。该函数调用成功后，必须将返回的 systemIndex 保存在应用程序中，并作为入口参数传递给其他 API 函数，用于连接指定的系统。

同一时刻，一个系统只能被一个应用程序占用。若其他应用程序想要使用该系统，必须先调用函数 NT_CloseSystem 来断开当前应用程序与系统之间的连接，否则将由于系统资源被占用而导致系统无法被再次打开。

2.2.1 接口与定位字符串

纳米定位系统控制器支持 USB 接口和以太网接口。在 DLL 的使用上，这两种接口的控制器，除了定位字符串不一样外，无任何区别。

定位字符串（systemLocator，以下简称定位符）用于指定函数 NT_OpenSystem 打开哪一个系统，定位符的一般形式如下所示：

```
usb:id:5535186640
network:192.168.1.200:2000
```

第一个定位字符串用来连接 USB 接口的控制器，第二个定位字符串用来连接以太网接口的控制器。

对于 USB 接口的控制器，定位字符串的格式如下：

```
usb:id:<id>
```

其中，<id>是每一台控制器的唯一标识，出厂时一般会打印在控制器背面。对于只有一个控制器连接到 PC 的情况，不再需要<id>对控制器进行区分，此时可以使用如下所示的简易定位符：

```
usb:ix:0
```

对于多台控制器连接到 PC 的情况，使用简易定位符只能初始化其中的一个系统。此时推荐使用函数 NT_FindSystems，此函数将扫描 PC 的所有 USB 端口，并将连接到 PC 的所有控制器的定位符以列表的形式返回。对于已经被成功初始化的系统，可以调用函数 NT_GetSystemLocator 来获取 systemIndex 指定系统的定位符。

对于以太网接口的控制器，定位字符串的格式如下：

```
network:<ip>:<port>
```

其中，<ip>是一个 IPv4 地址，由 0 到 255 之间的四个整数组成，用一个点分隔。<port>是一个整数，表示端口号。

以定位符 network:192.168.1.200:2000 为例，IP 地址是 192.168.1.200，端口号

是 2000.

有关以太网接口的配置，请参阅《纳米定位系统以太网接口配置指南》。

注意，定位字符串格式为英文，只支持小写，请要严格按照规范的格式输入。

2.2.2 通信模式

在调用函数 NT_OpenSystem 对系统进行初始化时，用户必须在同步通信模式或者异步通信模式之间进行选择。不同的模式不仅会影响 DLL 内部通信方式，还会影响 DLL 的使用方式。同步通信模式比较简单，但是灵活性较差。异步模式更加灵活，但是编程实现更加复杂。

简单来讲，在同步模式下，一个函数调用的完整过程是，向控制器发送命令->阻塞->接收应答信号。应答信号可能是命令请求查询的信息，命令接收完成的确认，也可能是错误代码。阻塞时间通常只有几个毫秒。

相反，在异步模式下，一个函数调用向控制器发送命令后会立即返回，命令请求查询的信息需要用户手动去获取。有关异步通信的更多信息，请参阅 2.3 节。

2.3 使用异步通信模式

本节旨在让您更好地理解异步通信模式。使用异步通信模式可以更好地满足您的需求。

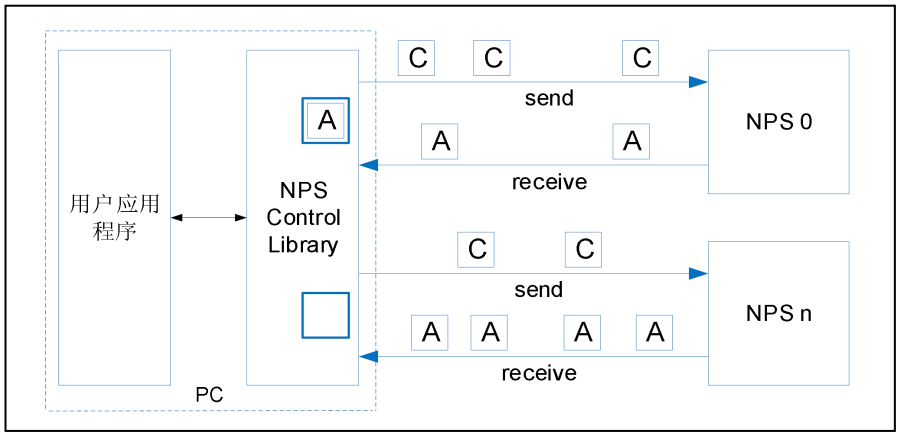
2.3.1 概述

可以这样想象，在控制器与 PC 之间有两条通信线，一条用于向系统发送命令，另一条用于接收来自系统的应答。在同步模式下，两条线路上的通信量是同步的，发送指令一定会有应答。在异步模式下，一条线路上的通信量与另外一条线路上的通信量无关，两条线路上的通信量是异步的。因此，当向系统发送命令时，可能不会有应答信号，这取决于命令的类型或者通道的配置。注意，如果几个命令被发送到系统的不同通道，来自每个通道的应答的顺序是未定义的。但是，来自单个通道的应答的顺序由发出命令的顺序定义。

总而言之，异步模式的使用包括两个部分：

- 发送命令：这些命令包括移动命令，配置命令等
- 检索和管理应答：这些包括状态信息，错误等

下图给出了在异步模式下，多个系统并行通信的结构图。其中，C 表示命令，A 表示应答。蓝色的框表示接收缓冲区。



2.3.2 发送命令

DLL 中 IIb 部分的大多数 API 函数都会向硬件发送命令，以调用某些功能或者获取某些信息。这些函数不会阻塞，会立即返回。请注意，甚至 `NT_GetStatus_A` 之类的函数也会立即返回，并且不会立即提供所需的信息。它们仅向硬件发送请求，请求的内容需要用户手动获取（请参阅下文）。

错误处理分为两个级别。DLL 在实际发送命令之前会进行一些有效的范围检查，如通信模式是否正确，输入参数是否超出规定范围等等。在此级别上检测到的错误将导致函数返回 `NT_OK` 以外的状态代码。但是，即使命令发送成功，也可能因为硬件上的限制而导致发送错误。在这种情况下，系统将返回一个错误包。

2.3.3 检索应答

在异步模式下，对于如 NT_StepMove_A 之类的控制类命令以及设置系统参数的指令，系统不会产生任何应答。对于 NT_GetPosition_A 之类的获取信息命令，系统产生的应答会以数据包的形式存放在接收缓冲区中（见上图），需要用户手动去获取，库提供了几种获取数据包的方法。

每一个系统都有自己的接收缓冲区，因此，来自给定系统所有通道的所有应答被依次存放在同一个缓冲区中。接收缓冲区被组织为 FIFO 缓冲区，以下函数用于访问缓冲区的头部。

- NT_ReceiveNextPacket_A:返回当前位于接收缓冲区头部的应答包。如果缓冲区为空且未给出超时时间（超时时间等于 0），则会返回 NT_NO_PACKET_TYPE 类型的数据包。如果缓冲区为空并给出了超时时间，则将返回在超时时间内到达的第一个数据包。如果在超时时间内没有收到数据包，则返回 NT_NO_PACKET_TYPE 类型的数据包。**返回的数据包将从接收缓冲区内删除**。

- NT_LookAtNextPacket_A:这个函数与函数 NT_ReceiveNextPacket_A 的功能相似，不同之处在于它不会将返回的数据包从接收缓冲区内删除。

- NT_DiscardPacket_A:此函数用于删除接收缓冲区中的数据包。如果接收缓冲区为空，该函数无效。

注意：

- 使用函数 NT_LookAtNextPacket_A 时，因为数据包不会从接收 FIFO 中删除，因此您只能查看 FIFO 头部的数据包。如果想查看 FIFO 头部下面的数据包，请调用函数 NT_DiscardPacket_A 删除 FIFO 头部的数据包，或者直接调用函数 NT_ReceiveNextPacket_A。

- 调用函数 NT_LookAtNextPacket_A，然后调用 NT_DiscardPacket_A 具有与调用 NT_ReceiveNextPacket_A 相同的效果。不同之处在于，应用程序（线程）可以在数据包被删除之前（可能有负责处理特定数据包的线程）根据需要经常查看数据包。

使用函数 NT_ReceiveNextPacket_A 时，用户可以通过超时参数 timeout 控制函数的阻塞时间。通常，该函数将在以下两个事件之一发生时返回：

- 收到数据包
- 发生超时

如果给出了较长甚至无限的超时时间（NT_TIMEOUT_INFINITE），并且始终没有接收到数据包时，您可以通过在另外一个线程调用函数 NT_CancelWaitForPacket_A 来结束此函数的阻塞。

2.3.4 数据包格式

数据包的结构定义如下：

```
typedef struct NT_packet {
        NT_PACKET_TYPE packetType; // type of packet
        NT_INDEX channelIndex; // source channel
        unsigned int data1; // data field
        signed int data2; // data field
        signed int data3; // data field
        unsigned int data4; // data field
} NT_PACKET;
```

对于函数 NT_ReceiveNextPacket_A 返回的任何数据包，都应首先检查其类型。数据包各个字段数据的有效性取决于数据包的类型。有关数据包类型及其含义的更多信息，请参阅 4.2 节。

2.4 闭环模式

2.4.1 传感器模式

为了让定位台能够识别当前的位置，传感器需要供电。但是，这会导致其产生热量(造成漂移效应)，因此在某些情况下(尤其是在对温度变化非常敏感或者无法散热的环境中，如真空)可能需要禁用传感器。为此，传感器提供了两种不同的工作模式，可以通过函数 NT_SetSensorEnabled_S 进行配置。

- *Disable* – 在此模式下，传感器的电源被关闭，这样能够避免热量的产生。此时若调用诸如 NT_GotoPositionAbsolute_S 之类的闭环命令，命令将不会被执行，并且返回错误信息 NT_SENSOR_DISABLED_ERROR。如果传感器的发热干扰了系统中的其他组件，这个“Disable”模式会变的很有用。（比如在 SEM 中，可以禁用传感器以免干扰到 SEM 电子束）

注意： 关闭传感器后，开环命令（NT_StepMove_S）仍然能够执行，但是定位台的位置信息将失效。如果用户需要获取位置信息，需要重新开启传感器的电源。请注意，位置计算是增量式的，如果定位台在传感器电源关闭期间执行了移动，那么即使重新开启电源，位置信息也将失效。

- *Enabled* – 在此模式下，传感器将持续供电。所有的移动命令都可以正常执行。

2.4.2 定义位置

定位台内部使用光栅尺作为传感器，光栅尺可以分为有零点（又称参考点）和无零点两种。使用无零点光栅尺的定位台，控制器在上电后无法知道其所处的物理位置，一般的做法是设定其起始位置为零点。函数 NT_SetPosition_S 就是用于此目的。该函数可以将当前位置设置为有效输入范围内的任意值，包括零点。需要注意的是，该函数仅使光栅尺的测量标尺发生偏移，但定位台的物理位置不会发生改变，也不会执行任何移动。

对于上述定位台，其物理位置必须通过系统外部的设备来确定，并且每次系统上电都要重新确定位置。

对于使用光栅尺带有零点的定位台，克服了上述中的不便，这种定位台可以通过函数 NT_FindReferenceMark_S 以自动的方式确定零点位置。

2.4.3 旋转定位台

旋转定位台在处理位移时不同于线性定位台。

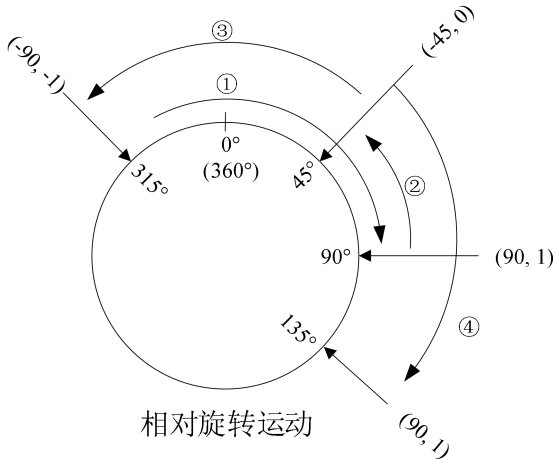
假设旋转定位台当前指向 45°，并按照命令要移动到 90°。可以通过两种方式完成：顺时针和逆时针。旋转位移由角度值(angle)和圈数(revolution)值组成，角度有效范围是 0~359,999,999，单位 u°。圈数值(revolution)代表已经完成了 360°的旋转，有效范围是 -32,768~32,767。

如果旋转定位台从 0°(angle=0,revolution=0)沿着正方向(顺时针)移动，角度值会增加。如果运动越界 360°(0°)角度值会从 0 开始，同时 revolution 会自动加 1，表示已经完成了一圈运动。反之亦然。

当使用 NT_GotoAngleAbsolute_S 发出绝对运动命令时，运动方向由给定的参数定义。在上面的例子中(从 45 度移动到 90 度)，方向可以通过指定 revolution 为 0 或-1 作为旋转参数来区分(假设当前旋转圈数为 0)。

注意当使用函数 NT_GotoAngleRelative_S 时旋转角度的有效输入范围值将扩展到 -359,999,999~359,999,999。

下面通过具体案列分析相对位置旋转运动和绝对位置旋转运动。简化运动命令表达方式:(angle,revolution)。



相对旋转运动

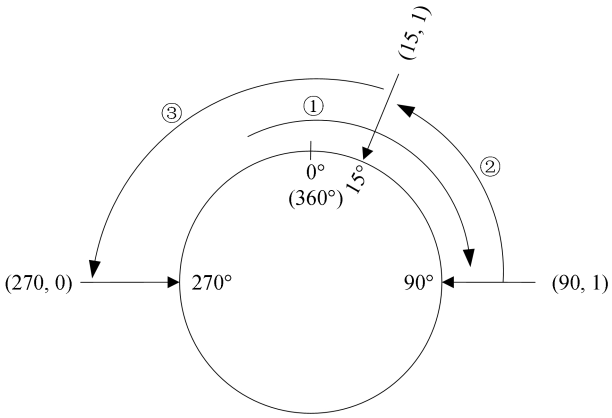
上图中共有 4 条相对运动命令。起点是(0,0)。

① (90, 1)：angle=90°，revolution=1；顺时针旋转越界 0°(360°)后再继续旋转 90°；

② (-45, 0)：angle=-45°，revolution=0；相对①的位置逆时针旋转 45°；

③ (-90,-1)：angle=-90°，revolution=-1；逆时针旋转到越界 0°(360°)后 revolution 自动-1，之后继续旋转，在相对②的位置逆时针旋转 90°后停止，此时指向的角度是 315°。经过①②③的运动后 revolution=0，已经旋转了 0 圈。

④ (90, 1)：angle=90°，revolution=1；假设还是从相对②的位置出发。顺时针旋转到越界 0°(360°)revolution 自动+1，之后继续旋转，并到达②的位置，此后再继续旋转 90°后停止，此时指向的角度是 135°，经过①②④的运动后 revolution=2，已经旋转了 2 圈。



绝对旋转运动

上图中共有 3 条绝对运动命令。起点是 0°。

① (90, 1)：angle=90°，revolution=1；顺时针旋转越界 0°(360°)后再继续旋转 90°；

② (15, 1)：angle=15°，revolution=1；逆时针旋转到第 1 圈的 15°位置；

③ (270, 0)：angle=270°，revolution=0；逆时针旋转到第 0 圈的 270°的位置；

2.5 其他

2.5.1 多个命令源

控制器支持 PC 应用程序的控制，同时也可以响应手动控制模块（操作杆）的命令。这样的设置既有优点也有缺点，主要有以下几点需要考虑：

系统的所有通道都可以接收来自 PC 应用程序或手动控制模块的命令。当用户在通过 PC 应用程序操作定位台的同时，也可以通过手动控制模块调整其位置（例如，当软件处于非活动状态时）。但是，在这种情况下，手动控制模块可能会覆盖来自应用程序的命令，反之亦然。这可能会导致意料之外的情况出现，为避免此类情况，应用程序可以完全（或暂时）禁用手动控制模块。更多信息，请参阅函数 `NT_SetHCMEnabled`。

3 库函数介绍

本章将对 DLL 中的所有 API 函数进行介绍，并给出示例程序，示例中出现的所有宏定义均在 DLL 所附带的头文件（NTControl.h）中定义。

3.1 初始化函数

NT_CloseSystem

函数原型：

```
NT_STATUS NT_CloseSystem(NT_INDEX systemIndex);
```

功能描述：

此函数用于关闭由 NT_OpenSystem 初始化的系统，并释放 DLL 为该系统分配的资源。在应用程序关闭前，一定要调用此函数，否则该系统将无法被其他应用程序使用。对于一个未关闭的系统，再次打开将失败并返回错误代码。

警告，未关闭的系统将导致资源泄露。

入口参数：

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向要关闭的系统。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if(result == NT_OK){
       //Closing previously aquired system
       NT_CloseSystem(ntHandle);
}
```

NT_OpenSystem

函数原型：

```
NT_STATUS NT_OpenSystem(NT_INDEX *systemIndex,
                        const char *systemLocator,
                        const char *options);
```

功能描述：

此函数用来初始化由 systemLocator 指定的系统。systemIndex 是系统索引，系统初始化成功后，由此函数分配。该参数将作为入口参数传递给其他 API 函数，用来寻址由 systemLocator 指定的系统。options 是配置选项表，用于配置 DLL 的通信模式。配置选项表允许同时写入多个配置，每种配置选项之间必须使用逗号或者换行符分隔。

options 的配置选项表：

- *reset* 在控制器上电的情况下进行硬件复位，**该项非必选项**。
- *async,sync* 使用 async 选项将通信模式设置为异步，使用 sync 选项将通信模式设置为同步，**该项为必选项**。
- *open-timeout<t>* 仅用于以太网接口。<t>是 PC 尝试连接纳米定位系统控制器的最大超时时间，单位为 ms。默认值是 2000ms。最大超时时间取决于操作系统。

入口参数

- *systemIndex* (unsigned 32bit)，输出 – 系统索引，指向由此函数打开的系统。
- *systemLocator* (const char pointer)，输入 – 目标系统的定位符。
- *options* (const char pointer)，输入 – 初始化系统的配置选项。请参阅 options 的配置选项表。

示例程序：

```
const char loc1[] = "usb:id:5535186640";
const char loc2[] = "network:192.168.1.200:2000";

NT_STATUS result;
NT_INDEX ntHandle1,ntHandle2;
// connect to a USB interface for async. communication
result = NT_OpenSystem(&ntHandle1, loc1, "async");
if(result != NT_OK){
      // handle error
}
// connect to a USB interface for sync. communication with 1.0 sec timeout
result = NT_OpenSystem(&ntHandle2, loc2, "sync,opne-timeout 1000");
if(result != NT_OK){
      // handle error
}
```

NT_FindSystems

函数原型：

```
NT_STATUS NT_FindSystems(const char *options,
                         char *outBuffer,
                         unsigned int *ioBufferSize);
```

功能描述：

此函数用于将所有连接到 PC 的控制器的定位符以列表的形式写入 outBuffer。options 用于对查找到的系统进行配置（目前未使用）。调用者必须在 outBuffer 中传递一个指向 char 缓冲区的指针，并将 ioBufferSize 设置为缓冲区的大小。调用成功后，函数将系统定位符列表写入 outBuffer，并将写入的字节数写入 ioBufferSize。如果提供的缓冲区太小而无法容纳生成的列表，则该缓冲区将不包含任何有效内容，但 ioBufferSize 包含所需的缓冲区大小。

此函数调用前不需要进行系统初始化。

入口参数

- *options* (const char)，输入 – 对查找到的系统进行配置，**目前未使用**。
- *outBuffer* (char)，输出 – 指向缓冲区的指针，函数调用成功后，该缓冲区中将保存所有可用系统的定位符。
- *ioBufferSize* (unsigned 32bit)，输入/输出 – 指定函数调用前 outBuffer 的大小。在函数调用之后，它保存写入 outBuffer 的字节数。

注意：对于以太网版本的控制器，此函数不可用。

示例程序：

```
char outBuffer[4096];
unsigned int bufferSize = sizeof(outBuffer);
NT_STATUS result = NT_FindSystems("", outBuffer, &bufferSize);
if(result == NT_OK){
       // outBuffer holds the locator strings, separated by '\n'
       // bufferSize holds the number of bytes written to outBuffer
}
```

NT_GetNumberOfChannels

函数原型：

```
NT_STATUS NT_GetNumberOfChannels(NT_INDEX systemIndex,
                                 unsigned int *channels);
```

功能描述：

此函数用于获取当前系统的可用通道数，即定位台的最大通道数。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channels* (unsigned 32bit)，输出 – 在函数调用成功后保存系统的可用通道数。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if(result != NT_OK){
       // handle error...
}
unsigned int number;
result = NT_GetNumberOfChannels(ntHandle,&number);
if (result == NT_OK) {
       // number holds the number of channels of the system
}
```

NT_GetSystemLocator

函数原型：

```
NT_STATUS NT_GetSystemLocator(NT_INDEX systemIndex,
                              char *outBuffer,
                              unsigned int *ioBufferSize);
```

功能描述：

此函数用于获取指定系统的定位符，获取到的定位符被保存至 outBuffer。调用此函数时，调用者必须传递一个足够大的缓冲区，并将缓冲区大小写入 ioBufferSize。调用成功后，函数将系统定位符写入到缓冲区 outBuffer 中，并将定位符字节数保存到 ioBufferSize 中。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。。
- *outBuffer* (char pointer)，输出 – 指向缓冲区的指针，用于在函数调用成功后保存系统的定位符。
- *ioBufferSize* (unsigned 32bit)，输入/输出 – 在函数调用之前指定 outBuffer 的大小。函数调用之后，保存写入 outBuffer 的字节数。

注意：对于以太网版本的控制器，此函数不可用。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
char outBuffer[4096];
unsigned int bufferSize = sizeof(outBuffer);
NT_STATUS result = NT_GetSystemLocator(ntHandle, outBuffer, &bufferSize);
if(result == NT_OK){
       // outBuffer holds the locator string
       // bufferSize holds the number of bytes written to outBuffer
}
```

NT_SetHCMEnabled

函数原型

```
NT_STATUS NT_SetHCMEnabled(NT_INDEX systemIndex,
                           unsigned int enabled);
```

功能描述：

此函数用于启用或禁用连接到系统的手动控制模块，以避免其在应用程序控制系统时产生干扰。（更多信息请参阅 2.5.1 节）。有三种模式可选：

- NT_HCM_DISABLED(0)：在此模式下，手动控制模块被禁用，屏幕也停止刷新。
- NT_HCM_ENABLED(1)：在此模式下，手动控制模块可使用，屏幕刷新正常。
- NT_HCM_CONTROLS_DISABLED(2)：在此模式下，手动控制模块被禁用，屏幕刷新正常。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。。
- *enabled* (unsigned 32bit)，输入 – 选择手动控制模块的工作模式。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if(result != NT_OK){
      // handle error...
}
// disable the Hand Control Module
result = NT_SetHCMEnabled(ntHandle,NT_HCM_DISABLED);
```

NT_GetVersionInfo

函数原型

```
NT_STATUS NT_GetVersionInfo(NT_INDEX systemIndex,
                                    char *infomation,
                                    unsigned int *size);
```

功能描述：

此函数用于获取控制器的版本信息。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *infomation(char)*，输出 – 返回版本信息的相关字符串。
- *size*(unsigned 32bit)，输入 – 字符串的长度。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if(result != NT_OK){
      // handle error...
}
// disable the Hand Control Module
char info[200];
unsigned int size = sizeof(info);
result = NT_GetVersionInfo(ntHandle, info, &size);
```

3.2 同步通信函数

说明：同步通信模式和异步通信模式下的所有函数都是线程安全的。

NT_GetPosition_S

函数原型

```
NT_STATUS NT_GetPosition_S(NT_INDEX systemIndex,
                           NT_INDEX channelIndex,
                           signed int *position);
```

功能描述：

此函数用于查询当前指定通道上定位台的位置。此函数只能用于带有光栅尺的定位台，传感器电源必须启用（请参阅函数 NT_SetSensorEnabled_S）。若不满足以上条件，函数将终止执行并返回错误代码。

此函数只能用于线性定位台，对旋转定位台使用此函数将返回错误代码。旋转定位台获取位置应使用函数 NT_GetAngle_S。

入口参数：

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- *position* (signed 32bit)，输出 – 函数调用成功后，保存定位台当前的位置值（单位为纳米）。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if(result != NT_OK){
       // handle error...
}
// get current position
signed int position;
result = NT_GetPosition_S(ntHandle,0,&position);
if(result == NT_OK){
       // current position is in position variable
}
```

NT_GetSensorEnabled_S

函数原型：

```
NT_STATUS NT_GetSensorEnabled_S(NT_INDEX systemIndex,
                                NT_INDEX channelIndex,
                                unsigned int *enabled);
```

功能描述：

此函数用于查询当前连接到系统的指定通道上的传感器的工作模式。有关传感器模式的更多信息，请参阅 2.4.1 节。

入口参数：

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- *enabled* (unsigned 32bit)，输出 – 函数调用成功后，保存传感器当前的工作模式，可能的返回值如下：
  - NT_SENSOR_DINTBLED(0)
  - NT_SENSOR_ENABLED(1)

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if(result != NT_OK){
      // handle error...
}
// read sensor mode
unsigned int enabled;
result = NT_GetSensorEnabled_S(ntHandle,0,&enabled);
```

NT_GetStatus_S

函数原型：

```
NT_STATUS NT_GetStatus_S(NT_INDEX systemIndex,
                         NT_INDEX channelIndex,
                         unsigned int *status);
```

功能描述：

此函数用于获取当前指定通道上定位台的运动状态（有关运动状态代码的更多信息，请参阅 4.3 节）。此函数可用于检查之前发出的移动命令是否已经完成。

注意， Status 刷新时间间隔大约为 2ms。

入口参数：

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- *status* (unsigned 32bit)，输出 – 函数调用成功后，保存定位台当前的运动状态。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if(result != NT_OK){
       // handle error...
}
// get current status
unsigned int status;
result = NT_GetStatus_S(0,0,&status);
```

NT_GetVoltageLevel_S

函数原型：

```
NT_STATUS NT_GetVoltageLevel_S(NT_INDEX systemIndex,
                               NT_INDEX channelIndex,
                               unsigned int *level);
```

功能描述：

此函数用于查询当前施加在指定通道定位台压电元件上的电压电平。**注意，**此函数需要在调用函数 NT_ScanMoveAbsolute_S 或者 NT_ScanMoveRelative_S 后使用，这些函数用来设置目标通道上的输出电压电平。

注意，扫描运动可能需要很长时间（取决于扫描速度和扫描距离），请确保扫描运动结束后再调用此函数，否则获取的电压电平没有意义。

入口参数：

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- *level* (unsigned 32bit)，输出 – 函数调用成功后，保存当前施加在压电元件上的电压电平。输出电压的范围为 0...4,095，其中 0 对应 0V，4,095 对应 100V。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if(result != NT_OK){
      // handle error...
}
// get current status
unsigned int status;
while(status != NT_STOPPED_STATUS){
    result = NT_GetStatus(ntHandle,0,&status)
    Sleep(100);
}
// get current voltage level
unsigned int level;
result = NT_GetVoltageLevel_S(ntHandle,0,&level);
```

NT_GotoPositionAbsolute_S

函数原型：

```
NT_STATUS NT_GotoPositionAbsolute_S(NT_INDEX systemIndex,
                                    NT_INDEX channelIndex,
                                    signed int position);
```

功能描述：

此函数用于控制指定通道上的定位台在闭环控制下移动到目标位置。此函数只能用于带有光栅尺的定位台，并且传感器电源必须启用（参阅函数 NT_SetSensorEnabled_S）。若不满足以上条件，函数将终止执行并返回错误代码。

执行此函数时，定位台的运动状态为 NT_TARGET_STATUS。默认情况下，当定位台达到目标位置后，将保持在目标位置，这对补偿漂移效应非常有用。此时定位台的运动状态为 NT_HOLDING_STATUS（请参阅函数 NT_GetStatus_S）。如果想让定位台退出闭环保持状态，请调用函数 NT_SetClosedLoopHoldEnabled_S。

默认情况下，如果在执行命令时检测到定位台已经运动到极限位置，运动将中止。如果不需要此功能，可以用函数 NT_LimitEnable_S 禁用极限判定功能。

此函数只能用于线性定位台，对旋转定位台使用此函数将返回错误代码。控制旋转定位台移动到目标位置应使用函数 NT_GotoAngleAbsolute_S。

入口参数：

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- *position* (signed 32bit)，输入– 绝对位置，以纳米为单位。

示例程序：

```
// move to zero position
result = NT_GotoPositionAbsolute_S(ntHandle,0,0);
```

NT_GotoPositionRelative_S

函数原型：

```
NT_STATUS NT_GotoPositionRelative_S(NT_INDEX systemIndex,
                                    NT_INDEX channelIndex,
                                    signed int diff);
```

功能描述：

此函数用于控制指定通道上的定位台在闭环控制下，移动到到相对于当前位置 diff 处的位置。此函数只能用于带有光栅尺的定位台，传感器电源必须启用（参阅函数 NT_SetSensorEnabled_S）。若不满足以上条件，函数将终止执行并返回错误代码。

默认情况下，如果在上一个命令完成之前系统又收到了相对定位命令，则定位台最终移动的距离值是两次命令的中 diff 值的叠加。如果不希望这样做，可以使用函数 NT_SetAccumulateRelativePositions_S 禁用此功能。

执行此函数时，定位台的运动状态为 NT_TARGET_STATUS。默认情况下，当定位台达到目标位置后，将保持在目标位置，这对补偿漂移效应非常有用，此时定位台的运动状态为 NT_HOLDING_STATUS（请参阅函数 NT_GetStatus_S）。如果想让定位台退出闭环保持状态，请调用函数 NT_SetClosedLoopHoldEnabled_S。

默认情况下，如果在执行命令时检测到定位台已经运动到极限位置，运动将中止。如果不需要此功能，可以用函数 NT_LimitEnable_S 禁用极限判定功能。

此函数只能用于线性定位台，对旋转定位台使用此函数将返回错误代码。控制旋转定位台执行相对位置运动应使用函数 NT_GotoAngleRelative_S。

入口参数：

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- *diff* (signed 32bit)，输入 – 相对位置，以纳米为单位。

示例程序：

```
// move 2 micro meters in negative direction
result = NT_GotoPositionRelative_S(ntHandle,0,-2000);
```

NT_ScanMoveAbsolute_S

函数原型：

```
NT_STATUS NT_ScanMoveAbsolute_S(NT_INDEX systemIndex,
                                NT_INDEX channelIndex,
                                unsigned int target,
                                unsigned int scanSpeed);
```

功能描述：

此函数用于直接控制施加到指定通道定位台压电元件上的电压电平，从当前值以速度 scanSpeed，线性变化至 target。定位台将执行一个从当前位置到目标位置的扫描移动（开环）。

执行此命令时，定位台的运动状态为 NT_SCANNING_STATUS（更多信息请参阅函数 NT_GetStatus_S）。

入口参数：

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- *target* (unsigned 32bit)，输入 – 绝对目标位置。有效输入范围为 0...4,095，0 对应 0V，4,095 对于 100V。
- *scanSpeed* (unsigned 32bit)，输入 – 扫描速度。有效输入范围是 1 ... 4,095,000。将 0-100V 等分成 4096 份，扫描速度表示每秒扫描电压变化的份数。比如，值为 1 时，在从 0 扫描到 4,095 需要 4,095 秒，而全速扫描（值为 4,095,000）只需一毫秒。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if(result != NT_OK){
      // handle error...
}
// scan to 50V
result = NT_ScanMoveAbsolute_S(ntHandle,0,2048,10000);
// scan to 0V within two seconds
Result = NT_ScanMoveAbsolute_S(ntHandle,0,0,1024);
```

NT_ScanMoveRelative_S

函数原型：

```
NT_STATUS NT_ScanMoveRelative_S(NT_INDEX systemIndex,
                                NT_INDEX channelIndex,
                                signed int diff,
                                unsigned int scanSpeed);
```

功能描述：

此函数用于直接控制施加到指定通道定位台压电元件上的电压电平，从当前值以速度 scanSpeed，线性变化 diff。定位台将执行一个相对于当前位置的扫描移动（开环）。

执行此命令时，定位台的运动状态为 NT_SCANNING_STATUS（请参阅函数 NT_GetStatus_S）。

入口参数:

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- *diff* (signed 32bit)，输入 – 相对目标位置。有效输入范围为-4095...4,095。 如果控制器最终得到的绝对扫描目标超过 0...4095 的有效输入范围，则扫描运动将在边界处停止。
- *scanSpeed* (unsigned 32bit)，输入 – 扫描速度。有效输入范围是 1 ... 4,095,000。将 0-100V 等分成 4096 份，扫描速度表示每秒扫描电压变化的份数。比如，值为 1 时，在从 0 扫描到 4,095 需要 4,095 秒，而全速扫描（值为 4,095,000）只需一毫秒。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if(result != NT_OK){
      // handle error...
}
// scan to 50V
result = NT_ScanMoveAbsolute_S(ntHandle,0,2048,10000);
// scan to 25V within one second
result = NT_ScanMoveRelative_S(ntHandle,0,-1024,1024);
// scan to (automatically stop at) 0V
result = NT_ScanMoveRelative_S(ntHandle,0,-3000,1024);
```

NT_SetAccumulateRelativePositions_S

函数原型：

```
NT_STATUS NT_SetAccumulateRelativePositions_S(NT_INDEX systemIndex,
                                              NT_INDEX channelIndex,
                                              unsigned int accumulate);
```

功能描述：

此函数需要与函数 NT_GotoPositionRelative_S 和 NT_GotoAngleRelative_S 一起使用，用于设置相对位置命令是否叠加。默认情况下，相对位置命令将被叠加，在上一次的闭环对位置命令未执行完之前，所有新的相对位置设定值都将叠加到之前的目标位置之上。如果相对位置命令不叠加，则新的相对位置命令将被立即执行。

示例：假设定位台当前处于零位置。连续快速发出两个相对移动命令，均以+1mm 作为相对目标。激活累积功能后，最终位置将为 2mm。在未激活累积的情况下，最终位置将发生变化（例如 1.2mm），取决于第二条命令到达的时间。

入口参数：

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- *accumulate* (unsigned 32bit)，输入 – 模式选择。有效输入为：
  NT_NO_ACCUMULATE_RELATIVE_POSITIONS(0)
  NT_ACCUMULATE_RELATIVE_POSITIONS(1)

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if(result != NT_OK){
       // handle error...
}
// disable accumulation of relative movement commands in closed-loop mode
result = NT_SetAccumulateRelativePositions_S(ntHandle,
0,NT_NO_ACCUMULATE_RELATIVE_POSITIONS);
```

NT_SetPosition_S

函数原型：

```
NT_STATUS NT_SetPosition_S(NT_INDEX systemIndex,
                           NT_INDEX channelIndex,
                           signed int position);
```

功能描述：

对于装有传感器的定位台，此函数用于将指定通道上定位台的位置或者角度重新定义为特定的值。定位器的测量标尺将相应地移动，但是定位器不会产生任何移动。

有关定义位置的更多信息，请参阅 2.4.2 节。

入口参数：

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- *position* (signed 32bit)，输入 – 定义当前位置为此值。如果是旋转定位台该参数范围是 0~359999999，单位 u°。旋转定位台的转数(revolution)默认设置为 0。

示例程序：

```
// set the position of the first positioner connected to the system to 3.5mm
result = NT_SetPosition_S(ntHandle,0,3500000);
```

NT_SetSensorEnabled_S

函数原型：

```
NT_STATUS NT_SetSensorEnabled_S(NT_INDEX systemIndex,
                                NT_INDEX channelIndex,
                                unsigned int enabled);
```

功能描述：

此函数用于设置当前系统中指定通道上的传感器的工作模式。

可以通过函数 NT_GetSensorEnabled_S 获取配置结果。有关传感器模式的更多信息，请参阅 2.4.1 节。

入口参数：

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 - 通道索引，索引值从 0 开始。
- *enabled* (unsigned 32bit)，输入 - 设置传感器的工作模式，有效输入为：
  - NT_SENSOR_DISABLED(0)
  - NT_SENSOR_ENABLED(1)

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if(result != NT_OK){
       // handle error...
}
// disable sensors
result = NT_SetSensorEnabled_S(ntHandle,0,NT_SENSOR_DISABLED);
```

NT_StepMove_S

函数原型：

```
NT_STATUS NT_StepMove_S(NT_INDEX systemIndex,
                        NT_INDEX channelIndex,
                        signed int steps,
                        unsigned int amplitude,
                        unsigned int frequency);
```

功能描述：

此函数用于控制指定通道上的定位台执行步进移动（开环）。此时，控制器通道输出的电压波形为锯齿波，参数 steps，amplitude，frequency 分别用来控制锯齿波的个数，幅值和频率。另外，step 的正负对应锯齿波的两种不同模式（第一种是电压值从默认电平上升至最大值，再陡落至最小值，再上升至默认电平。第二种是电压值先下降至最小值，再陡升至最大值，再下降至默认电平），对应定位台不同的运动方向。**注意，控制器输出的默认电平是 50v，其输出的锯齿波的起点及终点均为默认电平，且沿默认电平中心对称。**

当控制器收到命令 NT_Stop_S 时，控制器将停止输出波形，输出电压恢复至默认电平。

执行此命令时，定位台的运动状态为 NT_STEPPING_STATUS（请参阅函数 NT_GetStatus_S）。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- *steps* (signed 32bit)，输入 - 设置控制器输出锯齿波的个数及模式。有效的输入范围为-30,000...30,000。值为 0 时定位台将停止，值为±30,000 时定位台将执行无限制的移动，直到调用函数 NT_Stop_S 才会停止。steps **值越大，定位台行程越大。**
- *amplitude* (unsigned 32bit)，输入 - 设置控制器输出的锯齿波的幅值。该参数必须以 12 位值的形式给出（范围 0...4,095）。0 对应 0V，4,095 对应 100V。注意，DLL 中限制 amplitude 的最小输入为 100，若输入小于 100，函数将会终止执行并返回错误代码 NT_AMPLITUDE_TOO_LOW_ERROR 。**amplitude 值越大，单个波形下定位台的行程越大。**
- *frequency* (unsigned 32bit)，输入 - 设置控制器输出锯齿波的频率，单位为 Hz，有效的范围为 1...18,500。**frequency 值越大，定位台移动越快。**

示例程序：

```
// perform 100 steps with full amplitude at 1kHz
result = NT_StepMove_S(ntHandle,0,100,4095,1000);
```

NT_Stop_S

函数原型：

```
NT_STATUS NT_Stop_S(NT_INDEX systemIndex,
                    NT_INDEX channelIndex);
```

功能描述：

停止指定通道上定位台正在执行的任何形式的移动，无论是开环模式、闭环模式还是扫描模式。**注意，此命令也会关闭闭环命令的位置保持功能。**

调用此函数后，定位台的运动状态为 NT_STOPPED_STATUS（请参阅函数 NT_GetStatus_S）。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if(result != NT_OK){
       // handle error...
}
// perform 1,000 steps with half amplitude at 1kHz
result = NT_StepMove_S(ntHandle,0,1000,2048,1000);
// stop
result = NT_Stop_S(ntHandle,0);
/*
Note: In this example the positioner will start executing 1,000 steps after the
NT_StepMove_S command. Since NT_Stop_S is called right away, the number of steps
actually executed (before the movement is aborted) is a timing issue and may depend
on your PC machine speed and/or the USB connection to the hardware.
*/
```

NT_SetPIDParameter_S

函数原型：

```
NT_STATUS  NT_SetPIDParameter_S(NT_INDEX systemIndex,
                                NT_INDEX channelIndex,
                                unsigned int type,
                                float temp);
```

功能描述：

此函数用于设置闭环运动中 PID 控制的比例系数（P），积分系数（I），微分系数（D）。**注意，此函数每一次调用只能设置 PID 三个系数中的一个。**

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- *type* (unsigned 32bit)， 输入 – 设置输入参数的类型：
  - P:NT_PID1_P(0)
  - I:NT_PID1_I(1)
  - D:NT_PID1_D(2)
- *temp* (float)，输入 – 设置的 PID 参数值。默认 P 值为 0.015，I 值为 0.0005，D 值为 0。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if (result != NT_OK) {
        // handle error...
}
// set P of PID is 0.015
result = NT_SetPIDParameter_S(ntHandle,0,NT_PID1_P, 0.015);
```

NT_SetClosedLoopMaxFrequency_S

函数原型：

```
NT_STATUS  NT_SetClosedLoopMaxFrequency_S(NT_INDEX systemIndex,
                                          NT_INDEX channelIndex,
                                          unsigned int frequency);
```

功能描述：

对于带有传感器的定位台，此函数用于设置定位台执行闭环命令时被驱动的最大频率。每个通道可以单独设置此参数。只需要设置一次，后续的所有闭环指令都会采用新的驱动频率。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- *frequency(unsigned 32bit)*，输入 – 设置的频率值。范围为 50...8000。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if (result != NT_OK) {
        // handle error...
}
// set maximum closed-loop frequency to 4000
result = NT_SetClosedLoopMaxFrequency_S(ntHandle,0,4000);
```

NT_SetClosedLoopMoveSpeed_S

函数原型：

```
NT_STATUS  NT_SetClosedLoopMoveSpeed_S(NT_INDEX systemIndex,
                                       NT_INDEX channelIndex,
                                       unsigned int type,
                                       unsigned int speed);
```

功能描述：

此函数用来配置指定通道在执行闭环命令时的速度控制功能。默认情况下，速度控制被禁用，此时闭环运动只受最大驱动频率的影响（更多信息请参阅函数 NT_SetClosedLoopMaxFrequency_S）。开启速度控制后，如果当前控制器输出的驱动频率太小不能满足当前设定的速度，此时应该提高驱动频率，否则可能无法达到设定的移动速度。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- *type* (unsigned 32bit)，输入 – 可选的输入值如下：
  - NT_SPEED_DISABLED(0)
  - NT_SPEED_ENABLED(16)
- *speed* (unsigned 32bit)，输入 - 设置定位台匀速运动的速度值,线性定位台的单位是 nm/s,旋转定位台的单位是 u°/s。有效值范围是 0...5,000,000。0 默认禁用速度控制。

示例程序：

```
//Activate the speed control function, with a speed value of 100000nm/s
Result = NT_SetClosedLoopMoveSpeed_S(ntHandle,0,NT_SPEED_ENABLED,100000);
```

NT_GetClosedLoopMoveSpeed_S

函数原型：

```
NT_STATUS  NT_GetClosedLoopMoveSpeed_S(NT_INDEX systemIndex,
                                       NT_INDEX channelIndex,
                                       unsigned int *speed);
```

功能描述：

此函数用于用于获取函数 NT_SetClosedLoopMoveSpeed_S 设置的闭环移动速度。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- *speed(unsigned 32bit)*，输出 – 函数调用成功后，保存获取的闭环移动速度值。线性定位台的单位是 nm/s,旋转定位台的单位是 u°/s。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if (result != NT_OK) {
        // handle error...
}
//get the speed of channel 0
unsigned int speed;
result = NT_GetClosedLoopMoveSpeed_S(ntHandle,0,&speed);
```

NT_CalibrateSensor_S

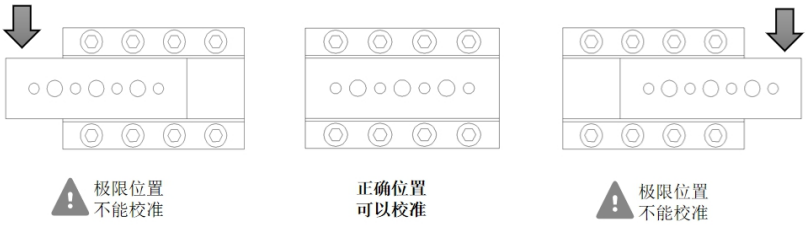
函数原型：

```
NT_STATUS  NT_CalibrateSensor_S(NT_INDEX systemIndex,
                                NT_INDEX channelIndex)
```

功能描述：

此函数用于提高定位台位置计算的精度。此函数只能用于带有光栅尺的定位台，传感器电源必须启用（请参阅函数 NT_SetSensorEnabled_S）。若不满足以上条件，函数将终止执行并返回错误代码。

如果机械设置发生变化（不同的定位器连接到不同的通道），则应为每个通道调用一次此函数。校准数据将保存到非易失性存储器中。如果机械设置保持不变，则控制器重新上电后无需调用此函数。在校准过程中，线性定位器将执行 40um 的移动，因此必须确保在执行校准命令前，定位台有足够的移动自由度。



入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。

示例程序：

```
//start calibration routine
result =  NT_CalibrateSensor_S(ntHandle,0);
unsigned int status;
do {
    NT_GetStatus_S(ntHandle,0,&status);
} while (status != NT_STOPPED_STATUS);
// done calibrating
```

NT_EmergencyStop_S

函数原型：

```
NT_STATUS  NT_EmergencyStop_S(NT_INDEX systemIndex,
                              NT_INDEX channelIndex)
```

功能描述：

此函数用于失能指定通道的模拟电压输出。默认情况下，所有通道的模拟电压输出都是使能的。一旦失能指定通道的模拟电压输出，后续针对该通道的所有命令都将无效。失能通道的模拟电压输出可以用于处理紧急情况。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。

示例程序：

```
unsigned int ntHandle;
const char loc[] = “usb:id:5535186640”;
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync”);
if (result != NT_OK) {
        // handle error...
}

//enable output of channel 0
result = NT_EmergencyStop_S(ntHandle,0);
```

NT_SetClosedLoopHoldEnabled_S

函数原型：

```
NT_STATUS  NT_SetClosedLoopHoldEnabled_S(NT_INDEX systemIndex,
                                         NT_INDEX channelIndex,
                                         unsigned int enabled)
```

功能描述：

此函数用于开启或关闭闭环保持功能。默认状态下，定位台在完成闭环命令后，将保持在目标位置，此时定位台的运动状态为 NT_HOLDING_STATUS。保持在目标位置时，由于闭环控制始终在对定位台的位置进行动态调节，因此定位台将产生微小的振动。如果不希望定位台在到达目标位置后产生振动，可以调用此函数关闭闭环保持功能。

闭环保持功能关闭后，定位台的运动状态为 NT_NO_HOLDING_STATUS（更多信息请参阅函数 NT_GetStatus_S）。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- *enabled (unsigned 32bit)*，输入 – 设置开启或关闭闭环保持功能，有效输入为
  - NT_CLOSELOOP_DISABLED(0)
  - NT_CLOSELOOP_ENABLED(1)

示例程序：

```
unsigned int ntHandle;
const char loc[] = “usb:id:5535186640”;
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync”);
if (result != NT_OK) {
        // handle error...
}
result = NT_SetClosedLoopHoldEnabled_S(ntHandle,0,NT_CLOSELOOP_ENABLED);
```

NT_FindReferenceMark_S

函数原型：

```
NT_STATUS   NT_FindReferenceMark_S(NT_INDEX systemIndex,
                                   NT_INDEX channelIndex,
                                    unsigned int direction,
                                    unsigned int enabled)
```

功能描述：

此函数只可应用于带有零点的定位台，用于寻找物理零点。成功找到零点后，定位台将保持在零点，此时定位台的移动状态为 NT_HOLDING_STATUS。如果设置了自动归零标志，在定位台到达物理零点后，会将当前位置和角度设置为零。

该命令执行后，调用函数 NT_GetPhysicalPositionKnown_S 查看零点是否成功找到。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- *direction* (unsigned 32bit)，输入 – 寻零点的方向，指从哪个方向开始找零点，有效输入为
  - NT_FIND_FORWARD(1)
  - NT_FIND_BACKWARD(2)
- *enabled* (unsigned 32bit)，输入 – 找到零点之后位移值是否自动归零，有效输入为
  - NT_AUTO_ZERO_DISABLED(0)
  - NT_AUTO_ZERO_ENABLED(1)

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if (result != NT_OK) {
        // handle error...
}
//move to reference mark and set to zero
result =
NT_FindReferenceMark_S(ntHandle,0,NT_FIND_FORWARD,NT_AUTO_ZERO_ENABLED);
```

NT_GetPhysicalPositionKnown_S

函数原型：

```
NT_STATUS  NT_GetPhysicalPositionKnown_S(NT_INDEX systemIndex,
                                         NT_INDEX channelIndex,
                                         unsigned int *known)
```

功能描述：

此函数用于查看定位台是否“知道”其物理位置。在控制器上电后，定位台的物理位置是未知的，默认情况下会将定位台当前所处的位置假设为零位置。

通过调用函数 NT_FindReferenceMark_S 找到定位台的零点后，物理位置就变为已知。如果软件程序重新启动并连接到一个一直在线的系统，此功能非常有用。如果物理位置已知，则不需要再次调用函数 NT_FindReferenceMark_S。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- *known* (unsigned 32bit)，输出 – 函数调用成功后，保存定位台是否已经“知道”其物理位置，可能的返回值如下：
  - NT_PHYSICAL_POSITION_UNKNOWN(0)
  - NT_PHYSICAL_POSITION_KNOWN(1)

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if(result != NT_OK){
       // handle error...
}
//check whether the physical position of channel 0 is known
unsigned int known;
result = NT_GetPhysicalPositionKnown_S(ntHandle,0,&known);
```

NT_LimitEnable_S

函数原型：

```
NT_STATUS  NT_LimitEnable_S(NT_INDEX systemIndex,
                            NT_INDEX channelIndex,
                            unsigned int enabled)
```

功能描述：

此函数用于开启/关闭物理极限位置判定功能，此函数只适用于线性定位台的闭环运动控制。默认情况下，极限位置判定功能是开启的，线性定位台在运动到两端的极限位置后，将保持在极限位置。如果关闭极限位置判定功能，当线性定位台移动到极限位置后，不会自动停止，继续执行闭环运动。**注意，这种情况可能会损坏定位台。**

旋转定位台没有极限位置，可以持续运动。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- *enabled* (unsigned 32bit)，输入 – 是否开启极限位置判定功能，有效输入为
  - NT_LIMIT_DISABLED(0)
  - NT_LIMIT_ENABLED(1)

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if (result != NT_OK) {
        // handle error...
}
// Disabled the limit function
result = NT_LimitEnable_S(ntHandle,0,NT_LIMIT_DISABLED);
```

NT_DisPlacementClear_S

函数原型：

```
NT_STATUS  NT_DisPlacementClear_S(NT_INDEX systemIndex,
                                  NT_INDEX channelIndex)
```

功能描述：

此函数用于将指定通道的位移值清零。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if (result != NT_OK) {
        // handle error...
}
//1 channel reset
result = NT_DisPlacementClear_S(ntHandle,0);
```

NT_GetClosedLoopMoveAcceleration_S

函数原型：

```
NT_STATUS  NT_GetClosedLoopMoveAcceleration_S
                            (NT_INDEX systemIndex,
                             NT_INDEX channelIndex,
                             unsignedint *acceleration);
```

功能描述：

此函数用于获取当前为指定通达配置的闭环运动加速度。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- *acceleration* (unsigned 32bit)，输出 – 函数调用成功后，保存指定通道的闭环运动加速度值。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if (result != NT_OK) {
        // handle error...
}
unsigned int acceleration;
result = NT_GetClosedLoopMoveAcceleration_S(ntHandle,0,&acceleration);
if (result == NT_OK) {
        // acceleration holds the current movement acceleration
}
```

NT_GetAngle_S

函数原型：

```
NT_STATUS NT_GetAngle_S(NT_INDEX systemIndex,
                        NT_INDEX channelIndex,
                        unsigned int *angle,
                        signed int *revolution);
```

功能描述：

此函数用于查询当前指定通道上旋转定位台的角度值。此函数只能用于带有光栅尺的定位台，传感器电源必须启用（请参阅函数 NT_SetSensorEnabled_S）。若不满足以上条件，函数将终止执行并返回错误代码。

注意，此函数只能用于旋转定位台，否则，函数将终止执行并返回错误代码。

旋转定位台位置的定义包括两个部分：角度值(angle)和圈数值(revolution)。旋转一圈等于 360°。定位台越过零点边界后时，angle 归零，revolution 自动加一或减一。角度值的范围是 0~359,999,999。

关于更多角度 angle 和圈数 revolution 的信息请看 2.4.3"旋转 Motor"。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- *angle(unsigned 32bit)*，输出 – 获取角度值，单位 u° 。
- *revolution(signed 32bit)*，输出 – 获取当前圈数 ° 。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if (result != NT_OK) {
        // handle error...
}
//get current position
unsigned int angle,revolution;
result = NT_GetAngle_S(ntHandle,0,&angle,&revolution);
if(result == NT_OK){
          //angle and revolution parameters have been updated
}
```

NT_SetSensorType_S

函数原型：

```
NT_STATUS NT_SetSensorType_S(NT_INDEX systemIndex,
                             NT_INDEX channelIndex,
                             unsigned int type);
```

功能描述：

当使用带有传感器的定位台时，此函数用于告诉控制器，指定通道上连接了哪种定位台。类型将影响位置的计算以及适用于通道的函数（例如，NT_GetPosition_S 和 NT_GetAngle_S）

请注意，每个通道都将设置存储到非易失存储器中。因此，不需要在控制器每次上电后调用此函数。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 - 通道索引，索引值从 0 开始。
- *type(unsigned 32bit)*，输入 – 指定 Motor 类型。当前可用类型如下：
  - LINEAR(1)
  - GON59(2)
  - GON78(3)
  - RS80(4)
  - RS60(5)
  - RS50(9)
  - RS40(8)
  - RS20(7)

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if (result != NT_OK) {
        // handle error...
}
//configure linear sensor for  channel 0
result = NT_SetSensorType_S(ntHandle,0,LINEAR);
```

NT_GotoAngleAbsolute_S

函数原型：

```
NT_STATUS NT_GotoAngleAbsolute_S(NT_INDEX systemIndex,
                                 NT_INDEX channelIndex,
                                 unsigned int angle,
                                 signed int revolution);
```

功能描述：

此函数用于控制指定通道上的定位台在闭环控制下旋转至特定的角度。此函数只能用于带有光栅尺的定位台，并且传感器电源必须启用（请参阅函数 NT_SetSensorEnabled_S）。若不满足以上条件，函数将终止执行并返回错误代码。

只有当通道连接旋转定位台，并且传感器类型配置为旋转型时，才可以使用此函数（详见 NT_SetSensorType_S），否则将返回错误代码。控制直线定位台移动到目标位置请使用函数 NT_GotoPositionAbsolute_S。

执行此函数时，定位台的运动状态为 NT_TARGET_STATUS。默认情况下，当定位台达到目标角度后，将保持在目标位置，这对补偿漂移效应非常有用，此时定位台的运动状态为 NT_HOLDING_STATUS（请参阅函数 NT_GetStatus_S）。如果想让定位台退出闭环保持状态，请调用函数 NT_SetClosedLoopHoldEnabled_S。

更多旋转运动介绍请参阅 2.4.3 节“旋转定位台”。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 - 通道索引，索引值从 0 开始。
- *angle* (unsigned 32bit)，输入 – 绝对运动角度，单位 u°。有效范围：0~359,999,999。
- *revolution* (signed 32bit)，输入 – 绝对运动圈数。有效范围：-32,768~32,767。

示例程序：

```
unsigned int ntHandle;
const char loc[] = “usb:id:5535186640”;
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, “sync”);
if (result != NT_OK) {
        // handle error...
}
//move to 90° angle for channel 0
result = NT_GotoAngleAbsolute_S(ntHandle,0,90000000,0);
```

NT_GotoAngleRelative_S

函数原型：

```
NT_STATUS NT_GotoAngleRelative_S(NT_INDEX systemIndex,
                                 NT_INDEX channelIndex,
                                 signed int angleDiff,
                                 signed int revolutionDiff);
```

功能描述：

此函数用于控制指定通道上的定位台在闭环控制下，旋转到相对于当前位置特定角度的位置。此函数只能用于带有光栅尺的定位台，并且传感器电源必须启用（请参阅函数 NT_SetSensorEnabled_S）。若不满足以上条件，函数将终止执行并返回错误代码。

只有当通道连接旋转定位台，并且传感器类型配置为旋转型时，才可以使用此函数（详见 NT_SetSensorType_S），否则将返回错误代码。控制直线定位台执行相对位置移动请使用函数 NT_GotoPositionAbsolute_S。

默认情况下，如果在上一个命令完成之前系统又收到了相对定位命令，则定位台最终移动的距离值是两次命令的中角度值的叠加。如果不希望这样做，可以使用函数 NT_SetAccumulateRelativePositions_S 禁用此功能。

执行此函数时，定位台的运动状态为 NT_TARGET_STATUS。默认情况下，当定位台达到目标角度后，将保持在目标位置，这对补偿漂移效应非常有用，此时定位台的运动状态为 NT_HOLDING_STATUS（请参阅函数 NT_GetStatus_S）。如果想让定位台退出闭环保持状态，请调用函数 NT_SetClosedLoopHoldEnabled_S。

更多旋转运动介绍请参阅 2.4.3 节“旋转定位台”。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 - 通道索引，索引值从 0 开始。
- *angleDiff(signed 32bit)*，输入 – 相对运动角度值，单位 u°。有效范围 -359,999,999~359,999,999
- *revolution (signed 32bit)*，输入 – 相对运动圈数。有效范围：-32,768~32,767

示例程序：

```
unsigned int ntHandle;
const char loc[] = “usb:id:5535186640”;
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, “sync”);
if (result != NT_OK) {
        // handle error...
}
//do one full turn plus another 90° in negative direction
result = NT_GotoAngleRelative_S(ntHandle,0, -90000000, -1);
```

NT_GetSensorType_S

函数原型：

```
NT_STATUS NT_GetSensorType_S(NT_INDEX systemIndex,
                             NT_INDEX channelIndex,
                             unsigned int *type);
```

功能描述：

当使用带有传感器的定位台时，此函数用于获取当前定位台的类型。例如：LINEAR。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 - 通道索引，索引值从 0 开始。
- **type(unsigned 32bit)*，输出 – Motor 类型。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if (result != NT_OK) {
        // handle error...
}
//configure linear sensor for  channel 0
unsigned int type;
result = NT_GetSensorType_S(ntHandle,0,&type);
```

NT_SetClosedLoopMoveAcceleration_S

函数原型：

```
NT_STATUS NT_SetClosedLoopMoveAcceleration_S(NT_INDEX systemIndex,
                                             NT_INDEX channelIndex,
                                             unsigned int enabled,
                                             unsigned int acceleration);
```

功能描述：

此函数用来配置指定通道在执行闭环命令时的加速度控制功能。能进行加速度使能（NT_ASPEED_ENABLED）。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- *type* (unsigned 32bit)，输入 – 可选的输入值如下：
  - NT_ASPEED_DISABLED(0)
  - NT_ASPEED_ENABLED(1)
- *acceleration*(unsigned 32bit)，输入 - 设置定位台的加速度，单位是 um/s²,旋转定位台的加速度单位是 m°/s²。有效值范围是 1-1,000。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if (result != NT_OK) {
        // handle error...
}
//Activate the speed control function, with a speed value of 100um/s²
Result = NT_SetClosedLoopMoveAcceleration_S(ntHandle,0,NT_ASPEED_ENABLED,100);
```

NT_GetClosedLoopMaxFrequency_S

函数原型：

```
NT_STATUS NT_GetCloseLoopMaxFrequency_S(NT_INDEX systemIndex,
                                        NT_INDEX channelIndex,
                                        unsigned int *frequency);
```

功能描述：

此函数用于获取设置的最大频率值。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- **frequency(unsigned 32bit)*，输出 – 最大频率值。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if (result != NT_OK) {
        // handle error...
}
//configure linear sensor for  channel 0
unsigned int frequency;
result = NT_GetCloseLoopMaxFrequency_S(ntHandle,0,&frequency);
```

NT_SetCheckPower_S

函数原型

```
NT_STATUS NT_SetCheckPower_S(NT_INDEX systemIndex);
```

功能描述：

此函数用于设置检测掉电的标志位为 1。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if(result != NT_OK){
       // handle error...
}
// Set power-off detection flag
result = NT_SetCheckPower_S(ntHandle);
```

NT_GetCheckPower_S

函数原型

```
NT_STATUS NT_GetCheckPower_S(NT_INDEX systemIndex,
                                        unsigned int *power);
```

功能描述：

此函数用于获取检测掉电的标志位。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *power*(unsigned 32bit)，输出 – 掉电标志位。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if(result != NT_OK){
       // handle error...
}
// Obtain power-off detection flag
unsigned int power;
result = NT_GetCheckPower_S(ntHandle, &power);
```

NT_SetPositionLimit_S

函数原型

```
NT_STATUS NT_SetPositionLimit_S(NT_INDEX systemIndex,
                                NT_INDEX channelIndex,
                                signed int minPosition,
                                signed int maxPosition);
```

功能描述：

对于带有集成传感器的定位器，此功能可用于设置线性定位器软极限。调用此函数后，定位器不会移动超过设置的极限位置。默认情况下，没有设置限制。

注：maxPosition 必须大于 minPosition，需要有一定的差值，否则定位器不会移动。如果两者具有相同的值，则禁用软件范围限制。此函数只能用于线性定位台，对旋转定位台使用此函数将返回错误代码。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- *minPosition*（signed 32bit），输入 – 以纳米为单位给出的绝对最小位置。
- *MaxPosition*（signed 32bit），输入 – 以纳米为单位给出的绝对最大位置。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if(result != NT_OK){
      // handle error...
}
// limit the travel range of  channel 0 to +/- 3mm
result = NT_SetPositionLimit_S(ntHandle,0,-3000000,3000000);
```

NT_GetPositionLimit_S

函数原型：

```
NT_STATUS NT_GetPositionLimit_S(NT_INDEX systemIndex,
                                NT_INDEX channelIndex,
                                signed int *minPosition,
                                signed int *maxPosition);
```

功能描述：

NT_SetPositionLimit_S 的反函数。可用于读取行程范围限制，当前配置用于线性通道。

注意：如果没有设置限制，则 minPosition 和 maxPosition 都将为 0。此函数只能用于线性定位台，对旋转定位台使用此函数将返回错误代码。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- **minPosition*(signed 32bit)，输出 – 如果调用成功，此参数将以纳米为单位保持最小位置。
- **maxPosition*(signed 32bit)，输出 – 如果调用成功，此参数将以纳米为单位保持最大位置。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if (result != NT_OK) {
        // handle error...
}
// retrieve the travel range limit of  channel 0
signed int min,max;
result = NT_GetPositionLimit_S(ntHandle,0,&min,&max);
```

NT_SetAngleLimit_S

函数原型

```
NT_STATUS NT_SetAngleLimit_S(NT_INDEX systemIndex,
                             NT_INDEX channelIndex,
                             signed int minRevolution,
                             signed int maxRevolution,
                             signed int minAngle,
                             signed int maxAngle);
```

功能描述：

对于带有集成传感器的定位器，这个功能用来设置旋转极限位置。（线性位置极限设置函数是 SA_SetAngleLimit_S）。默认禁用极限设置。

注意：maxAngle 必须大于 minAngle，需要有一定的差值，否则定位器不会移动。如果两者具有相同的值，则禁用软件范围限制。此函数只能用于旋转定位台，对线性定位台使用此函数将返回错误代码。

入口参数

- *systemIndex* (unsigned 32bit)，输入 - 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 - 通道索引，索引值从 0 开始。
- *minRevolution*（signed 32bit），输入 - 最小转数。有效范围是-32768-32767。
- *maxRevolution*（signed 32bit），输入 - 最大转数。有效范围是-32768-32767。
- *minAngle*（signed 32bit），输入 - 最小角度，单位为微度。有效范围是-359999999-359999999。
- *maxAngle*（signed 32bit），输入 - 最大角度，单位为微度。有效范围是-359999999-359999999。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if(result != NT_OK){
      // handle error...
}
// Limit the travel range of channel 0 to 0 turns -10 ° to 1 turn 10 °
result = NT_SetAngleLimit_S(ntHandle,0,0,1,- 10000000,10000000);
```

NT_GetAngleLimit_S

函数原型：

```
NT_STATUS NT_GetAngleLimit_S(NT_INDEX systemIndex,
                             NT_INDEX channelIndex,
                             signed int *minRevolution,
                             signed int *maxRevolution,
                             signed int *minAngle,
                             signed int *maxAngle);
```

功能描述：

NT_SetAngleLimit_S 的反函数。获取旋转限制范围，当前配置用于旋转通道。

注意：如果没有设置限制，则 minAngle 和 maxAngle 都将为 0。此函数只能用于旋转定位台，对线性定位台使用此函数将返回错误代码。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。
- **minRevolution*(signed 32bit)，输出 – 如果调用成功，此参数将保持最小转数。
- **maxRevolution*(signed 32bit)，输出 – 如果调用成功，此参数将保持最小转数。
- **minAngle*(signed 32bit)，输出 – 如果调用成功，此参数将以微度为单位保持最小位置。
- **maxAngle*(signed 32bit)，输出 – 如果调用成功，此参数将以微度为单位保持最大位置。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "sync");
if (result != NT_OK) {
        // handle error...
}
// retrieve the travel range limit of channel 0
signed int minA,maxA;
signed int minR, maxR;
result = NT_GetAngleLimit_S(ntHandle,0,&minR,&maxR,&minA,&maxA);
```

3.3 异步通信函数

异步通信模式的大多数函数与同步模式的大多数函数具有相同的功能。主要区别是函数名称的后缀。有关这一部分函数的详细信息，请参考同步函数，本节不在赘述。但是，有些函数的功能及用法与同步模式的函数不同，本节仅对这些函数的功能及用法进行介绍。

NT_ReceiveNextPacket_A

函数原型：

```
NT_STATUS NT_ReceiveNextPacket_A(NT_INDEX systemIndex,
                                 unsigned int timeout,
                                 NT_PACKET *packet);
```

功能描述：

此函数用于从系统接收数据包。如果接收到一个数据包，则该数据包将返回并由调用者使用。如果未收到任何数据包，则返回 NT_NO_PACKET_TYPE 类型的数据包。有关数据包的更多信息，请参阅第 2.3.3 节。

注意，参数 timeout 决定此函数的阻塞时间。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *timeout* (unsigned 32bit)，输入 – 指定等待传入数据的时间，单位为毫秒。如果在此时间段内没有收到数据包，函数将返回 NT_NO_PACKET_TYPE 类型的数据包。设置为 NT_TIMEOUT_INFINITE 时，函数将阻塞挂起直到接收到数据包。在这种情况下，可以通过调用函数 NT_CancelWaitForPacket_A 手动解除阻塞。
- *packet* (NT_PACKET)，输出 – 如果调用成功，将保存接收到的数据包。数据包各个字段的内容取决于数据包的类型。有关数据包类型的更多信息，请参阅 4.2 节。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "async");
if (result != NT_OK) {
        // handle error...
}
// request position of positioner
result = NT_GetPosition_A(ntHandle,0);
// receive answer, but don't wait longer than five seconds
NT_PACKET packet;
result = NT_ReceiveNextPacket_A(ntHandle,5000,&packet);
```

NT_DiscardPacket_A

函数原型：

```
NT_STATUS NT_DiscardPacket_A(NT_INDEX systemIndex);
```

功能描述：

此函数可用于丢弃接收到的数据包。如果数据包是较早收到的，但没有被使用（没有从接收缓冲区中删除），则此功能可能会将其丢弃。如果接收缓冲区为空，则此函数无效。 有关数据包的更多信息，请参阅第 2.3.3 节。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。

示例程序：

```
// request position of positioner
result = NT_GetPosition_A(ntHandle,0);
// poll until answer is there
NT_PACKET packet;
packet.packetType = NT_NO_PACKET_TYPE;
while (packet.packetType != NT_POSITION_PACKET_TYPE) {
       result = NT_LookAtNextPacket_A(ntHandle,1000,&packet);
       if (packet.packetType != NT_POSITION_PACKET_TYPE)
                 NT_DiscardPacket_A(ntHandle);
}
```

NT_CancelWaitForPacket_A

函数原型：

```
NT_STATUS NT_CancelWaitForPacket_A(NT_INDEX systemIndex);
```

功能描述：

此函数可以与函数 NT_LookAtNextPacket_A 结合使用。在从系统接收到数据包或者发生指定的超时前，函数 NT_LookAtNextPacket_A 将阻塞。特别地，当函数超时时间设置为 NT_TIMEOUT_INFINITE 时，可能会导致函数一直阻塞。此时可以从其他线程调用函数 NT_CancelWaitForPacket_A 以解除函数 NT_LookAtNextPacket_A 的阻塞，并让它返回错误代码 NT_CANCELED_ERROR。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "async");
if (result != NT_OK) {
            // handle error...
}
NT_PACKET dataPacket;
NT_STATUS result;
result = NT_ReceiveNextPacket_A(ntHandle,NT_TIMEOUT_INFINITE,&dataPacket);
switch (result) {
        case NT_OK:
                // handle received packet
                break;
        case NT_CANCELED_ERROR:
               //a different thread has called the NT_CancelWaitForPacket_A
               //function to manually unblock NT_ReceiveNextPacket_A above
                break;
        default:
                // handle other errors
                break;
}
```

NT_LookAtNextPacket_A

函数原型：

```
NT_STATUS NT_LookAtNextPacket_A(NT_INDEX systemIndex,
                                unsigned int timeout,
                                NT_Packet *packet);
```

功能描述：

此函数用于查看接收缓冲区中的数据包，但不会删除数据包。超时参数决定此函数是否阻塞。如果没有收到数据包，则返回类型为 NT_NO_PACKET_TYPE 的数据包。如果接收到了数据包，则返回接收到的数据包。此函数不会将数据包从接收缓冲区中删除，因此可以经常“查看”数据包。调用函数 NT_DiscardPacket_A 和函数 NT_ReceivePacket_A 可以删除接收缓冲区中的数据包。更多信息，请参阅第 2.3.3 节。

入口参数：

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *timeout* (unsigned 32bit)，输入 – 指定等待传入数据的时间，单位为毫秒。如果在此时间段内没有收到数据包，函数将返回 NT_NO_PACKET_TYPE 类型的数据包。设置为 NT_TIMEOUT_INFINITE 时，函数将阻塞挂起直到接收到数据包。在这种情况下，可以通过调用函数 NT_CancelWaitForPacket_A 手动解除阻塞。
- *packet* (NT_PACKET)，输出 – 如果调用成功，将保存接收到的数据包。数据包各个字段的内容取决于数据包的类型。有关数据包类型的更多信息，请参阅 4.2 节。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "async");
if (result != NT_OK) {
        // handle error...
}
// request position of positioner
result = NT_GetPosition_A(ntHandle,0);
// receive answer, but don't wait longer than five seconds
NT_PACKET packet;
result = NT_LookAtNextPacket_A(ntHandle,5000,&packet);
NT_DiscardPacket();
if ((packet.packetType == NT_POSITION_PACKET_TYPE) && (packet.channelIndex ==
0)) {
// position is in packet.data2
}
```

NT_GetPosition_A

函数原型：

```
NT_STATUS NT_GetPosition_A(NT_INDEX systemIndex,
                           NT_INDEX channelIndex);
```

功能描述：

对应此函数的同步版本，用于查询目标通道上定位台的位置。但是， 此函数仅仅发送查询命令，位置信息需要通过其他函数（如 NT_ReceiveNextPacket_A）获取。此函数只能用于带有光栅尺的定位台，传感器电源必须启用（请参阅函数 NT_SetSensorEnabled_S）。若不满足以上条件，函数将终止执行并返回错误代码。

被寻址的频道将返回 NT_POSITION_PACKET_TYPE 类型的数据包。data2 字段将保留当前位置。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "async");
if (result != NT_OK) {
        // handle error...
}
// request current position
result = NT_GetPosition_A(ntHandle,0);
NT_PACKET packet;
// wait for answer, but not longer than one second
result = NT_ReceiveNextPacket_A(ntHandle,1000,&packet);
if (result != NT_OK) {
        // handle error
} else {
        if ((packet.packetType == NT_POSITION_PACKET_TYPE) &&
            (packet.channelIndex == 0)) {
            // current position is in packet.data2
        } else {
            // handle packet otherwise
        }
}
```

NT_GetSensorEnabled_A

函数原型：

```
NT_STATUS NT_GetSensorEnabled_A(NT_INDEX systemIndex);
```

功能描述：

对应此函数的同步版本，用于查询传感器当前的工作模式。但是， 此函数仅仅发送查询命令，工作模式需要通过其他函数（如 NT_ReceiveNextPacket_A）获取。

系统将返回 NT_SENSOR_ENABLED_PACKET_TYPE 类型的数据包。data1 字段将保留当前位置。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。

示例程序：

```
unsigned int
ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "async");
if (result != NT_OK) {
        // handle error...
}
// request current sensor mode
result = NT_GetSensorEnabled_A(ntHandle,0);
NT_PACKET packet;
// wait for answer, but not longer than one second
result = NT_ReceiveNextPacket_A(mscHandle,1000,&packet);
if (result != NT_OK) {
        // handle error
} else {
        if (packet.packetType == NT_SENSOR_ENABLED_PACKET_TYPE) {
                // sensor mode is in packet.data1
        } else {
                // handle packet otherwise
        }
}
```

NT_GetStatus_A

函数原型：

```
NT_STATUS NT_GetStatus_A(NT_INDEX systemIndex,
                         NT_INDEX channelIndex);
```

功能描述：

对应此函数的同步版本，用于查询目标通道上的定位台的运动状态。但是， 此函数仅仅发送查询命令，运动状态需要通过其他函数（如 NT_ReceiveNextPacket_A）获取。

被寻址的通道将返回 NT_STATUS_PACKET_TYPE 类型的数据包。data1 字段将保留当前位置。

请参阅 4.3 节以获取状态代码列表。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "async");
if (result != NT_OK) {
        // handle error...
}
// request current status
result = NT_GetStatus_A(ntHandle,0);
NT_PACKET packet;
// wait for answer, but not longer than one second
result = NT_ReceiveNextPacket_A(ntHandle,1000,&packet);
if (result != NT_OK) {
        // handle error
} else {
        if ((packet.packetType == NT_STATUS_PACKET_TYPE) &&
            (packet.channelIndex == 0)) {
            // positioner movement status code is in packet.data1
        } else {
            // handle packet otherwise
        }
}
```

NT_GetVoltageLevel_A

函数原型：

```
NT_STATUS NT_GetVoltageLevel_A(NT_INDEX systemIndex,
                               NT_INDEX channelIndex);
```

功能描述：

对应此函数的同步版本，用于查询当前施加到指定通道定位台压电元件上的电压电平。但是，此函数仅仅发送查询命令，电压电平需要通过其他函数（如 NT_ReceiveNextPacket_A）获取。

被寻址的通道将返回 NT_VOLTAGE_LEVEL_PACKET_TYPE 类型的数据包。data1 字段将保留电压电平。

入口参数

- *systemIndex* (unsigned 32bit)，输入 – 系统索引，指向一个已初始化的系统。
- *channelIndex* (unsigned 32bit)，输入 – 通道索引，索引值从 0 开始。

示例程序：

```
unsigned int ntHandle;
const char loc[] = "usb:id:5535186640";
NT_STATUS result = NT_OpenSystem(&ntHandle, loc, "async");
if (result != NT_OK) {
        // handle error...
}
// request current voltage level
result = NT_GetVoltageLevel_A(ntHandle,0);
NT_PACKET packet;
// wait for answer, but not longer than one second
result = NT_ReceiveNextPacket_A(ntHandle,1000,&packet);
if (result != NT_OK) {
        // handle error
} else {
        if ((packet.packetType == NT_VOLTAGE_LEVEL_PACKET_TYPE) &&
            (packet.channelIndex == 0)) {
            // voltage level is in packet.data1
        } else {
            // handle packet otherwise
        }
}
```

4 附录

4.1 函数状态代码

所有库函数被调用后均会返回函数执行成功或失败的状态码，本章节列出所有可能的状态代码，并提供简短说明。

代码	标识 描述
0	NT_OK 函数调用成功。
1	NT_INITIALIZATION_ERROR 初始化库时发生错误，请先将所有系统断开与 PC 的连接并复位，然后再尝试初始化库。
2	NT_NOT_INITIALIZED_ERROR 对一个未初始化的系统进行了函数调用。在与硬件通信之前先对系统进行初始化。
3	NT_NO_SYSTEMS_FOUND_ERROR 初始化库时，PC 未检测到系统。请检查 usb 电缆的连接，并确保正确安装驱动程序。注意：控制器与计算机连接并上电后，可能需要几秒钟的时间才能检测到系统。
4	NT_TOO_MANY_SYSTEMS_ERROR 允许连接到计算机的系统数量限制为 32 个。如果您有更多控制器连接到了计算机，请断开一些。
5	NT_INVALID_SYSTEM_INDEX_ERROR 传递给函数的系统索引无效，系统索引从 0 开始，如果 N 是获得的系统数量，那么系统索引的有效范围为 0 ...（N - 1）。
6	NT_INVALID_CHANNEL_INDEX_ERROR 传递给函数的通道索引无效，通道索引从 0 开始，如果 N 是系统上可用的通道数量，那么通道索引的有效范围是 0 ...（N - 1）。
7	NT_TRANSMIT_ERROR 向硬件发送数据时发生错误，请重启系统。
8	NT_WRITE_ERROR 向硬件发送数据时发生错误，请重启系统。
9	NT_INVALID_PARAMETER_ERROR 传递给函数的参数无效。请检查功能文档中的有效范围。
10	NT_READ_ERROR 从硬件接收数据时发生错误，请重启系统。
12	NT_INTERNAL_ERROR 内部通信错误，请重启系统。
13	NT_WRONG_MODE_ERROR 所调用的函数与初始化时选择的通信模式不匹配（请参阅 2.4 节）。
14	NT_PROTOCOL_ERROR 内部协议错误，请重启系统。

15 NT_TIMEOUT_ERROR
硬件无响应，确保所有电缆连接正确，然后重启系统。
19 NT_WRONG_CHANNEL_TYPE_ERROR
当调用函数与通道类型不符时，则会发生此错误。
21 NT_INVALID_SYSTEM_LOCATOR_ERROR
如果定位字符串不符合支持的格式，则由函数 NT_OpenSystem 返回。
22 NT_INPUT_BUFFER_OVERFLOW_ERROR
当用于存储从系统接收到的数据包的输入缓冲区已满时，则会发生此错误。
23 NT_QUERYBUFFER_SIZE_ERROR
如果用户提供的缓冲区太小而无法保存返回的数据，则会发生此错误。
129 NT_NO_SENSOR_PRESENT_ERROR
如果调用的函数需要传感器反馈，但并未检测到系统有传感器连接，则发生此错误。
130 NT_AMPLITUDE_TOO_LOW_ERROR
输入的参数 amplitude 太低。
131 NT_AMPLITUDE_TOO_HIGH_ERROR
输入的参数 amplitude 太高。
132 NT_FREQUENCY_TOO_LOW_ERROR
输入的参数 frequency 太低。
133 NT_FREQUENCY_TOO_HIGH_ERROR
输入的参数 frequency 太高。
135 NT_SCAN_TARGET_TOO_HIGH_ERROR
扫描移动的目标位置太远。
136 NT_SCAN_SPEED_TOO_LOW_ERROR
扫描运动命令给出的扫描速度参数过低。
137 NT_SCAN_SPEED_TOO_HIGH_ERROR
扫描运动命令给出的扫描速度参数过高。
140 NT_SENSOR_DISABLED_ERROR
如果调用的函数需要传感器反馈，检测到系统连接传感器但被禁用，则会发送此错误。

4.2 数据包类型

注意，数据包哪些字段的数据有效取决于数据包的类型，因此，建议在使用对数据进行处理前先检查数据包的类型。

代码	标识 描述	有效字段
0	NT_NO_PACKET_TYPE 此类型的数据包不代表实际的数据包。它仅表示未接收到任何数据包。所有字段均无效。	None
1	NT_ERROR_PACKET_TYPE 如果函数执行失败或者发生其他错误，将返回此类型的数据包。channelIndex 字段指示数据包来自哪个通道，data1 字段保存错误代码（请参阅 4.1 节）	channelIndex, data1
2	NT_POSITION_PACKET_TYPE 调用函数 NT_GetPosition_A 成功将返回此数据包。channelIndex 字段指示数据包来自哪个通道，data2 字段保存当前位置（单位为纳米）。	channelIndex, data2
4	NT_STATUS_PACKET_TYPE 调用函数 NT_GetStatus_A 成功将返回此数据包。channelIndex 字段指示数据包来自哪个通道，data1 字段保存当前运动状态。	channelIndex, data1
6	NT_VOLTAGE_LEVEL_PACKET_TYPE 调用函数 NT_GetVoltageLevel_A 成功将返回此数据包。channelIndex 字段指示数据包来自哪个通道，data1 字段保存当前施加到定位台压电元件上的电平。返回值的范围是 0..4095。0 对应 0V，4095 对应 100V。	channelIndex, data1
8	NT_SENSOR_ENABLED_PACKET_TYPE 调用函数 NT_GetSensorEnabled_A 成功将返回此数据包。由于传感器的使能状态时系统全局的，因此在此类型的数据包中未定义 channelIndex。data1 字段保存当前配置的传感器模式，并且只能是以下值： NT_SENSOR_DISABLED， NT_SENSOR_ENABLED，	channelIndex, data1

4.3 通道状态码

下表列出了由函数 NT_GetStatus_S 返回的指定通道上定位台的状态代码。

代码	标识 描述
0	NT_STOPPED_STATUS 定位台当前不执行任何移动。
1	NT_STEPPING_STATUS 定位台当前正在执行开环移动。(参阅函数 NT_StepMove_S)。
2	NT_SCANNING_STATUS 定位台当前正在执行扫描移动。(如：NT_ScanMoveAbsolute_S)。
3	NT_HOLDING_STATUS 定位台保持在当前位置。(如：NT_GotoPositionAbsolute_S)。
4	NT_TARGET_STATUS 定位台当前正在执行闭环移动。
5	NT_SENSOR_CLOSED_STATUS 传感器未打开。
6	NT_NO_HOLDING_STATUS 定位台结束闭环保持状态。
7	NT_REFERENCING_STATUS 定位台正在执行寻找 index。
8	NT_CALIBRATING_STATUS 定位台执行校准状态。
10	NT_PHY_LIMIT_STATUS 定位台运动到物理极限位置。
11	NT_SOFT_LIMIT_STATUS 定位台运动到软件极限位置。
13	NT_SHORT_CIRCUIT_STATUS 定位台输出端发生短路。

纳特斯（苏州）科技有限公司

地址：苏州市高铁新城长三角国际研发社区启动区

电话：13166018168

传真：021-64283339

邮箱：info@nators.com

网址：www.nators.com

